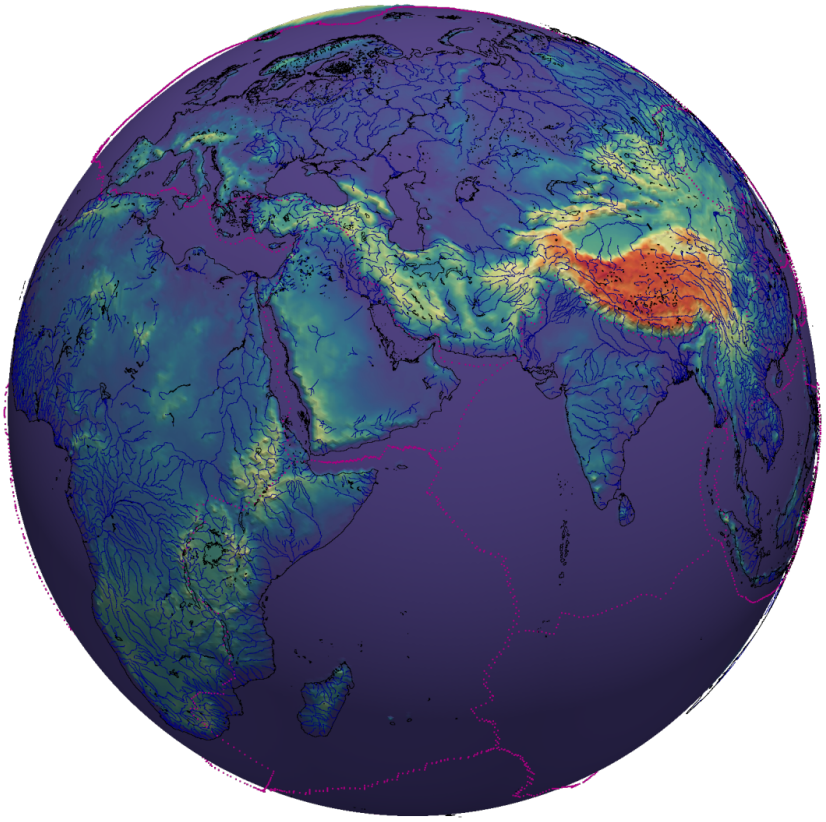
FieldStone



C. Thieulot

All the stones

C. Thieulot

January 19, 2025

Contents

1 Manufactured solution ($Q_1 \times P_0$+penalty) 19

2 Stokes sphere in 2D ($Q_1 \times P_0$+penalty) 24

3 Convection in a 2D box ($Q_1 \times P_0$+penalty) 26

4 The lid driven cavity ($Q_1 \times P_0$+penalty) 31

5 SolCx benchmark ($Q_1 \times P_0$+penalty) 34

6 SolKz benchmark ($Q_1 \times P_0$+penalty) 36

7 SolVi benchmark ($Q_1 \times P_0$+penalty) 38

8 The indentor benchmark ($Q_1 \times P_0$+penalty) 41

9 The annulus benchmark ($Q_1 \times P_0$+penalty) 47

10 Stokes sphere in 3D ($Q_1 \times P_0$+penalty, $Q_2 \times P_0$+penalty) 49

11 Stokes sphere in 3D ($Q_1 \times P_0$ - mixed) 56

12 consistent pressure recovery ($Q_1 \times P_0$+penalty) 58

13 the Particle in Cell technique (1) - the effect of averaging ($Q_1 \times P_0$+penalty) 65

14 solving the full saddle point problem ($Q_1 \times P_0$) 70

15 saddle point problem with Schur complement approach - benchmark ($Q_1 \times P_0$) 74

16 saddle point problem with Schur complement approach - Stokes sphere ($Q_1 \times P_0$) 77

17 Solving the full saddle point problem in 3D ($Q_1 \times P_0$, $Q_2 \times Q_1$) 87

18 D&H mms and more: Solving the full saddle point problem ($Q_2 \times Q_1$) 92

19 D&H mms: Solving the full saddle point problem ($Q_3 \times Q_2$) 97

20 The Busse benchmark ($Q_1 \times P_0$) 99

21 The annulus benchmark ($Q_2 \times Q_1$) 105

22 The stabilised $Q_1 \times Q_1$ element 110

23 compressible flow (1) - analytical benchmark ($Q_1 \times P_0$) 117

24 compressible flow (2) - convection box ($Q_1 \times P_0$) 120

25 Rayleigh-Taylor instability (1) - instantaneous ($Q_2 \times Q_1$, $Q_2 \times P_{-1}$) **136**

26 Slab detachment benchmark (1) - instantaneous ($Q_1 \times P_0$) **143**

27 Consistent Boundary Flux ($Q_1 \times P_0$) **150**

28 convection 2D box - Tosi *et al.* , 2015 ($Q_1 \times P_0$) **154**

29 open boundary conditions ($Q_1 \times P_0$) **167**

30 conservative velocity interpolation **170**

31 conservative velocity interpolation 3D **184**

32 2D analytical sol. from stream function ($Q_1 \times P_0$) **187**

33 Convection in an annulus ($Q_1 \times P_0$) **190**

34 2D plane strain elastic problems (Q_1) **195**

35 analytical sol. in annulus from stream function ($Q_1 \times P_0$) **199**

36 the annulus geometry elastic aquarium (Q_1) **202**

37 marker advection and population control **205**

38 convection in a 700km box with T b.c. at the top ($Q_1 \times P_0$-penalty) **207**

39 The dilation rate approach of Choi and Petersen [69] ($Q_2 \times Q_1$) **211**

40 Instantaneous Rayleigh-Taylor instability ($Q_2 \times Q_1$) **229**

41 Stokes flow and structural restoration ($Q_2 \times Q_1$) **232**

42 The no flow test ($Q_1 \times P_0$, $Q_2 \times Q_1$, $Q_3 \times Q_2$, $Q_4 \times Q_3$) **238**

43 Time-dependent advection problems (Q_1, Q_2) **242**

44 the flat slab **260**

45 Rotating cone on a triangular (P_1) mesh **262**

46 MMS1 with Crouzeix-Raviart ($P_2^+ \times P_{-1}$) elements **264**

47 triangular MINI ($P_1^+ \times P_1$) element **266**

48 D&H with $Q_1 \times P_0$, $Q_2 \times Q_1$, $Q_3 \times Q_2$and $Q_4 \times Q_3$elements **269**

49 Consistent Boundary Flux method on D&H benchmark with 4 elements **275**

50 Lithosphere extension ($Q_2 \times Q_1$) **276**

51 Convection in a triangular domain (MINI) **279**

52 Serendipity element in 2D ($Q_2 \times Q_1$, $Q_2^{(8)} \times Q_1$, $QH8 - C1 \times Q_1$) **283**

53 The sinking block benchmark ($Q_2 \times Q_1$, $Q_2^{(8)} \times Q_1$) **287**

54 Arbitrary Lagrangian Eulerian formulations **290**

55 Subduction as a thin-sheet problem **301**

56 The Diamond-square algorithm **308**

57 DG-FEM: 1D steady state diffusion **310**

58 Elastic disk under compression **314**

59 Ice Flow Down an Inclined Plane **323**

60 DG-FEM: 1D advection **330**

61 Channel flow with Herschel-Bulkley rheology ($Q_2 \times Q_1$) **333**

62 Subduction a la Quinquis Case 1 ($P_2^+ \times P_{-1}$) **336**

63 failure in cemented granular material **343**

64 Elasto-viscous benchmarks **349**

65 steady-state advection-diffusion (Q_1) **362**

66 something about sph **368**

67 Newtonian subduction setups (sinking block, subduction) **370**

68 the corner flow - Subduction benchmark (2) **384**

69 Spherical shell **396**

70 Visco-Viscoplastic rheology in crustal model ($Q_2 \times Q_1$) **399**

71 S40RTS in an annulus **404**

72 quadrilateral MINI - the $Q_1^+ \times Q_1$ element **408**

73 using python functionalities ($Q_1 \times P_0$+ penalty) **428**

74 the annulus benchmark with $Q_1^+ \times Q_1$ elements (MINI) **434**

75 the 3D stokes sphere with $Q_1^+ \times Q_1$ elements (MINI) **437**

76 mms and stokes sphere with $Q_2 \times P_{-1}$ elements **444**

77 Rotated $Q_1 \times P_0$ element & DSSY element **449**

78 $Q_1 \times P_0$ macro-element **456**

79 DG-FEM: 2D steady state diffusion **472**

80 Fortin's $Q_1^+ \times P_0$ in 2D **476**

81 Fortin's $Q_1^+ \times P_0$ in 3D **487**

82 The 3D MINI element with 2 bubble functions ($Q_1^{++} \times Q_1$) **493**

83 Thermal structure of oceanic and continental lithosphere (Q_1, Q_2) **500**

84 computing gravity above objects **505**

85 Using the S20RTS and S40RTS datasets 513

86 Thermal re-equilibrium of the Lith. Mantle during post-rift stage ($Q_1 \times P_0$) 523

87 Newton method for power-law rheologies ($Q_2 \times Q_1$) 527

88 convection in a stratified mantle ($Q_1 \times P_0$, $Q_2 \times Q_1$, $Q_3 \times Q_2$) 557

89 on the topic of integrated deformation 566

90 Thick-walled elastic cylinder under uniform boundary pressure (axisymmetric) 586

91 Stokes sphere in a cylinder, axisymmetric formulation ($Q_2 \times Q_1$) 589

92 Stokes sphere in a cylinder, axisymmetric formulation, $P_2^+ \times P_{-1}$ 592

93 Buoyancy-driven flow with ($P_2^+ \times P_{-1}$) 595

94 visualising UUP07 in Paraview 607

95 van Keken *et al.* R-T instability with mesh adaptivity ($P_2^+ \times P_{-1}$) 609

96 Blob in axisymmetric planet ($P_2 \times P_1$, $P_2^+ \times P_{-1}$) 615

97 using the CRUST 1.0 dataset 621

98 Single WINTERC layer gravity benchmark 625

99 reading in the whole WINTERC 5.4 dataset 631

100 the MOLA Mars topography & computing gravity 634

101 Y12M solver for $Q_1 \times P_0$+penalty Fortran codes 643

102 conformal mesh refinement in 2D 646

103 conformal mesh refinement in 3D 650

104 grad-div stabilisation ($Q_2 \times Q_1$, $Q_2 \times P_{-1}$) 652

105 solving the flexure equation with FDM 656

106 plumes in axisymmetric chunk ($Q_2 \times Q_1$) 660

107 Convection in a porous medium 668

108 topography above lower crustal bordering the Tibetan plateau 669

109 Poiseuille flow with obstacle in 3D ($Q_2 \times Q_1$) 674

110 Convection in a 2D box - BA vs EBA ($Q_2 \times Q_1$) 678

111 Manufactured solution for compressible elastic deformation in a disc 687

112 MINI, $P_2 \times P_1$, $Q_2 \times Q_1$, $Q_2 \times P_{-1}$ and C-R elts 691

113 Gravity of a const. density tetrahedron & hexahedron following Werner & Scheeres (1997) 710

114 Joint Gravity and Stokes inversion of a 2D circular anomaly 733
115 $Q_1 \times P_0$-stab element for isoviscous flow 736
116 $Q_1 \times P_0$-stab element with viscosity contrasts 742
117 circular inclusion with bespoke mesh 743
118 subduction initiation in the 80's ($Q_2 \times Q_1$) 745
119 on computing the lithostatic pressure in geodynamical models 750
120 2D Stokes flow. Which FE pair is best? 760
121 Olivine deformation maps (Gueydan & Précigout, 2014) 820
122 Runge-Kutta methods & particle advection 824
123 Boussinesq & Love elastic problems 829
124 Plate with a hole elastic problems 833
125 Building Voronoi cells 841
126 Darcy stuff 844
127 circular inclusion in simple shear ($P_2^+ \times P_{-1}$) 846
128 fluid flow in porous media - Simpson book chapter 10 (Q_1) 848
129 Visco-elastic materials - Simpson book chapter 12 (Q_2) 854
130 Reaction-diffusion (Q_1) - Simpson book chapter 7 871
131 creating triangular meshes with python 875
132 gravity field of a mass circle 877
133 The annulus benchmark ($Q_2 \times Q_1$) with Doube Jacobian 880
134 CMT Catalog Files 885
135 SLAB2 data visualisation 888
136 RUM slabs visualisation 890
137 Configurational entropy 892
138 Magnetic Terrain Effect 895
139 Evolution of a 1D River Profile 901
140 Evolution of a Fluvially Dissected Landscape 903
141 The 'Beaumont' mantle temperature trick 917
142 Calculating stress and strain rate in 2D viscous inclusion-matrix systems 922
143 Passive margins getting squeezed in the mantle convection vice 932

144 Dave's magma 940

145 Converting vtu files to png or ascii 941

146 Jan's project 943

147 Solving the Stokes system with LINALG and more ($Q_2 \times Q_1$) 944

148 Erik's project 953

149 van Keken et al 2008 benchmark with bespoke quad mesh 967

150 Optimisation of FE matrix building process 973

151 Free slip on curved geometries ($Q_2 \times Q_1$) 986

152 Temperature in mantle wedge (van Keken et al 2008) 998

153 Vorticity-Streamfunction - isoviscous 2D case with free slip 999

154 Vorticity-Streamfunction - isoviscous 2D case with no slip 1005

155 Vorticity-Streamfunction - Convection in 2D box 1013

156 Vorticity-Streamfunction & the Lorenz equations - Kellogg & Turcotte (1990) 1020

157 Solving the 1D diffusion equation with the Method of Lines 1034

158 Solving the Stokes Eqs. with FDM in 2D 1037

159 Computing plastic yield envelopes 1049

160 A simple look at dynamic topography 1052

161 Implementing the $Q_{2,1} \times Q_{1,2} \times Q'_1$ element 1063

162 Testing the $Q_1^\star \times P_0$ element [WIP] 1082

163 A false good idea doing parallel pic 1089

164 Solving the 1D wave equation with FEM 1094

165 Solving the 2D wave equation with FEM 1100

166 Solving the 1D wave equation with FDM 1104

167 Shear heating as a heat source, replicating Stüwe (1998) 1106

168 Edge-driven convection: a simple study by Kim and So [190] 1113

169 equilibration of partially melted metamorphic rocks, replicating Stüwe (1995) 1120

170 Title of the stone 1129

A Classification 1131
A.1 Stones I wrote/adapted/used for reviewing a paper . . . 1131
A.2 Stones which are replicating a paper . . . 1131
A.3 Stones which served as a basis for a publication . . . 1131
A.4 Stones (partially) based on BSc/MSc thesis . . . 1131

Contributors 1153
Elements 1154

Naming convention for every stone and elefant

These have evolved in the first two years so that the early stones may not follow these conventions (yet).

Variable	meaning
`mV`	number m_v of velocity nodes in the element
`mP`	number m_p of pressure nodes in the element
`mT`	number m_T of temperature nodes in the element
`ndofV`	number of velocity degrees of freedom per velocity node
`ndofP`	number of pressure degrees of freedom per pressure node
`NV`	total number of velocity nodes
`NfemV`	total number of velocity degrees of freedom
`NfemP`	total number of pressure degrees of freedom
`Nfem`	total number of degrees of freedom
`nelx,nely,nelz`	number of elements in each direction
`nel`	total number of elements
`nqel`	number of quadrature points per element
`xq,yq,zq`	coordinates of a quadrature point
`uq,vq,wq`	velocity at a quadrature point
`pq`	pressure at a quadrature point
`etaq`	viscosity value at a quadrature point
`rhoq`	density value at a quadrature point
`dt`	time step δt
`gx,gy,gz`	components of the gravity vector
`jcob`	determinant of the Jacobian matrix

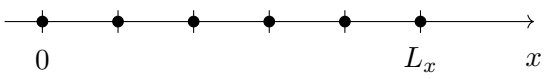
Array	meaning
iconV	connectivity array for velocity nodes
iconP	connectivity array for pressure nodes
qcoords_r	contains the r coordinates of the nel quadrature points
qcoords_s	contains the s coordinates of the nel quadrature points
qcoords_t	contains the t coordinates of the nel quadrature points
qweights	contains the weights associated to the nel quadrature points
xV,yV,zV	coordinates of the NV velocity nodes
xP,yP,zP	coordinates of the NP pressure nodes
bc_fix	boolean array of size NfemV. Is true if a boundary condition is specify at a dof
bc_val	value of the boundary condition at a given dof
u,v,w	velocity components
p	pressure
q	pressure projected onto the velocity nodes
NNNV	velocity basis functions values at a quadrature point
dNNNVdr	velocity basis functions derivatives values at a quadrature point
dNNNVds	velocity basis functions derivatives values at a quadrature point
dNNNVdt	velocity basis functions derivatives values at a quadrature point
NNNP	pressure basis functions values at a quadrature point
b_mat	elemental gradient matrix $\boldsymbol{B}$
K_el	elemental stiffness matrix $\mathbb{K}_{el}$
G_el	elemental gradient matrix $\mathbb{G}_{el}$
N_mat	elemental $\boldsymbol{N}$ matrix
a_mat	fully assembled Stokes matrix
A_sparse	Stokes matrix in lil_matrix format
exx,eyy,exy,...	components of the strain rate tensor (elemental or nodal array)
tauxx,tauyy,tauxy,..	components of the deviatoric stress tensor (elemental or nodal array)
jcb	2x2 or 3x3 Jacobian matrix
jcbi	inverse of the Jacobian matrix
rhs	right hand side vector
sol	solution vector returned by the solver

Function	meaning
NNV	function which returns the value of all velocity basis functions $\mathcal{N}_i^{\mathrm{v}}$ at a given location
dNNVdr	function which returns the value of all velocity basis functions $\mathcal{N}_i^{\mathrm{v}}$ derivative with respect to r at a given location
dNNVds	function which returns the value of all velocity basis functions $\mathcal{N}_i^{\mathrm{v}}$ derivative with respect to s at a given location
dNNVdt	function which returns the value of all velocity basis functions $\mathcal{N}_i^{\mathrm{v}}$ derivative with respect to t at a given location
NNP	function which returns the value of all pressure basis functions $\mathcal{N}_i^{P}$ at a given location
bx	
by	
bz	

Stone 00: mesh generation in 1D, 2D, 3D

The node layout

Let us start in one dimension. We wish to generate a set of N discrete and equidistant x coordinates between 0 and L_x:



The distance between two consecutive coordinates is

$$h = \frac{L_x}{N-1}$$

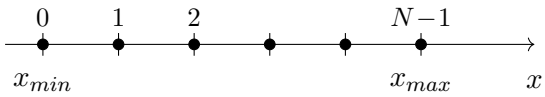
This simply translates as follows in Python:

```
N=10
Lx=1
h=Lx/(N-1)
for i in range(0,N):
    print(i*h)
```

Since i ranges from 0 to $N-1$ (because ... python!) the generated values go from 0 to L_x. Obviously I am doing this because I later wish to reuse these coordinates so I also wish to store them in an array. I therefore need to declare and array of size N which will contain all N coordinates:

```
import numpy as np
N=10
Lx=1
h=Lx/(N-1)
x=np.zeros(NV,dtype=np.float64)
for i in range(0,N):
    x[i]=i*h
```

What if now I wished the N nodes (i.e. the coordinates of all points) to be placed between two arbitrary coordinates x_{min} and x_{max}?

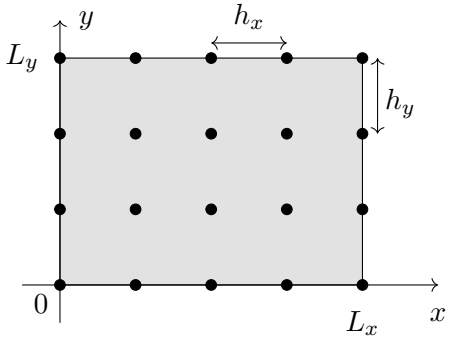


In this case the length of the domain is $L_x = x_{max} - x_{min}$, and the above code becomes:

```
import numpy as np
N=10
xmin=-4
xmax=3
h=(xmax-xmin)/(N-1)
x=np.zeros(N,dtype=np.float64)
for i in range(0,N):
    x[i]=xmin+i*h
```

Note the presence of x_{min} in the last line!

Unfortunately the world is definitely not one-dimensional, so I may want to build a two-dimensional grid spanning the domain $[0, L_x] \times [0, L_y]$ as depicted here:



There are now $N = N_x \times N_y$ nodes in the mesh, and we need to define the mesh spacing in both the x and y direction:

$$h_x = \frac{L_x}{N_x - 1} \qquad h_y = \frac{L_y}{N_y - 1}$$

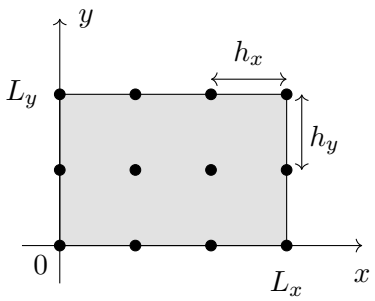
From our experience with the 1D case, it seems logical to resort to a double for loop, one in the x direction, one in the y direction. However we must now make a decision as to which loop is inside the other (inner loop vs. outer loop). In essence, I must decide between

```
#approach 1
for i in range(0,Nx):
    for j in range(0,Ny):
```

and

```
#approach 2
for j in range(0,Ny):
    for i in range(0,Nx):
```

As it turns out, I need not decide, both options are equally valid as we will see. I now fix $N_x = 4$ and $N_y = 3$ so that the mesh contains 12 nodes:



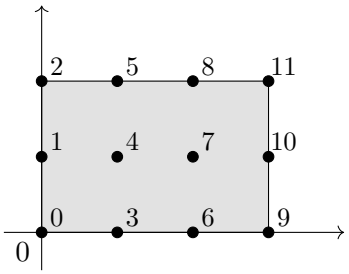
Looking at approach #1, I can include a print statement inside the loops as follows:

```
#option 1
Nx=4
Ny=3
for i in range(0,Nx):
    for j in range(0,Ny):
        print('i=',i,'; j=',j)
```

If I was to run this code it would display (I first exhaust the j values from the inner most loop before I switch to a different i value):

```
i= 0   ;  j= 0
i= 0   ;  j= 1
i= 0   ;  j= 2
i= 1   ;  j= 0
i= 1   ;  j= 1
i= 1   ;  j= 2
i= 2   ;  j= 0
i= 2   ;  j= 1
i= 2   ;  j= 2
i= 3   ;  j= 0
i= 3   ;  j= 1
i= 3   ;  j= 2
```

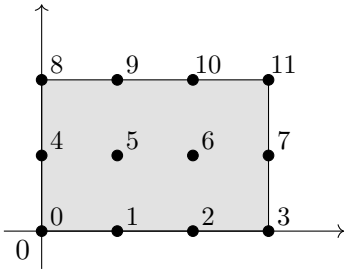
This means that the code is going through the nodes in the following order:



Turning now to approach #2, the same print statement will now yield:

```
i= 0  ;  j= 0
i= 1  ;  j= 0
i= 2  ;  j= 0
i= 3  ;  j= 0
i= 0  ;  j= 1
i= 1  ;  j= 1
i= 2  ;  j= 1
i= 3  ;  j= 1
i= 0  ;  j= 2
i= 1  ;  j= 2
i= 2  ;  j= 2
i= 3  ;  j= 2
```

and this corresponds then to the following order:



In the end, we see that both approaches yield a valid numbering scheme, one is 'lines first' , the other 'columns first'.

In what follows I focus on approach #2, only because it seems more 'natural' to me than the other one (it goes from left to right, then bottom to top). Our proto-code then looks like this:

```
Lx=10
Ly=8
Nx=4
Ny=3
N=Nx*Ny
hx=Lx/(Nx-1)
hy=Ly/(Ny-1)
x=np.zeros(N,dtype=np.float64)
y=np.zeros(N,dtype=np.float64)
for j in range(0,Ny):
    for i in range(0,Nx):
        x[?]=i*hx
        y[?]=j*hy
```

Note that I have left two question marks. One may want to replace these by i and j, following our experience in the 1D case. However, remember that i (j) ranges between 0 and $N_x-1=3$ (0 and $N_y-1=2$ respectively) but we need to assign and store the coordinates of all 12 nodes.

So what should I write instead of question marks? Effectively, I need some form of counter so that every time the inner loop is executed the counter is incremented by one. When all i, j combinations are exhausted this counter should have reached the last node. Such a counter can be implemented as follows:

```
counter=0
for j in range(0,Ny):
    for i in range(0,Nx):
        print('counter=',counter,'i=',i,';␣j=',j,j*Nx+i)
        counter+=1
```

It must be initialised to zero, and it is incremented $N_x \cdot N_y$ times. Running this code would yield

```
counter=  0 i= 0 ; j= 0
counter=  1 i= 1 ; j= 0
counter=  2 i= 2 ; j= 0
counter=  3 i= 3 ; j= 0
counter=  4 i= 0 ; j= 1
counter=  5 i= 1 ; j= 1
counter=  6 i= 2 ; j= 1
counter=  7 i= 3 ; j= 1
counter=  8 i= 0 ; j= 2
counter=  9 i= 1 ; j= 2
counter= 10 i= 2 ; j= 2
counter= 11 i= 3 ; j= 2
```

We see that the counter takes all 12 values between 0 and 11, so it is the index I am looking for. Finally, the 2D code (or approach #2) looks like:

```
Lx=10
Ly=8
Nx=4
Ny=3
N=Nx*Ny
hx=Lx/(Nx-1)
hy=Ly/(Ny-1)
x=np.zeros(N,dtype=np.float64)
y=np.zeros(N,dtype=np.float64)
counter=0
for j in range(0,Ny):
    for i in range(0,Nx):
        x[counter]=i*hx
        y[counter]=j*hy
        counter+=1
```

It is trivial to verify that one can swap the two lines with the 'for' statements to obtain the approach #1 version of the code.

Before I move to the three-dimensional case, I wish to mention a slightly different approach than the 'counter' one. Looking back at the numbering generated by approach #2, it is easy to see that the node number (i.e. a value between 0 and 11) can be also computed with $j \cdot N_x + i$. You can indeed verify that this expression yields the same values as the counter here above for all the i, j combinations. However one must realise that this formula is not valid for approach #1. The code then becomes:

```
Lx=10
Ly=8
Nx=4
Ny=3
N=Nx*Ny
hx=Lx/(Nx-1)
hy=Ly/(Ny-1)
x=np.zeros(N,dtype=np.float64)
y=np.zeros(N,dtype=np.float64)
for j in range(0,Ny):
    for i in range(0,Nx):
        x[j*Nx+i]=i*hx
        y[j*Nx+i]=j*hy
```

Also, if the domain is not $[0, L_x] \times [0, L_y]$ but $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$, the code becomes

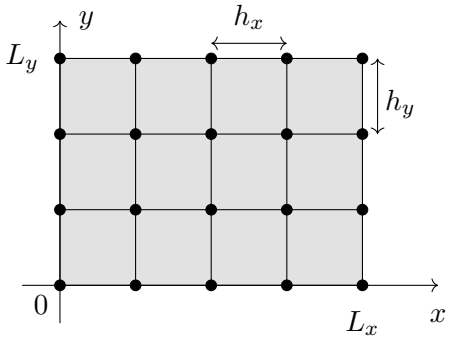
```
xmin=-3
xmax=5
ymin=-1
ymax=4
Nx=4
Ny=3
N=Nx*Ny
hx=(xmax-xmin)/(Nx-1)
hy=(ymax-ymin)/(Ny-1)
x=np.zeros(N,dtype=np.float64)
y=np.zeros(N,dtype=np.float64)
for j in range(0,Ny):
    for i in range(0,Nx):
        x[j*Nx+i]=xmin+i*hx
        y[j*Nx+i]=ymin+j*hy
```

Extending the codes above to three-dimensions is rather trivial, especially when the counter approach is used. One must simply be aware of the order of the loops (there are 3 =6 approaches).

```
xmin=-3
xmax=5
ymin=-1
ymax=4
zmin=1
zmax=6
Nx=4
Ny=3
Nz=5
N=Nx*Ny*Nz
hx=(xmax-xmin)/(Nx-1)
hy=(ymax-ymin)/(Ny-1)
hz=(zmax-zmin)/(Nz-1)
x=np.zeros(N,dtype=np.float64)
y=np.zeros(N,dtype=np.float64)
z=np.zeros(N,dtype=np.float64)
counter
for k in range(0,Nz):
    for j in range(0,Ny):
        for i in range(0,Nx):
            x[counter]=xmin+i*hx
            y[counter]=ymin+j*hy
            z[counter]=zmin+k*hz
            counter+=1
```

You will find near identical codes in (for example) Stone 1 and 10.

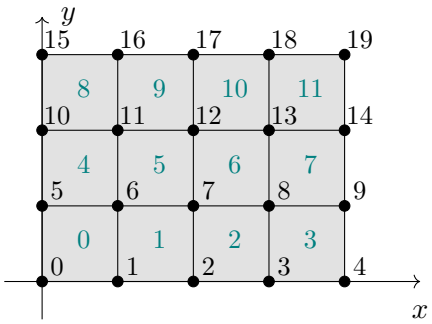
In some cases, we wish to store the coordinates of the mesh nodes because the nodes make a regular grid of quadrilaterals on which ODEs or PDEs will be solved:



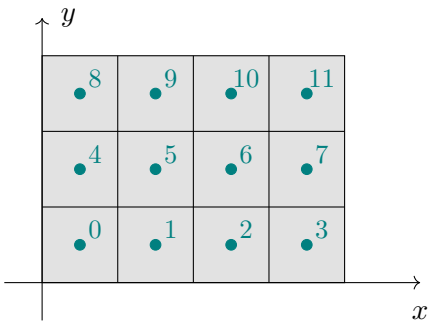
There are N_x nodes in the x direction but only $N_x - 1$ cells/elements[1]. Likewise there are N_y nodes in the y direction but only $N_y - 1$ cells/elements. In total there are $N_e = (N_x - 1)(N_y - 1)$ elements. As for

[1]Because of my bias towards the Finite Element Method, I will refer to these as elements.

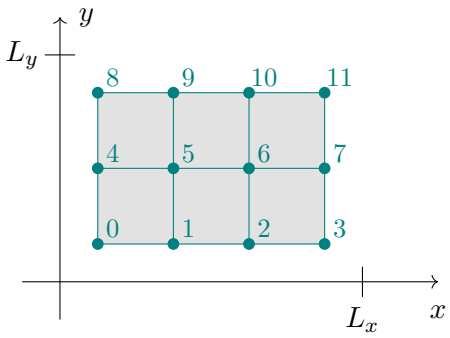
the cell center nodes, we must adopt a systematic way of numbering them. If I keep using approach #2 for the nodes, the following numbering of elements seems natural:



In some applications we want to also store the coordinates of the center of all cells, i.e. the N_e coordinates of these points:



Given this numbering these points form a regular grid:



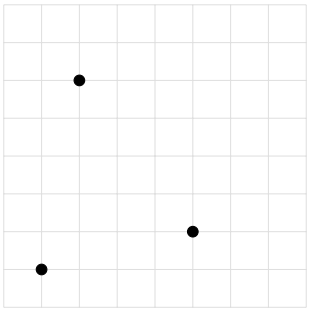
This 'new' domain is bound by $x'_{min} = h_x/2$, $x'_{max} = L_x - h_x/2$ in the x direction and $y'_{min} = h/2$, $y'_{max} = L_y - h_y/2$ in the y direction. We note that the spacings h_x and h_y between the centers is the same as for the nodes. We store the coordinates (x_c, y_c) of the cell centers in two arrays of length N_e and the code is then simply

```
Lx=9
Ly=7
Nx=4
Ny=3
N=Nx*Ny
hx=(xmax-xmin)/(Nx-1)
hy=(ymax-ymin)/(Ny-1)
#mesh nodes coordinates
x=np.zeros(N,dtype=np.float64)
y=np.zeros(N,dtype=np.float64)
counter=0
for j in range(0,Ny):
    for i in range(0,Nx):
        x[counter]=i*hx
        y[counter]=j*hy
        counter+=1
```

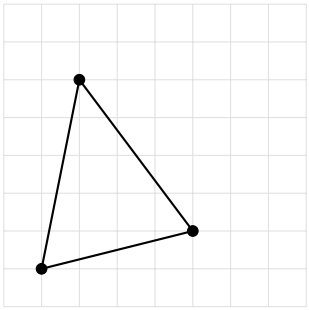
```
#cell centers coordinates
Ne=(Nx-1)*(Ny-1)
xc=np.zeros(Ne,dtype=np.float64)
yc=np.zeros(Ne,dtype=np.float64)
xmin=hx/2
xmax=Lx-hx/2
ymin=hy/2
ymax=Ly-hy/2
counter=0
for j in range(0,Ny-1):
    for i in range(0,Nx-1):
        xc[counter]=xmin+i*hx
        yc[counter]=ymin+j*hy
        counter+=1
```

The connectivity array

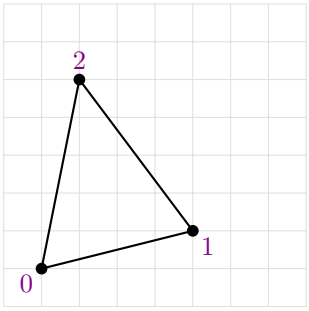
Let us consider 3 points in a plane:



Since they are not aligned they form a triangle:



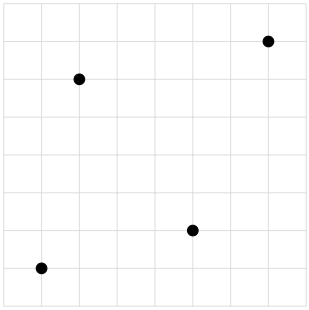
When it comes to assigning these points an identity (or a number), I can arbitrarily assign them the following numbers:



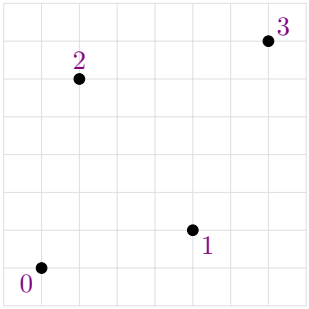
counterclockwise

This triangle is then made of nodes 0, 1, and 2.

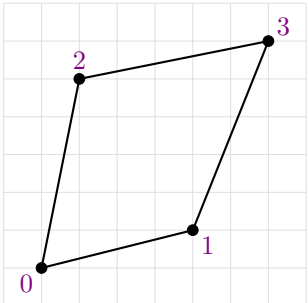
Let us now add one more point in the plane as follows:



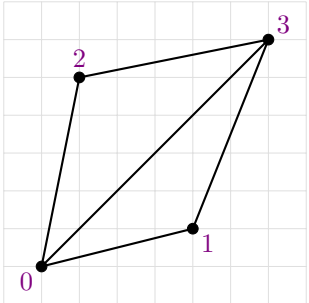
I will here number them as follows:



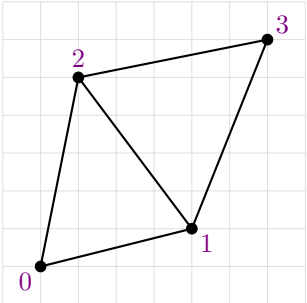
I can either decide to see them as the vertices of a quadrilateral, or as the vertices of two triangles. However, there are two possibilities for the triangles:



1 quadrilateral



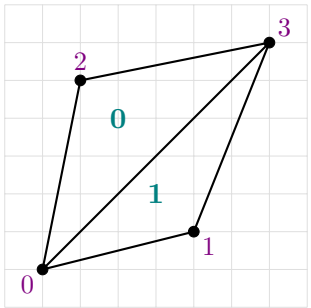
2 triangles (option 1)



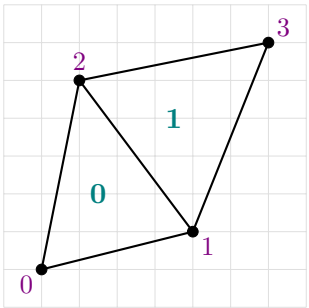
2 triangles (option 2)

When it comes to the quadrilateral, assuming a counterclockwise numbering, I can say that this quadrilaterals is then made of nodes 0, 1, 3 and 2. Note that I could also say it is made of nodes 3, 2, 0 and 1.

Looking at the cases with 2 triangles I first need to label/number the triangles themselves:



2 triangles (option 1)



2 triangles (option 2)

In option 1, and still using a counterclockwise approach, triangle 0 is made of nodes {0,3,2} and triangle 1 is made of nodes {0,1,3}. In option 2, triangle 0 is made of nodes {0,1,2} and triangle 1 is made of nodes {1,3,2}.

If I keep adding points, I will ultimately end up with a mesh which counts many quadrilaterals and/or triangles. In order to characterise this mesh, I need two numbers: *nel*, the number of elements (or cells)

and N the total number of vertices. However, this is not enough because I also need to know which vertex belongs to which element.

Concretely, I need to store for each element a list of $m = 3$ (triangles) or $m = 4$ (quadrilaterals) numbers. The resulting array is often called the connectivity array. In FieldStone this array is always `icon`, of size $m \times nel$.

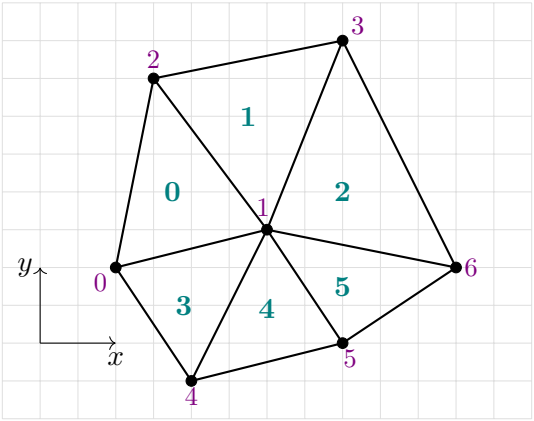
In the cases above, $m = 3$, $nel = 2$, $N = 4$ and

$$\texttt{icon}_{\text{option 1}} = \begin{pmatrix} 0 & 0 \\ 3 & 1 \\ 2 & 3 \end{pmatrix} \qquad \texttt{icon}_{\text{option 2}} = \begin{pmatrix} 0 & 1 \\ 1 & 3 \\ 2 & 2 \end{pmatrix}$$

The way to understand these is as follows: for option 1,

- icon[0,0]='the identity of node #0 of element #0' → 1
- icon[1,0]='the identity of node #1 of element #0' → 4
- icon[2,0]='the identity of node #2 of element #0' → 3
- icon[0,1]='the identity of node #0 of element #1' → 1
- icon[1,1]='the identity of node #1 of element #1' → 2
- icon[2,1]='the identity of node #2 of element #1' → 4

Finally, let us consider the following case:



This mesh counts $nel = 6$ elements/cells and $N = 7$ points. The connectivity array is then of size 3×6:

$$\texttt{icon} = \begin{pmatrix} 0 & 1 & 1 & 0 & 4 & 5 \\ 1 & 3 & 6 & 4 & 5 & 6 \\ 2 & 2 & 3 & 1 & 1 & 1 \end{pmatrix}$$

with

$$\texttt{x} = \begin{pmatrix} 1 & 3 & 1.5 & 4 & 2 & 4 & 5.5 \end{pmatrix} \tag{1}$$

$$\texttt{y} = \begin{pmatrix} 1 & 1.5 & 3.5 & 4 & \text{-} & 0 & 1 \\ & & & & 0.5 & & \end{pmatrix} \tag{2}$$

Let us then compute the barycenter coordinates of all triangles in arrays `xb` and `yb` of size $N = 7$. We need to loop over all triangles and for each compute the average of its nodes coordinates:

```
for iel in range(0,nel):
    xb[iel]=(x[icon[0,iel]]+x[icon[1,iel]]+x[icon[2,iel]])/3
    yb[iel]=(y[icon[0,iel]]+y[icon[1,iel]]+y[icon[2,iel]])/3
```

The same concepts of course apply in 3D. In the case of tetrahedra then $m = 4$ and in the case of hexahedra then $m = 8$.

Stone 1

Manufactured solution ($Q_1 \times P_0$+penalty)



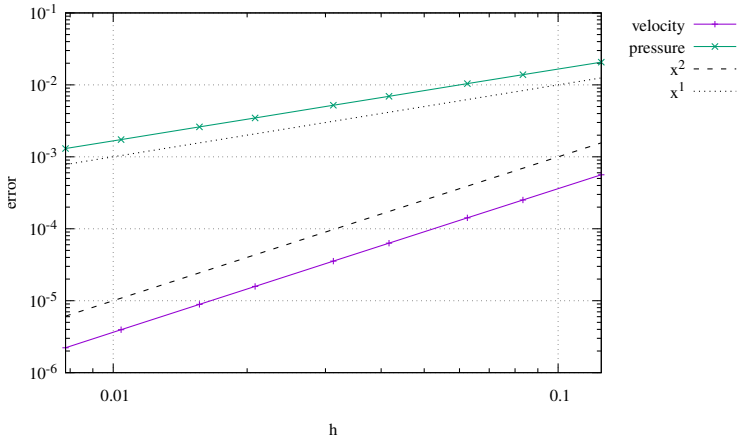
python_codes/fieldstone_01/text.tex

2D
box
Q1P0
penalty
analytical_benchmark
stokes
matplotlib
error_convergence

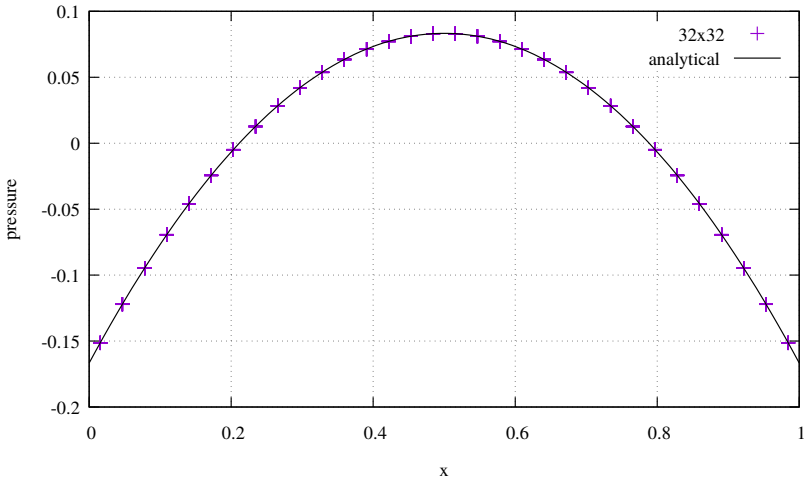
P J F Codes: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_01

The python stone was developed in collaboration with Job Mos. The julia stone was developed by Jort Jansen..

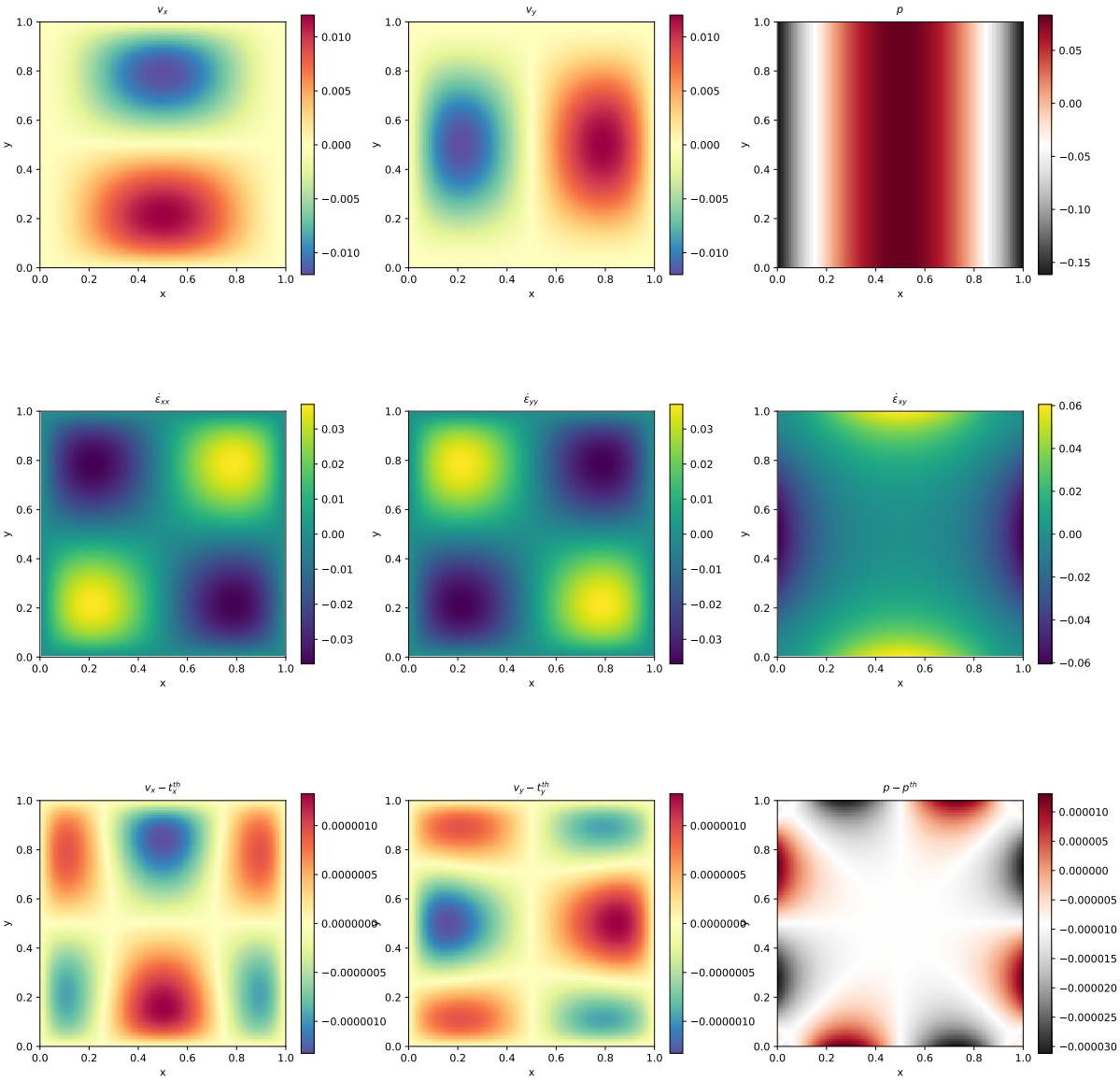
This benchmark is taken from Donea & Huerta (2003) [94] and is described fully in section 18.2. In order to illustrate the behavior of selected mixed finite elements in the solution of stationary Stokes flow, we consider a two-dimensional problem in the square domain $\Omega = [0,1] \times [0,1]$, which possesses a closed-form analytical solution.



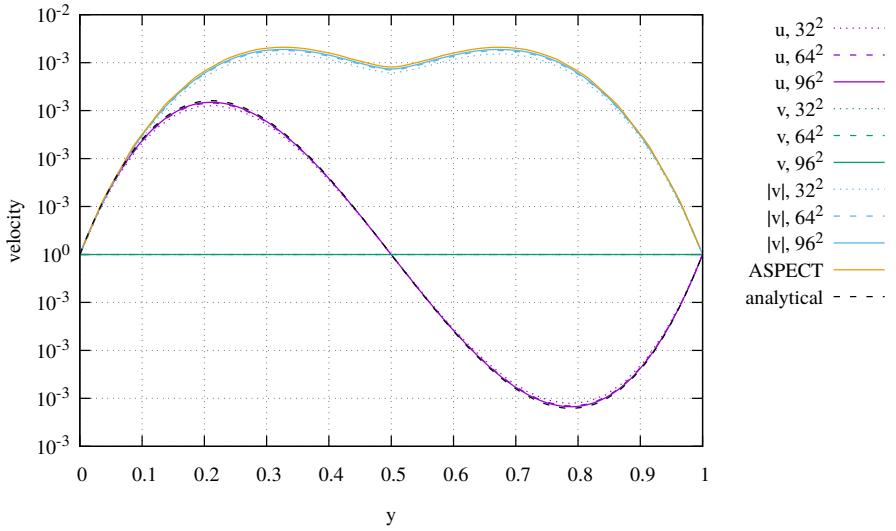
Quadratic convergence for velocity error, linear convergence for pressure error, as expected.



Pressure field



One can also compute vertical/depth averages of the velocity, see Eq. (18.13). Note that the averages are computed somewhat naively: instead of computing integrals in the x-direction at many depths I simply take an arithmetic average over rows of nodes. At high resolution the profiles converge to the analytical profiles for the individual components and to the results obtained with ASPECT . Please check STONE 110 for a better implementation.



Vertical average of velocity. ASPECT was run with the 'depth average' postprocessor added to the prm file.

About the Fortran code

I wrote the first ever version of this STONE in Fortran90 around 2011. It is available in the *simplefem* folder in this stone.

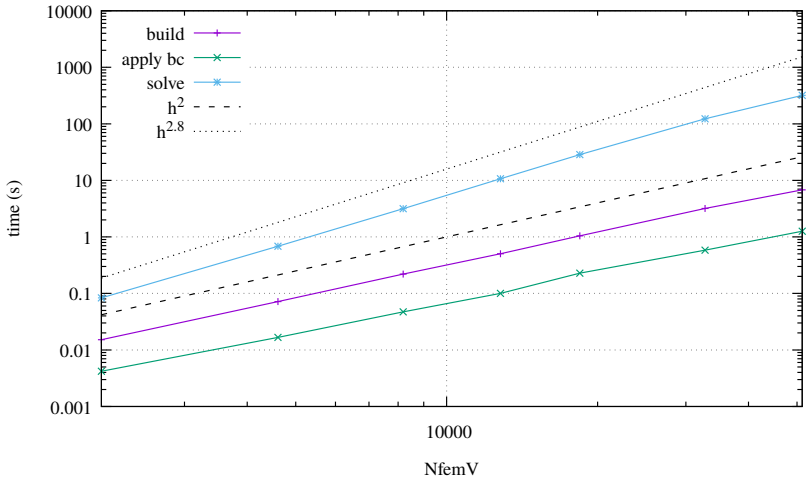
The code can be compiled as follows:

```
> gfortran -O3 linpack_d.f90 blas_routines.f simplefem.f90
```

and run as follows:

```
> ./a.out
```

The solver used here does not make use of a sparse storage and relies on a dense matrix set of subroutine from BLAS and LinPACK[1].



Despite a very naive application of the boundary conditions to the entire assembled FE matrix, we find here that this part of the code is not taking long at all.

There is no export of the results to ParaView format, only ascii text files.

[1] https://en.wikipedia.org/wiki/LINPACK

About the Julia code

The python code was translated to julia in a rather straightforward manner. The matrix is still a full array (no sparse storage during the build process) and the boundary conditions are applied naively. Even with a 80×80 resolution the memory use is under 2Gb. Note that there seems to be a limit as to the maximum size of the matrix array and for instance 96×96 yields to a crash.

In this version the matrix is stored as a full array. Before the linear system is solved, however, we must convert it to sparse storage:

```
Sps_amat=sparse(a_mat)
```

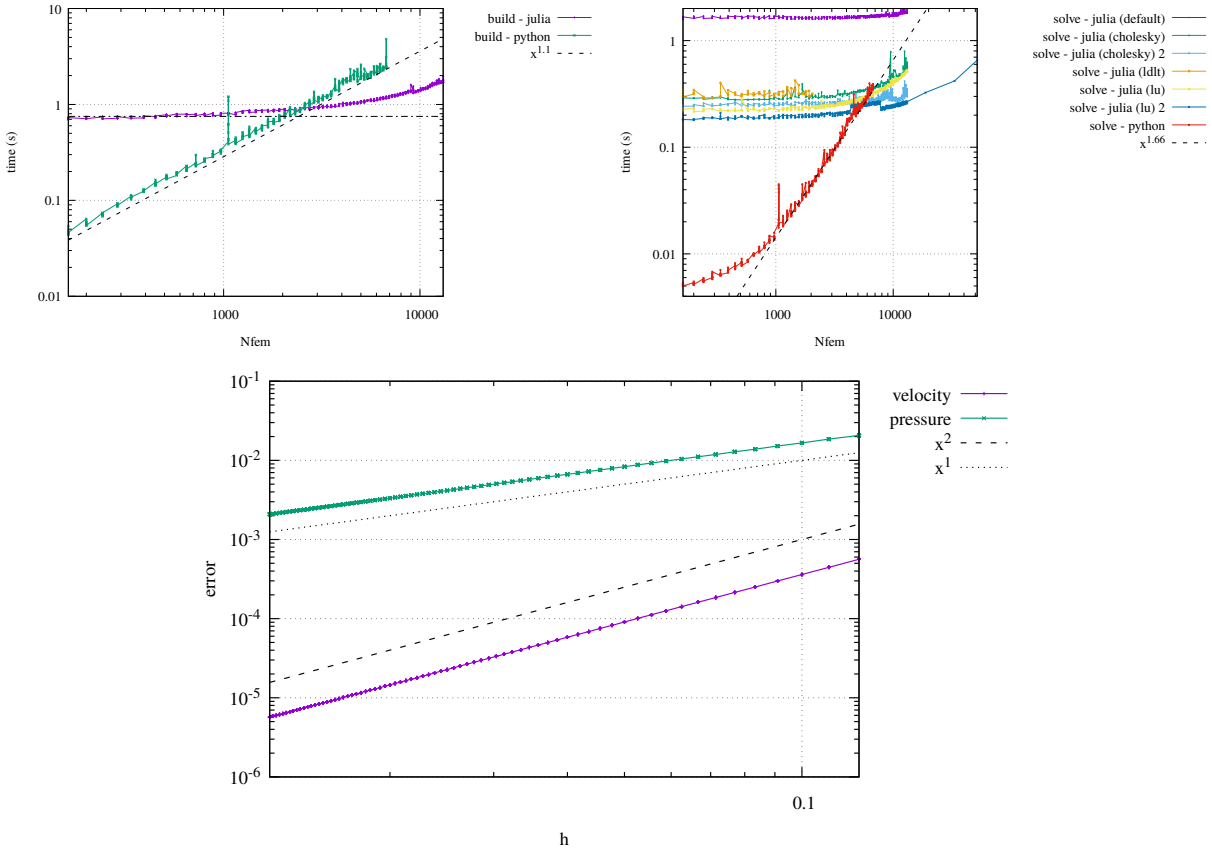
When it comes to the solve, we have various options. The first and most simple one is similar to Matlab's backslash:

```
sol= Sps_amat\rhs
```

Another possibility is cholesky[2] or ldlt or lu:

```
sol= cholesky(Sps_amat)\rhs
sol= ldlt(Sps_amat)\rhs
sol= lu(Sps_amat)\rhs
```

The computed errors are of course identical to those of the python codes but we find that at low resolutions the python code is much faster than the julia code: this is likely due to the time taken by julia to compile the code.



Matrix building and solving times as obtained for nelx from 8 until 80, each resolution being run 10 times.

We find that the LU-based solver is the fasted in this case.

[2]This method uses the CHOLMOD library from SuiteSparse: `https://people.engr.tamu.edu/davis/suitesparse.html`

Stone 2

Stokes sphere in 2D ($Q_1 \times P_0$+penalty)

FEM

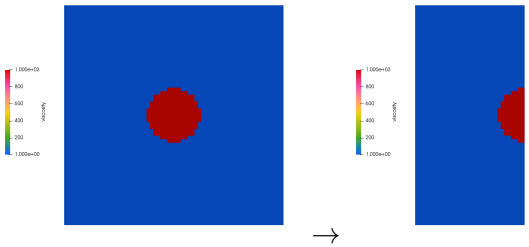
2D
box
Q1P0
vrms
penalty
buoyancy_driven
stokes_equation
viscous_rheology
paraview_output

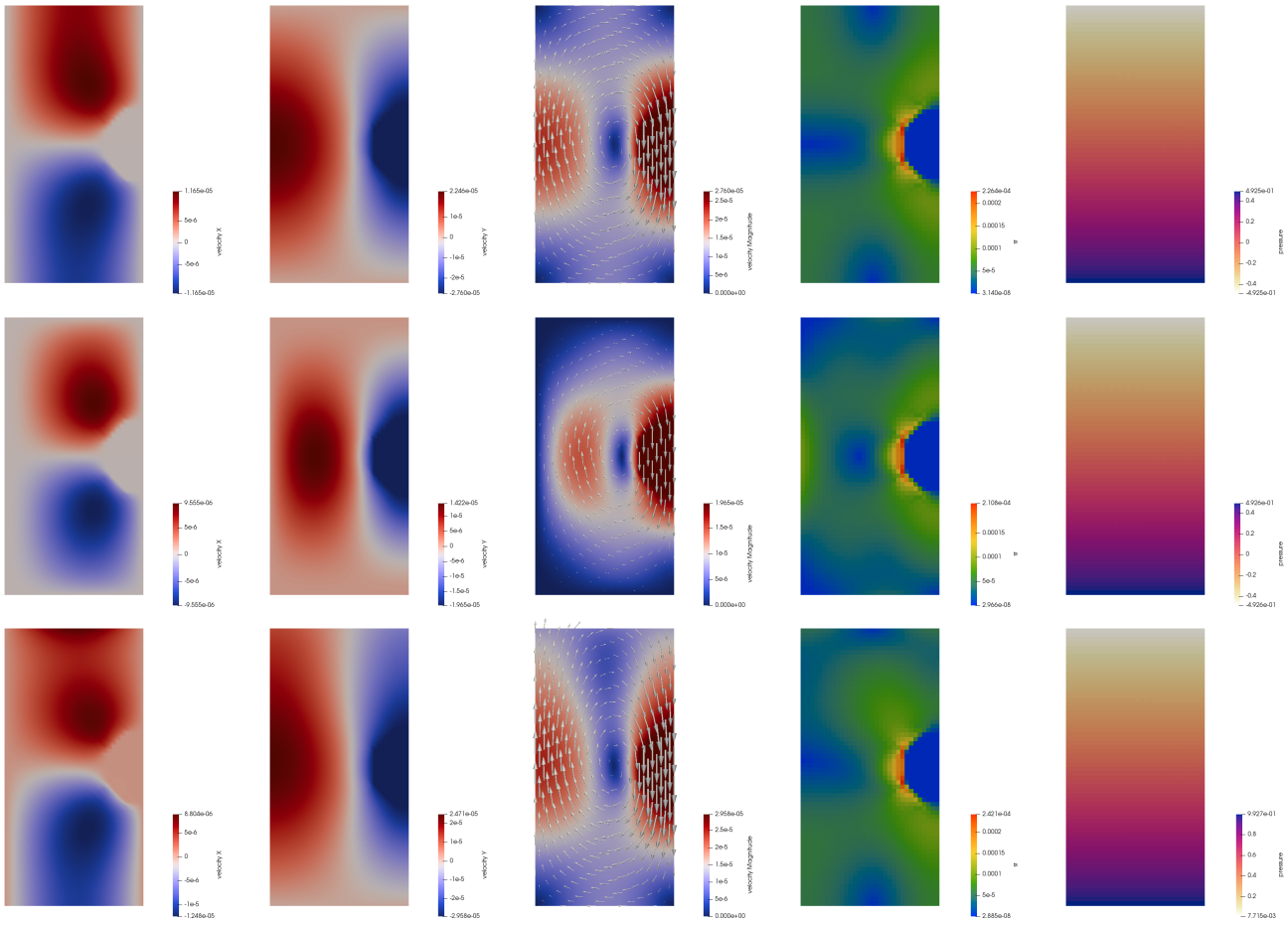
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_02`

This stone carries out the benchmark of Section 19.24 with $Q_1 \times P_0$ elements using the penalty formulation. The domain is a unit square and the fluid is characterised by $\rho = 1$ and $\eta = 1$ while the sphere is characterised by $\rho = 1.01$ and $\eta = 1000$. The gravity vector is $\vec{g} = (0, -1)$.

Boundary conditions are free slip on all sides (`bc_type=0`), no slip on all sides (`bc_type=1`) or free slip on the sides and bottom and open on top (`bc_type=2`). Viscosity and density directly computed at the quadrature points. The results are presented in Section 19.24 and compared to those obtained with other codes.

Also, because of the inherent symmetry we can solve the system on only half the domain:





Results on a 32×64 grid. Top is free-slip, middle is no-slip, bottom is open top.

Stone 3

Convection in a 2D box ($Q_1 \times P_0$+penalty)



2D
box
Q1P0
penalty
numerical_benchmark
stokes_equation
energy_equation
thermo−mechanical
time_stepping
viscous_rheology
buoyancy_driven
vrms
nusselt_number
rayleigh_number
matplotlib
steady_state

P J Codes: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_03`

The julia stone was developed by Jort Jansen..

This benchmark deals with the 2-D thermal convection of a fluid of infinite Prandtl number in a rectangular closed cell. In what follows, I carry out the case 1a, 1b, and 1c experiments as shown in Blankenbach *et al.* (1989) [23]: steady convection with constant viscosity in a square box.

The temperature is fixed to zero on top and to ΔT at the bottom, with reflecting symmetry at the sidewalls (i.e. $\partial_x T = 0$) and there are no internal heat sources. Free-slip conditions are implemented on all boundaries.

The Rayleigh number is given by

$$\mathsf{Ra} = \frac{\alpha g_y \Delta T h^3}{\kappa \nu} = \frac{\alpha g_y \Delta T h^3 \rho^2 C_p}{k \eta} \tag{3.1}$$

In what follows, I use the following parameter values: $L_x = L_y = 1$,$\rho_0 = c_P = k = \eta = 1$, $T_0 = 0$, $\alpha = 10^{-2}$, $g = 10^2 Ra$ and I run the model with $Ra = 10^4, 10^5$ and 10^6.

The initial temperature field is given by

$$T(x,y) = (1-y) - 0.01 \cos(\pi x) \sin(\pi y) \tag{3.2}$$

The perturbation in the initial temperature fields leads to a perturbation of the density field and sets the fluid in motion. Depending on the initial Rayleigh number, the system ultimately reaches a steady state after some time.

The Nusselt number (i.e. the mean surface temperature gradient over mean bottom temperature) is computed as follows [23]:

$$\mathsf{Nu} = L_y \frac{\int_0^{L_x} \frac{\partial T}{\partial y}(y = L_y)\, dx}{\int_0^{L_x} T(y = 0)\, dx} \tag{3.3}$$

Note that in our case the denominator is equal to 1 since $L_x = 1$ and the temperature at the bottom is prescribed to be 1.

If no convection is taking place (i.e. Ra is too low) then only diffusion is taking place and the steady state temperature field is then $T(y) = 1 - y$, so that $\partial_y T = -1$ and we find that the Nusselt number is simply 1.

The steady state root mean square velocity and Nusselt number measurements are indicated in the following Table alongside those of [23] and [317]. (Note that this benchmark was also carried out by many others (see Section 19.28).

		Blankenbach et al (1989)	Tackley (1994)	32x32	48x48	64x64
$\mathsf{Ra} = 10^4$	V_{rms}	42.864947 ± 0.000020	42.775	42.89111	42.87938	42.87360
	Nu	4.884409 ± 0.000010	4.878	4.783252	4.837800	4.857373
$\mathsf{Ra} = 10^5$	V_{rms}	193.21454 ± 0.00010	193.11	194.1032	193.6264	193.4519
	Nu	10.534095 ± 0.000010	10.531	9.604454	10.08087	10.26830
$\mathsf{Ra} = 10^6$	V_{rms}	833.98977 ± 0.00020	833.55			
	Nu	21.972465 ± 0.000020	21.998			

Steady state Nusselt number Nu and v_{rms} measurements as reported in the literature.

Food for thought: Looking at the mass, momentum and energy conservation equations, we see that that they are coupled: the temperature enters the rhs of the momentum equation since the density depends on the temperature (Boussinesq approximation) while the velocity is present in the advection term of the energy equation. One should then solve all three equations with u, v, p, T as unknowns. However this is rarely done in practice and often the system is solved in a segregated way: first solve for u, v, p assuming T known, then solving for T assuming u, v, p known. If small time steps are used this is a reasonable approach, or, like in this case, when one wishes to compute the steady state of the system rather than an accurate time-evolution of the system. Better schemes are available and one example thereof is explained in Kronbichler et al (2012) [200].

Also, the current version of this STONE uses a Crank-Nicolson approach for the time discretisation of the $\partial T/\partial t$ term as explained in Section 9.3.

Something must be said about how the Nusselt number is computed. Its calculation requires the integral of the temperature gradient along an edge. Because it is much simpler to compute the temperature gradient in the middle of the element alongside other quantities such as pressure, this (elemental) quantity is used in the Nusselt number calculations, which makes it inaccurate and therefore explains the discrepancy between the computed values and those of other publications. Check the Consistent Boundary Flux method in Section 15.13.

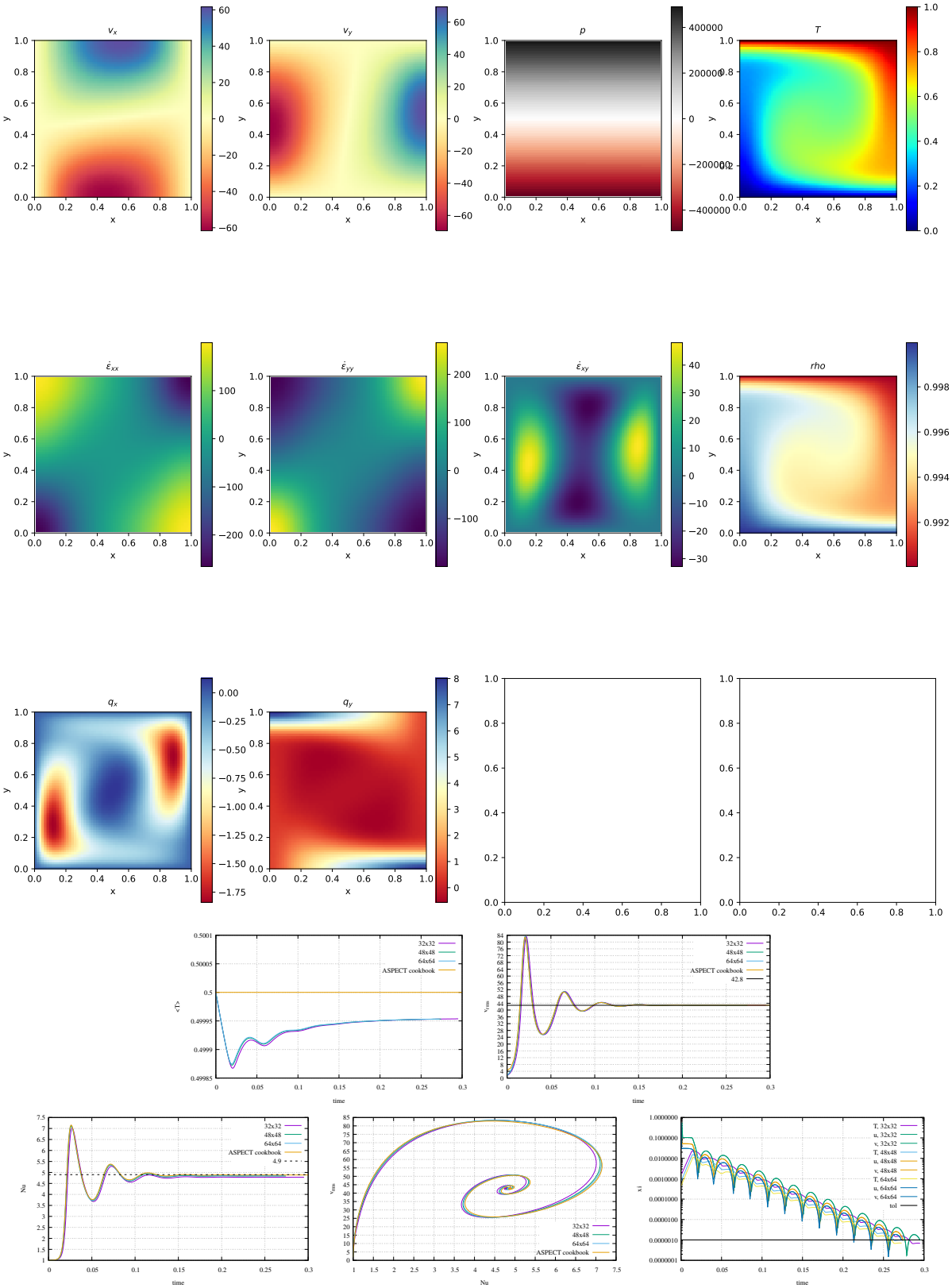
Finally, it is expected that the thickness of the boundary layers decreases with higher Rayleigh number values. As a consequence, in order to appropriately capture those, one needs a higher resolution than at low Ra numbers. This explains why acceptable results are obtained for $\mathsf{Ra} = 10^4$ even at low resolution 32×32.

Note that the results obtained with ASPECT at comparable resolution ought to be more accurate since ASPECT relies on quadratic elements for both velocity and temperature.

A few remarks about the code. It relies on $Q_1 \times P_0$ elements with a penalty formulation so that the pressure degrees of freedom are not taken into account. The temperature is also bi-linear (Q_1) so that the temperature and velocity nodes are identical and no distinction is made between them.

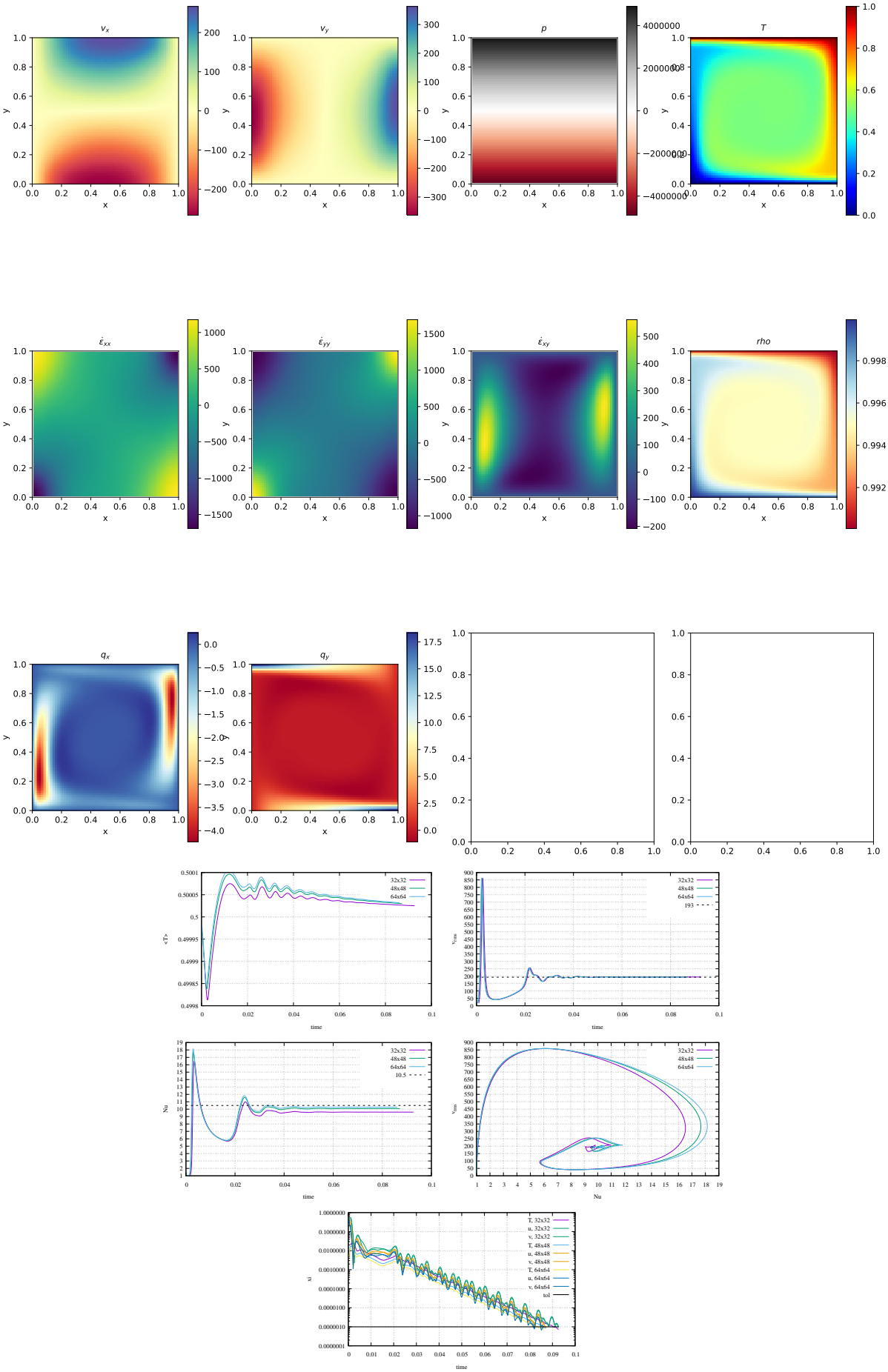
Steady state has been reached when the relative velocity and temperature difference between two consecutive time steps is less than a tolerance set to 10^{-6}.

Results for $\mathrm{Ra} = 10^4$.

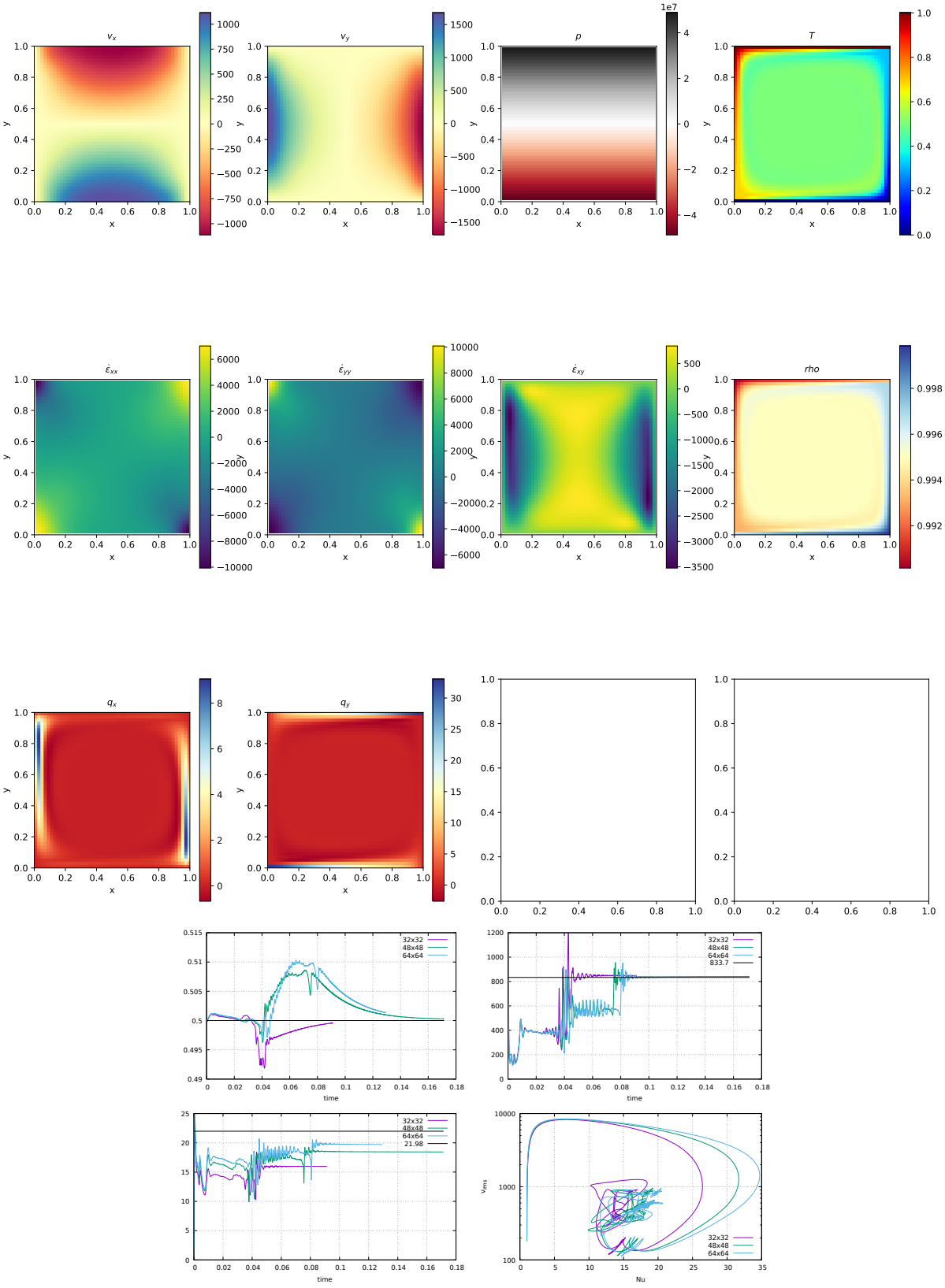


Note that the ASPECT data have been shifted so that the location of the first vrms peak coincides with the one of the 64x64 run.

Results for $\mathrm{Ra} = 10^5$.



Results for $\mathsf{Ra} = 10^6$.



I have tested the influence of the CFL number (0.5 instead of 1) and it does not change much at all.

Stone 4

The lid driven cavity ($Q_1 \times P_0$+penalty)



```
2D
box
Q1P0
penalty
stokes_equation
paraview_output
```

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_04`

The lid driven cavity is a famous Computational Fluid Dynamics test case [181, 130, 247, 30, 40, 142, 291] and has been studied in countless publications with a wealth of numerical techniques (see [108] for a succinct review) and also in the laboratory [199].

It models a plane flow of an isothermal isoviscous fluid in a rectangular (usually square) lid-driven cavity. The boundary conditions are no slip on left, right and bottom. The gravity is set to zero as the flow is entirely driven by the moving lid.

The lid driven cavity problem (`ldc=0`)

In the standard case, the upper side of the cavity moves in its own plane at unit speed, while the other sides are fixed. This thereby introduces a discontinuity in the boundary conditions at the two upper corners of the cavity and yields an uncertainty as to which boundary (side or top) the corner points belong to. In this version of the code the top corner nodes are considered to be part of the lid. If these are excluded the recovered pressure showcases an extremely large checkboard pattern.

This benchmark is usually dicussed in the context of low to very high Reynolds number with the full Navier-Stokes equations being solved (with the noticeable exception of [277, 278, 65, 102] which focus on the Stokes equation). In the case of the incompressible Stokes flow, the absence of inertia renders this problem instantaneous so that only one time step/Stokes solve is needed.

The lid driven cavity problem - regularisation I (`ldc=1`)

We avoid the top corner nodes issue altogether by prescribing the horizontal velocity of the lid as follows:

$$u(x) = 16x^2(1-x)^2. \tag{4.1}$$

In this case the velocity and its first derivative is continuous at the corners. This is the so-called regularised lid-driven cavity problem [251]. The factor 16 ensures that $\max_{[0,1]}(u) = 1$.

The lid driven cavity problem - regularisation II (`ldc=2`)

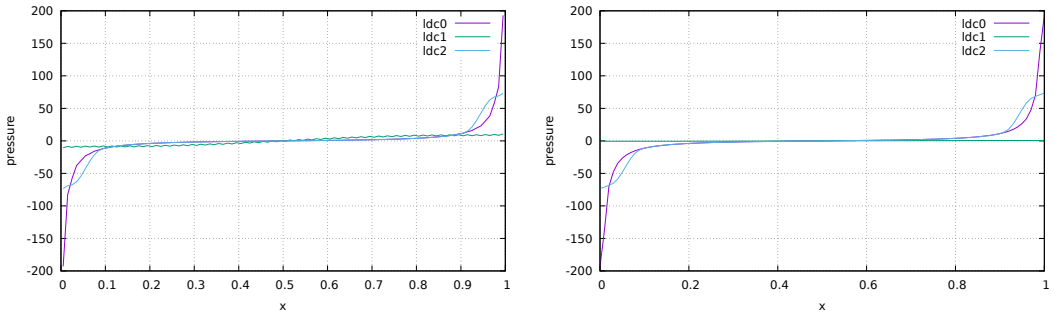
Another regularisation was presented in de Frutos *et al.* [118] and also in Appendix D.4 of John [171] (2016). Here, a regularized lid driven cavity is studied which is consistent in the sense that $\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$ holds also at the corners of the domain. There are no-slip conditions at the boundaries $x = 0$, $x = 1$, and $y = 0$.

The velocity at $y = 1$ is given by

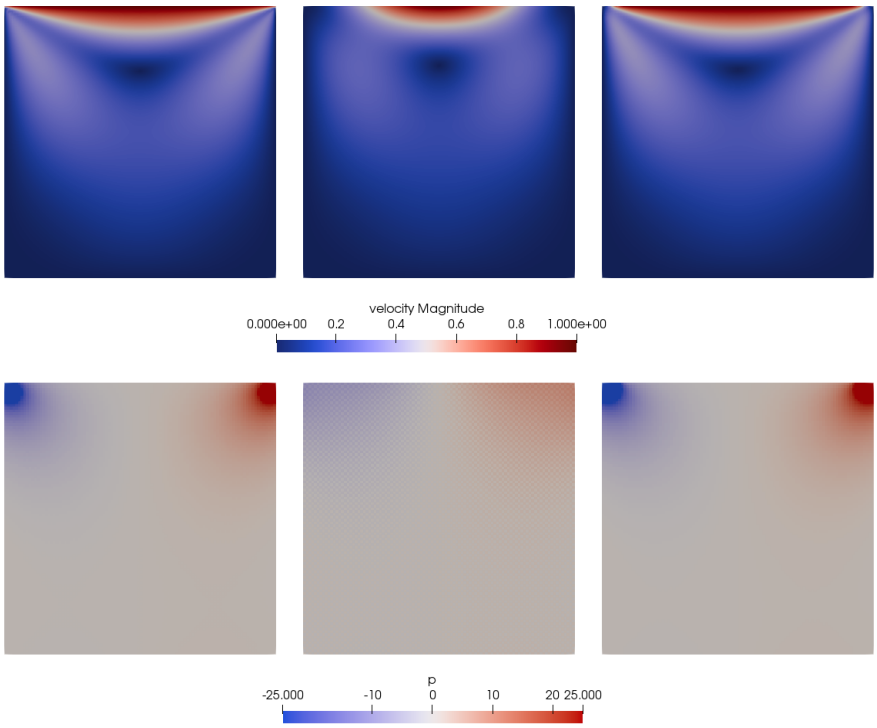
$$\begin{aligned} u(x) &= 1 - \frac{1}{4}\left(1 - \cos(\frac{x_1 - x}{x_1}\pi)\right)^2 \qquad x \in [0, x_1] \\ u(x) &= 1 \qquad x \in [x_1, 1 - x_1] \\ u(x) &= 1 - \frac{1}{4}\left(1 - \cos(\frac{x - (1 - x_1)}{x_1}\pi)\right)^2 \qquad x \in [1 - x_1, 1] \end{aligned} \tag{4.2}$$

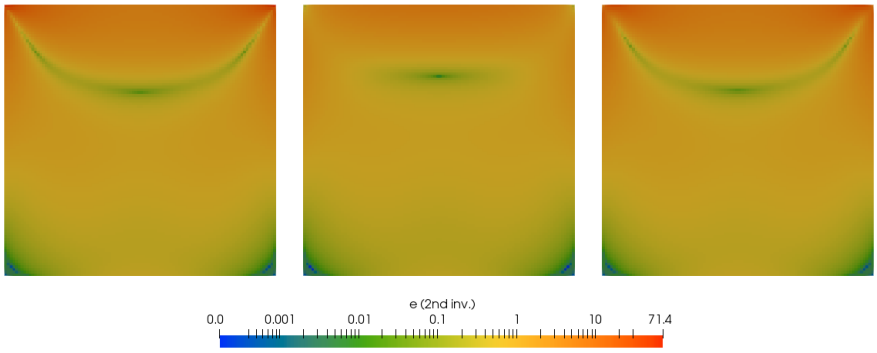
Results are obtained with $x_1 = 0.1$.

A 100×100 element grid is used. A zero vertical velocity is prescribed at the top and the exact form of the prescribed horizontal velocity is controlled by the `ldc` parameter.



Left: pressure p at $y = L_y$; Right: smoothed pressure q at $y = L_y$.




e (2nd inv.)
0.0
0.001
0.01
0.1
1
10
71.4

Stone 5

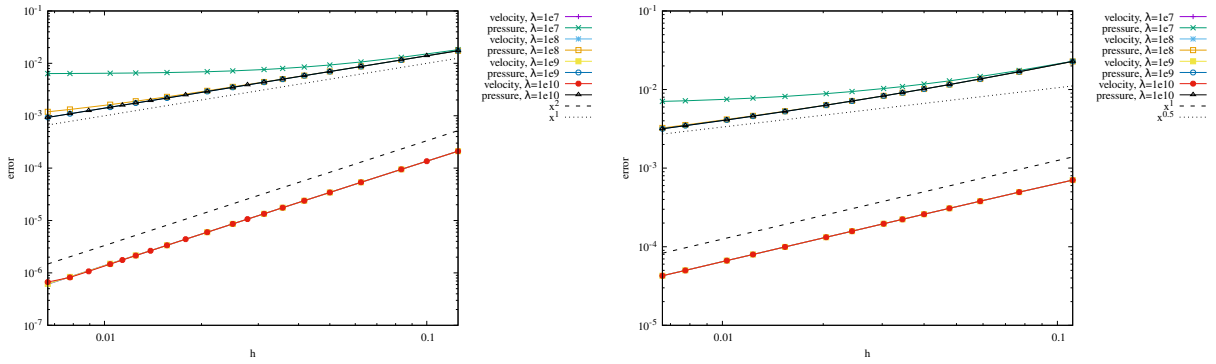
SolCx benchmark ($Q_1 \times P_0$+penalty)



P F Codes: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_05`

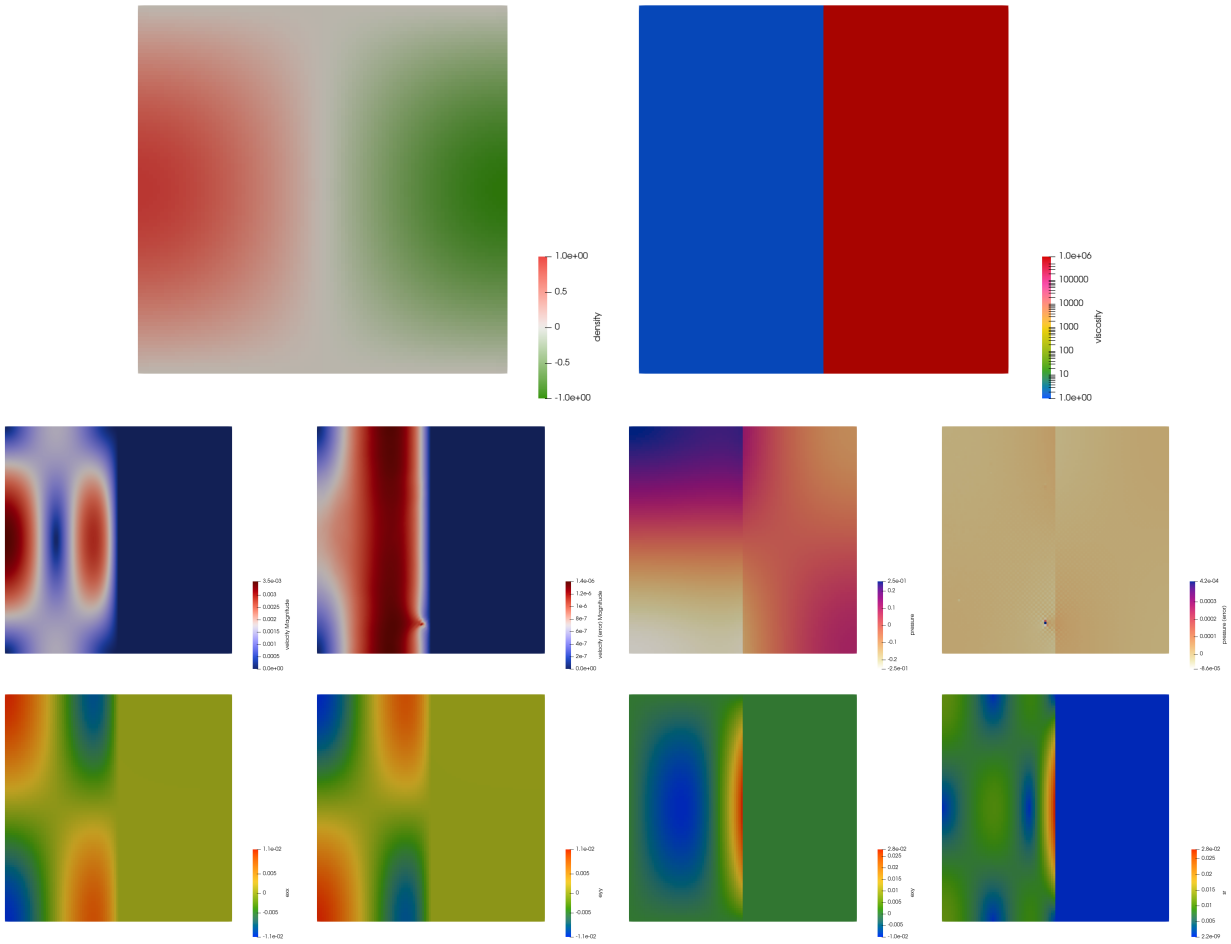
Last revision: Oct. 28th, 2024.

The experiment is fully described in Section 18.3. The viscosity is prescribed at the quadrature points If the number of elements is even in the x-direction direction, all elements (and their associated quadrature points) have a constant viscosity (1 or 10^6). If it is odd, then the elements situated at the viscosity jump have half their integration points with $\eta = 1$ and half with $\eta = 10^6$ (which is a pathological case since the used quadrature rule inside elements cannot represent accurately such a jump).



Velocity and pressure error convergence as a function of mesh size and for various values of the penalty parameter. Left: even number of elements in each direction; Right: odd numbers.

Because of the high viscosity in the right part of the domain, the penalty parameter should be high enough to insure an incompressible flow and thereby recover the expected convergence rate (at least for the even case). Note that values higher than $\lambda = 10^{10}$ yield erroneous solutions due to round-off errors.



Results obtained on 100×100 mesh for $\lambda = 10^{10}$.

F Note that a Fortran90 version of this code is available in the same folder.

Stone 6

SolKz benchmark ($Q_1 \times P_0$+penalty)



P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_06`

Last revision: Oct. 28th, 2024.

The SolKz benchmark [260] is similar to the SolCx benchmark. but the viscosity is now a function of the space coordinates:

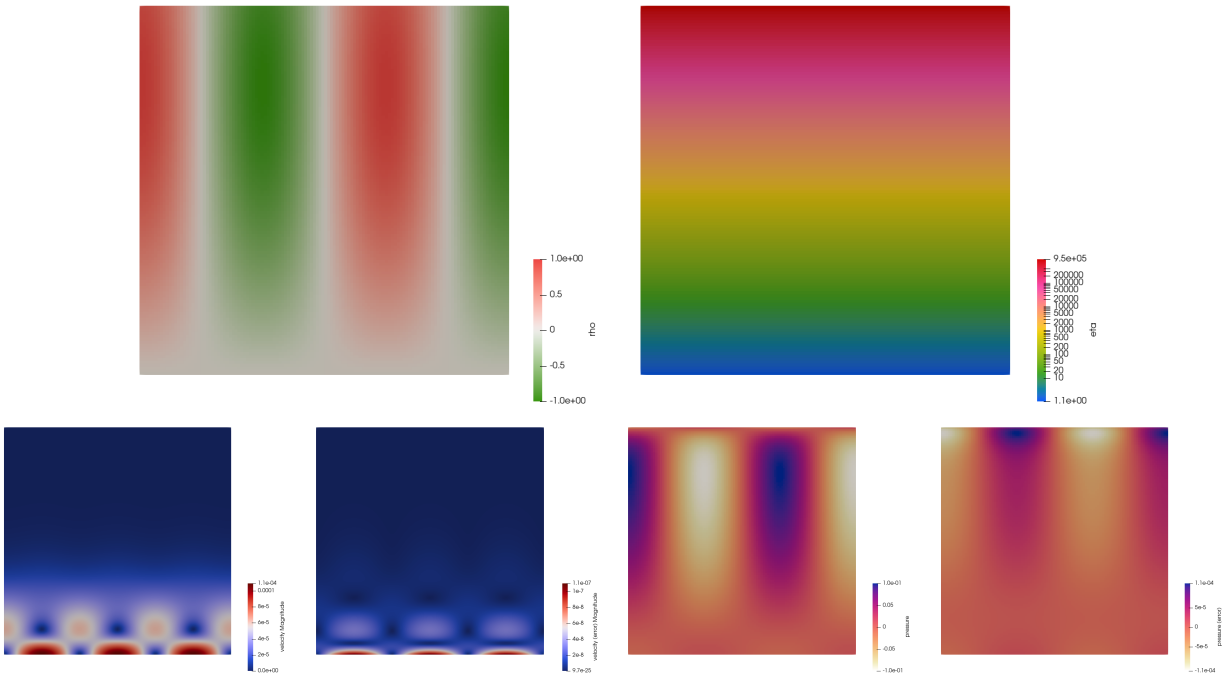
$$\eta(y) = \exp(By) \quad \text{with} \quad B = 13.8155 \tag{6.1}$$

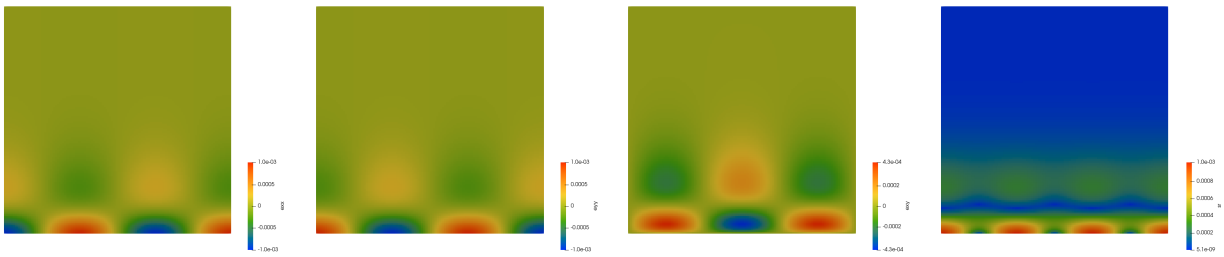
It is however not a discontinuous function but grows exponentially with the vertical coordinate so that its overall variation is again 10^6. The forcing is again chosen by imposing a spatially variable density variation as follows:

$$\rho(x,y) = \sin(2y)\cos(3\pi x) \tag{6.2}$$

Free slip boundary conditions are imposed on all sides of the domain. This benchmark is presented in Zhong (1996) [371] as well and is studied in Duretz *et al.* (2011) [97] and Gerya *et al.* (2013) [126].

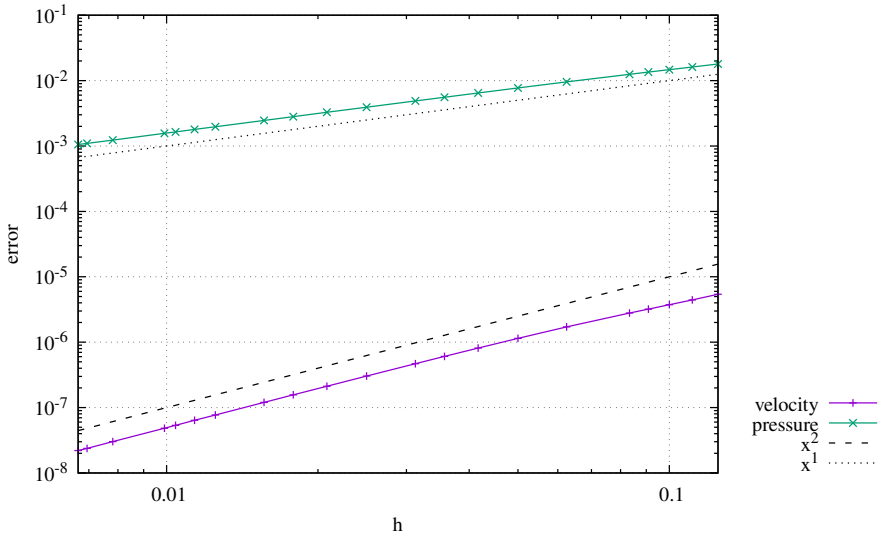
Note that the matrx storage and imposition of boundary conditions is still of the naive type in the code.





Results obtained on 128×128 mesh.

We recover a quadratic convergence for the velocity error and a linear convergence for the pressure error:



Velocity and pressure error as a function of mesh size.

Stone 7

SolVi benchmark ($Q_1 \times P_0$+penalty)



2D
box
Q1P0
penalty
analytical_benchmark
error_convergence
stokes_equation
vrms
matplotlib

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_07`

Following SolCx and SolKz, the SolVi inclusion benchmark solves a problem with a discontinuous viscosity field, but in this case the viscosity field is chosen in such a way that the discontinuity is along a circle. Given the regular nature of the grid used by a majority of codes and the present one, this ensures that the discontinuity in the viscosity never aligns to cell boundaries. This in turns leads to almost discontinuous pressures along the interface which are difficult to represent accurately. Schmid & Podlachikov (2003) [283] derived a simple analytic solution for the pressure and velocity fields for a circular inclusion under simple shear and it was used in [88, 315, 97, 200, 126].

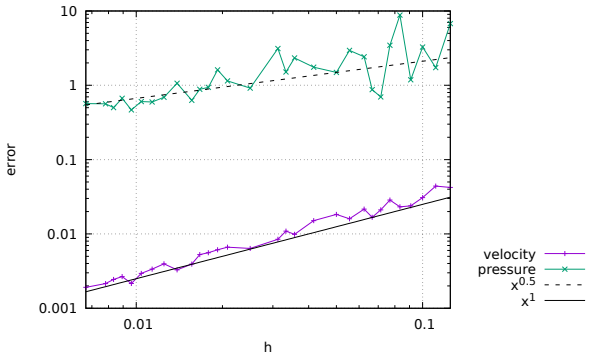
Because of the symmetry of the problem, we only have to solve over the top right quarter of the domain. The analytical solution requires a strain rate boundary condition (e.g., pure shear) to be applied far away from the inclusion. In order to avoid using very large domains and/or dealing with this type of boundary condition altogether, the analytical solution is evaluated and imposed on the boundaries of the domain. By doing so, the truncation error introduced while discretizing the strain rate boundary condition is removed.

A characteristic of the analytic solution is that the pressure is zero inside the inclusion, while outside it follows the relation

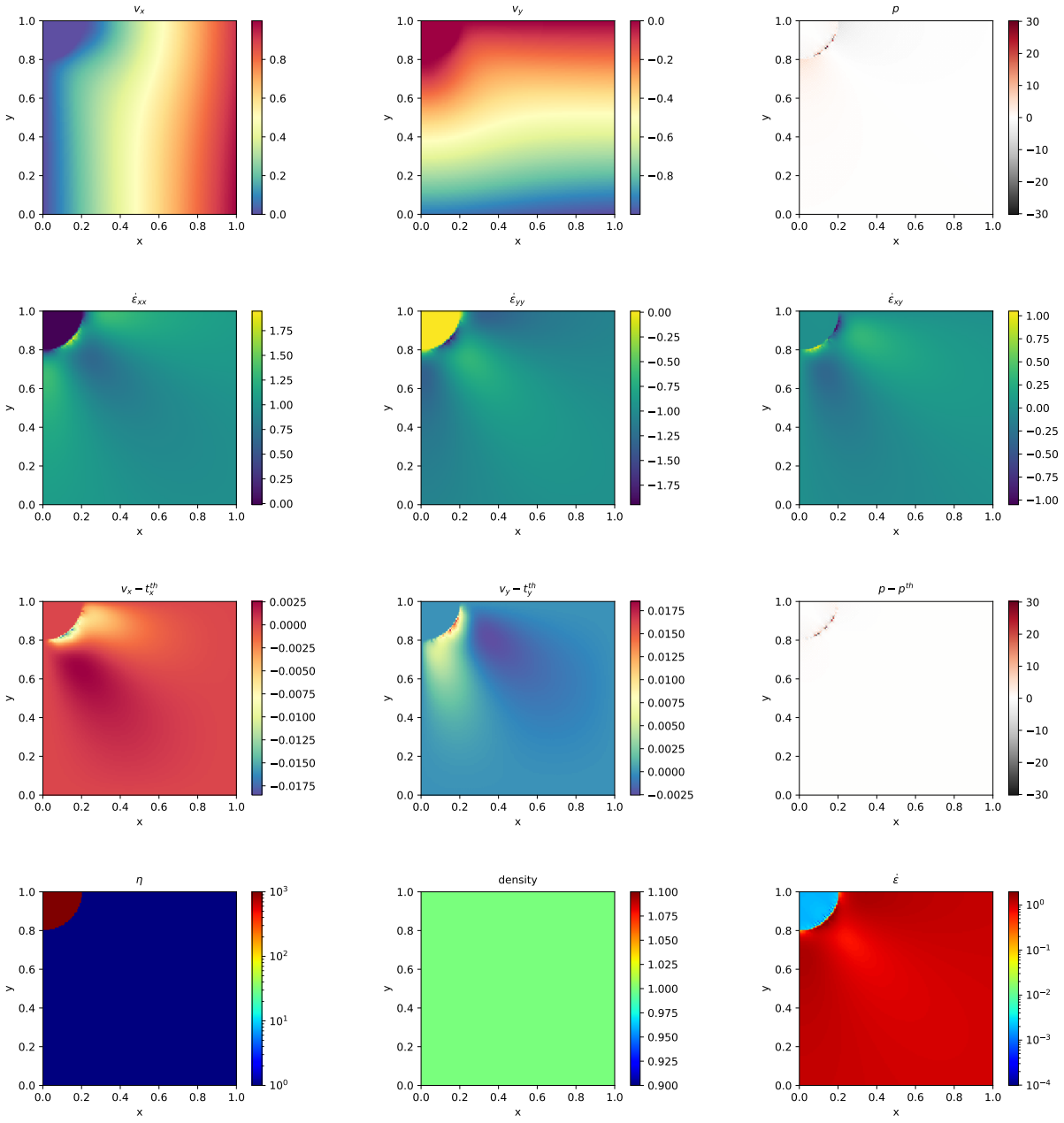
$$p_m = 4\dot{\epsilon}\frac{\eta_m(\eta_i-\eta_m)}{\eta_i+\eta_m}\frac{r_i^2}{r^2}\cos(2\theta) \tag{7.1}$$

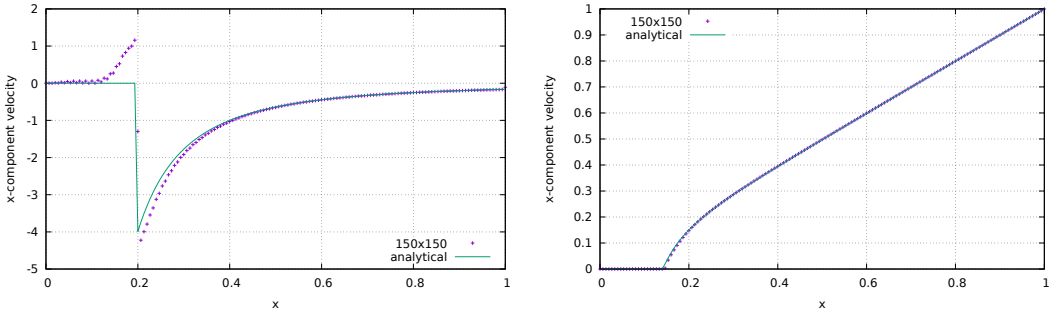
where $\eta_i = 10^3$ is the viscosity of the inclusion and $\eta_m = 1$ is the viscosity of the background media, $\theta = \tan^{-1}(y/x)$, and $\dot{\epsilon} = 1$ is the applied strain rate.

Deubelbeiss & Kauss (2008) [88] thoroughly investigated this problem with various numerical methods (FEM, FDM), with and without tracers, and conclusively showed how various averagings lead to different results. Duretz et al (2011) [97] obtained a first order convergence for both pressure and velocity, while Kronbichler et al (2012) [200] and Gerya et al (2013) [126] showed that the use of adaptive mesh refinement in respectively the FEM and FDM yields convergence rates which depend on refinement strategies.



Velocity and pressure error convergence as a function of mesh size.





Left: Pressure at the bottom of the domain. Right: u on the diagonal $x = y$.

Stone 8

The indentor benchmark ($Q_1 \times P_0$+penalty)



2D
box
Q1P0
penalty
analytical_benchmark
stokes_equation
nonlinear
viscous_rheology
plastic_rheology
residual

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_08`

The punch benchmark is one of the few boundary value problems involving plastic solids for which there exists an exact solution. Such solutions are usually either for highly simplified geometries (spherical or axial symmetry, for instance) or simplified material models (such as rigid plastic solids) [175].

In this experiment, a rigid punch indents a rigid plastic half space; the slip line field theory gives exact solutions as shown in section 19.10. The plane strain formulation of the equations and the detailed solution to the problem were derived in the Appendix of Thieulot, Fullsack, and Braun [330] (2008) and are also presented in Gerbault, Poliakov, and Daignieres [125] (1998).

The two dimensional punch problem has been extensively studied numerically for the past 40 years, see for example Zienkiewicz, Humpheson, and Lewis [373] (1975), Zienkiewicz, Huang, and Pastor [372] (1995), Christiansen and Andersen [72] (1999), Huh, Lee, and Yang [161] (1999), Christiansen and S. Pedersen [73] (2001), Yu and Tin-Loi [359] (2006), Rabczuk, Areias, and Belytschko [255] (2007), Bui, Fukugawa, Sako, and Ohno [47] (2008). It has been used to draw a parallel with the tectonics of eastern China in the context of the India-Eurasia collision, see Tapponnier and Molnar [319] (1976) and Molnar and Tapponnier [236] (1977). It is also worth noting that it has been carried out in one form or another in series of analogue modelling articles concerning the same region, with a rigid indenter colliding with a rheologically stratified lithosphere [250, 86, 173].

Numerically, the one-time step punch experiment is performed on a two-dimensional domain of purely plastic von Mises material. Given that the von Mises rheology yield criterion does not depend on pressure (see Section 5.11), the density of the material and/or the gravity vector is set to zero. Sides are set to free slip boundary conditions, the bottom to no slip, while a vertical velocity $(0, -v_p)$ is prescribed at the top boundary for nodes whose x coordinate is within $[L_x/2 - \delta/2, L_x/2 + \delta/2]$.

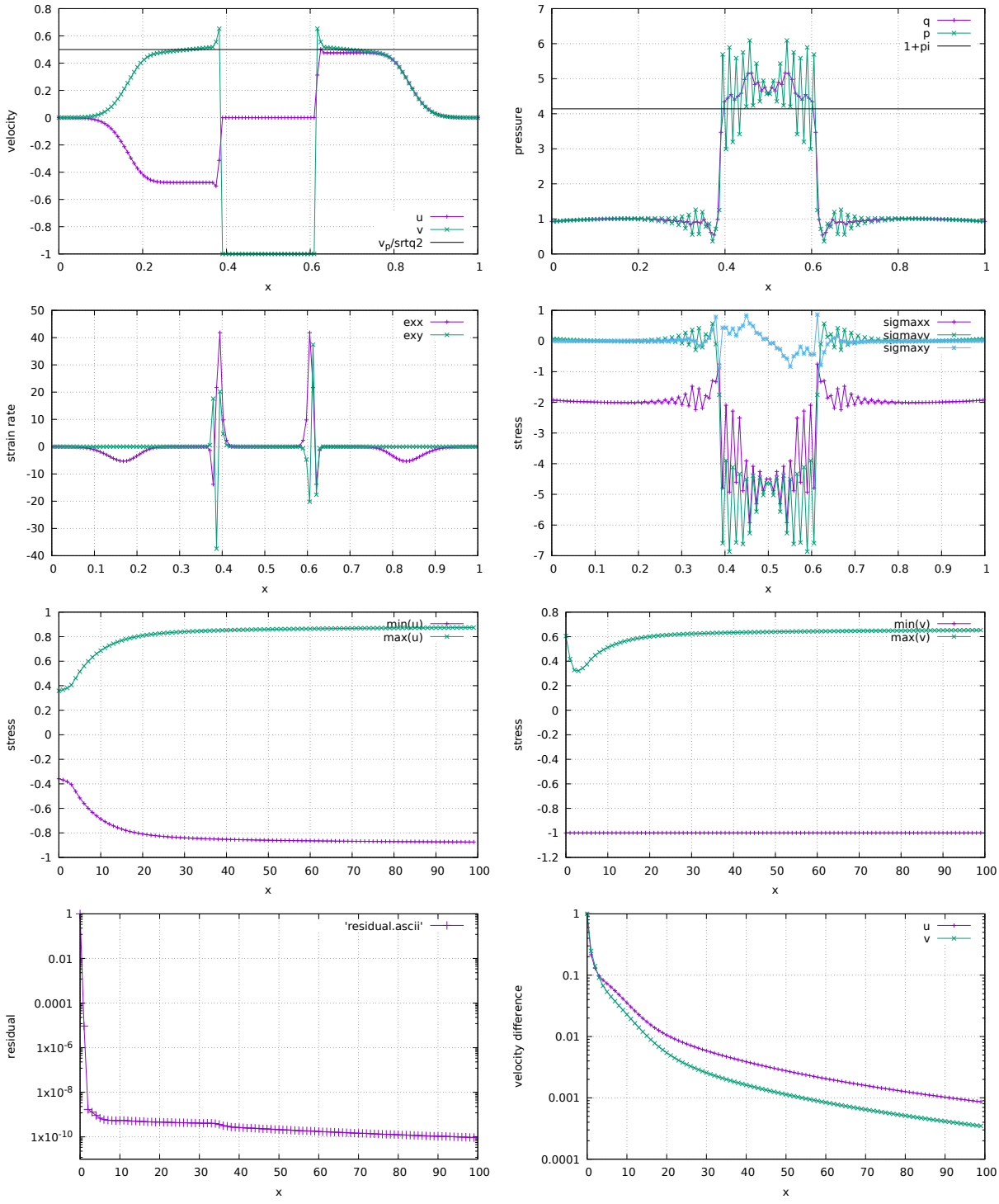
The following parameters are used: $L_x = 1$, $L_y = 0.5$, $\eta_{min} = 10^{-3}$, $\eta_{max} = 10^3$, $v_p = 1$, $\delta = 0.11111$ and the yield value of the material is set to $\sigma_Y = 1$.

The analytical solution predicts that the angle of the shear bands stemming from the sides of the punch is $\pi/4$, that the pressure right under the punch is $1 + \pi$, and that the velocity of the rigid blocks on each side of the punch is $v_p/\sqrt{2}$ (this is simply explained by invoking conservation of mass).

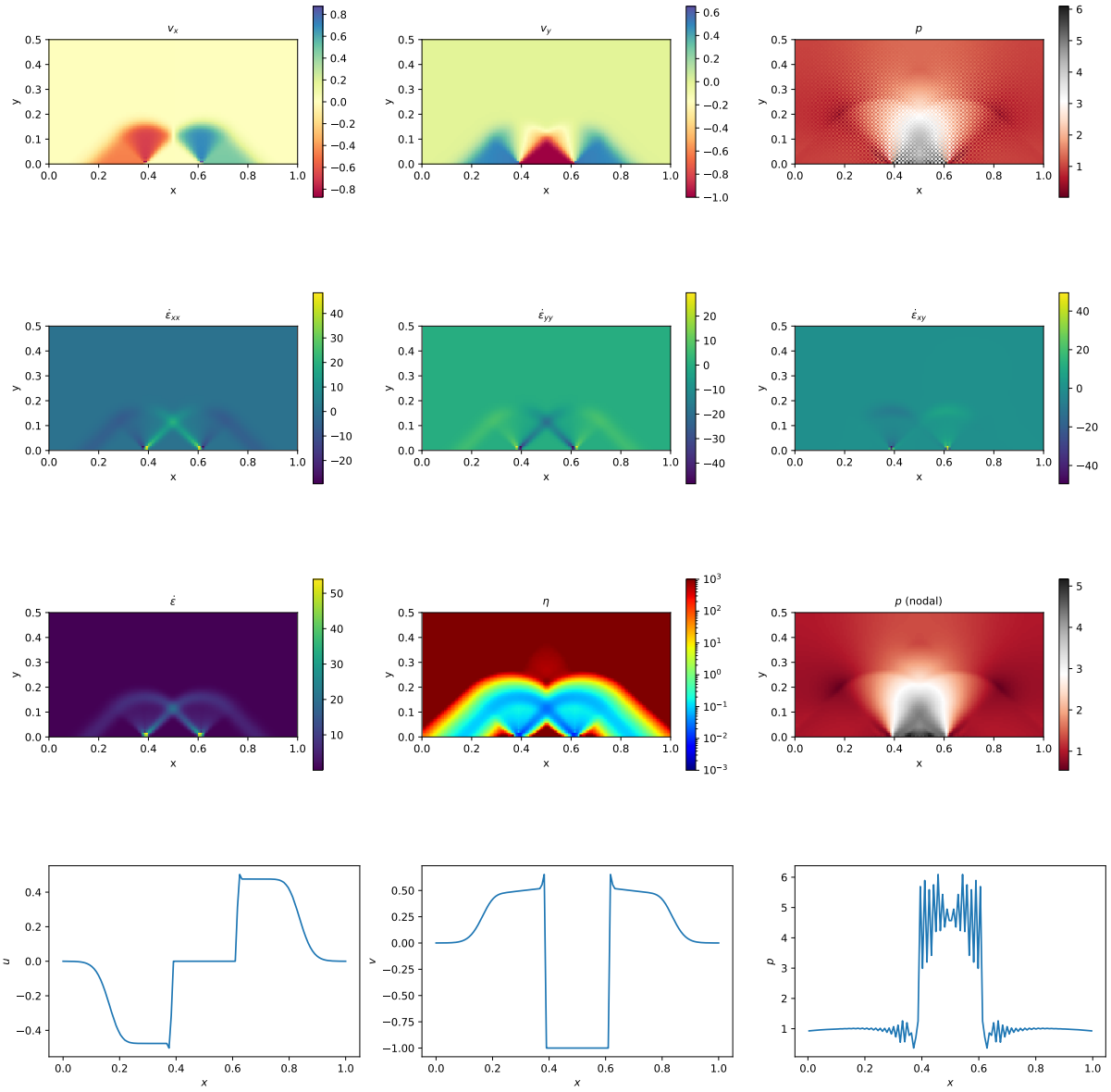
In what follows I show results of the rough and smooth punch for a 124×68 mesh. The difference between the two lies in the nature of the kinematic boundary conditions under the punched area. 'rough' means that the indentor also fixes the horizontal velocity component to zero while it is free in the smooth case.

We see that the smooth punch does not trigger the checkerboard pressure modes as much as the rough case and we recover nicely the analytical pressure under the punch [330, 133].

Rough punch

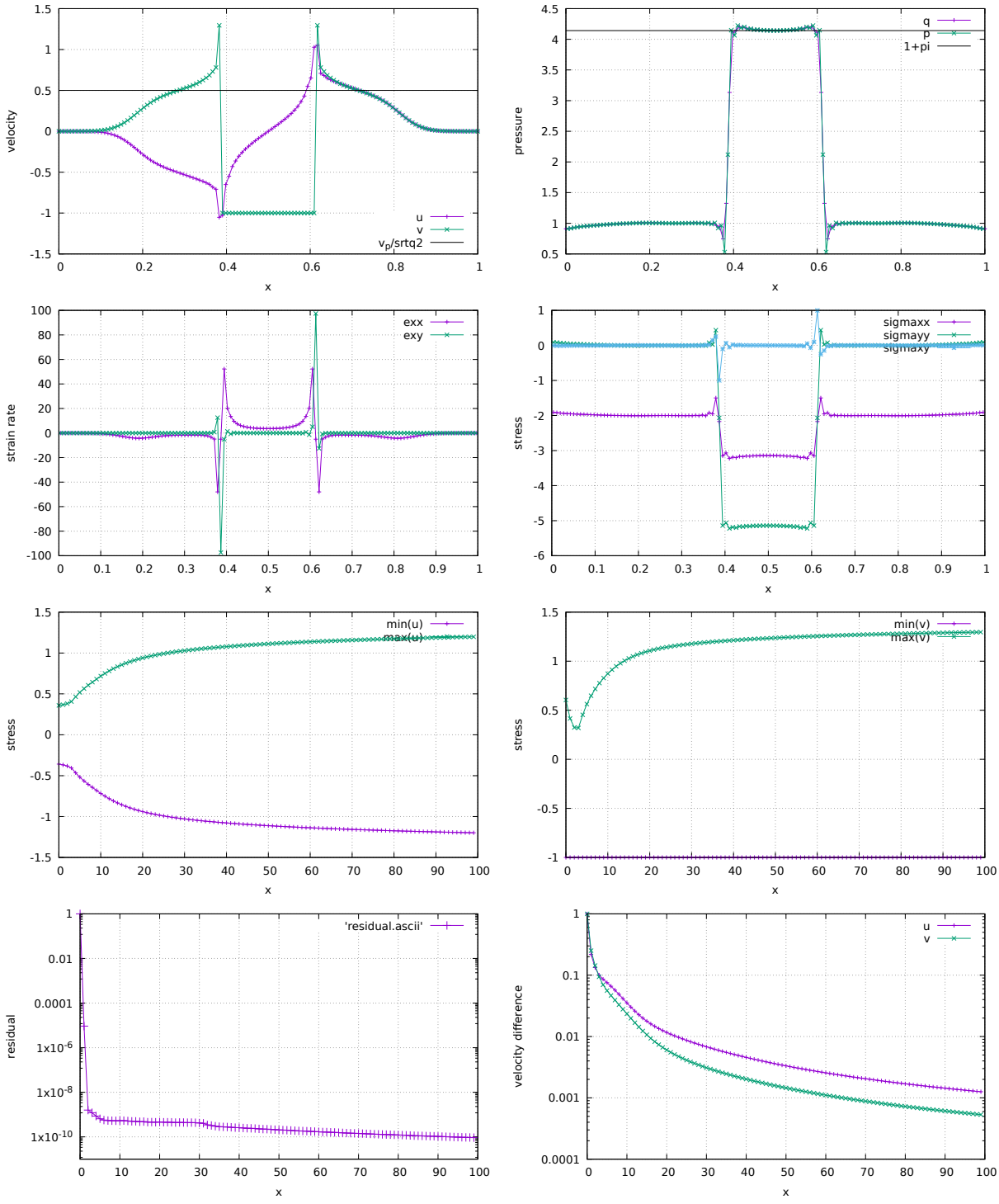


a,b,c,d) velocity, pressure, strainrate and stress at the top of the domain; e,f) min/max value of u and v; g,h) residual and normalised velocity difference.

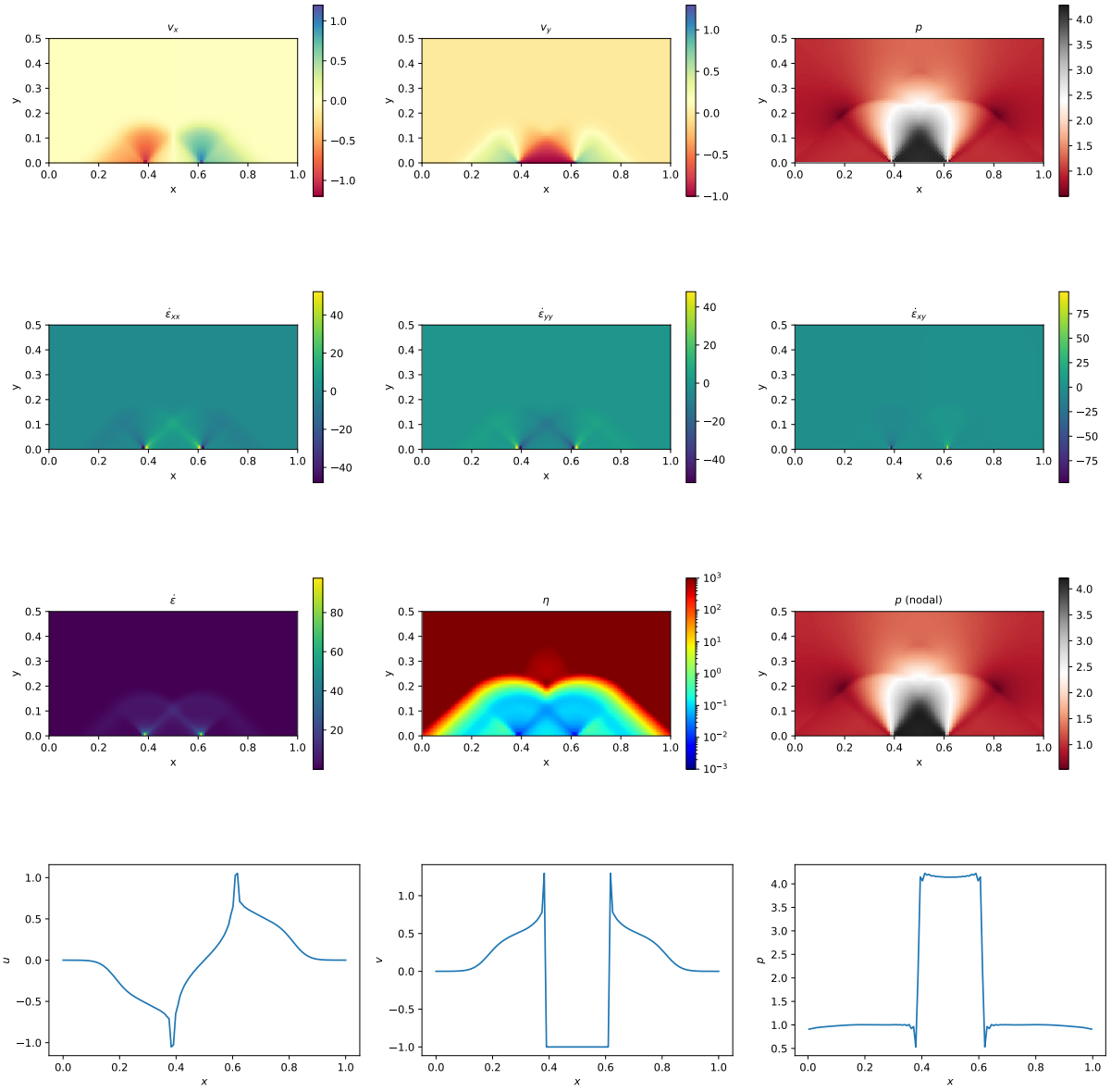

v_x
v_y
p
$\dot{\varepsilon}_{xx}$
$\dot{\varepsilon}_{yy}$
$\dot{\varepsilon}_{xy}$
$\dot{\varepsilon}$
η
p (nodal)
x
y

u
v
p
x

Smooth punch



a,b,c,d) velocity, pressure, strainrate and stress at the top of the domain; e,f) min/max value of u and v; g,h) residual and normalised velocity difference.


v_x
v_y
p
$\dot{\varepsilon}_{xx}$
$\dot{\varepsilon}_{yy}$
$\dot{\varepsilon}_{xy}$
$\dot{\varepsilon}$
η
p (nodal)
x
y

u
v
p
x

Stone 9

The annulus benchmark ($Q_1 \times P_0$+penalty)



2D
annulus
Q1P0
penalty
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
paraview_output

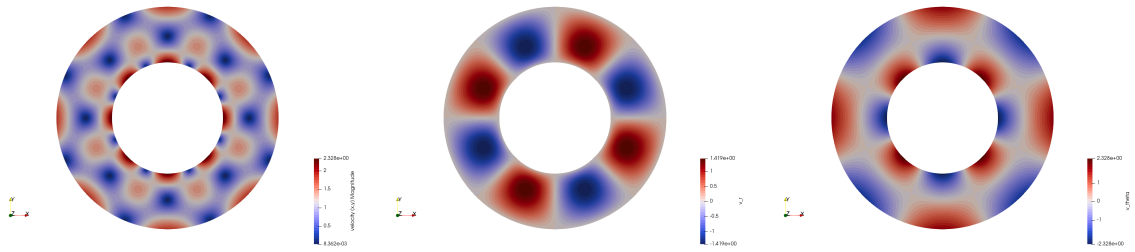
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_09`

This fieldstone was developed in collaboration with Prof. E.G.P. Puckett.

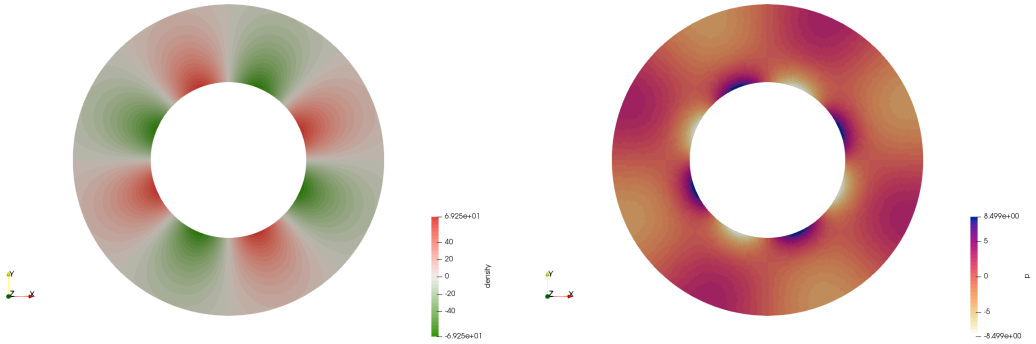
The benchmark is described fully in Section 19.18 in which an analytical solution to the isoviscous incompressible Stokes equations is derived in an annulus geometry. The velocity and pressure fields are as follows:

$$v_r(r,\theta) = g(r) k \sin(k\theta), \tag{9.1}$$
$$v_\theta(r,\theta) = f(r) \cos(k\theta), \tag{9.2}$$
$$p(r,\theta) = kh(r) \sin(k\theta), \tag{9.3}$$
$$\rho(r,\theta) = \aleph(r) k \sin(k\theta), \tag{9.4}$$

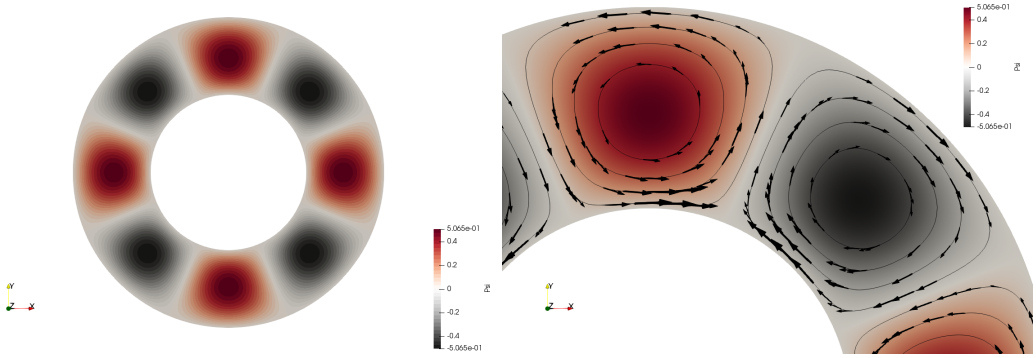
with $f(r)$, $g(r)$, $h(r)$ given in Section 19.18. The parameters A and B are chosen so that $v_r(R_1) = v_r(R_2) = 0$, i.e. the velocity is tangential to both inner and outer surfaces. The gravity vector is radial and of unit length. In the present case, we set $R_1 = 1$, $R_2 = 2$ and $C = -1$. The parameter k can take values 0,1,2,3,...



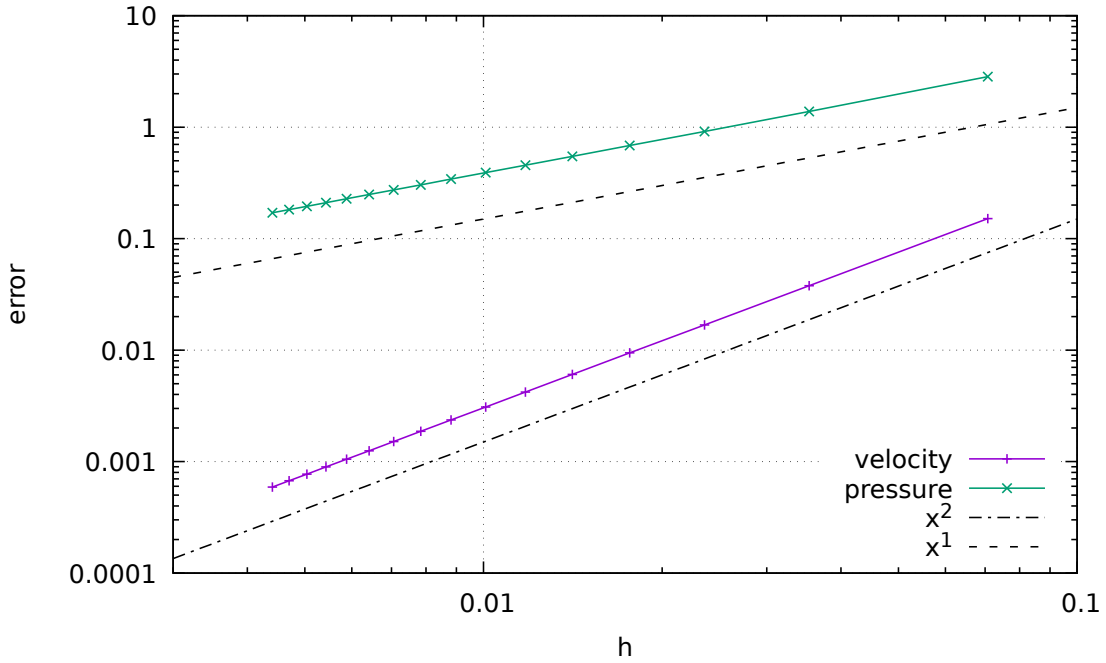
Left to right: velocity norm, r component, θ component



Left: density field; right: pressure field.



Left: ψ field; right: ψ isolines with velocity arrows.



Stone 10

Stokes sphere in 3D ($Q_1 \times P_0$+penalty, $Q_2 \times P_0$+penalty)



3D
box
Q1P0
penalty
sparse_matrix
numerical_benchmark
stokes_equation
viscous_rheology
buoyancy_driven
vrms
paraview_output

P J F Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_10`

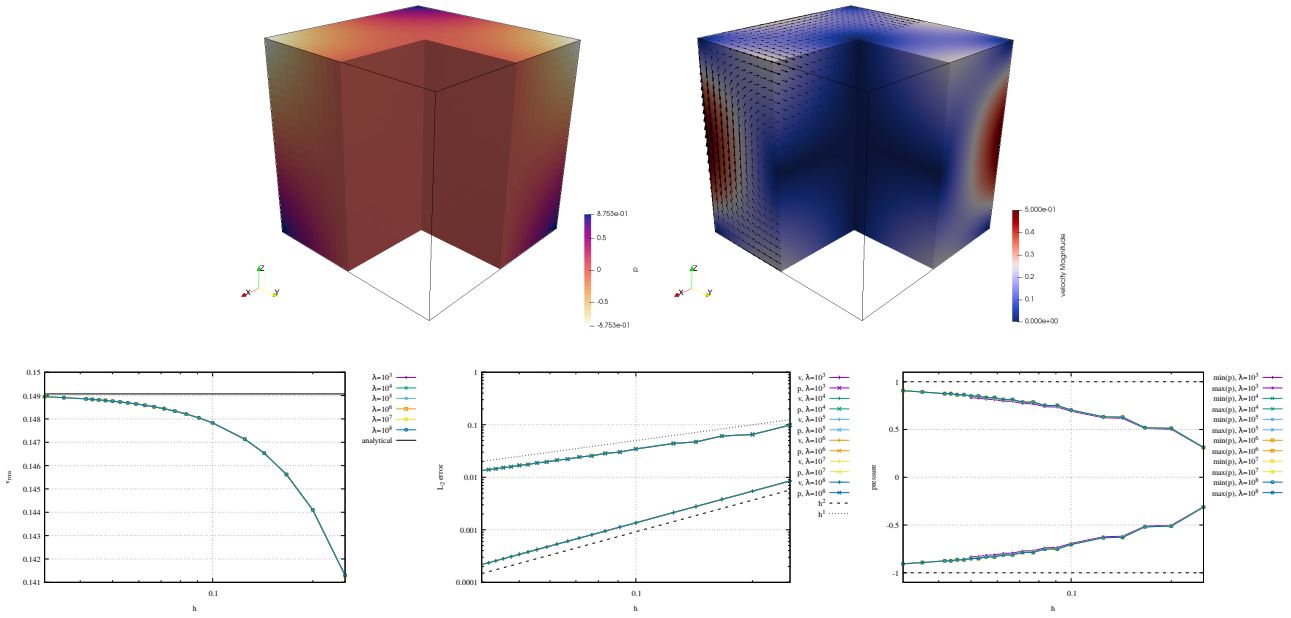
Last revision: Sept. 27th, 2024.

Manufactured solution (`experiment=5`)

It is described in Section 18.30. The velocity and pressure fields are:

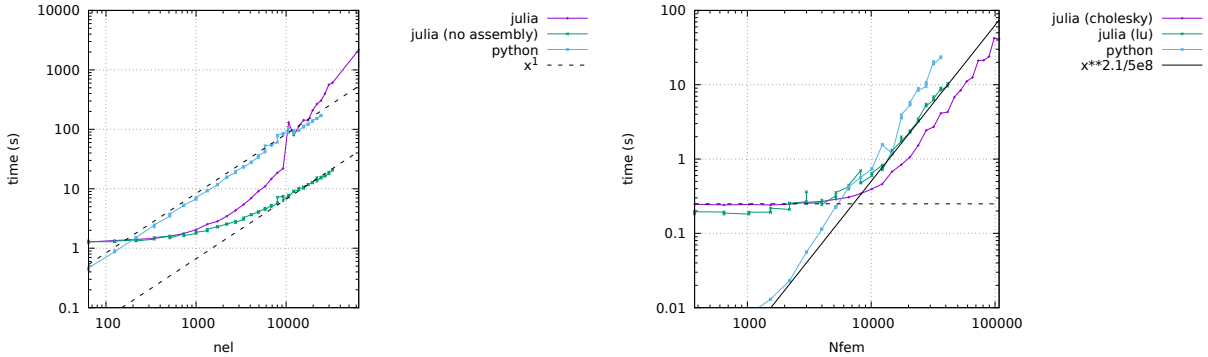
$$u(x,y,z) = x(1-x)(1-2y)(1-2z) \tag{10.1}$$
$$v(x,y,z) = (1-2x)y(1-y)(1-2z) \tag{10.2}$$
$$w(x,y,z) = -2(1-2x)(1-2y)z(1-z) \tag{10.3}$$
$$p(x,y,z) = (2x-1)(2y-1)(2z-1) \tag{10.4}$$

This flow field has the built-in property that there is no flux through the boundaries, i.e. the normal component of the velocity field is zero on each face.



We recover the expected second order velocity error convergence and first order pressure error convergence. Surprisingly, we find that the results are not sensitive to the value of the penalty parameter λ.

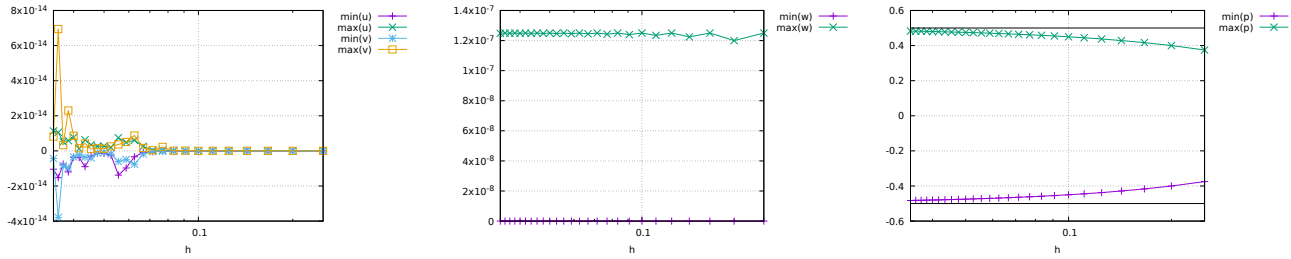
This STONE is implemented in both python and julia, in order to look at performance aspects. Both versions of the code are as close to each other as possible. The FE build and solve times are recorded and we find that the build time grows linearly with the number of elements in the python case but not in the julia case. I suspected that the sparse storage of julia (as opposed to a linked list converted to sparse matrix in python) makes it slow so I ran the code without the assembly and indeed it is then linear with the number of elements. I need to find a better way!!



Matrix building and system solve times for both python and julia codes. Obtained with penalty=1e6. Each resolution is run 3 times.

Aquarium (`experiment=0`)

FS on all sides, $\rho = \eta = g_z = 1$, $\lambda = 10^6$



Statistics of u, v, w, p.

We expect a zero velocity field. This is the case (down to machine precision) for the u and v components, but since the penalty method yields a weakly compressible flow, the vertical component w is larger but

remains small. The pressure is expected to be lithostatic and it should be -0.5 at the top and +0.5 at the bottom. Pressure statistics seem to converge towards these values.

Stokes sphere

The domain is a unit cube. Free slip boundary conditions are imposed on all sides. The mesh counts nelx*nely*nelz=nel elements and nnx*nny*nnz=NV nodes. The density and the viscosity are prescribed in the domain by means of two functions: the density is set to $1 + \delta\rho$ inside a sphere of radius 0.123456789 centered at (0.5,0.5,0.5) and 1 outside. The viscosity is 1000 inside the sphere and 1 outside. The gravity vector is set to $\vec{g} = (0, 0, -1)$. This is the benchmark presented in Section **??**.

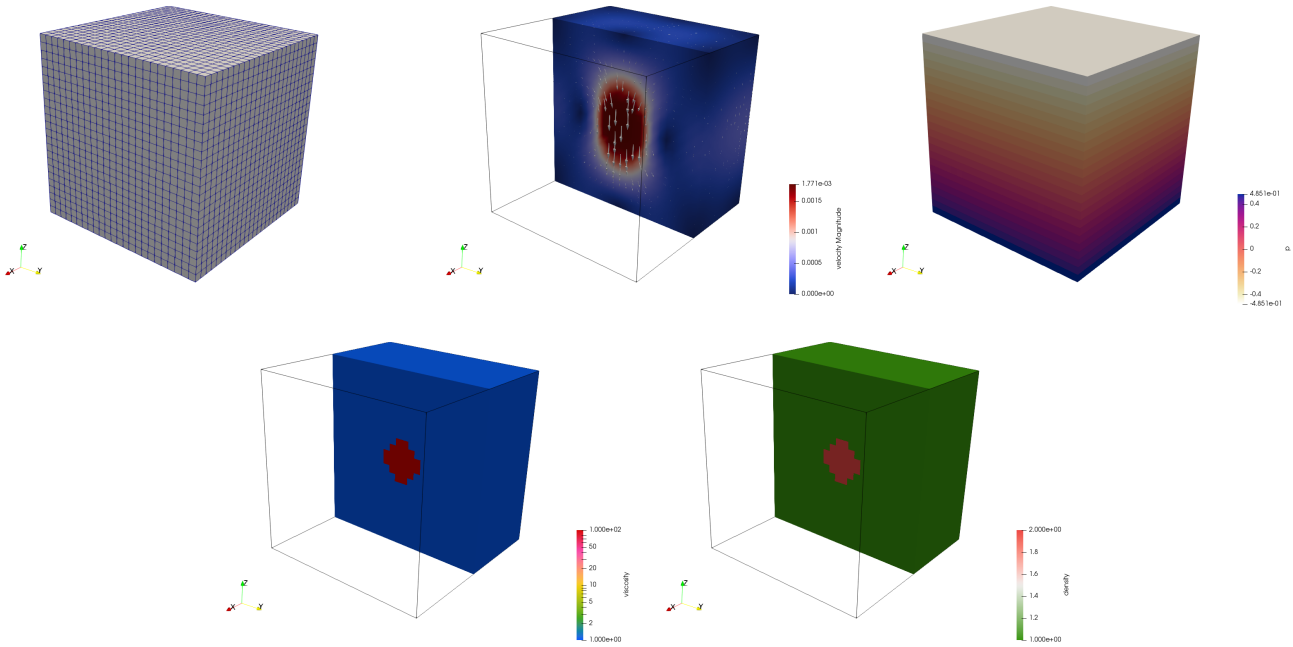
The FE matrix size grows even faster now than in the previous 2D case so choosing the right matrix storage is of paramount importance.

Three experiments are carried out:

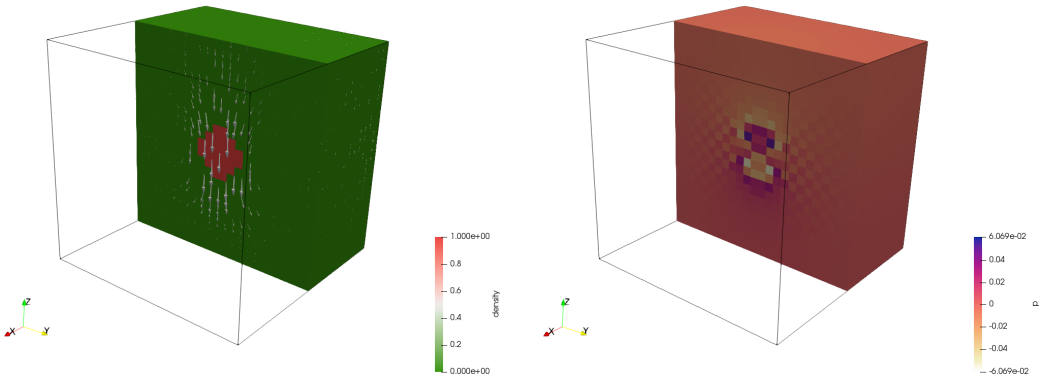
Exp. 1: the one described above. Free slip or no slip boundary conditions. We see that the pressure field is dominated by the lithostatic signal.

Exp. 2: same as experiment 1, but a reference density of 1 is substracted to all densities, so that the sphere density is 1 and the density of the surrounding fluid is now 0. In essence, we remove a 'background' density which does not participate in the flow generation, and thereby get rid of the lithostatic signal of the pressure. Free slip or no slip boundary conditions.

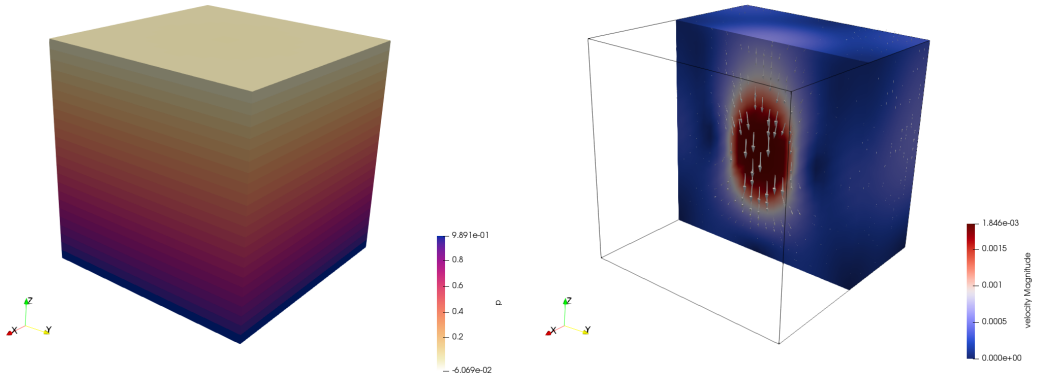
Exp. 3: same as experiment 2, but the top boundary is now open (free surface)



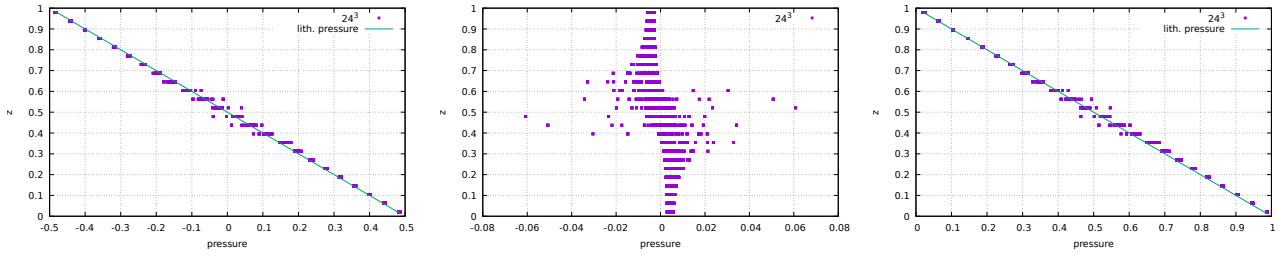
Exp. 1: resolution $24 \times 24 \times 24$



Exp. 2: Density and pressure fields. Resolution 24x24x24



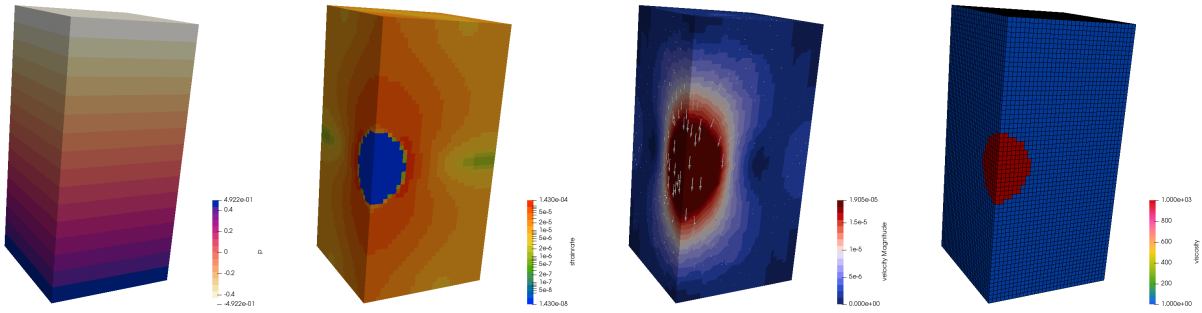
Exp. 3: pressure and velocity fields. Resolution 24x24x24



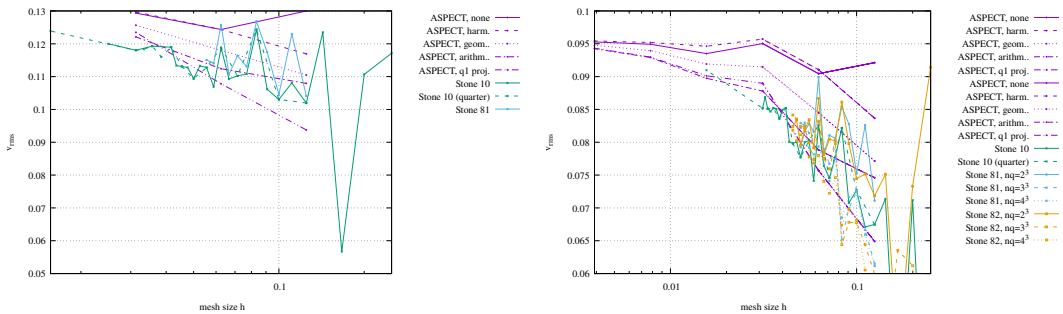
Elemental pressure for all elements as a function of their vertical (middle) coordinate for (from left to right) experiments 1, 2 and 3.

Note that a similar fortran code is present in the folder.

If the option 'quarter' is true, then the model takes advantage of the symmetries of the problem and runs it in a quarter of the original domain, so that the domain is then $0.5 \times 0.5 \times 1$. Measurements/results are identical to the full cube ones, but at a much lower cost.



Resolution $32 \times 32 \times 64$ - FS



Left: FS; Right: NS. Results obtained with this stones and other ones, as well as ASPECT. See Section **??** for all results.

Trying $Q_2 \times Q_0$+penalty... because why not

Based on a discussion with A. Regorda I have coded a $Q_2 \times Q_0$ (still with penalty) version of the STONE : `stoneQ2Q0.py`. The following results are obtained with `script_Q2Q0`.

I first naively used a 1 point integration (as for the $Q_1 \times P_0$ element) only for the penalty term, only to realise that the penalty term $\mathbb{K}_\lambda$ should be under-integrated, but not necessarily using a single point.

This sparked the study on the influence of the quadrature rule of the penalty term on the accuracy of results.

I also implemented the consistent pressure recovery described in Section 15.8.5. This results in a nodal pressure field (which I call q) which is solution of

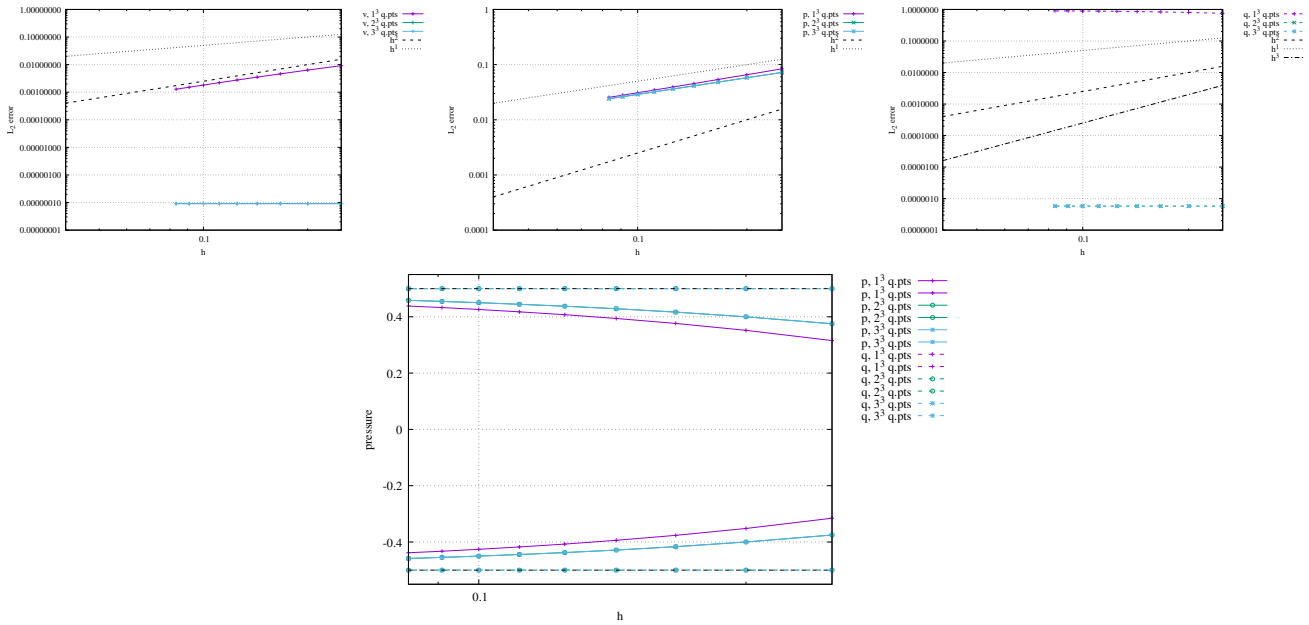
$$\boldsymbol{M} \cdot \vec{Q} = \vec{f}$$

where $\boldsymbol{M}$ is the Q_2 mass matrix that is integrated with the same quadrature as the viscous block $\mathbb{K}_\eta$ but the rhs $\vec{f}$ is integrated with the same quadrature as the penalty block $\mathbb{K}_\lambda$. In the end I also keep the pressure p computed in the middle of the element using $p = \lambda \vec{\nabla} \cdot \vec{\mathrm{v}}$.

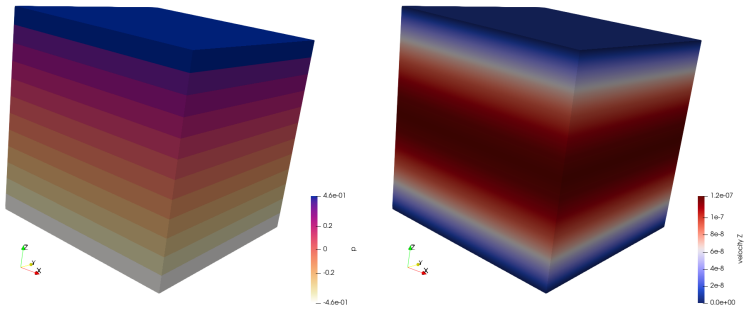
Finally to save time I have used the fact that the Jacobian of the coordinate change for cuboids is a diagonal matrix containing terms $h_x/2$, $h_y/2$, $h_z/2$ on the diagonal. This means that this can be computed once and for all for all elements.

Aquarium (`experiment=0`)

Fress slip on all sides, $\rho = \eta = g_z = 1$, $\lambda = 10^6$



We expect a zero velocity field. This is the case (down to machine precision) for the u and v components, but since the penalty method yields a weakly compressible flow, the vertical component w is much larger then u and v. The pressure is expected to be lithostatic and it should be -0.5 at the top and +0.5 at the bottom. Pressure statistics to converge towards these values.

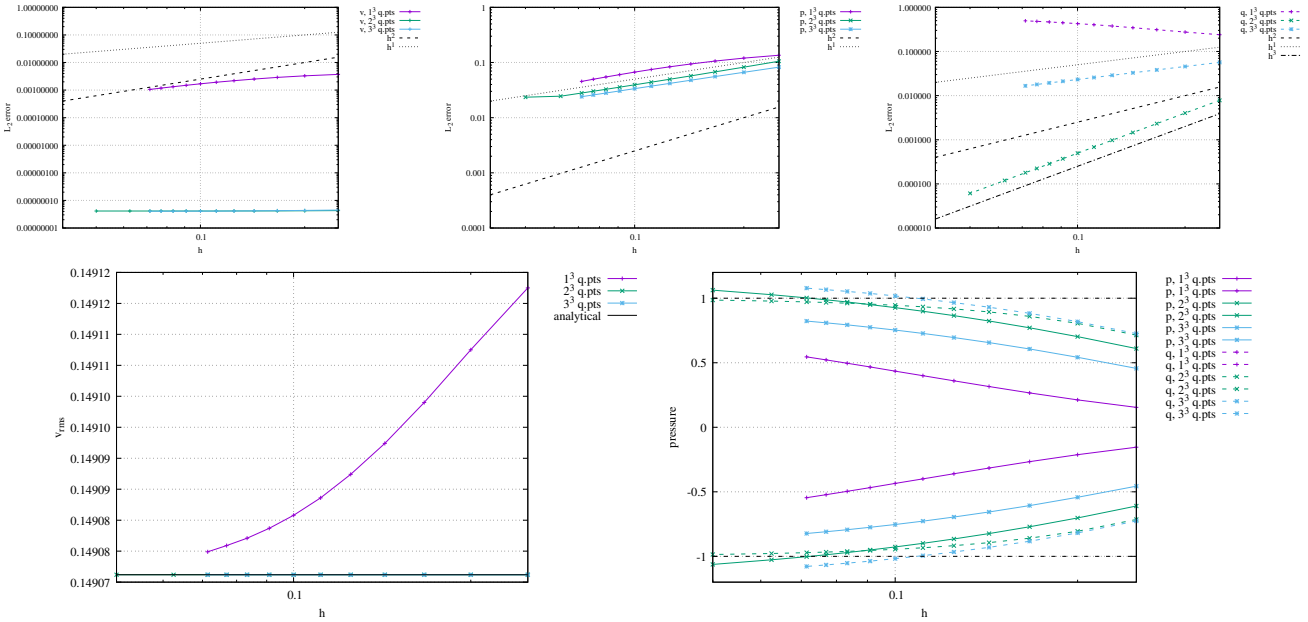


w and p for resolution $20 \times 20 \times 20$.

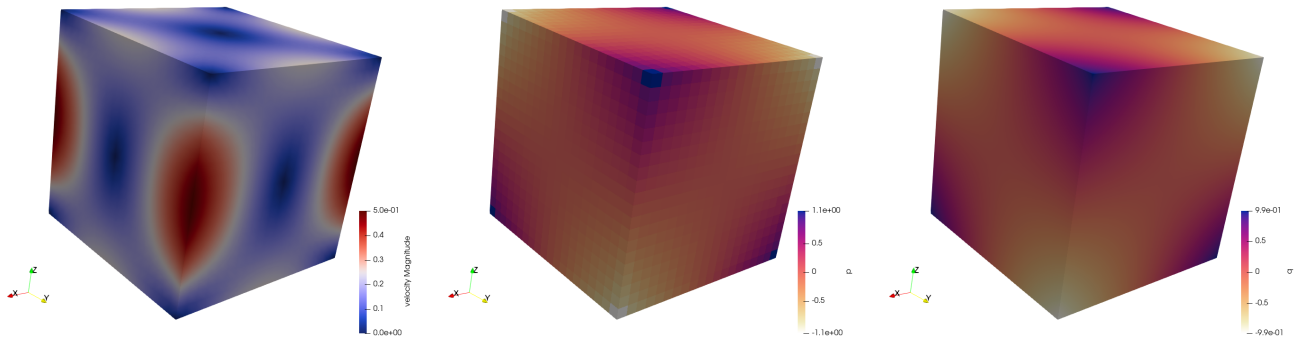
Note that at resolution 20x20x20 build time is 10min, and solve time is 48 min! Given this 'performance' this will limit a lot the tests we can carry out...

Manufactured solution (`experiment=5`)

See Section 18.30 for the analytical expressions of the velocity, pressure and body force vector.



Since the analytical solution of this experiment is quadratic then the velocity basis functions can exactly represent the solution inside each element and the accuracy of the velocity solution depends on the penalty parameter and not on resolution. We clearly see that using only one integration point for the penalty term results in lesser accurate velocities. Interestingly we obtain a superconvergence (third order) of the q error for a 2x2x2 quadrature and only a linear convergence for the q error for a 3x3x3 quadrature. The pressure q obtained with only 1 quad point yields results that are so off that they do not show on the plots. Finally we note that we obtain pressures that often exceed the analytical bounds for min and max pressure except for q with 2x2x2 quadrature. This seems to indicate that a 2x2x2 quadrature is preferable. We keep investigating in the next section with a more suitable manufactured solution benchmark.

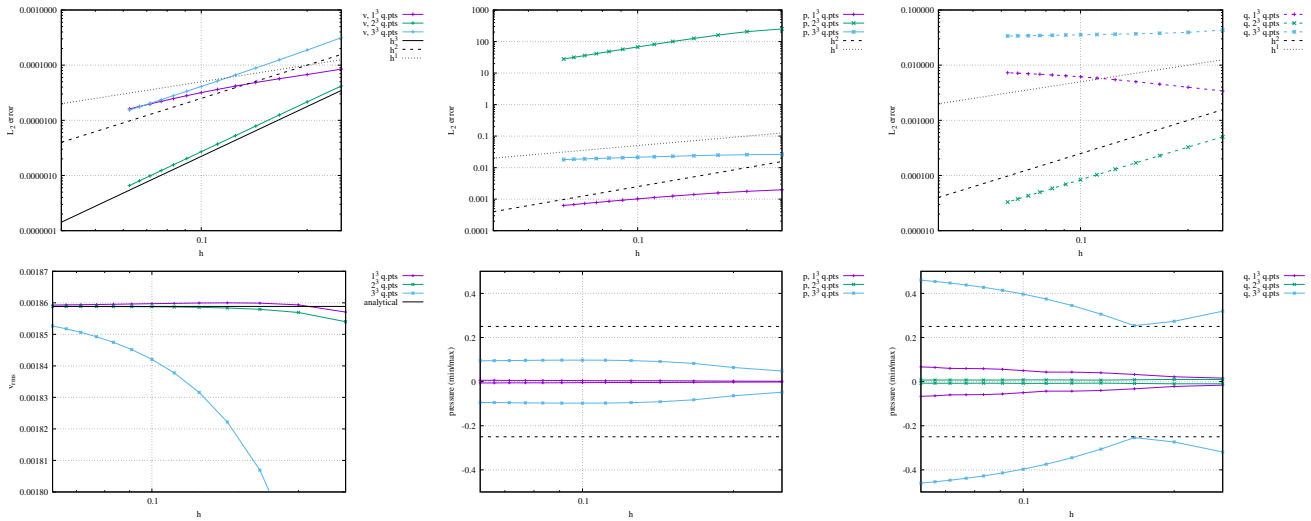


$||\vec{v}||$, p and q for resolution $20 \times 20 \times 20$.

Manufactured solution (`experiment=6`)

After the observations made about the previous experiment we then turn to another 3D manufactured solution as described in Section 18.30.

$$
\begin{aligned}
f(x) &= x^2(1-x)^2 && (10.5)\\
g(y) &= y^2(1-y)^2 && (10.6)\\
h(z) &= z^2(1-z)^2 && (10.7)\\
u(x,y,z) &= fg'h' && (10.8)\\
v(x,y,z) &= f'gh' && (10.9)\\
w(x,y,z) &= -2f'g'h && (10.10)\\
p(x,y,z) &= -f'g'h' && (10.11)
\end{aligned}
$$



We find that:

- The velocity error exhibits cubic convergence *only* for $2 \times 2 \times 2$ quadrature and is the smallest.
- the convergence of the p pressure is puzzling
- the convergence of the q pressure is second order (!) for $2 \times 2 \times 2$ quadrature and is also the smallest.

The conclusion is clear: the penalty term $\mathbb{K}_\lambda$ must be integrated with $2 \times 2 \times 2$ quadrature.
TODO: determine analytical min max of pressure

Stone 11

Stokes sphere in 3D ($Q_1 \times P_0$ - mixed)



3D
box
Q1P0
numerical_benchmark
stokes_equation
viscous_rheology
buoyancy_driven
paraview_output
pressure_normalisation

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_11`

Last revision: Sept. 5th, 2024.

The setup is identical to the one of the STONE 10.

The difference lies in how we solve the Stokes equation. This stone does not rely on the penalty method (Section 10.7) but instead used a mixed formulation, i.e. we solve for both velocity and pressure at the same time (see Section 10.8).

In the case when free slip boundary conditions are applied on all 6 faces of the cube we know that there is a pressure nullspace, i.e. that the pressure can only be computed up to a constant. In order to remove this nullspace one must add an additional constraint We here choose to (somewhat arbitrarily) enforce that the average pressure over the whole domain is zero:

$$< p >= \frac{1}{|\Omega|} \int_\Omega p \, dV = 0$$

Since the code relies of discontinuous zero-th order polynomial shape functions for pressure this condition simply writes (and since $|\Omega| = 1$):

$$< p >= \frac{1}{|\Omega|} \int_\Omega p \, dV = \sum_e \int_{\Omega_e} p dV = \sum_e p_e V_e = h^3 \sum_e p_e = 0$$

Dividing all by h^3, the condition becomes simply:

$$p_1 + p_2 + p_3 + ... + p_{nel} = 0$$

How this constraint is incorporated in the Stokes matrix is explained in Section 10.16.1.

It is also important to remember that if one now switches to a free surface at the top then the null space is absent from the equations and the constraint should be removed/switched off.

The pressure normalisation is controled by the boolean *pnormalise* in the code. Since the pressure constraint adds a line to the global FE, we then logically have:

```
if pnormalise:
    a_mat = np.zeros((Nfem+1,Nfem+1),dtype=np.float64)
    rhs   = np.zeros(Nfem+1,dtype=np.float64)
else:
    a_mat = np.zeros((Nfem,Nfem),dtype=np.float64)
    rhs   = np.zeros(Nfem,dtype=np.float64)
```

Once the solve has been done, we retrieve the separate velocity and pressure fields as follows:

```
u,v,w=np.reshape(sol[0:NfemV],(nnp,3)).T
p=sol[NfemV:Nfem]
```

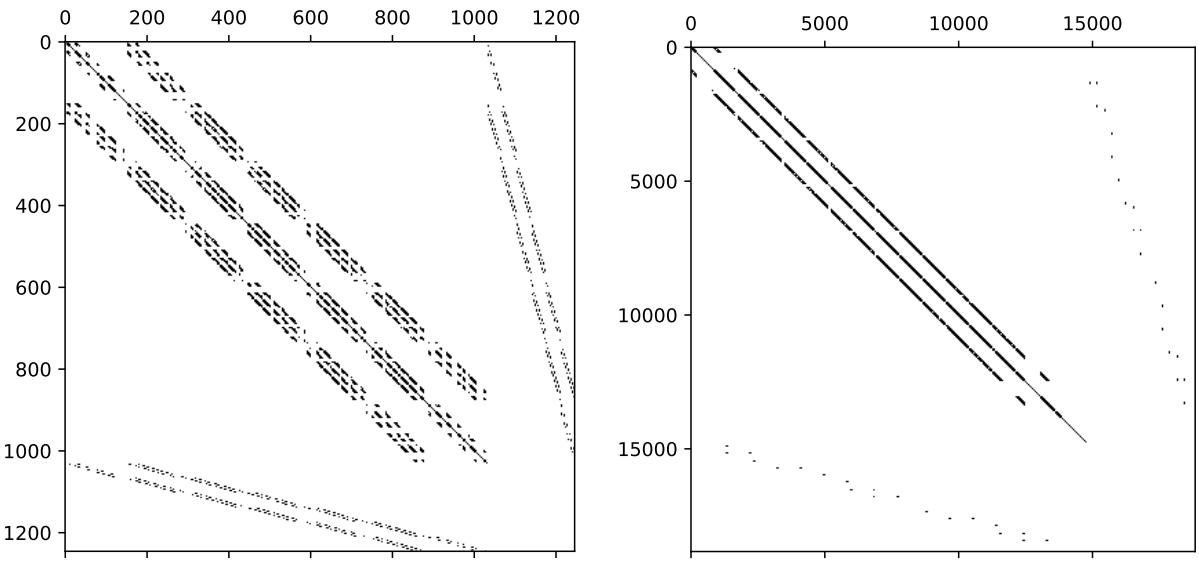
and the Lagrange multiplier is a scalar at the bottom of the array:

```
if pnormalise:
    print("     ----> Lagrange multiplier: %.4es" % sol[Nfem])
```

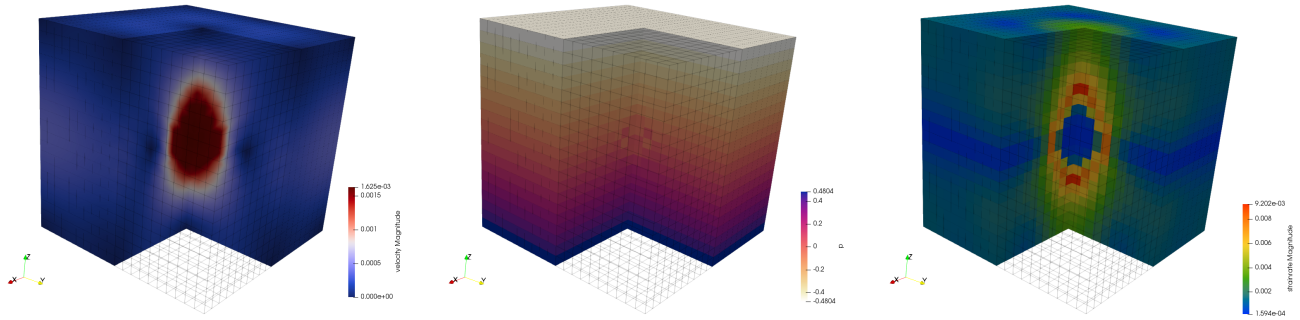
It is quite easy in python to visualise the matrix structure with the spy function:

```
plt.spy(a_mat)
```

and we see that we indeed recover the block structure with a zero block for the pressure-pressure entries.



Sparsity pattern of the Stokes matrix for a 6x6x6 mesh (left) and a 16x16x16 mesh (right).



Velocity, pressure and effective strain rate fields for a $20 \times 20 \times 20$ mesh.

Remark: the matrix building in this STONE is rather naive. The $\mathbb{K}$ and $\mathbb{G}$ matrices are built as full matrices and later assembled in a_mat which is also a full matrix, only to be converted to CSR format before calling the solver. In subsequent stones I usually only build a_mat in lil_matrix format.

Stone 12

consistent pressure recovery ($Q_1 \times P_0$+penalty)



2D
box
penalty
Q1P0
analytical_benchmark
error_convergence
stokes_equation
paraview_output

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_12`

We start from the analytical benchmark of Section 18.2 and we use $Q_1 \times P_0$ elements with a penalty formulation. We have seen in STONE 1 how to recover the elemental pressure as a postprocessing step. However, the discontinous nature of the pressure field (and the presence of a parasitic checkerboard mode) can be problematic for many reasons (pressure enters the rheology, equation of state, plotting, ...). We then wish to project the elemental pressure onto the nodes of the mesh while at the same time filtering the checkerboard mode out.

Terminology: in general, when a discontinuous elemental pressure is used in a stone, it is called p while its projection onto the velocity nodes is called q. In this case we compute several nodal pressures as explained in Section 15.8:

- q_1: smoothed pressure obtained with the center-to-node approach (scheme 1).
- q_2: the nodal pressure obtained by smoothing the elemental pressure using element areas (scheme 2).
- q_3: the nodal pressure obtained by smoothing the elemental p using triangle areas (scheme 3).
- q_4: smoothed pressure obtained with the center-to-node approach with inverse element area weighing.
- q_5: is not computed because same as 6.
- q_6: consistent pressure recovery through FE (scheme 5 & 6)
- q_7: consistent pressure recovery with lumped mass matrix.
- q_8: bilinear interpolation.

Since the setups showcase Dirichlet boundary conditions on all four sides, the pressure solution is normalised so that $\int p \, dV = 0$.

Remark. *Nodes on edges and corners may need special treatment as documented in Sani et al [277] or Lee et al (1979) [208] which is not done here. Nodes on the boundary showcase the largest errors. Maybe then use CBF instead for these?*

Remark. *Lee et al (1979) [208] discuss two different ways to compute the lumped mass matrix.*

Remark. *In the same folder there is a fortran90 version of the consistent recovery algorithm.*

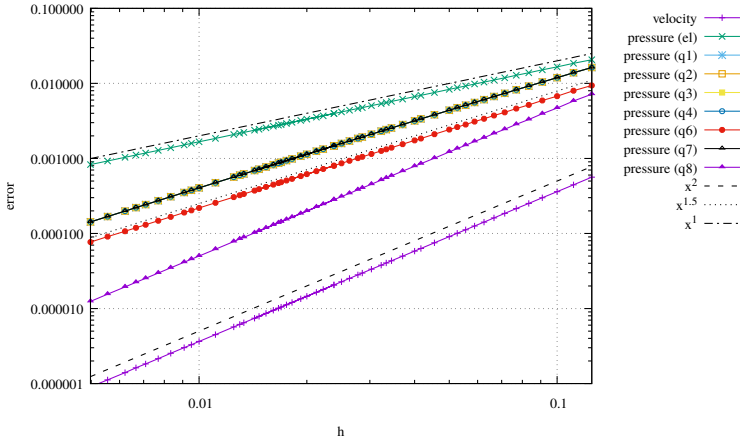
I have created a simple checkerboard filter. Let $\phi = \pm 1$ denote this field. The filter is applied as follows:

$$p_{filtered} = p - \left(\frac{1}{|\Omega|} \iint_{\Omega} p\phi dV \right) \phi$$

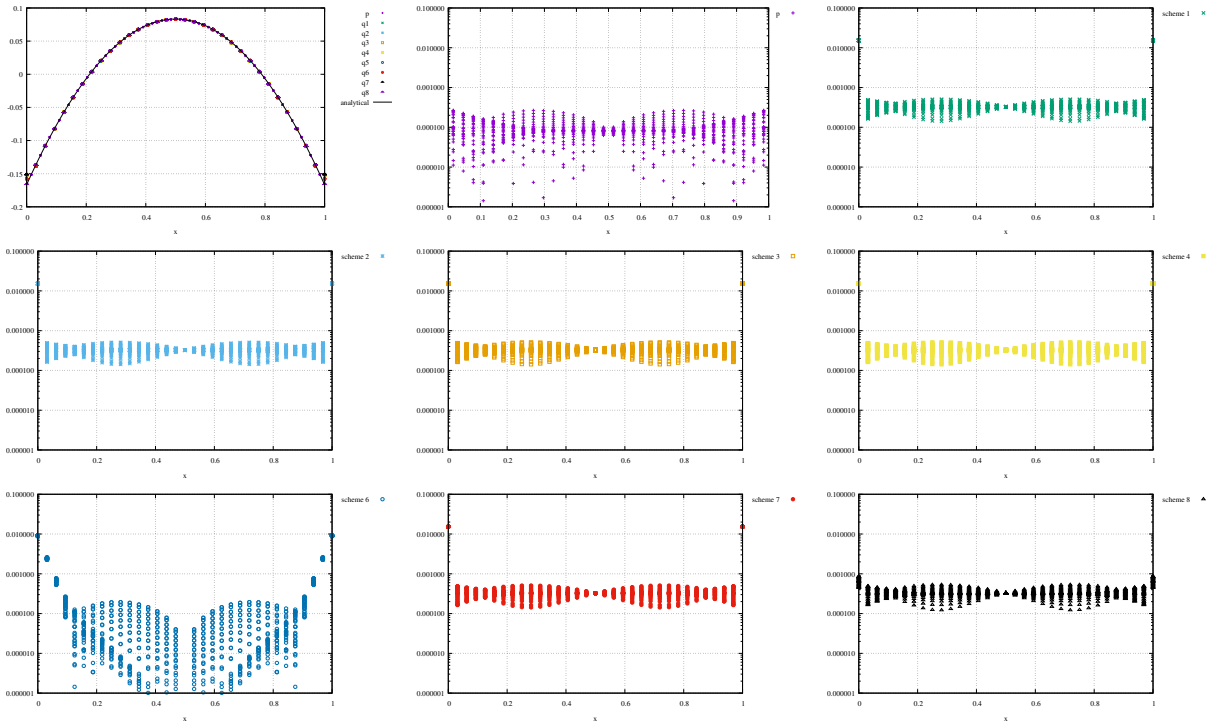
It is in fact a a projection. If $p = \alpha\phi$ where α is a scalar, then it is trivial to show that $p_{filtered} = 0$. Likewise, if $p = constant$ then $p_{filtered} = p$. This idea is probably to be found in the relevant publications of Section 15.8 in some form or shape.

There is much more work to be done on this topic/stone! Look at cost vs accuracy!

Regular mesh made of square elements We compute the error convergence for p, q_i $(i = 1, ...8)$:

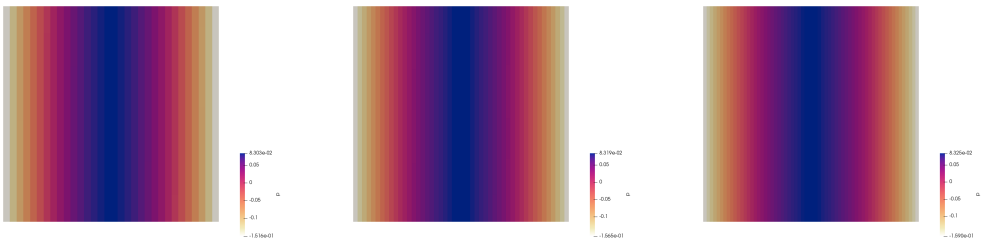


The elemental pressure error converges like h^1, q_{1-7} converge like $h^{1.5}$ and q^8 converges like h^2.



Left: pressure fields as a function of the x-coordinate. Right: absolute error with regards to the analytical solution. All on 32x32 mesh.

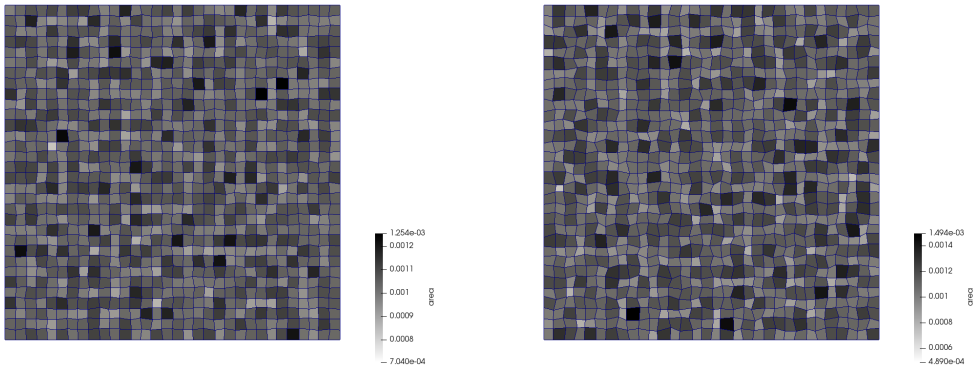
It is worth noticing that the checkerboard is (visually) not present:



Elemental pressure field for 32x32, 48x48 and 64x64 resolutions.

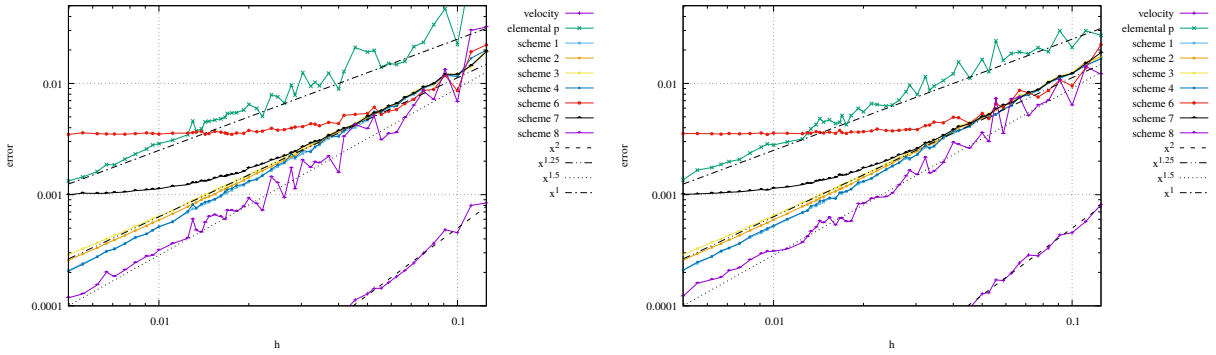
This means that the algorithms above fulfill an interpolation function, but not a smoothing one.

Adding randomness to internal node positions We now add a random value ξh to the location of all nodes which are not on the boundary where h=L_x/nelx and we set $\xi = 20\%$. In this case a 32x32 mesh looks as follows:



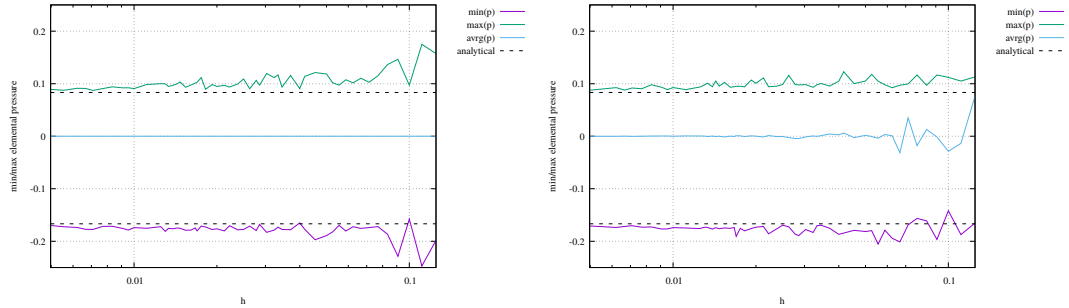
Mesh 32x32 elements. Left: $\xi = 0.1$; Right: $\xi = 0.2$

We repeat the same exercise as before on such a mesh and look at the errors



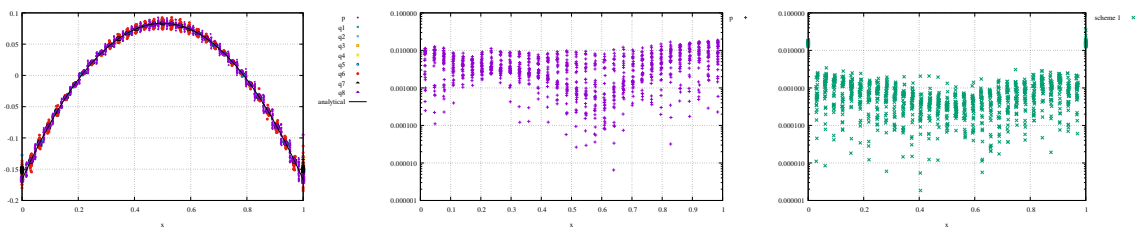
Velocity and pressure errors for $\xi = 0.2$. Left: no filter, right: with filter.

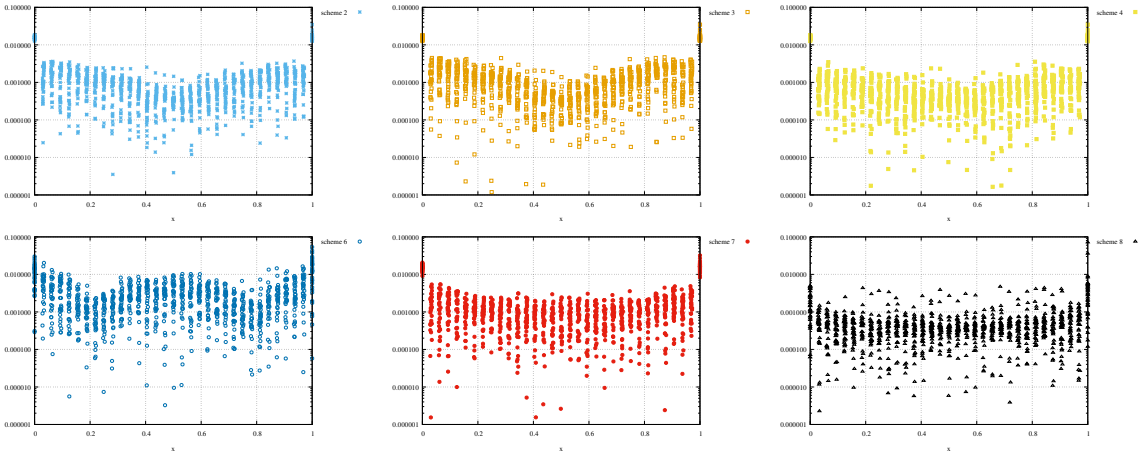
It is easy to see the drastic effect that the filter has on the min/max of the elemental pressure:



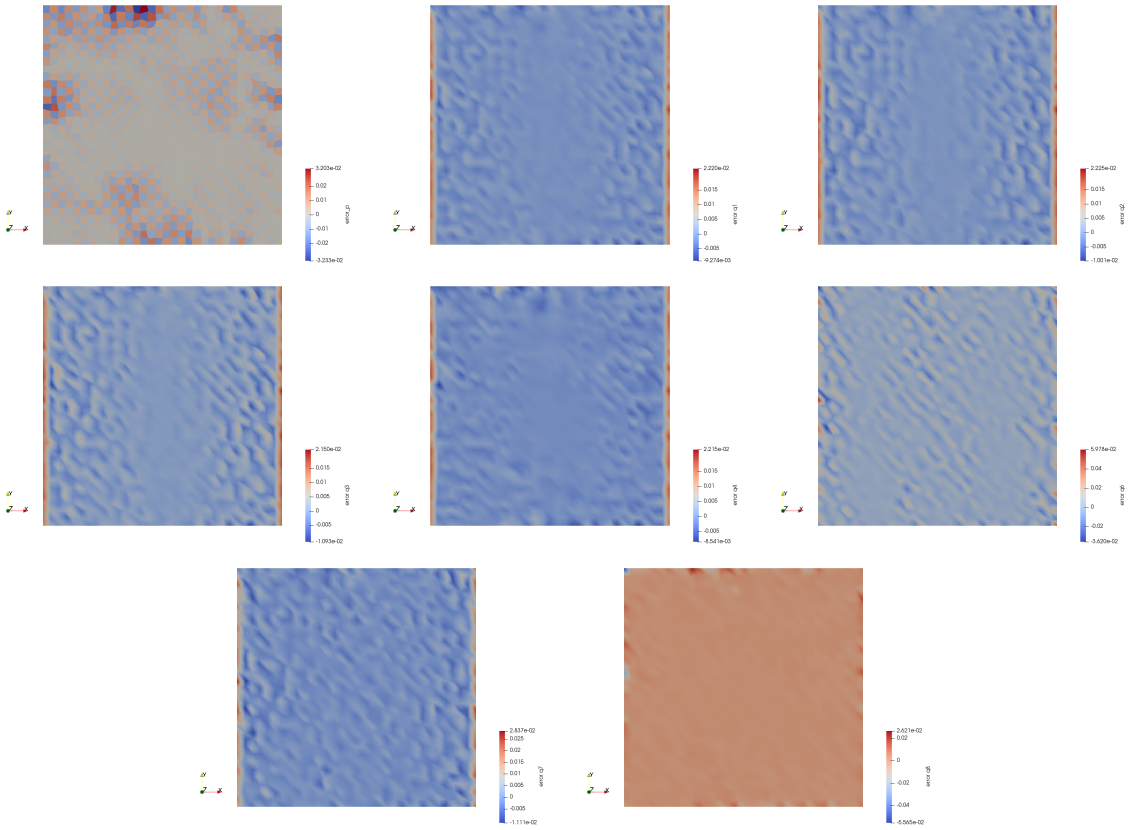
min/max value of elemental pressure field: Left: no filter, right: with filter

Rather surprisingly we find that q_1 still proves to be the most accurate of all pressures and it converges with $h^{1.5}$(?) as before. Because checkerboard modes are triggered the convergence of the elemental pressure is more chaotic but on average linear. The q_6 and q_7 fields seem to unexpectedly stop converging above a given resolution, which probably comes from the lack of adequate treatment of sides and corners. All others converge with $h^{1.5}$ and q_4 seems a bit more chaotic than the others. Scheme 8 seems to be the best here again but with an erratic convergence about $h^{1.5}$





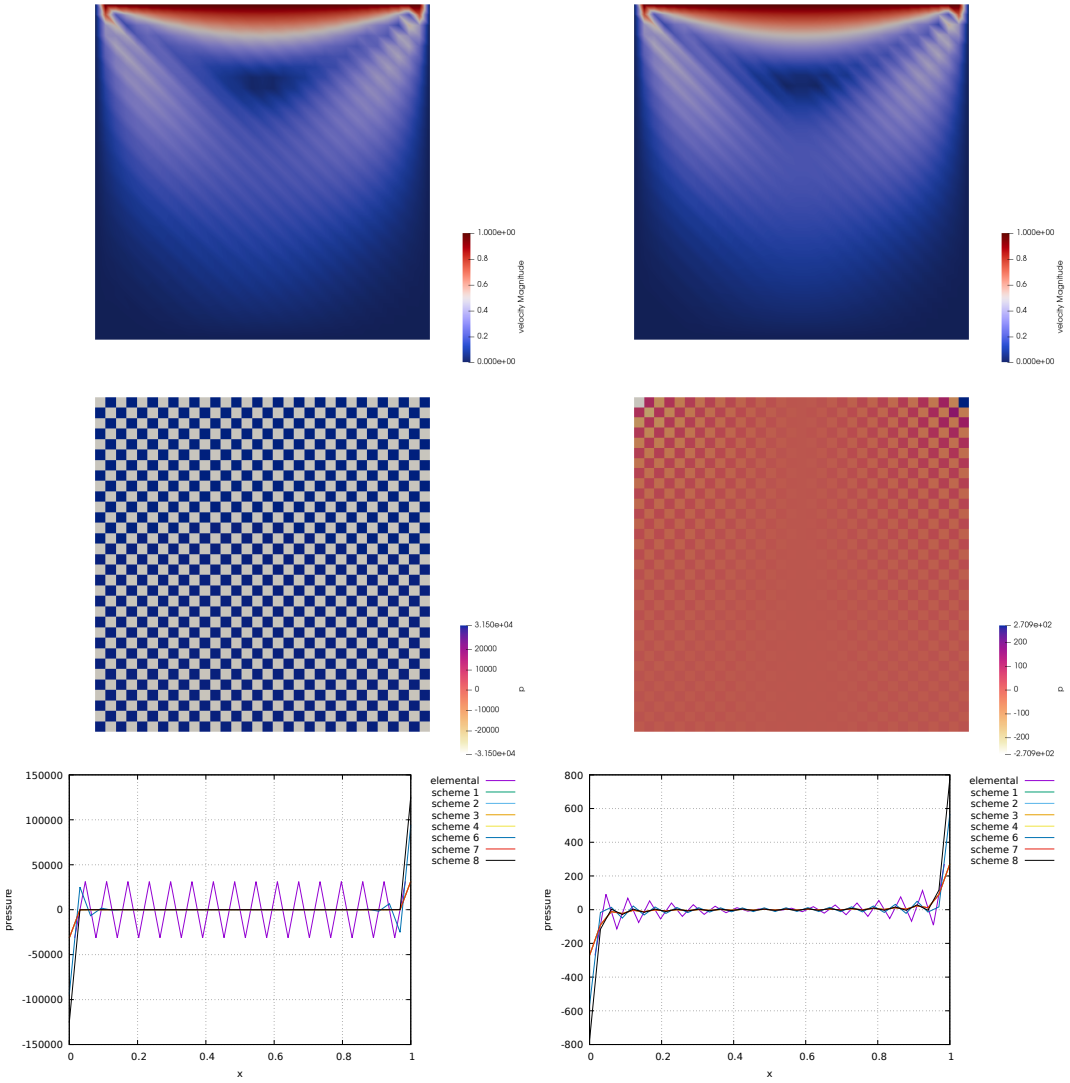
all obtained in 32x32 mesh



Pressure error for the elemental and nodal pressure fields. Note the presence of a spatially varying checkboard pattern in the elemental pressure field.

Obtained on 32x32 mesh.

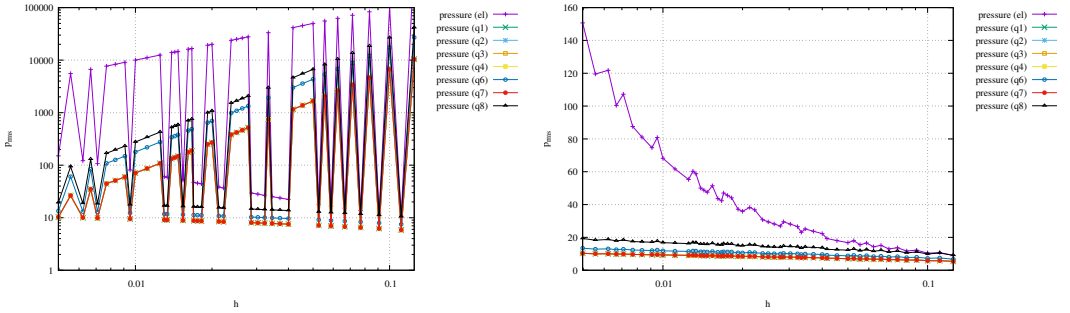
Lid driven cavity The domain is a unit square. No slip boundary conditions are prescribed on left, right and bottom boundary. Unit velocity is prescribed on the top, including both top corners, in order to generate a velocity/pressure discontinuity which triggers the checkerboard. This is rather succesful (no filter is applied yet):

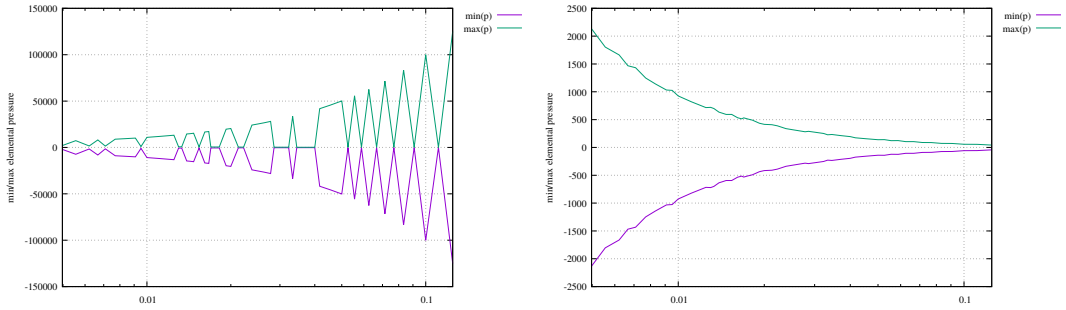


Left: 32x32, Right: 33x33. Last row is pressures at the top of the domain.

We recover the well known fact that even number of elements are more prone to checkerboard of high amplitude, but odd numbers do not preclude their presence. The conclusion is unescapable: there is no nodal pressure filed which does not showcase positive/negative oscillations.

It is easy to see the drastic effect that the filter has on the min/max of the elemental pressure:

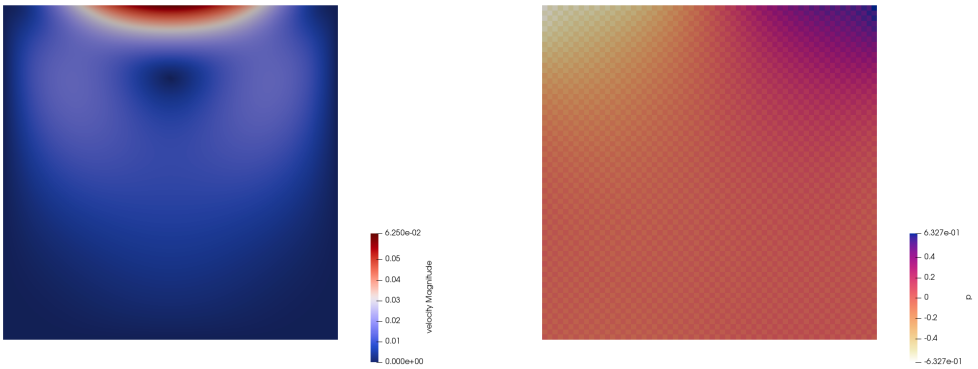




Left: filter off; Right: filter on

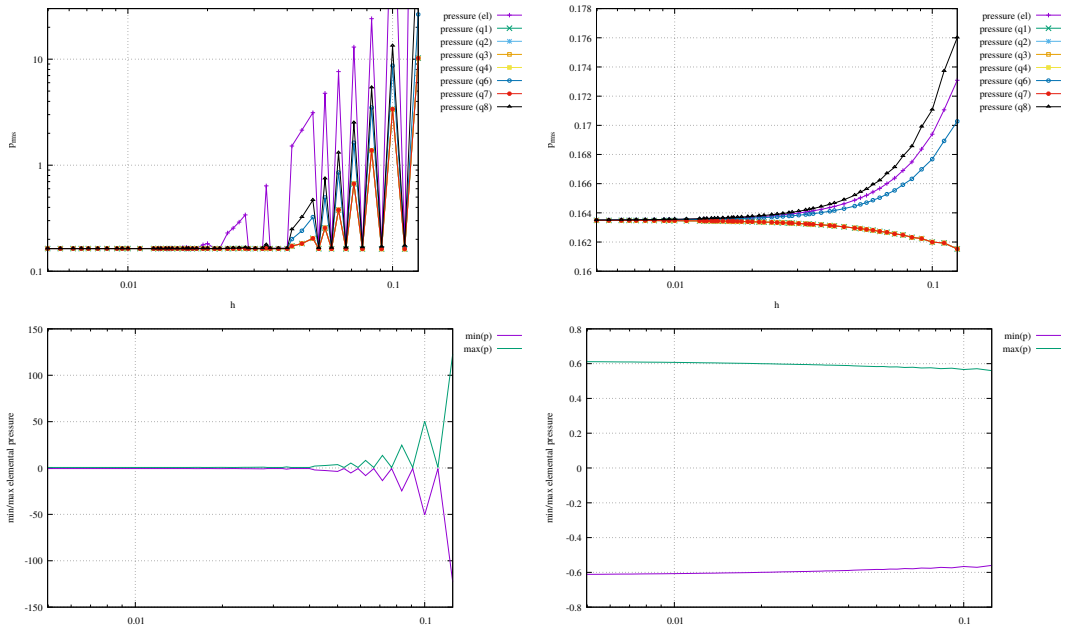
Only schemes 6 and 7 are unaffected by the filter since they

Regularised Lid driven cavity See stone 4. The velocity on the top is given by $u(x) = x^2(1-x)^2$. This has the advantage of removing the discontinuity at the corners and thereby yielding a problem which has a non-singular solution. However, wee see that the checkerboard mode is still present:



Resolution 32x32

There is no analytical solution. When the pressure errors are computed in the code the results are actually root mean square pressures (if a zero analytical pressure is used). We see that these all converge to a single value. Here again we see that the filter has a drastic effect on the min/max of the elemental pressure.



Left: filter off; Right: filter on

Stone 13

the Particle in Cell technique (1) - the effect of averaging ($Q_1 \times P_0$+penalty)

PIC FEM BSc

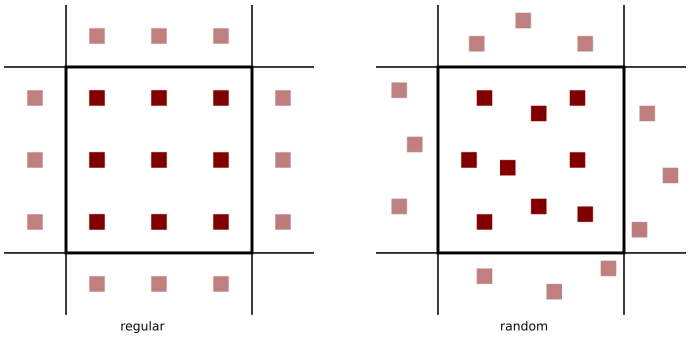
2D
box
Q1P0
penalty
numerical_benchmark
error_convergence
stokes_equation
viscous_rheology
buoyancy_driven
vrms
paraview_output
matplotlib
PIC

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_13`

This fieldstone was developed in collaboration with BSc student Eric Hoogen.

After the initial setup of the grid, markers can then be generated and placed in the domain. One could simply randomly generate the marker positions in the whole domain but unless a *very* large number of markers is used, the chance that an element does not contain any marker exists and this will prove problematic. In order to get a better control over the markers spatial distribution, one usually generates the marker per element, so that the total number of markers in the domain is the product of the number of elements times the user-chosen initial number of markers per element.

Our next concern is how to actually place the markers inside an element. Two methods come to mind: on a regular grid, or in a random manner, as shown on the following figure:



In both cases we make use of the shape functions: we generate the positions of the markers (random or regular) in the reference element first (r_{im}, s_{im}), and then map those out to the real element as follows:

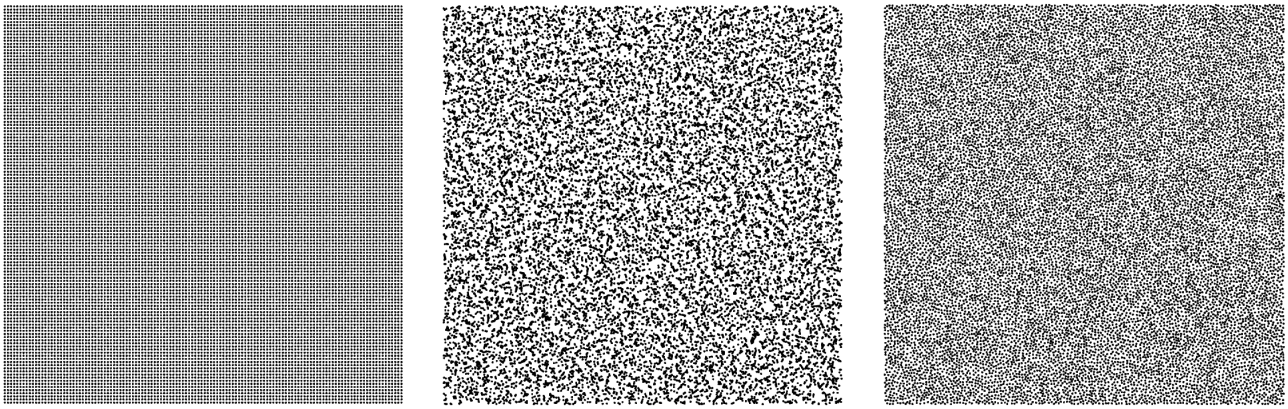
$$x_{im} = \sum_i^m \mathcal{N}_i(r_{im}, s_{im}) x_i \qquad\qquad y_{im} = \sum_i^m \mathcal{N}_i(r_{im}, s_{im}) y_i \tag{13.1}$$

where x_i, y_i are the coordinates of the vertices/nodes of the element.

A third option consists in the use of the so-called Poisson-disc sampling which produces points that are tightly-packed, but no closer to each other than a specified minimum distance, resulting in a more natural pattern [1]. Note that the Poisson-disc algorithm fills the whole domain at once, not element after element.

say smthg about avrg dist

insert here theory and link about Poisson disc



Left: regular distribution, middle: random, right: Poisson disc.

16384 markers (32×32 grid, 16 markers per element).

When using *active* markers, one is faced with the problem of transferring the properties they carry to the mesh on which the PDEs are to be solved. As we have seen, building the FE matrix involves a loop over all elements, so one simple approach consists of assigning each element a single property computed as the average of the values carried by the markers in that element. Often in colloquial language "average" refers to the arithmetic mean:

$$\langle \phi \rangle_{am} = \frac{1}{n} \sum_k^n \phi_i \tag{13.2}$$

where $< \phi >_{am}$ is the arithmetic average of the n numbers ϕ_i. However, in mathematics other means are commonly used, such as the geometric mean:

$$\langle \phi \rangle_{gm} = \left(\prod_i^n \phi_i \right)^{1/n} \qquad \text{or} \qquad \log \langle \phi \rangle_{gm} = \frac{1}{n} \left(\sum_i^n \log \phi_i \right) \tag{13.3}$$

and the harmonic mean[2]:

$$\langle \phi \rangle_{hm} = \left(\frac{1}{n} \sum_i^n \frac{1}{\phi_i} \right)^{-1} \tag{13.4}$$

Furthermore, there is a well known inequality for any set of positive numbers,

$$\langle \phi \rangle_{am} \quad \geq \quad \langle \phi \rangle_{gm} \quad \geq \quad \langle \phi \rangle_{hm} \tag{13.5}$$

which will prove to be important later on.

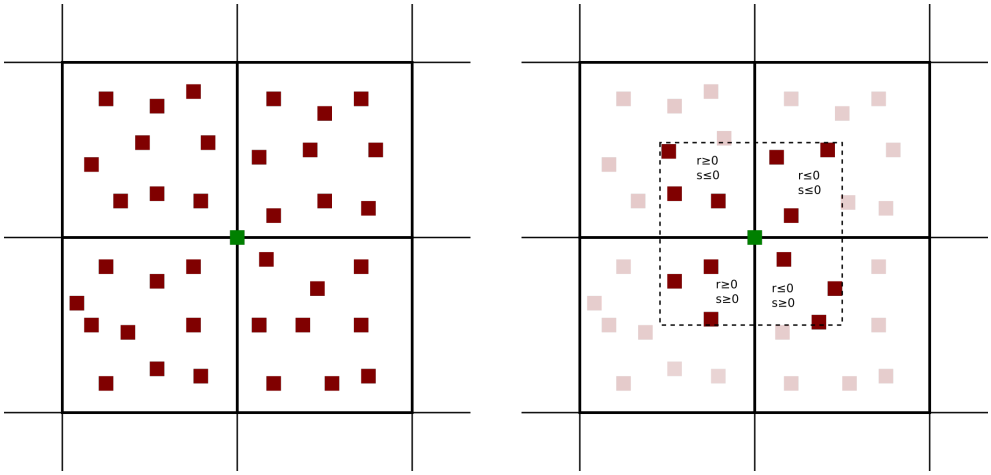
Let us now turn to a simple concrete example: the 2D Stokes sphere. There are two materials in the domain, so that markers carry the label "mat=1" or "mat=2". For each element an average density and

[1] https://en.wikipedia.org/wiki/Supersampling

[2] is this the one we use?

viscosity need to be computed. The majority of elements contains markers with a single material label so that the choice of averaging does not matter (it is trivial to verify that if $\phi_i = \phi_0$ then $\langle\phi\rangle_{am} = \langle\phi\rangle_{gm} = \langle\phi\rangle_{hm} = \phi_0$. Remain the elements crossed by the interface between the two materials: they contain markers of both materials and the average density and viscosity inside those depends on 1) the total number of markers inside the element, 2) the ratio of markers 1 to markers 2, 3) the type of averaging.

This averaging problem has been studied and documented in the literature, see for example Schmeling et al. [282] (2008), Deubelbeiss and Kaus [88] (2008), Thielmann, May, and Kaus [324] (2014), Pusok, Kaus, and Popov [252] (2017).



Nodal projection. Left: all markers inside elements to which the green node belongs to are taken into account. Right: only the markers closest to the green node count.

Let k be the green node of the figures above. Let (r, s) denote the coordinates of a marker inside its element. For clarity, we define the follow three nodal averaging schemes:

- nodal type A:

$$f_k = \frac{\text{sum of values carried by markers in 4 neighbour elements}}{\text{number of markers in 4 neighbour elements}}$$

- nodal type B:

$$f_k = \frac{\text{sum of values carried by markers inside dashed line}}{\text{number of markers in area delimited by the dashed line}}$$

- nodal type C

$$f_k = \frac{\text{sum of values carried by markers in 4 neighbour elements } *\mathcal{N}_p(r,s)}{\text{sum of } \mathcal{N}_p(r,s)}$$

where $\mathcal{N}_p$ is the Q_1 basis function corresponding to node p defined on each element. Since these functions are 1 on node k and then linearly decrease and become zero on the neighbouring nodes, this effectively gives more weight to those markers closest to node k.

This strategy is adopted in May, Brown, and Le Pourhiet [228] (2014) and May, Brown, and Le Pourhiet [227] (2015) (although it is used to interpolate onto the nodes of $Q_2 \times P_{-1}$ elements). It is formulated as follows:
"We assume that an arbitrary material point property f, is discretized via $f(\boldsymbol{x}) \simeq \delta(\boldsymbol{x} - \boldsymbol{x}_p) f_p$. We then utilize an approximate local L_2 projection of f_p onto a continuous Q_1 finite element space. The corner vertices of each Q_2 finite element define the mesh f_p is projected onto. The local reconstruction for a node i is defined by

$$\hat{f}_i = \frac{\int_{\Omega_i} N_i(\boldsymbol{x}) f(\boldsymbol{x})}{\int_{\Omega_i} N_i(\boldsymbol{x})} \simeq \frac{\sum_p N_i(\boldsymbol{x}_p) f_p}{\sum_p N_i(\boldsymbol{x}_p)}$$

where the summation over p includes all material points contained within the support Ω_i of the trilinear interpolant N_i".

Numerical setup

The setup is a Stokes sphere experiment with radius $R = 0.123$ placed in the center of the domain which is a 1×1 square. The viscosity of the sphere has been set to 10^3 while the viscosity of the surrounding fluid is 1. The average density is always computed with an arithmetic mean.

The code relies on $Q_1 \times P_0$ elements with a penalty formulation for simplicity. The penalty parameter is set to $\lambda = 10^7$. No-slip boundary conditions are applied on all four sides.

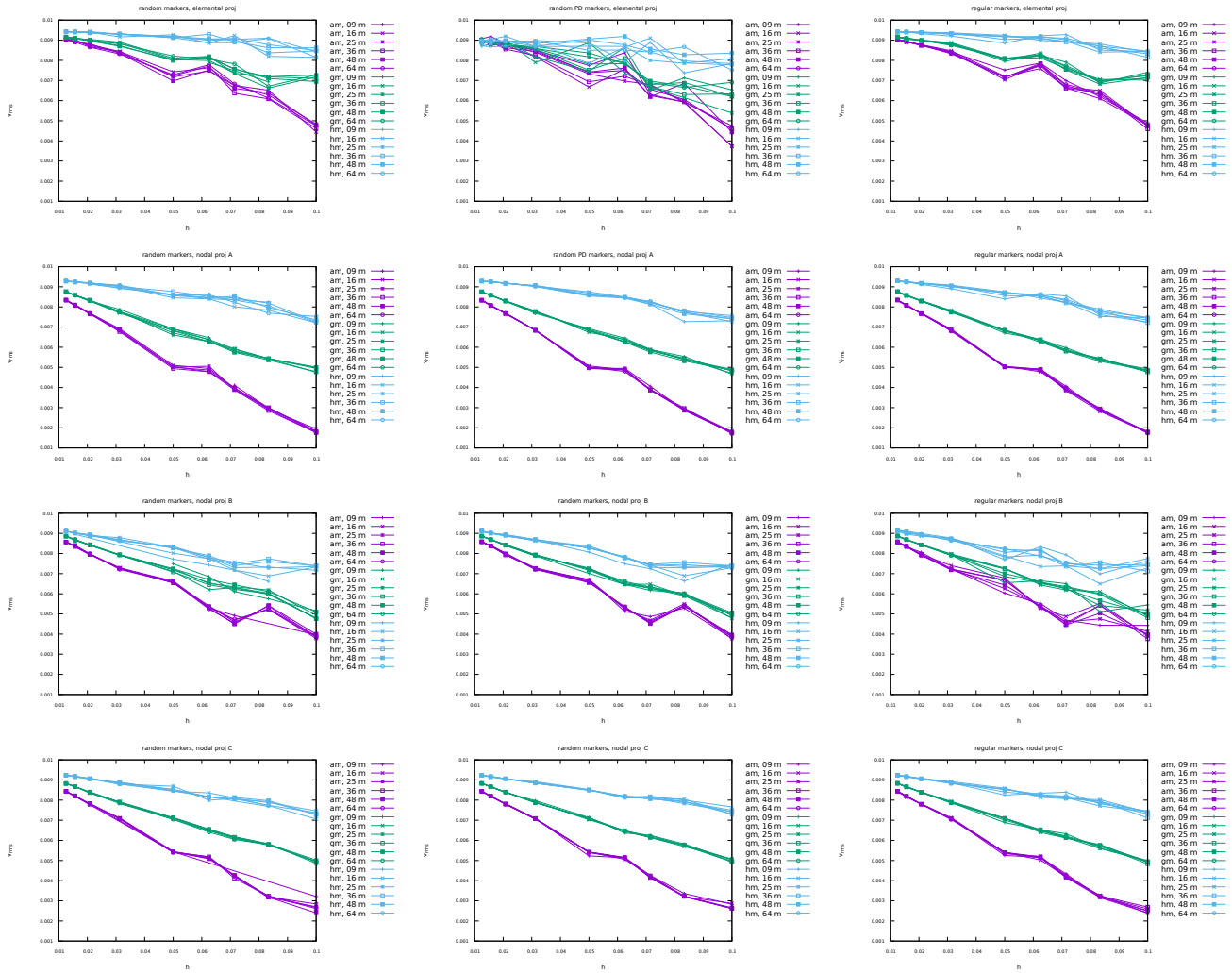
The bash script *script_runall* runs the code for many resolutions, both initial marker distribution and all four averaging types.

The script loops over the number of markers per dimension (3,4,5,6...), the type of projection (1: ,2: use all four elements around node, ,3: use all four quadrants around node ,4: use all four elements around node with weighing) and also the type of averaging (avrg=1: arithmetic mean; avrg=2: geometric mean; avrg=3: harmonic mean).

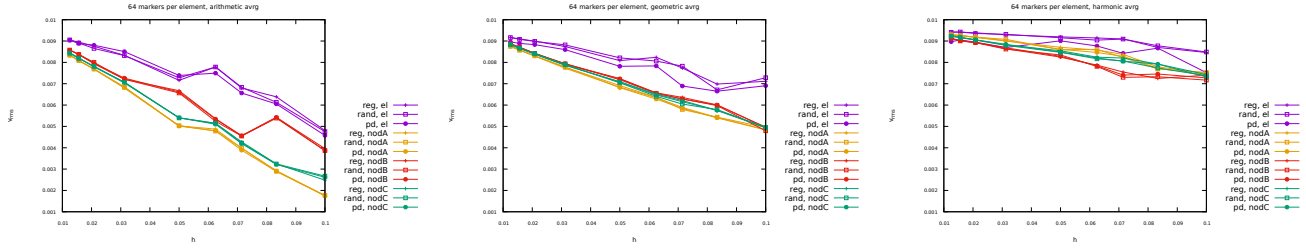
Conclusions

- With increasing resolution ($h \to 0$) ν_{rms} values seem to converge towards a single value, irrespective of the number of markers.
- At low resolution, say 32×32 (i.e. h=0.03125), ν_{rms} values for the three averagings differ by about 10%. At higher resolution, say 128×128, ν_{rms} values are still not converged.
- The number of markers per element plays a role at low resolution, but less and less with increasing resolution.
- Results for random and regular marker distributions are not identical but follow a similar trend and seem to converge to the same value.
- elemental values yield better results (espcecially at low resolutions)
- harmonic mean yields overal the best results

Root mean square velocity results are shown hereunder:



Left column: random markers, middle column: Poisson disc, right column: regular markers. First row: elemental projection, second row: nodal 1 projection, third row: nodal 2 projection, fourth row: nodal 3 projection.



Left to right: arithmetic, geometric, harmonic averaging for viscosity.

To Do, open questions, future work?

- change finite element pair: from Q1P0 to Q2Q1.
- export solution and fields to vtu format

Stone 14

solving the full saddle point problem ($Q_1 \times P_0$)



2D
box
Q1P0
sparse_matrix
analytical_benchmark
stokes_equation
matplotlib
pressure_scaling
pressure_normalisation

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_14

Last revision: Sept. 7th, 2024

Donea & Huerta manufactured solution

The details of the numerical setup are presented in Section 18.2.

In this stone we do not use the penalty formulation and therefore keep both velocity and pressure as unknowns. Therefore we end up having to solve the following saddle point system:

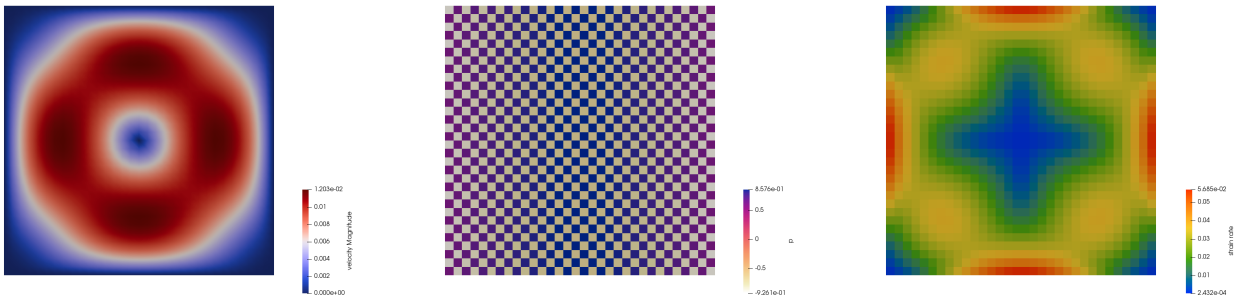
$$\begin{pmatrix} \mathbb{K} & \mathbb{G} \\ \mathbb{G}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{\mathcal{V}} \\ \vec{P} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ \vec{h} \end{pmatrix} \qquad \text{or,} \qquad \mathbb{A} \cdot \vec{\mathcal{X}} = \vec{b}$$

Each block $\mathbb{K}$, $\mathbb{G}$ and vector $\vec{f}$, $\vec{h}$ is built separately for each element in the code and assembled into the matrix $\mathbb{A}$ and vector $\vec{b}$ which are then passed to the solver.

Each element has $m_\nu = 4$ vertices so in total $ndofV \cdot m_\nu = 8$ velocity dofs and a single pressure dof, situated in the center of the element by convention. The total number of velocity dofs is therefore $NfemV = NV \cdot ndofV$ while the total number of pressure dofs is $NfemP = nel$. The total number of dofs is then $Nfem = NfemV + NfemP$.

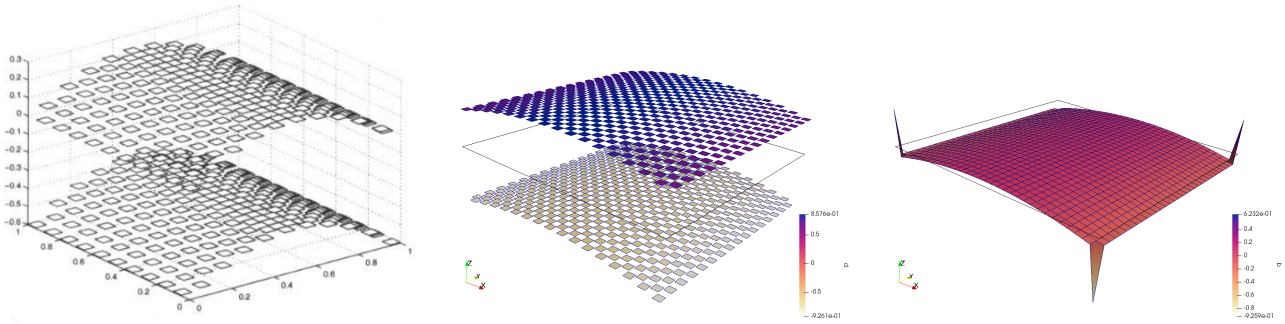
As a consequence, matrix $\mathbb{K}$ has size $(NfemV, NfemV)$ and matrix $\mathbb{G}$ has size $(NfemV, NfemP)$. Vector $\vec{f}$ is of size $NfemV$ and vector $\vec{h}$ is of size $NfemP$.

The pressure nullspace is removed by imposing that $\int_\Omega p \; dV = 0$ through a Lagrange multiplier (see Section 10.16.1).



From left to right: velocity, pressure and second invariant of strain rate for 32×32 mesh for the Donea & Huerta manufactured solution.

Unlike the results obtained with the penalty formulation (see Section 1), the pressure showcases a very strong checkerboard pattern, similar to the one in Donea and Huerta [94] (2003). The amplitude of this checkerboard is unpredictable and changes from one resolution to the other.



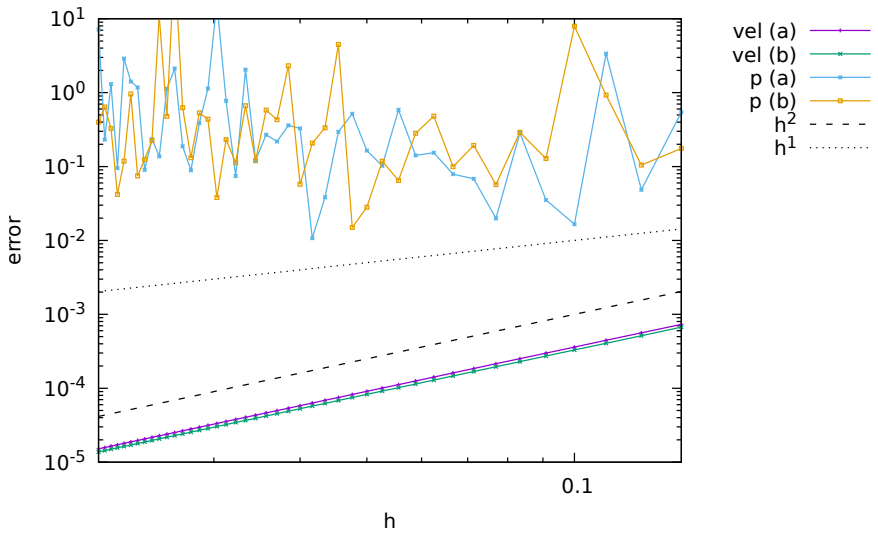
Left: pressure solution as shown in [94]; Middle: elemental pressure solution obtained with fieldstone on a 32×32 grid; Right: nodal pressure. Note that the vtu file was build in such a way so as to allow for this representation of the discontinuous pressure field.

The nodal pressure (obtained with a simple center-to-node algorithm) fails to recover a correct pressure at the four corners.

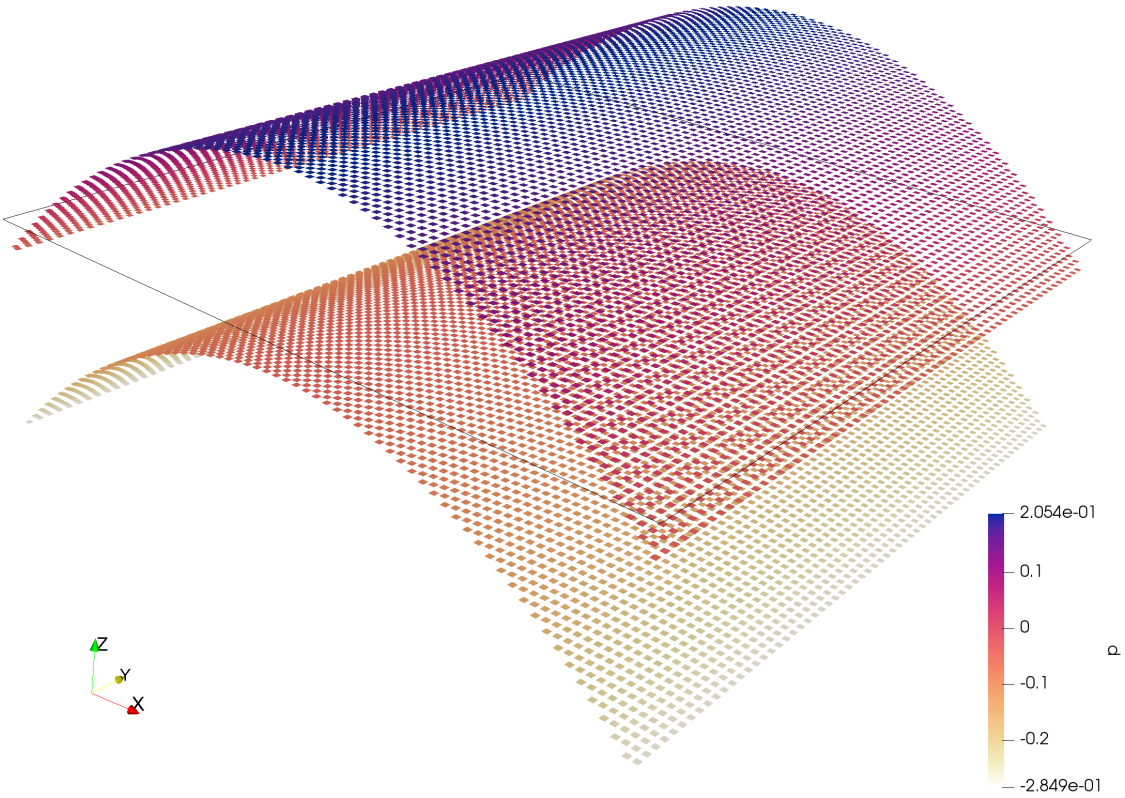
I have also explored the two formulations for the $\boldsymbol{C}$ matrix needed to compute $\mathbb{K}$ (see Section 10.8):

$$\boldsymbol{C}^{(a)} = \eta \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{C}^{(b)} = \eta \begin{pmatrix} 4/3 & -2/3 & 0 \\ -2/3 & 4/3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It seems that option (b) yields a slightly better velocity accuracy, but the pressure checkerboard yields large pressure error measurements in both cases:



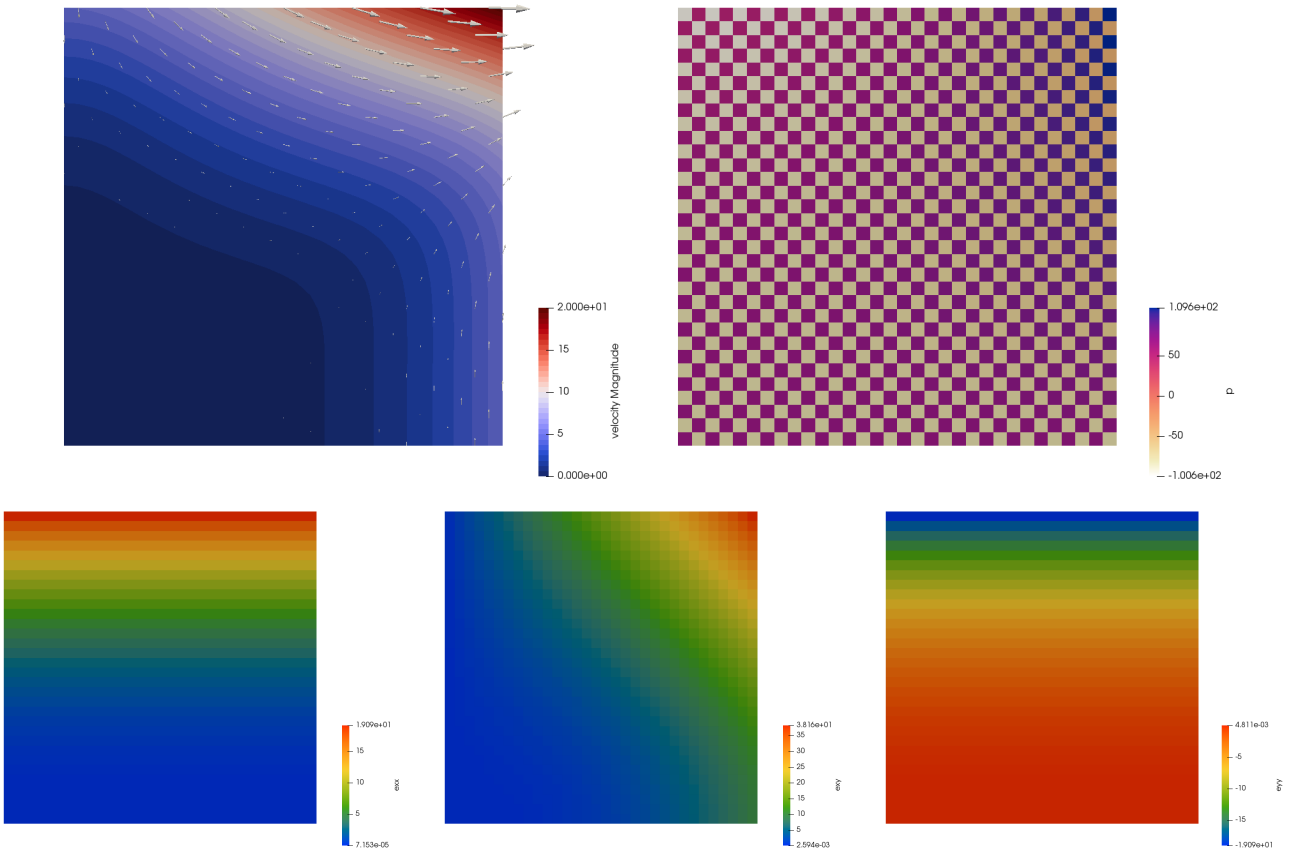
Velocity and pressure L_2 error as a function of the element size h.



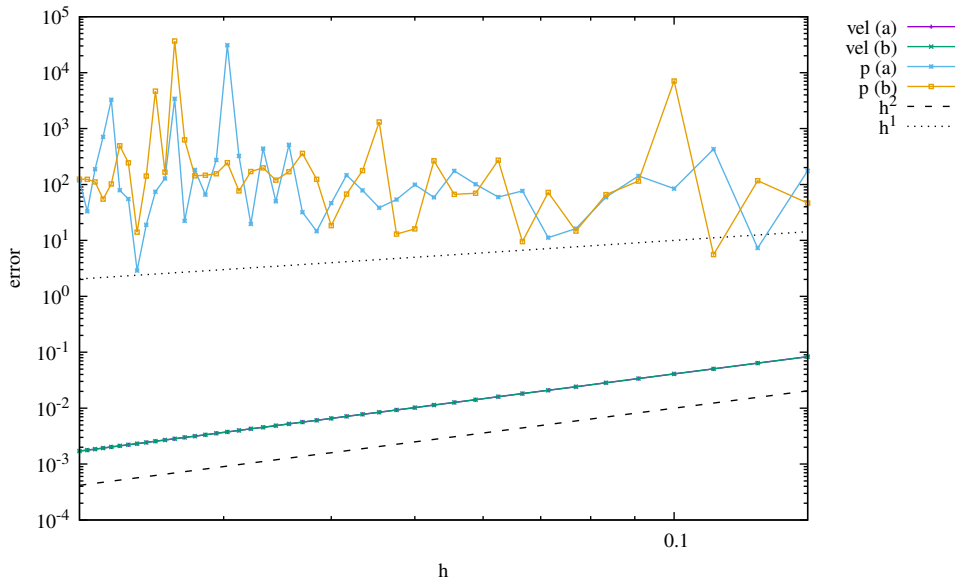
Elemental pressure for 128 × 128 mesh.

Burman & Hansbo manufactured solution

Details of this solution are in Section 18.32.



Mesh is 32×32 - results obtained with $\boldsymbol{C}^{(b)}$



Stone 15

saddle point problem with Schur complement approach - benchmark ($Q_1 \times P_0$)

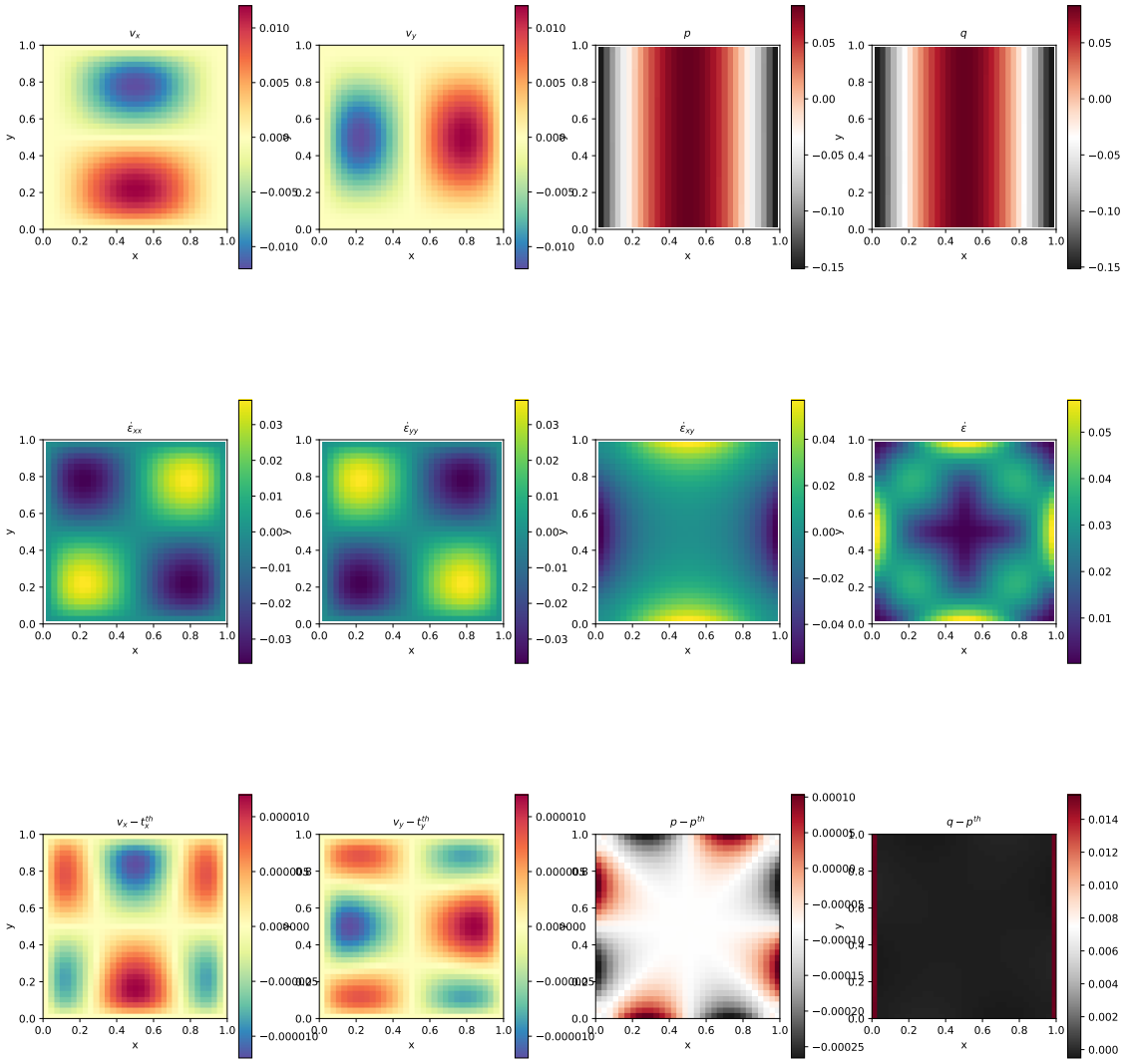


2D
box
Q1P0
analytical_benchmark
error_convergence
stokes_equation
residual
matplotlib
schur_complement

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_15`

Last revision: Sept. 7th, 2024.

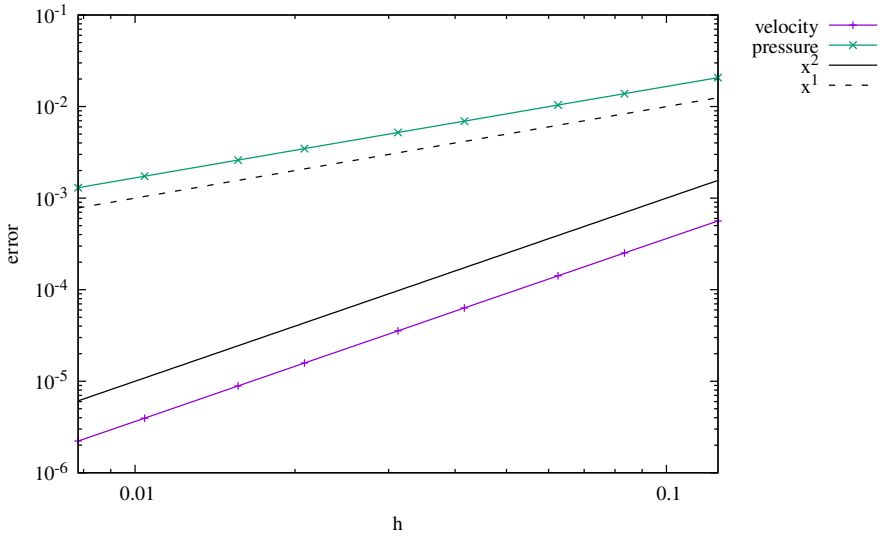
The details of the numerical setup are presented in Section 18.2. The main difference resides in the Schur complement approach to solve the Stokes system, as presented in Section 10.13 (see **solver_cg**). This iterative solver is very easy to implement once the blocks $\mathbb{K}$ and $\mathbb{G}$, as well as the rhs vectors f and h have been built.



Solution plot automatically generated with matplotlib.

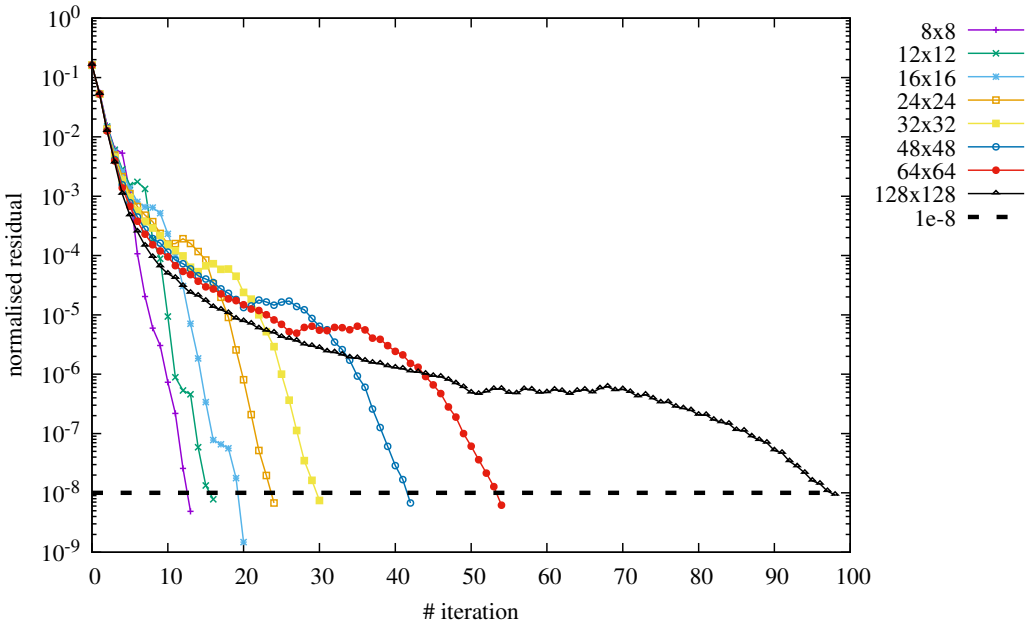
Rather interestingly the pressure checkerboard modes are not nearly as present as in STONE 1 which uses a full matrix approach.

Looking at the discretisation errors for velocity and pressure, we of course recover the same rates and values as in the full matrix case.



Finally, for each experiment the normalised residual (see **solver_cg**) was recorded. We see that all things equal the resolution has a strong influence on the number of iterations the solver must perform to reach the

required tolerance. This is one of the manifestations of the fact that the $Q_1 \times P_0$ element is not a stable element: the condition number of the matrix increases with resolution. We will see that this is not the case of stable elements such as $Q_2 \times Q_1$.



To Do, open questions, future work?

- build S and have python compute its smallest and largest eigenvalues as a function of resolution?

Stone 16

saddle point problem with Schur complement approach - Stokes sphere ($Q_1 \times P_0$)

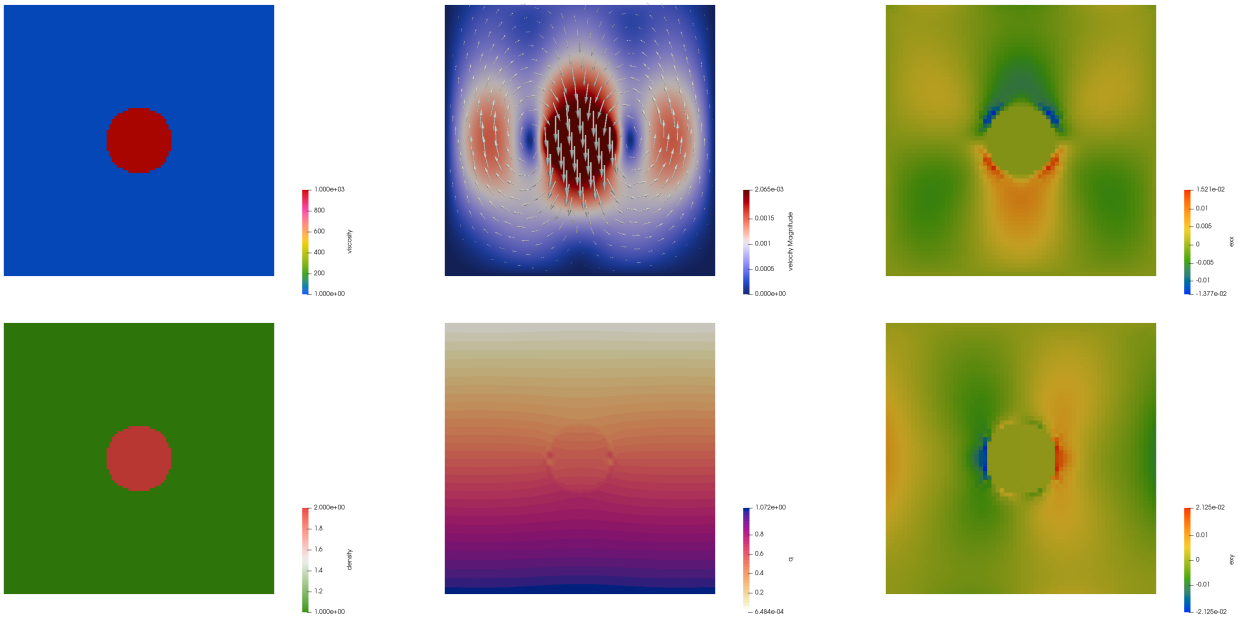


Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_16`

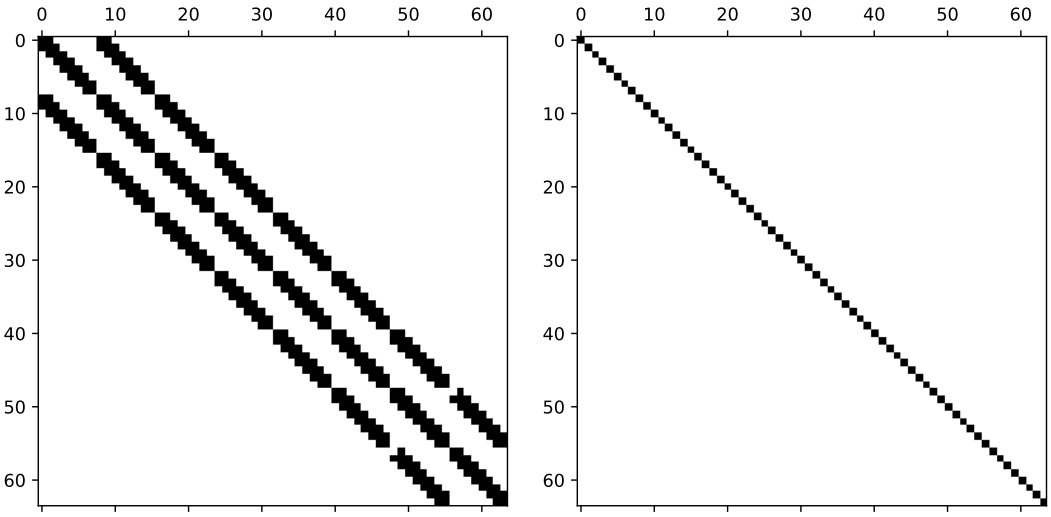
Last revision: Sept. 8th, 2024.

We are revisiting the 2D Stokes sphere problem, but this time we use the Schur complement approach to solve the Stokes system. We choose an open boundary at the top so that we do not have to deal with the pressure nullspace. No-slip boundary conditions are prescribed on the sides and bottom boundaries. Density is prescribed directly onto the quadrature points.

Because there are viscosity contrasts in the domain, it is advisable to use the Preconditioned Conjugate Gradient as presented in Section 10.13 (see **solver_pcg**).
The solver is implemented in the `schur_complement_cg_solver.py` file in the same folder.



Viscosity, density, velocity and pressure fields, resolution 96 × 96.



Resolution 8x8 elements. Matrix sparsity pattern for preconditioner #2 (left) and preconditioners 0,1,3,4 (right).

We have designed 5 preconditioners:

- `precond_type=0` It is the unit matrix (so it does nothing).
- `precond_type=1` This is a very simple one as it is diagonal and built element by element:

$$\mathbb{M}_{e,e} = \frac{h_x h_y}{\eta_e}$$

  where e is an element and η_e the viscosity evaluated in its center. Note that averages based on quadrature point values could also be considered (or any other kind of projection).

- `precond_type=2`

$$\mathbb{M} = \mathbb{G}^T (diag[\mathbb{K}])^{-1} \mathbb{G}$$

- `precond_type=3`

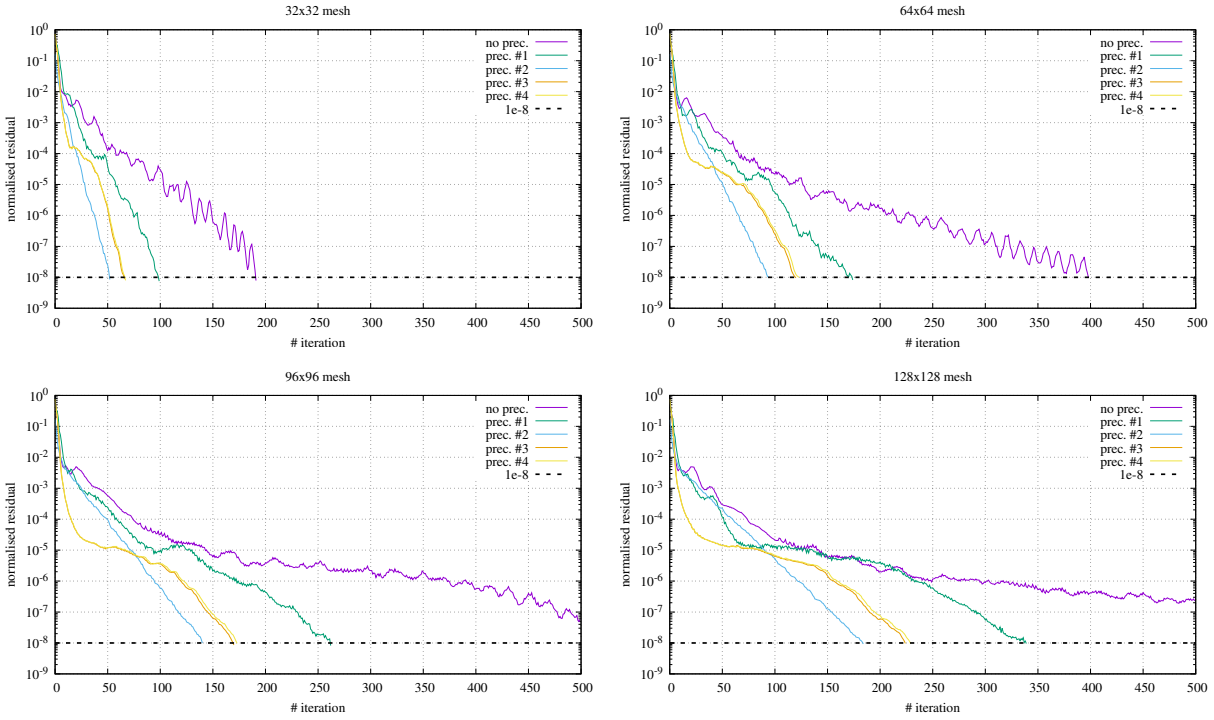
$$\mathbb{M} = diag\left[\mathbb{G}^T (diag[\mathbb{K}])^{-1} \mathbb{G}\right]$$

- `precond_type=4` Same as 2, but instead of using the diagonal of $\mathbb{G}^T (diag[\mathbb{K}])^{-1} \mathbb{G}$ we lump the matrix instead.

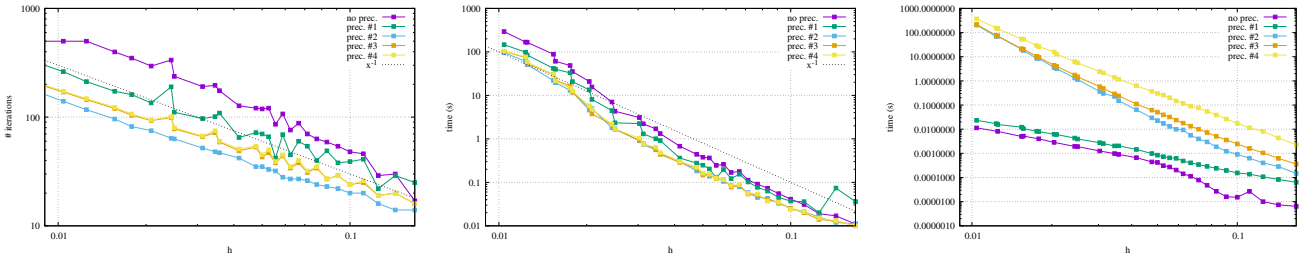
There are also multiple ways to prescribe the viscosity on the quadrature points:

- viscosity_field=1: viscosity is directly computed on the quadrature points;
- viscosity_field=2: viscosity is elemental;
- viscosity_field=3: viscosity is assigned to nodes and then interpolater onto quadrature points with an arithmetic averaging;
- viscosity_field=4: viscosity is assigned to nodes and then interpolater onto quadrature points with a geometric averaging;
- viscosity_field=5: viscosity is assigned to nodes and then interpolater onto quadrature points with a harmonic averaging.

viscosity_field=1

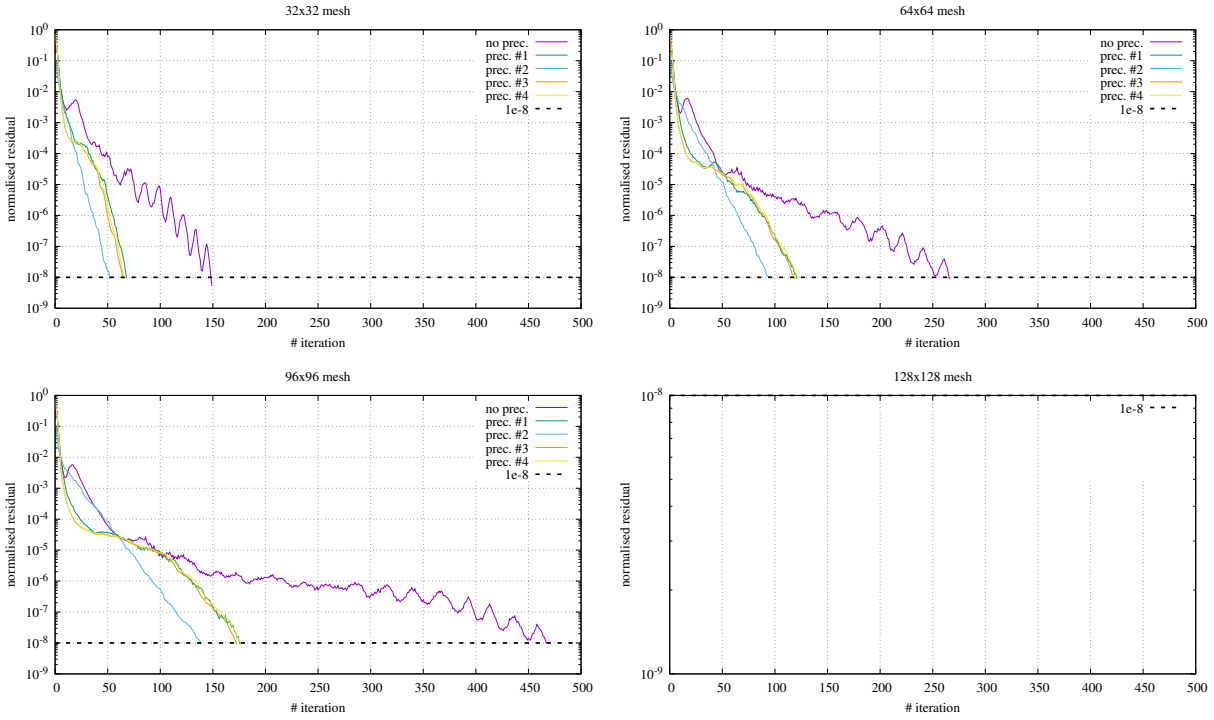


viscosity_field=1: Residual inside the solver as a function of the preconditioner type for 4 mesh resolutions.

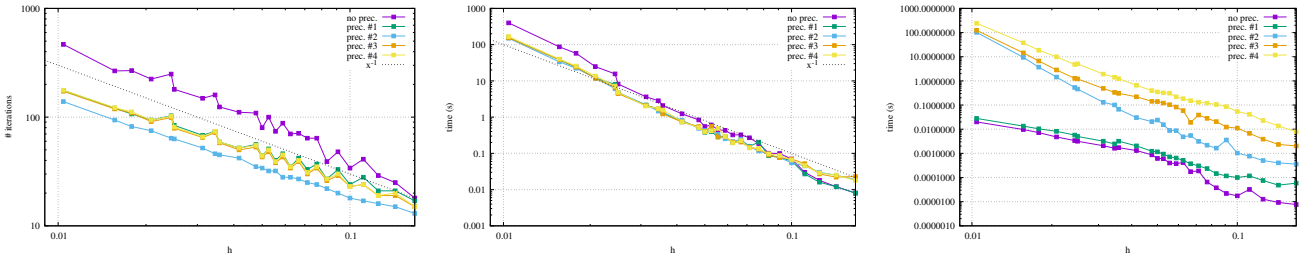


viscosity_field=1: Left: Number of iterations inside the solver; Middle: time spent in the solver. Right: time spent to build the preconditioner matrix.

viscosity_field=2

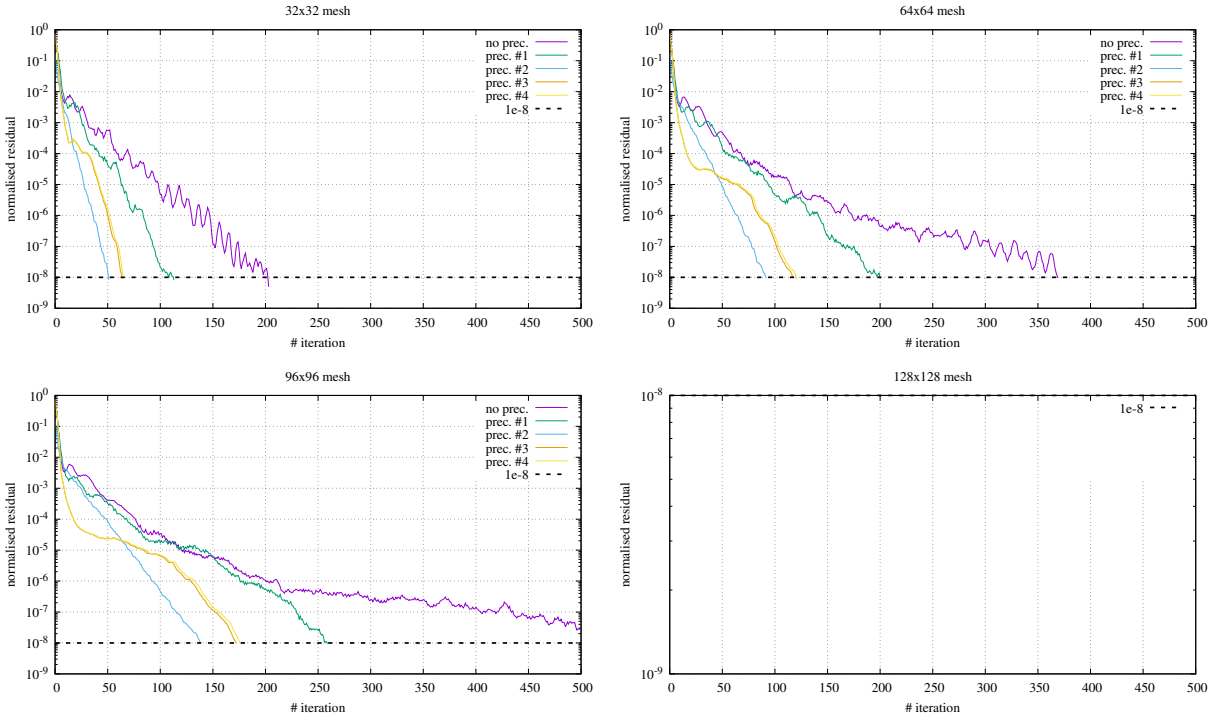


viscosity_field=2: Residual inside the solver as a function of the preconditioner type for 4 mesh resolutions.

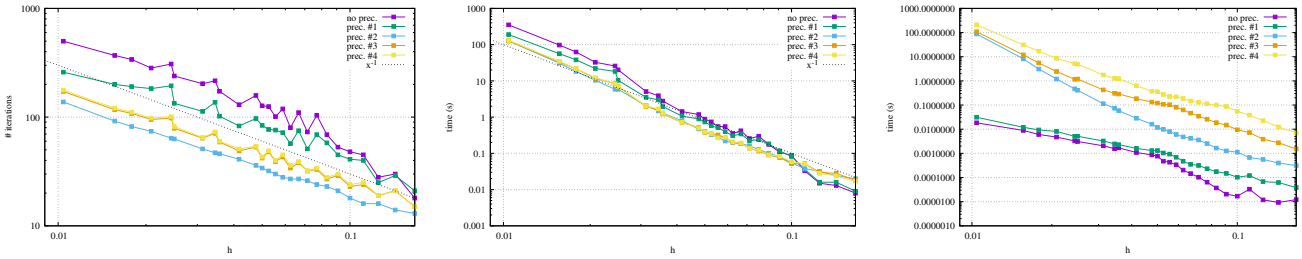


viscosity_field=2: Left: Number of iterations inside the solver; Middle: time spent in the solver. Right: time spent to build the preconditioner matrix.

viscosity_field=3

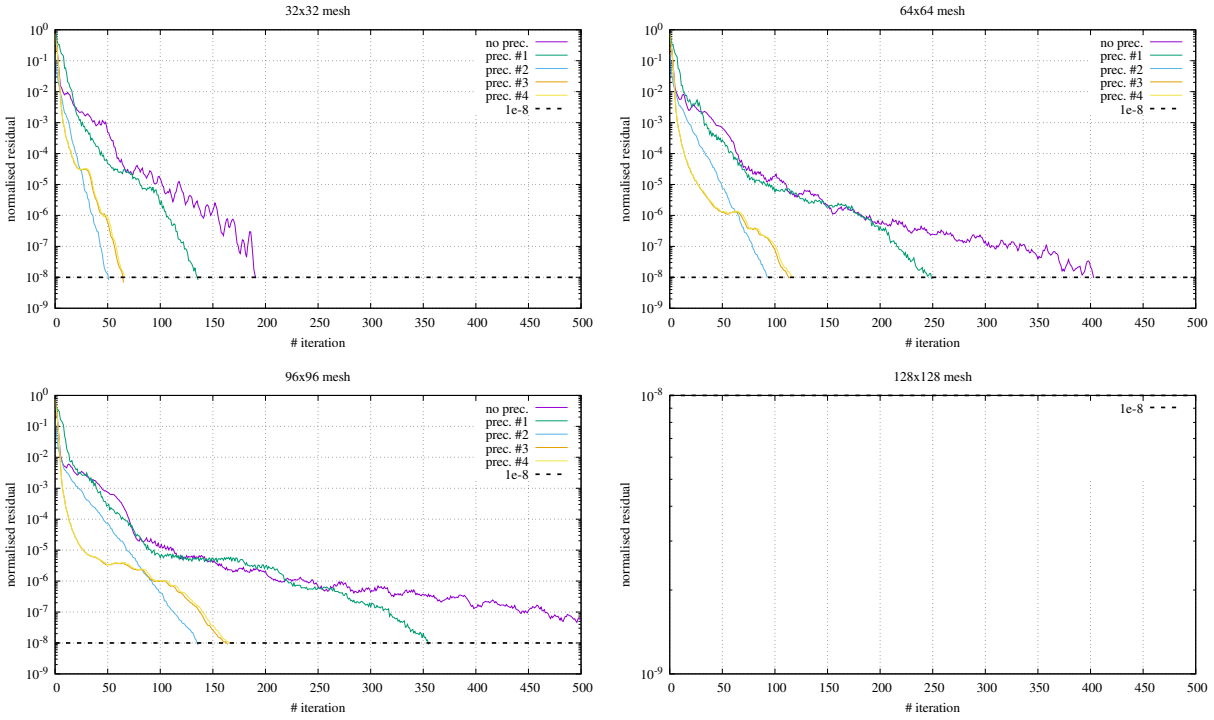


viscosity_field=3: Residual inside the solver as a function of the preconditioner type for 4 mesh resolutions.

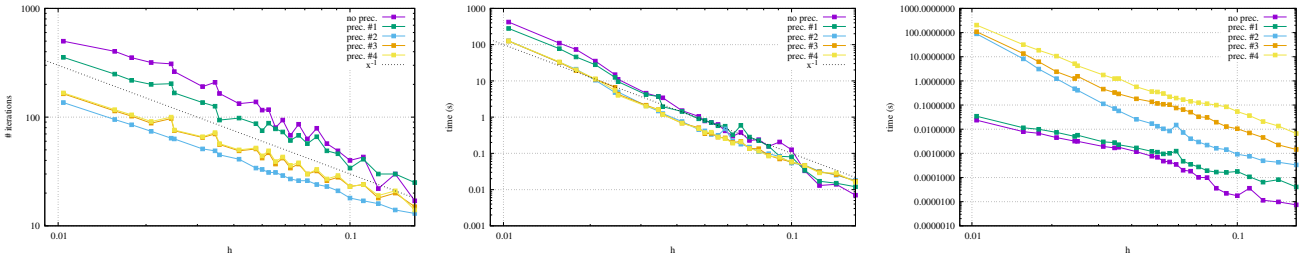


viscosity_field=3: Left: Number of iterations inside the solver; Middle: time spent in the solver. Right: time spent to build the preconditioner matrix.

viscosity_field=4

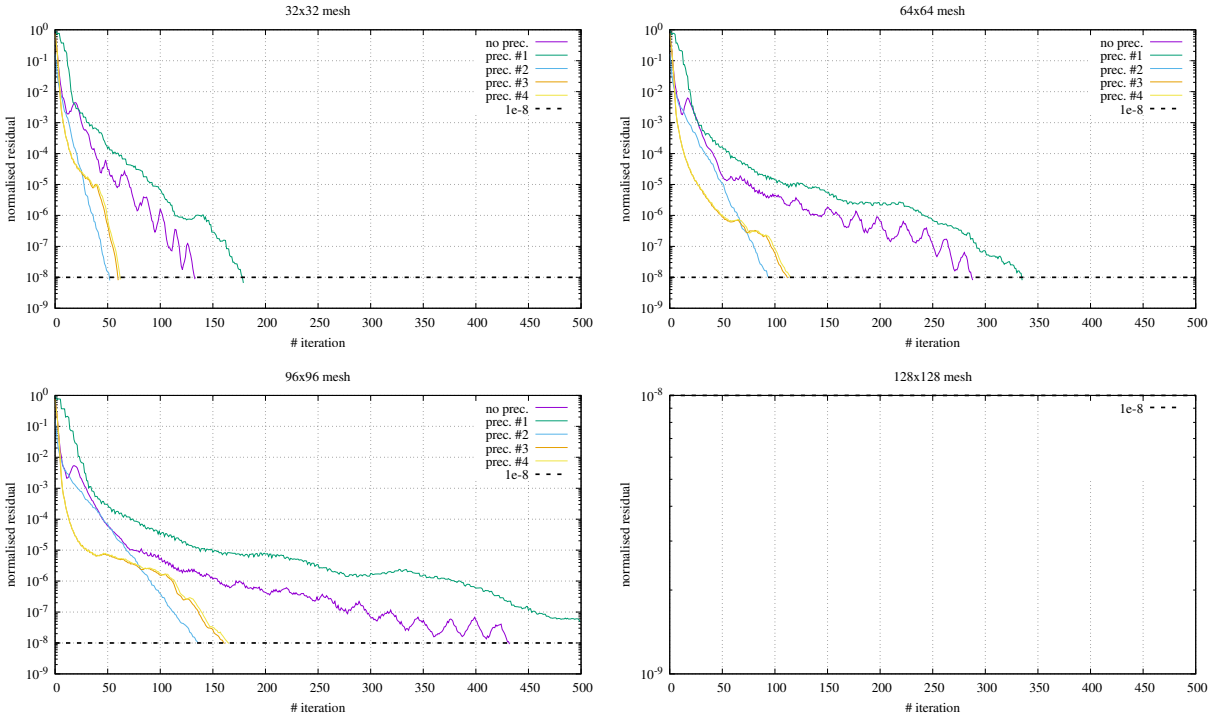


viscosity_field=4: Residual inside the solver as a function of the preconditioner type for 4 mesh resolutions.

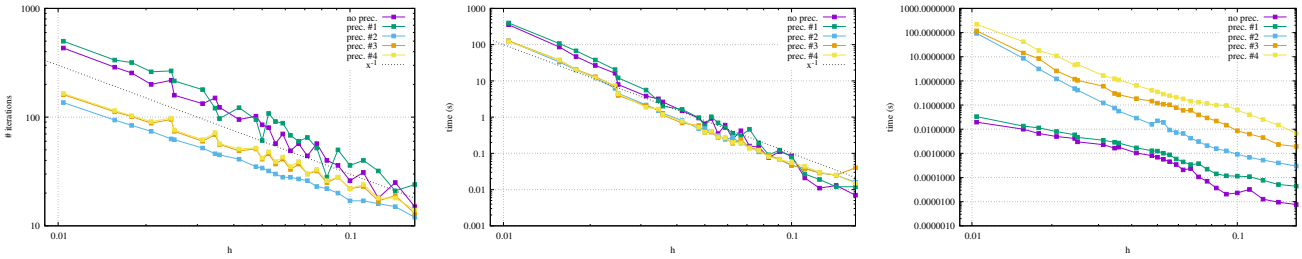


viscosity_field=4: Left: Number of iterations inside the solver; Middle: time spent in the solver. Right: time spent to build the preconditioner matrix.

viscosity_field=5

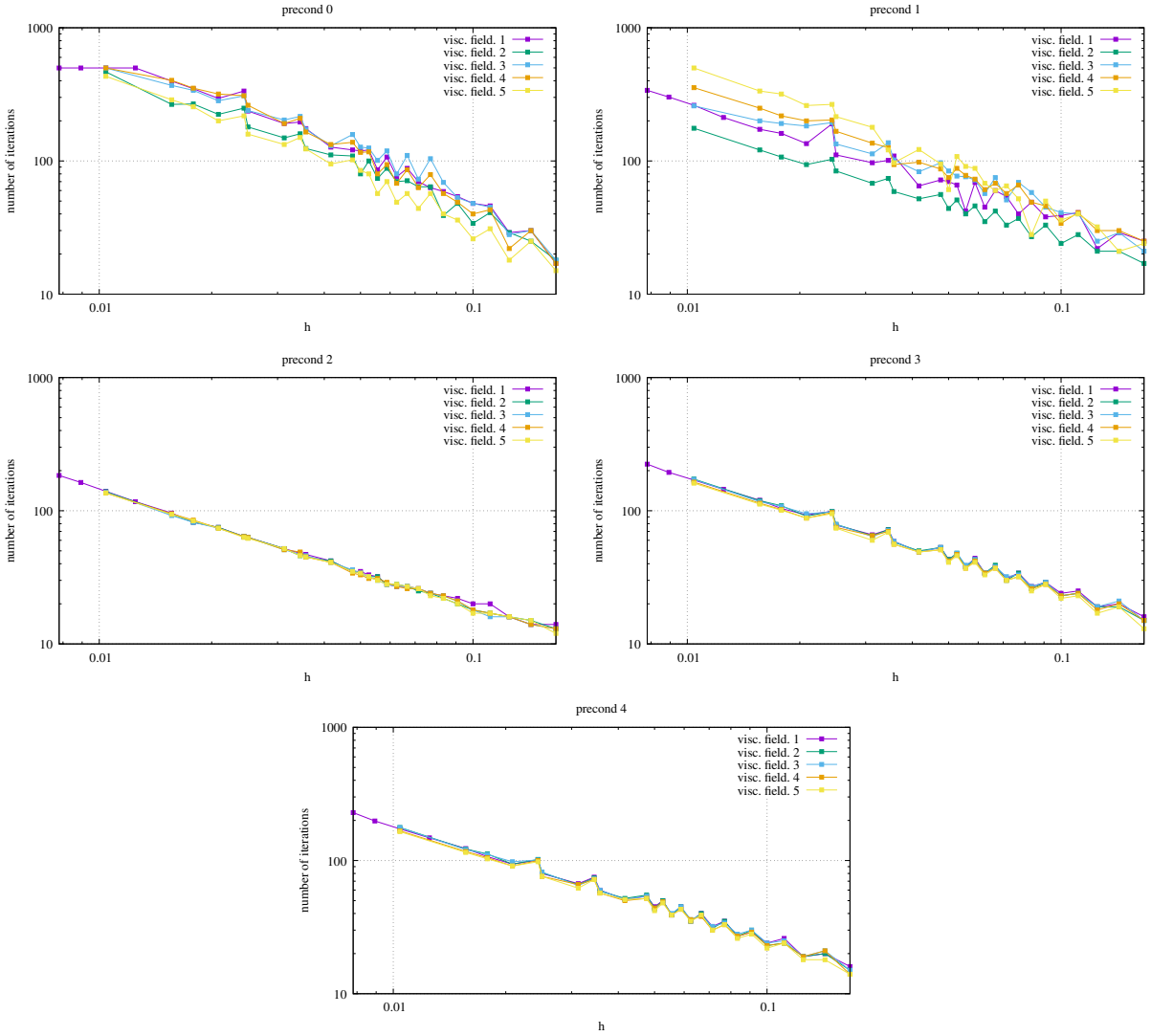


viscosity_field=5: Residual inside the solver as a function of the preconditioner type for 4 mesh resolutions.

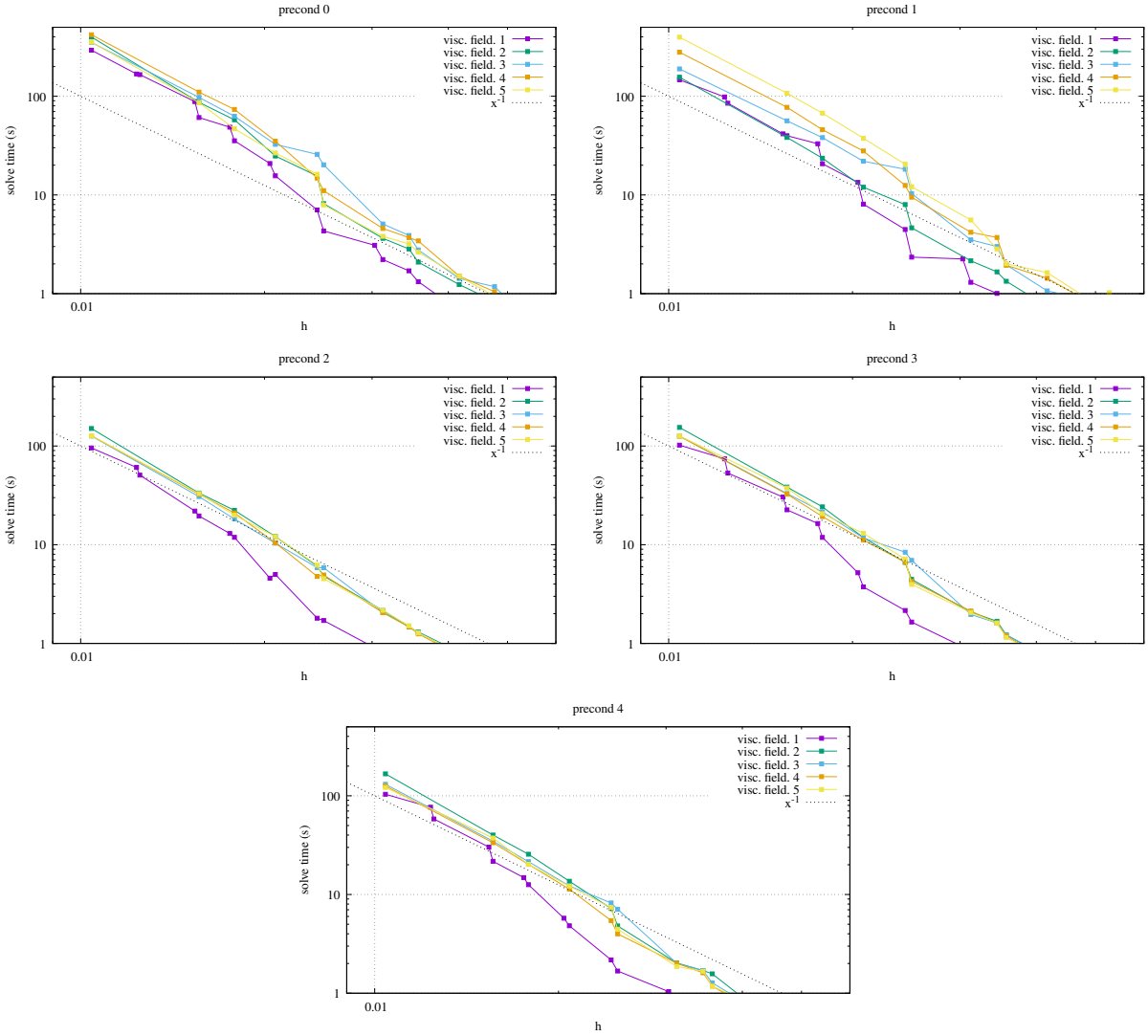


viscosity_field=5: Left: Number of iterations inside the solver; Middle: time spent in the solver. Right: time spent to build the preconditioner matrix.

On can also plot these same results (here the number of iterations in the outer solver), but this time per preconditioner type:



On can also plot these same results (here the solve time), but this time per preconditioner type:



These plots essentially prove in a rather concrete way why the $Q_1 \times P_0$ element is not an element which should be used: the number of iterations grows with the number of element while each iteration itself becomes more costly because the size of the matrix $\mathbb{K}$ increases too.

We see that the choice of preconditioner is important. Preconditioner #2 gives the best and most predictable results but its preconditioner matrix is not diagonal so that its use implies solving with it at each solver iteration, which has a cost: although at resolution 96x96 it requires the least iterations, the total time spent in the solver does not reflect this fact as well. It is also expensive to build!

Finally preconditioners 3 and 4 show extremely similar results, with a little advantage for #3, i.e. lumping does not help at all.

Since we have used the python solver as a black box inside our solver for $\mathbb{K}$ and $\mathbb{M}$ there would be much more to look into in this direction.

Remark. *I have tried lil_matrix and csr format on the K, G and M matrices but it makes the times to build these explode, so I stick to full matrices.*

Stone 17

Solving the full saddle point problem in 3D ($Q_1 \times P_0$, $Q_2 \times Q_1$)



3D
box
Q2Q1
analytical_benchmark
error_convergence
stokes_equation
paraview_output
pressure_normalisation

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_17`

When using $Q_1 \times P_0$ elements, this benchmark fails because of the Dirichlet b.c. on all 6 sides and all three components. However, as we will see, it does work well with $Q_2 \times Q_1$ elements.

This benchmark begins by postulating a polynomial solution to the 3D Stokes equation (see Dohrmann and Bochev [93] (2004)):

$$\vec{v} = \begin{pmatrix} x + x^2 + xy + x^3 y \\ y + xy + y^2 + x^2 y^2 \\ -2z - 3xz - 3yz - 5x^2 yz \end{pmatrix} \tag{17.1}$$

and

$$p = xyz + x^3 y^3 z - 5/32 \tag{17.2}$$

While it is then trivial to verify that this velocity field is divergence-free, the corresponding body force of the Stokes equation can be computed by inserting this solution into the momentum equation with a given viscosity η (constant or position/velocity/strain rate dependent). The domain is a unit cube and velocity boundary conditions simply use Eq. (**??**). Following Burstedde et al. [57] (2013), the viscosity is given by the smoothly varying function

$$\eta = \exp(1 - \beta(x(1-x) + y(1-y) + z(1-z))) \tag{17.3}$$

One can easily show that the ratio of viscosities $\eta^\star$ in the system follows $\eta^\star = \exp(-3\beta/4)$ so that choosing $\beta = 10$ yields $\eta^\star \simeq 1808$ and $\beta = 20$ yields $\eta^\star \simeq 3.269 \times 10^6$.

We start from the momentum conservation equation:

$$-\vec{\nabla} p + \vec{\nabla} \cdot (2\eta \dot{\vec{\varepsilon}}) = \vec{f}$$

The x-component of this equation writes

$$f_x = -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x}(2\eta\dot{\epsilon}_{xx}) + \frac{\partial}{\partial y}(2\eta\dot{\epsilon}_{xy}) + \frac{\partial}{\partial z}(2\eta\dot{\epsilon}_{xz}) \tag{17.4}$$

$$= -\frac{\partial p}{\partial x} + 2\eta\frac{\partial}{\partial x}\dot{\epsilon}_{xx} + 2\eta\frac{\partial}{\partial y}\dot{\epsilon}_{xy} + 2\eta\frac{\partial}{\partial z}\dot{\epsilon}_{xz} + 2\frac{\partial \eta}{\partial x}\dot{\epsilon}_{xx} + 2\frac{\partial \eta}{\partial y}\dot{\epsilon}_{xy} + 2\frac{\partial \eta}{\partial z}\dot{\epsilon}_{xz} \tag{17.5}$$

Let us compute all the block separately:

$$\begin{aligned}
\dot{\epsilon}_{xx} &= 1 + 2x + y + 3x^2y \\
\dot{\epsilon}_{yy} &= 1 + x + 2y + 2x^2y \\
\dot{\epsilon}_{zz} &= -2 - 3x - 3y - 5x^2y \\
2\dot{\epsilon}_{xy} &= (x + x^3) + (y + 2xy^2) = x + y + 2xy^2 + x^3 \\
2\dot{\epsilon}_{xz} &= (0) + (-3z - 10xyz) = -3z - 10xyz \\
2\dot{\epsilon}_{yz} &= (0) + (-3z - 5x^2z) = -3z - 5x^2z
\end{aligned}$$

In passing, one can verify that $\dot{\epsilon}_{xx} + \dot{\epsilon}_{yy} + \dot{\epsilon}_{zz} = 0$. We further have

$$\begin{aligned}
\frac{\partial}{\partial x}2\dot{\epsilon}_{xx} &= 2(2 + 6xy) \\
\frac{\partial}{\partial y}2\dot{\epsilon}_{xy} &= 1 + 4xy \\
\frac{\partial}{\partial z}2\dot{\epsilon}_{xz} &= -3 - 10xy \\
\frac{\partial}{\partial x}2\dot{\epsilon}_{xy} &= 1 + 2y^2 + 3x^2 \\
\frac{\partial}{\partial y}2\dot{\epsilon}_{yy} &= 2(2 + 2x^2) \\
\frac{\partial}{\partial z}2\dot{\epsilon}_{yz} &= -3 - 5x^2 \\
\frac{\partial}{\partial x}2\dot{\epsilon}_{xz} &= -10yz \\
\frac{\partial}{\partial y}2\dot{\epsilon}_{yz} &= 0 \\
\frac{\partial}{\partial z}2\dot{\epsilon}_{zz} &= 2(0) \\
\frac{\partial p}{\partial x} &= yz + 3x^2y^3z \qquad (17.6) \\
\frac{\partial p}{\partial y} &= xz + 3x^3y^2z \qquad (17.7) \\
\frac{\partial p}{\partial z} &= xy + x^3y^3 \qquad (17.8)
\end{aligned}$$

Pressure normalisation Here again, because Dirichlet boundary conditions are prescribed on all sides the pressure is known up to an arbitrary constant. This constant can be determined by (arbitrarily) choosing to normalised the pressure field as follows:

$$\int_\Omega p \, d\Omega = 0 \tag{17.9}$$

This is a single constraint associated to a single Lagrange multiplier λ and the global Stokes system takes the form

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} & 0 \\ \mathbb{G}^T & 0 & \mathcal{C} \\ 0 & \mathcal{C}^T & 0 \end{pmatrix} \begin{pmatrix} V \\ P \\ \lambda \end{pmatrix}$$

In this particular case the constraint matrix $\mathcal{C}$ is a vector and it only acts on the pressure degrees of freedom because of Eq.(17.9). Its exact expression is as follows:

$$\int_\Omega p\, d\Omega = \sum_e \int_{\Omega_e} p\, d\Omega = \sum_e \int_{\Omega_e} \sum_i N_i^p p_i\, d\Omega = \sum_e \sum_i \left(\int_{\Omega_e} N_i^p\, d\Omega \right) p_i = \sum_e \mathcal{C}_e \cdot \boldsymbol{p}_e$$

where $\boldsymbol{p}_e$ is the list of pressure dofs of element e. The elemental constraint vector contains the corresponding pressure basis functions integrated over the element. These elemental constraints are then assembled into the vector $\mathcal{C}$.

Constant viscosity

Choosing $\beta = 0$ yields a constant velocity $\eta(x,y,z) = \exp(1) \simeq 2.718$ (and greatly simplifies the right-hand side) so that

$$\frac{\partial}{\partial x}\eta(x,y,z) = 0 \tag{17.10}$$

$$\frac{\partial}{\partial y}\eta(x,y,z) = 0 \tag{17.11}$$

$$\frac{\partial}{\partial z}\eta(x,y,z) = 0 \tag{17.12}$$

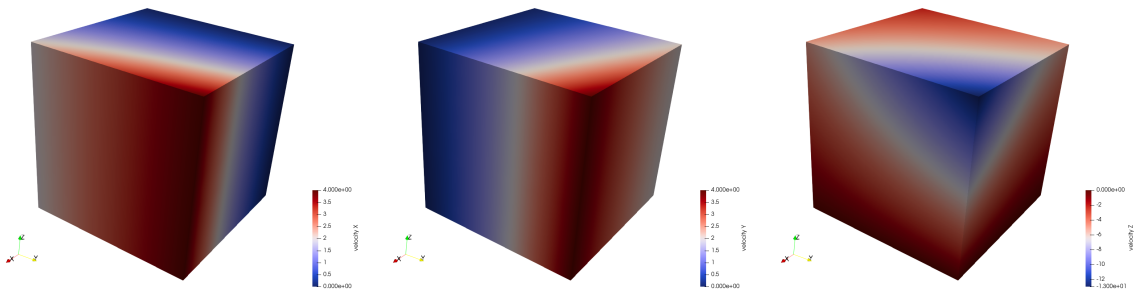
and

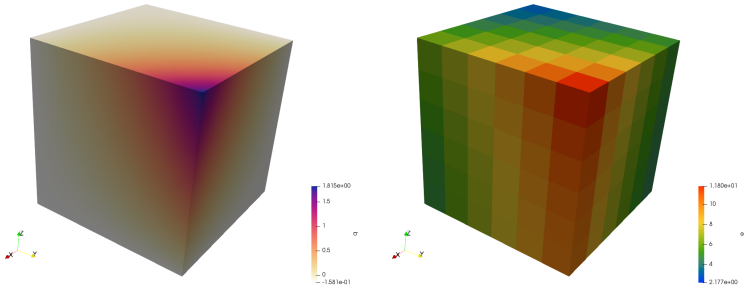
$$\begin{aligned}
f_x &= -\frac{\partial p}{\partial x} + 2\eta\frac{\partial}{\partial x}\dot{\epsilon}_{xx} + 2\eta\frac{\partial}{\partial y}\dot{\epsilon}_{xy} + 2\eta\frac{\partial}{\partial z}\dot{\epsilon}_{xz} \\
&= -(yz + 3x^2y^3z) + 2(2+6xy) + (1+4xy) + (-3-10xy) \\
&= -(yz + 3x^2y^3z) + \eta(2+6xy) \\
f_y &= -\frac{\partial p}{\partial y} + 2\eta\frac{\partial}{\partial x}\dot{\epsilon}_{xy} + 2\eta\frac{\partial}{\partial y}\dot{\epsilon}_{yy} + 2\eta\frac{\partial}{\partial z}\dot{\epsilon}_{yz} \\
&= -(xz + 3x^3y^2z) + \eta(1+2y^2+3x^2) + \eta 2(2+2x^2) + \eta(-3-5x^2) \\
&= -(xz + 3x^3y^2z) + \eta(2+2x^2+2y^2) \\
f_z &= -\frac{\partial p}{\partial z} + 2\eta\frac{\partial}{\partial x}\dot{\epsilon}_{xz} + 2\eta\frac{\partial}{\partial y}\dot{\epsilon}_{yz} + 2\eta\frac{\partial}{\partial z}\dot{\epsilon}_{zz} \\
&= -(xy + x^3y^3) + \eta(-10yz) + 0 + 0 \\
&= -(xy + x^3y^3) + \eta(-10yz)
\end{aligned}$$

Finally

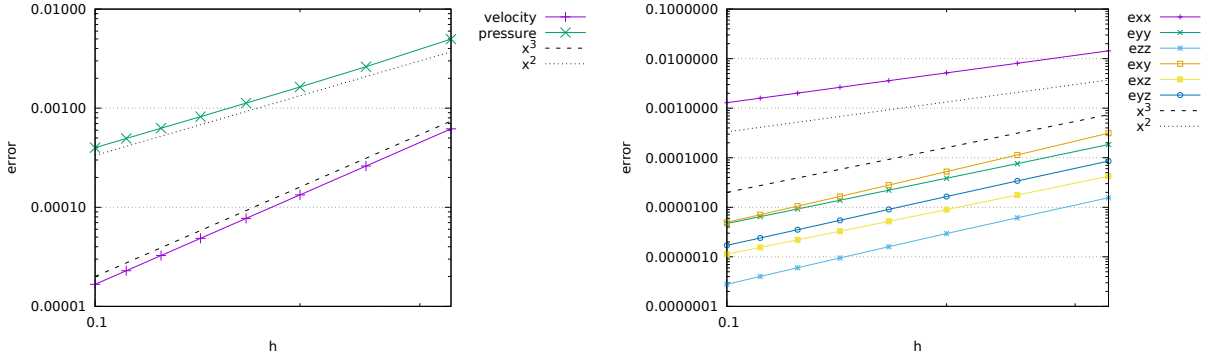
$$\vec{f} = -\begin{pmatrix} yz + 3x^2y^3z \\ xz + 3x^3y^2z \\ xy + x^3y^3 \end{pmatrix} + \eta \begin{pmatrix} 2+6xy \\ 2+2x^2+2y^2 \\ -10yz \end{pmatrix} \tag{17.13}$$

Remark. *There seems to be a sign problem with Eq.(26) in [57].*





Velocity, pressure and strain rate fields on a 6x6x6 grid.



Since the strain rate is the spatial derivative of the velocity field we should find that it converges like h^2 as expected. However I am quite puzzled as to why the convergence of $\dot{\varepsilon}_{xx}$ is quadratic but at the same time almost 2 orders of magnitude higher than the other 5 components which in turn converge abnormally fast like h^3.

Variable viscosity

The spatial derivatives of the viscosity are then given by

$$\begin{aligned}
\frac{\partial}{\partial x}\eta(x,y,z) &= -(1-2x)\beta\eta(x,y,z) \\
\frac{\partial}{\partial y}\eta(x,y,z) &= -(1-2y)\beta\eta(x,y,z) \\
\frac{\partial}{\partial z}\eta(x,y,z) &= -(1-2z)\beta\eta(x,y,z)
\end{aligned}$$

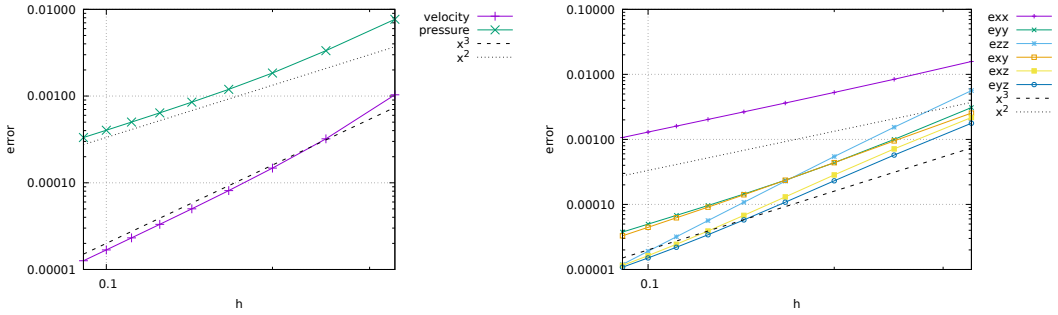
and the right-hand side by

$$\begin{aligned}
\vec{f} &= -\begin{pmatrix} yz+3x^2y^3z \\ xz+3x^3y^2z \\ xy+x^3y^3 \end{pmatrix} + \eta \begin{pmatrix} 2+6xy \\ 2+2x^2+2y^2 \\ -10yz \end{pmatrix} - (1-2x)\beta\eta(x,y,z) \begin{pmatrix} 2\dot{\epsilon}_{xx} \\ 2\dot{\epsilon}_{xy} \\ 2\dot{\epsilon}_{xz} \end{pmatrix} \\
&\quad -(1-2y)\beta\eta(x,y,z) \begin{pmatrix} 2\dot{\epsilon}_{xy} \\ 2\dot{\epsilon}_{yy} \\ 2\dot{\epsilon}_{yz} \end{pmatrix} - (1-2z)\beta\eta(x,y,z) \begin{pmatrix} 2\dot{\epsilon}_{xz} \\ 2\dot{\epsilon}_{yz} \\ 2\dot{\epsilon}_{zz} \end{pmatrix} \\
&= -\begin{pmatrix} yz+3x^2y^3z \\ xz+3x^3y^2z \\ xy+x^3y^3 \end{pmatrix} + \eta \begin{pmatrix} 2+6xy \\ 2+2x^2+2y^2 \\ -10yz \end{pmatrix} - (1-2x)\beta\eta \begin{pmatrix} 2+4x+2y+6x^2y \\ x+y+2xy^2+x^3 \\ -3z-10xyz \end{pmatrix} \\
&\quad -(1-2y)\beta\eta \begin{pmatrix} x+y+2xy^2+x^3 \\ 2+2x+4y+4x^2y \\ -3z-5x^2z \end{pmatrix} - (1-2z)\beta\eta \begin{pmatrix} -3z-10xyz \\ -3z-5x^2z \\ -4-6x-6y-10x^2y \end{pmatrix}
\end{aligned} \tag{17.14}$$

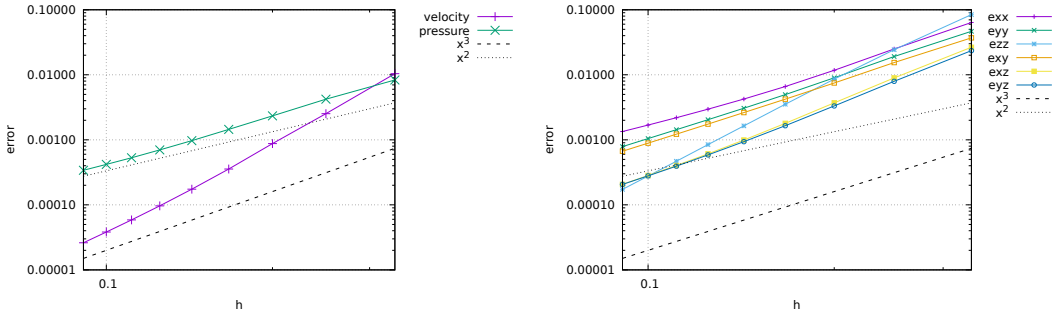
Note that at $(x,y,z)=(0,0,0)$, $\eta=\exp(1)$, and at $(x,y,z)=(0.5,0.5,0.5)$, $\eta=\exp(1-3\beta/4)$ so that the maximum viscosity ratio is given by

$$\eta^\star = \frac{\exp(1-3\beta/4)}{\exp(1)} = \exp(-3\beta/4)$$

By varying β between 1 and 22 we can get up to 7 orders of magnitude viscosity difference.

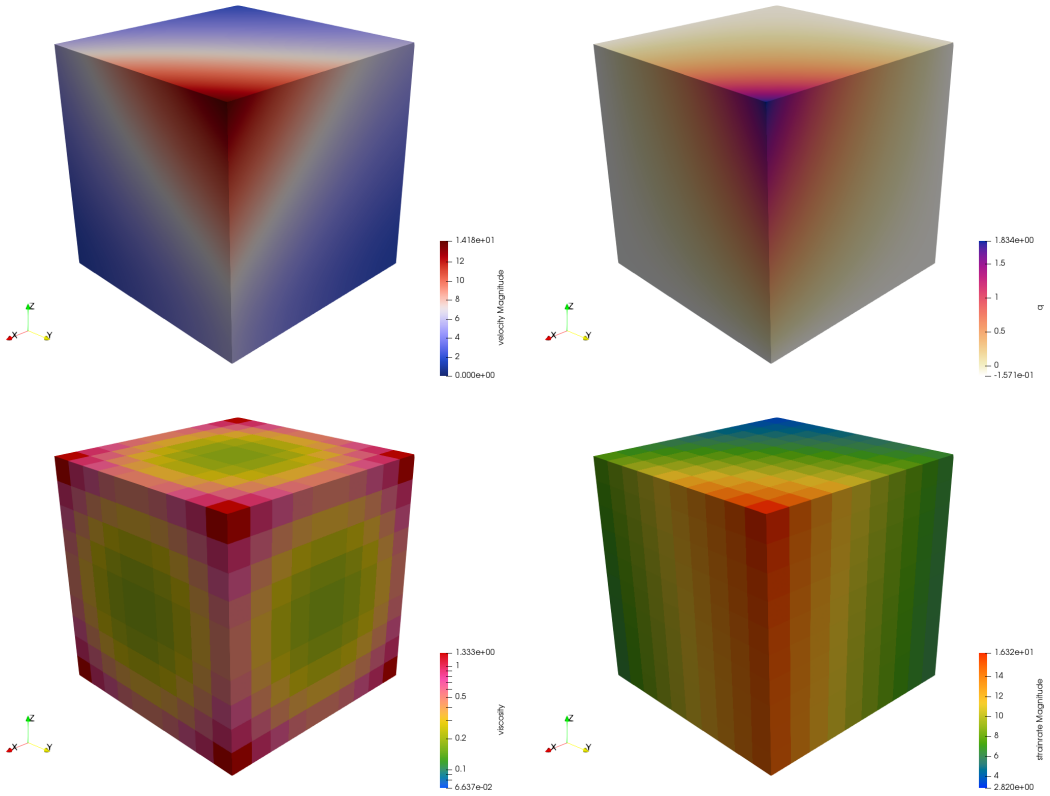


Obtained for $\beta = 5$



Obtained for $\beta = 10$

We see that with increasing viscosity contrasts higher and higher resolutions are needed to attain the expected convergence rates for velocity and pressure. As for strain rate convergence, as puzzled as before ...



Obtained for a 10x10x10 mesh and $\beta = 5$

Stone 18

D&H mms and more: Solving the full saddle point problem ($Q_2 \times Q_1$)



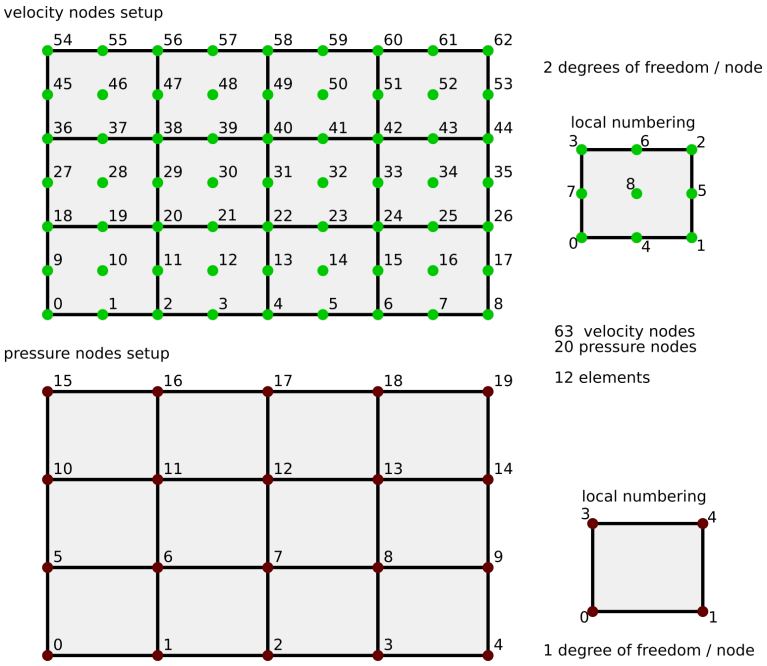
2D
box
Q2Q1
vrms
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
paraview_output
pressure_normalisation

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_18`

The code relies on $Q_2 \times Q_1$elements. Each element has $m_V = 9$ vertices so in total $ndof_V \times m_V = 18$ velocity dofs and $ndof_P * m_P = 4$ pressure dofs. The total number of velocity dofs is therefore $NfemV = NV \times ndofV$ while the total number of pressure dofs is $NfemP = NP \times ndofP$. The total number of dofs is then $Nfem = NfemV + NfemP$.

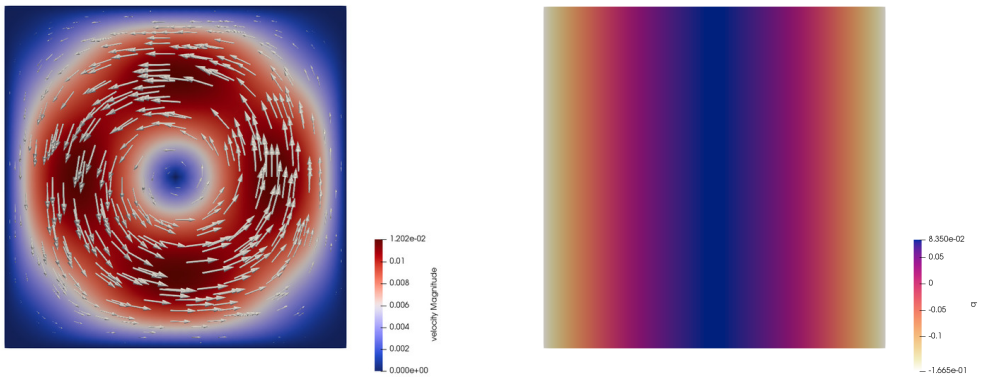
As a consequence, matrix $\mathbb{K}$ has size $NfemV, NfemV$ and matrix $\mathbb{G}$ has size $NfemV, NfemP$. Vector f is of size $NfemV$ and vector h is of size $NfemP$.

When needed the pressure nullspace is removed by imposing (Lagrange multiplier) that $\int_\Omega p \, dV = 0$.

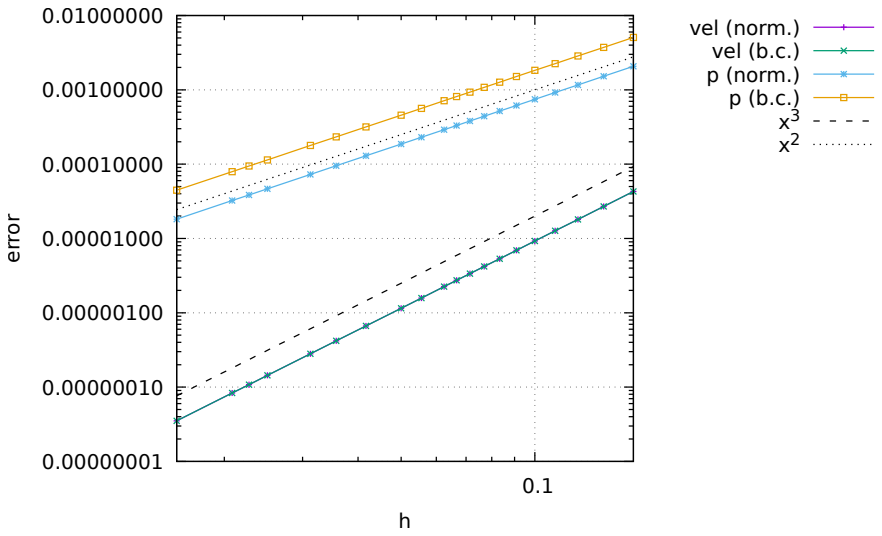


Donea & Huerta benchmark

The details of the numerical setup are presented in Section 18.2. It is triggered by setting bench=1.



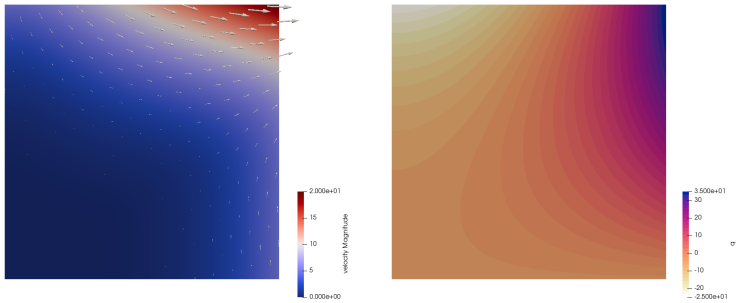
velocity and pressure fields for 32 × 32 elements grid



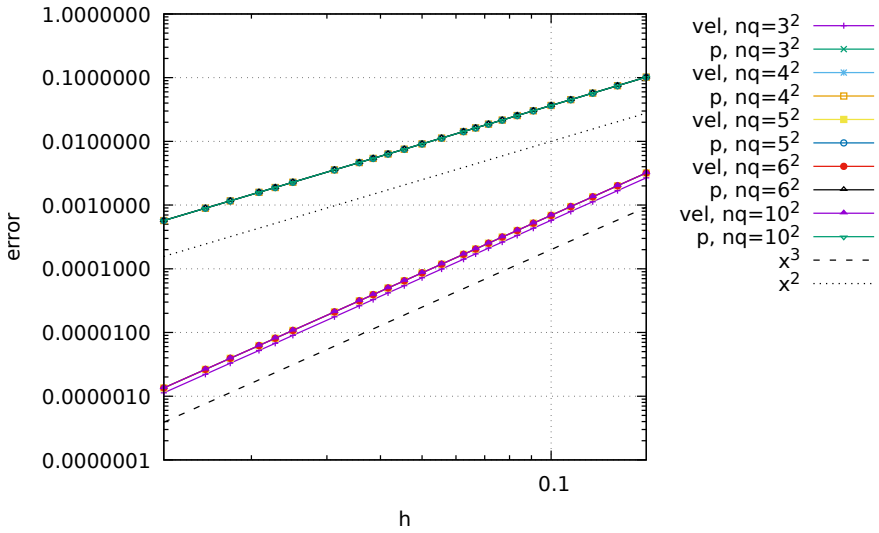
Velocity and pressure error convergence for both nullspace removal techniques. The zero average pressure approach yields smaller errors.

Burman & Hansbo manufactured solution

The details of the numerical setup are presented in Section 18.32.



velocity and pressure fields for 64 × 64 elements grid

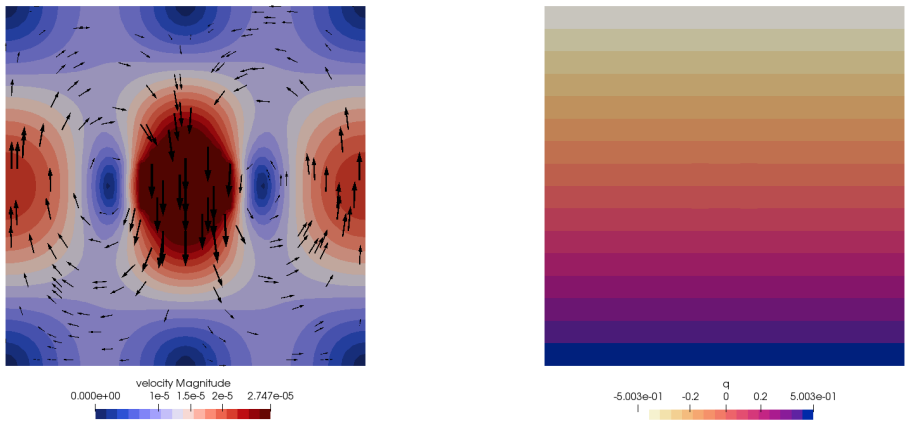


Velocity and pressure error convergence. Zero average.

No good reason to run such higher numbers of quadrature points.

Stokes sphere benchmark

The setup is described in Section 19.24. It is triggered by setting bench=2.

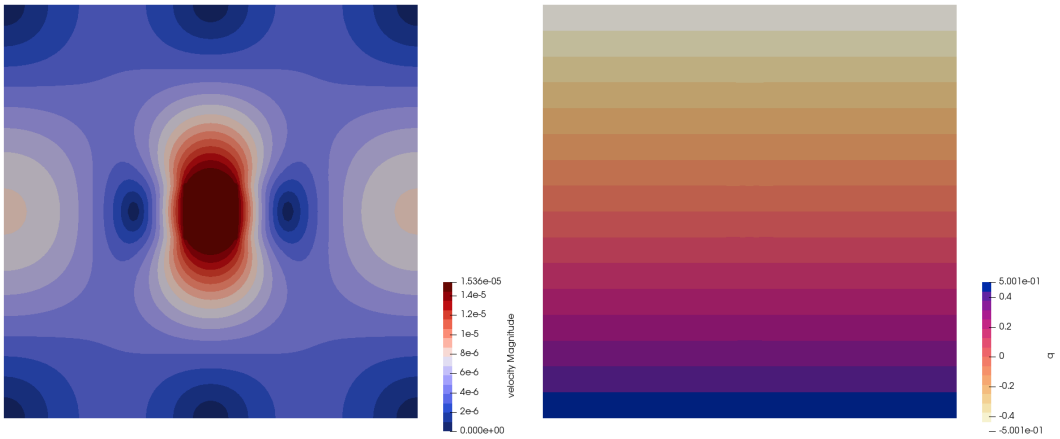


velocity and pressure fields for 32 × 32 elements grid and 3^2 quadrature points per element.

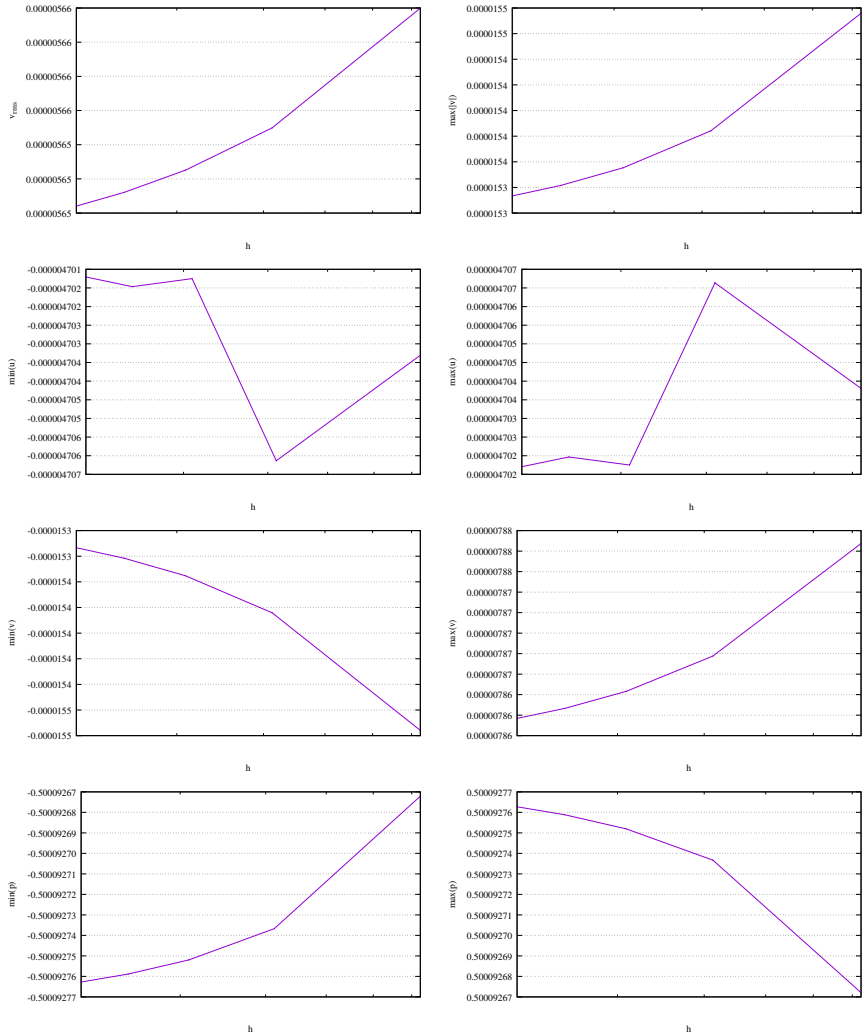
Sinking block benchmark

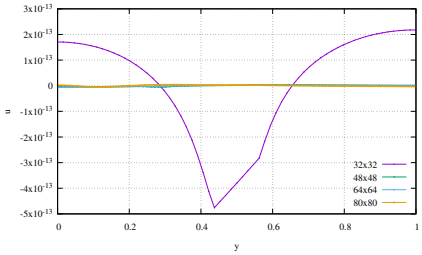
The setup is described in Section 19.22. It is triggered by setting bench=3.

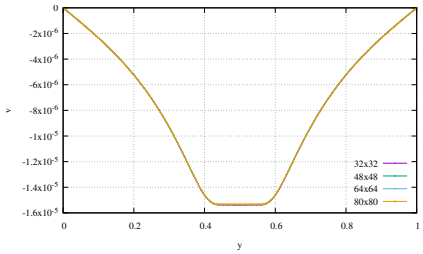
Free slip boundary conditions (FS)

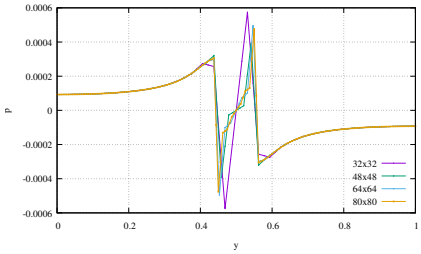


velocity and pressure fields for 48×48 elements grid and 3^2 quadrature points per element.

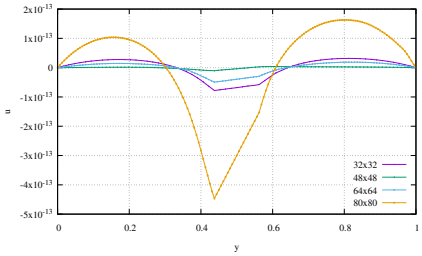


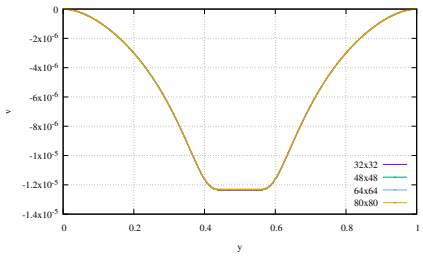


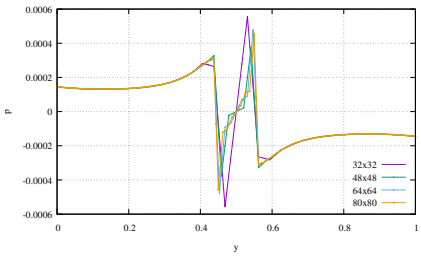




No slip boundary conditions (NS)







Stone 19

D&H mms: Solving the full saddle point problem ($Q_3 \times Q_2$)

```
2D
box
Q3Q2
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
paraview_output
pressure_normalisation
```

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_19`

The details of the numerical setup are presented in Section 18.2.

Each element has $m_V = 16$ vertices so in total $ndof_V \times m_V = 32$ velocity dofs and $ndof_P * m_P = 9$ pressure dofs. The total number of velocity dofs is therefore $NfemV = nnp \times ndofV$ while the total number of pressure dofs is $NfemP = nel$. The total number of dofs is then $Nfem = NfemV + NfemP$.

As a consequence, matrix $\mathbb{K}$ has size $NfemV, NfemV$ and matrix $\mathbb{G}$ has size $NfemV, NfemP$. Vector f is of size $NfemV$ and vector h is of size $NfemP$.

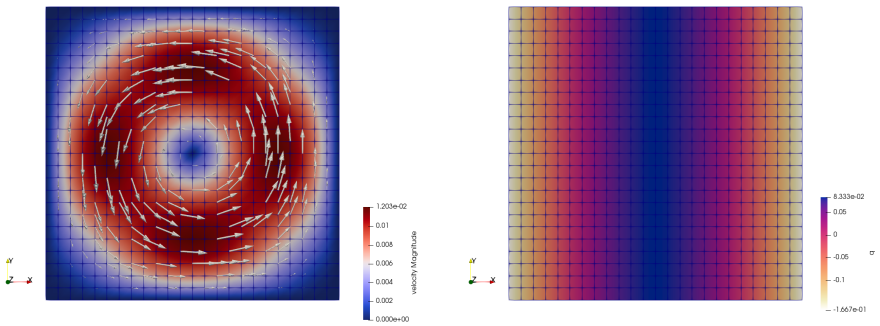
```
60===61===62===63===64===65===66===67===68===70
||             ||             ||             ||
50   51   52   53   54   55   56   57   58   59
||             ||             ||             ||
40   41   42   43   44   45   46   47   48   49
||             ||             ||             ||
30===31===32===33===34===35===36===37===38===39
||             ||             ||             ||
20   21   22   23   24   25   26   27   28   29
||             ||             ||             ||
10   11   12   13   14   15   16   17   18   19
||             ||             ||             ||
00===01===02===03===04===05===06===07===08===09
```

Example of 3x2 mesh. nnx=10, nny=7, nnp=70, nelx=3, nely=2, nel=6

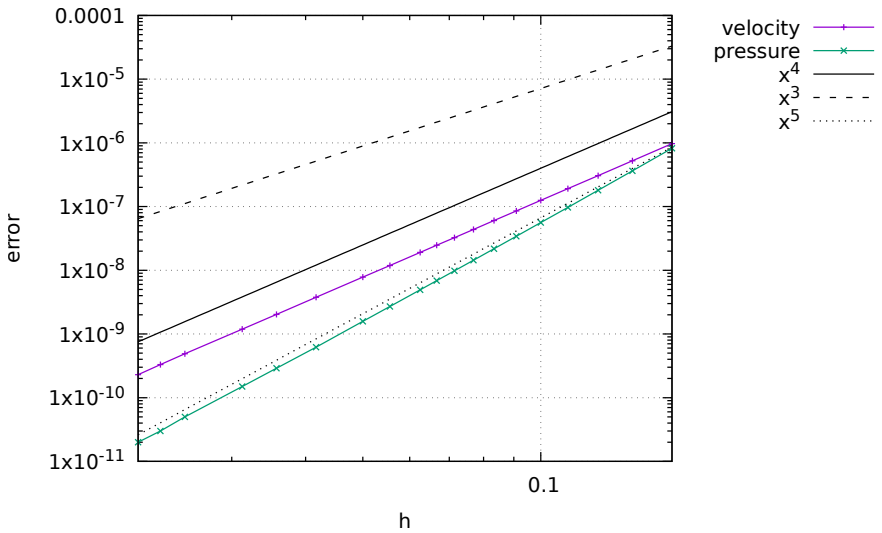
```
12===13===14===15          06=====07=====08
||   ||   ||   ||          ||     ||     ||
08===09===10===11          ||     ||     ||
```

```
||   ||   ||   ||          03=====04=====05
04===05===06===07          ||     ||     ||
||   ||   ||   ||          ||     ||     ||
00===01===02===03          00=====01=====02

Velocity (Q3)              Pressure (Q2)

(r,s)_{00}=(-1,-1)         (r,s)_{00}=(-1,-1)
(r,s)_{01}=(-1/3,-1)       (r,s)_{01}=(0,-1)
(r,s)_{02}=(+1/3,-1)       (r,s)_{02}=(+1,-1)
(r,s)_{03}=(+1,-1)         (r,s)_{03}=(-1,0)
(r,s)_{04}=(-1,-1/3)       (r,s)_{04}=(0,0)
(r,s)_{05}=(-1/3,-1/3)     (r,s)_{05}=(+1,0)
(r,s)_{06}=(+1/3,-1/3)     (r,s)_{06}=(-1,+1)
(r,s)_{07}=(+1,-1/3)       (r,s)_{07}=(0,+1)
(r,s)_{08}=(-1,+1/3)       (r,s)_{08}=(+1,+1)
(r,s)_{09}=(-1/3,+1/3)
(r,s)_{10}=(+1/3,+1/3)
(r,s)_{11}=(+1,+1/3)
(r,s)_{12}=(-1,+1)
(r,s)_{13}=(-1/3,+1)
(r,s)_{14}=(+1/3,+1)
(r,s)_{15}=(+1,+1)
```

The reduced coordinates of the quadrature points and their weights are to be found in Section 7.1.1.



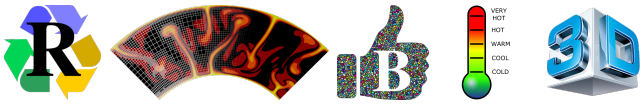
8x8 grid. Note that the 16-node Q_3 element does not exist in vtk so it is split into 9 Q_1 elements for visualisation purposes only.



velocity error rate is cubic, pressure superconvergent since the pressure field is quadratic and therefore lies into the Q2 space.

Stone 20

The Busse benchmark ($Q_1 \times P_0$)



3D
box
Q1P0
sparse_matrix
numerical_benchmark
stokes_equation
energy_equation
thermo_mechanical
time_stepping
viscous_rheology
buoyancy_driven
vrms
nusselt_number
rayleigh_number
paraview_output
pressure_scaling
steady_state

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_20`

This three-dimensional benchmark was first proposed by Busse et al (1993) [59]. It has been subsequently carried out in Tackley (1994) [317], Trompert & Hansen (1998) [334], Albers (2000) [2], O'Neill *et al.* (2006) [244], Davies *et al.* (2011) [84], Kronbichler *et al.* [200]. We here focus on Case 1 of [59]: an isoviscous bimodal convection experiment at $\mathsf{Ra} = 3 \cdot 10^5$.

The domain is of size $a \times b \times h$ with $a = 1.0079h$, $b = 0.6283h$ with $h = 2700\,\mathrm{km}$. It is filled with a Newtonian fluid characterised by $\rho_0 = 3300\mathrm{kg\,m^{-3}}$, $\alpha = 10^{-5}\mathrm{K^{-1}}$, $\mu = 8.0198 \times 10^{23}$ Pa s, $k = 3.564\ \mathrm{W\,m^{-1}\,K^{-1}}$, $C_p = 1080\ \mathrm{J\,K^{-1}\,kg^{-1}}$. The gravity vector is set to $\vec{g} = (0, 0, -10)^T \mathrm{m\,s^{-2}}$. The temperature is imposed at the bottom ($T = 3700$ °C) and at the top ($T = 0$ °C).

Note that using these numbers (as provided in the original paper), we arrive at Ra=29967.01, which is not exactly $3 \cdot 10^5$ as announced. Also, the heat diffusivity $\kappa = k/\rho_0 C_p$ is *exactly* 10^{-6} $\mathrm{m^2\,s^{-1}}$.

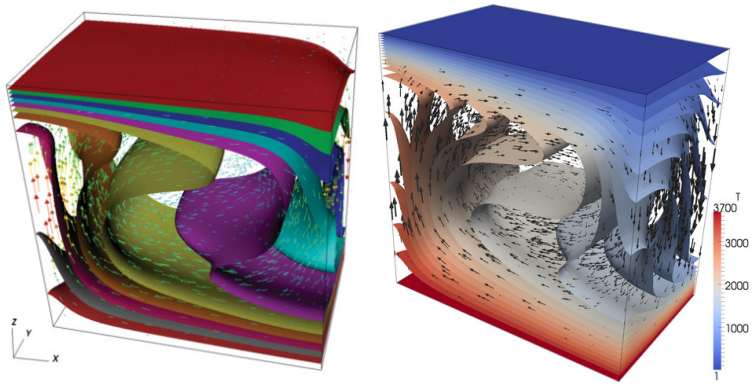
The various measurements presented in [59] are listed hereafter:

- The Nusselt number Nu computed at the top surface following Eq. (**??**):

$$\mathsf{Nu} = L_z \frac{\int\int_{z=L_z} \frac{\partial T}{\partial y} dxdy}{\int\int_{z=0} T dxdy}$$

- the root mean square velocity v_{rms} and the temperature mean square velocity T_{rms}

- The vertical velocity w and temperature T at points $\vec{r}_1 = (0, 0, L_z/2)$, $\vec{r}_2 = (L_x, 0, L_z/2)$, $\vec{r}_3 = (0, L_y, L_z/2)$ and $\vec{r}_4 = (L_x, L_y, L_z/2)$;
- the vertical component of the heat flux Q at the top surface at all four corners.



Left: Velocity field and isosurfaces of the temperature at steady state obtained with ASPECT . Taken from Kronbichler et al (2012) [200]. Right: same, obtained with ELEFANT [325].

Given the dimensions of the domain, here are resolutions that would yield (roughly) cubic elements:

1.0079×2700km	nelx=	16	20	24	28	32	36	40
0.6283×2700km	nely=	10	13	15	18	20	23	25
1.0000×2700km	nelz=	16	20	24	28	32	36	40

Methodology

In what follows I highlight a few important points which are key to understanding how the code is put together and works.

```
load needed modules and functions
define parameters
build V grid (xV,yV,zV)
build V connectivity (iconV)
define b.c. for velocity (bc_fixV,bc_valV)
build T grid (xT,yT,zT)
build T connectivity (iconT)
define b.c. for temperature (bc_fixT,bc_valT)
initial temperature field
.-------------------------> istep ----------------------.
|  build K,G,f,h                                         |
|  assemble them in A,rhs                                |
|  solve                                                 |
|  split solution vector in u,v,w,p                      |
|  {u,v,w}=relax*{u,v,w}+(1-relax)*{u,v,w}               |
|  compute vrms                                          |
|  build A, rhs for temperature                          |
|  solve for temperature T                               |
|  T=relax*T+(1-relax)*T                                 |
|  compute elemental strainrate                          |
|  compute nodal strainrate                              |
|  compute nodal pressure                                |
|  measure V and T at mid side edges, Nu ...             |
|  export to vtu and ascii files                         |
.--------------------------<-------------------------.
```

I fist load the shape functions which are in two separate files:

```
from shape_functionsV import NNV,dNNVdr,dNNVds,dNNVdt
from shape_functionsT import NNT,dNNTdr,dNNTds,dNNTdt
```

There are NV=nnx*nny*nnz velocity nodes and NT=NV temperature nodes.

The velocity grid is built: xV, yV, zV, iconV, and these are copied in xT, yT, zT and iconT for the temperature grid.

The initial temperature field is built as follows:

```
for i in range(0,NT):
   T[i]= (Temperature2-Temperature1)/Lz*zT[i]+Temperature1 \
       + 100*(np.cos(np.pi*xT[i]/Lx) + np.cos(np.pi*yT[i]/Ly))*np.sin(np.pi*zT[i]/Lz)
```

The $\boldsymbol{C}$ matrix of Eq. (**??**) is then built:

```
c_mat = np.array([[2,0,0,0,0,0],\
                  [0,2,0,0,0,0],\
                  [0,0,2,0,0,0],\
                  [0,0,0,1,0,0],\
                  [0,0,0,0,1,0],\
                  [0,0,0,0,0,1]],dtype=np.float64)
```

Getting to steady state In this case we are not so much interested in the path to steady state since the published values/measurements are *at* steady state. The mass and momentum conservation equations for incompressible Stokes flow do not contain a time derivative but the energy conservation equation does. At steady state the terms $\partial_t T$ is zero by definition so we must solve the following equation[1] (see Section **??**):

$$\vec{\mathbf{v}} \cdot \vec{\nabla} T - \vec{\nabla} \cdot k \vec{\nabla} T = 0 \tag{20.1}$$

The Finite Element discretisation of this equation yields

$$(\boldsymbol{K}_a + \boldsymbol{K}_d) \cdot \vec{T} = \vec{0}$$

which is much simpler than the time-dependent one and avoids to carefully consider the time-discretisation altogether.

We now have to solve three coupled equations:

$$-\vec{\nabla} p + \vec{\nabla} \cdot (2\eta \dot{\boldsymbol{\varepsilon}}^d) + \rho(T)\vec{g} = \vec{0} \tag{20.2}$$

$$\vec{\nabla} \cdot \vec{\mathbf{v}} = 0 \tag{20.3}$$

$$\vec{\mathbf{v}} \cdot \vec{\nabla} T - \vec{\nabla} \cdot k \vec{\nabla} T = 0 \tag{20.4}$$

We could then proceed to write the weak forms of these equations and cast these as we have done before, but this time considering velocity, pressure and temperature as unknowns of a (very) large system:

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} & . \\ \mathbb{G}^T & 0 & . \\ . & . & . \end{pmatrix} \cdot \begin{pmatrix} V \\ P \\ T \end{pmatrix} = \begin{pmatrix} . \\ . \\ . \end{pmatrix}$$

Once this matrix is filled a single solve will yield the steady state velocity, pressure and temperature fields!

The term $\rho(T)$ will naturally end up in the (1,3) block of the assembled matrix as matrix $\boldsymbol{L}$. The mass conservation equation in thisform is independent of temperature so the (2,3) block will be zero. The diffusion term $\boldsymbol{K}_d$ naturally finds its way to the (3,3) block and since there is not occurence of pressure in the energy equation the (3,2) block will be zero. We then obtain:

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} & \boldsymbol{L} \\ \mathbb{G}^T & 0 & 0 \\ . & 0 & \boldsymbol{K}_d \end{pmatrix} \cdot \begin{pmatrix} V \\ P \\ T \end{pmatrix} = \begin{pmatrix} . \\ . \\ . \end{pmatrix}$$

[1] Adiabatic heating, shear heating and internal heating are not considered in the benchmark

The last remaining term is the advection term of the energy equation and it is a problematic one: it features the product of the velocity by the temperature. As such it is nonlinear and one cannot either put it in the (3,1) block nor in the (3,3) block.

This approach fails, which is why the problem is solved iteratively. The idea is simple: when solving the coupled mass and momentum equations, assume temperature known, and when solving the energy equation assume velocity known. We can then alternatively solve one system and then the other, constantly updating the fields when re-building the matrices or right hand sides.

However, it is well known that a straightforward implementation of this algorithm does not work in practice, i.e. it fails to converge, which is why a relaxation scheme is often implemented.

1. Solve for velocity and pressure
2. relax velocity

```
u=relax*u+(1-relax)*u_old
v=relax*v+(1-relax)*v_old
w=relax*w+(1-relax)*w_old
```

3. Solve for temperature
4. relax temperature

```
T=relax*T+(1-relax)*T_old
```

5. check for convergence:

```
if np.abs(Nu-Nu_old)<1.e-5:
    break
```

6. store old fields

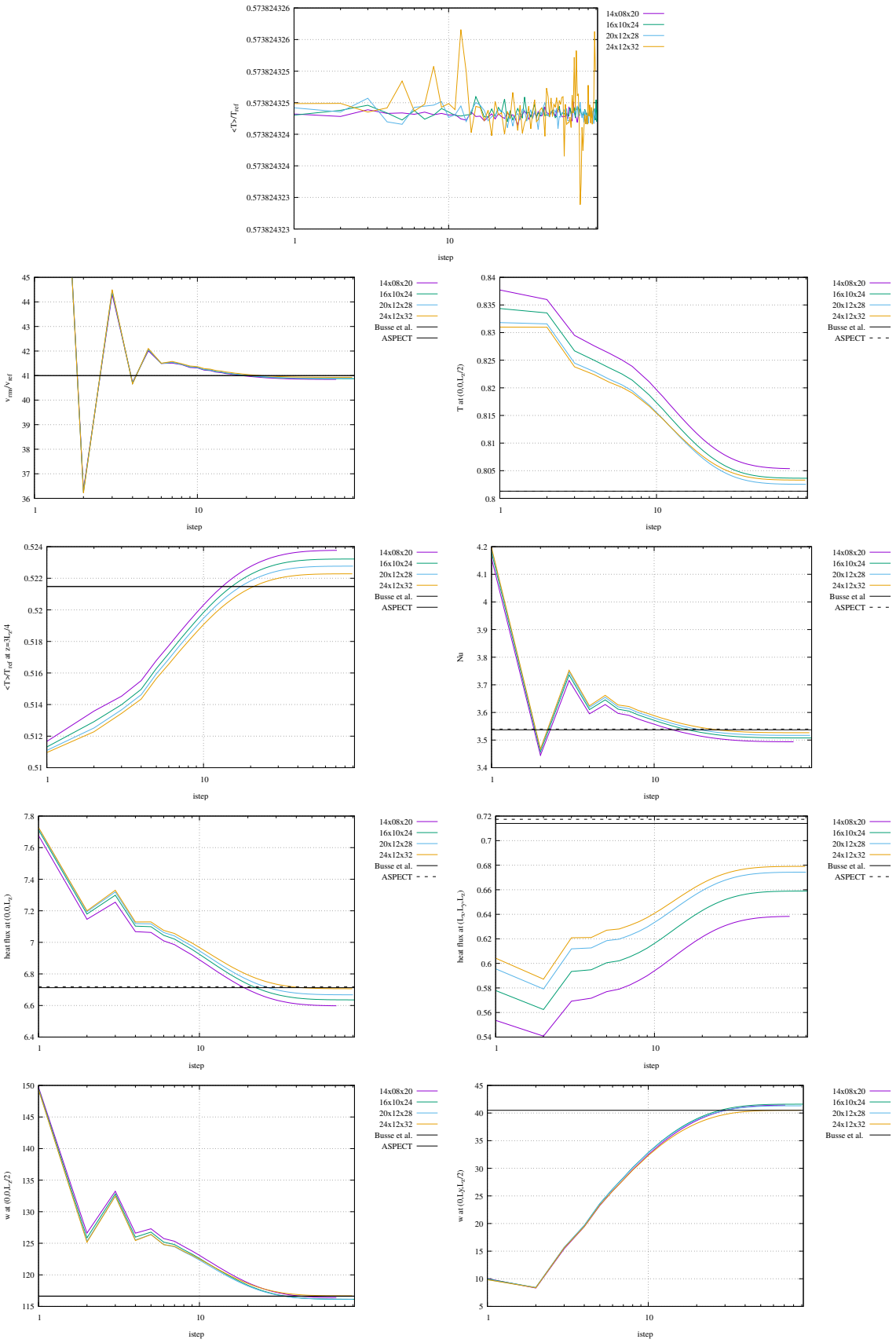
```
u_old=u
v_old=v
w_old=w
T_old=T
```

In this case I found that a relaxation coefficient of 1 actually works so it is used by default.

Computing nodal derivatives Another interesting approach here is how the strain rate is computed on the nodes as well as the temperature gradient. The strain rate is not needed for the required measurements of this benchmark but the Nusselt number calculations require $\partial T/\partial z$ at the top boundary. The idea is simple: loop over all elements, and for each element loop over its support nodes (in this case the $2 \times 2 \times 2$ nodes of the Q_1), compute the required derivative there and add its contribution to the nodal field. Finally divide the obtained field by the number of elements each node is part of.

Results

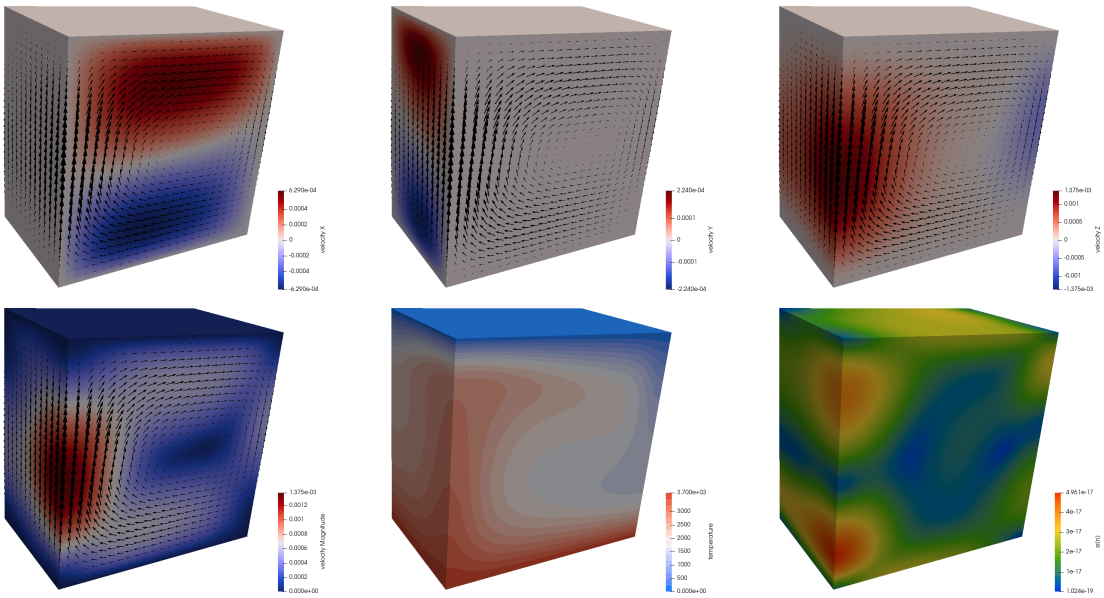
Given how the code is written and how costly 3D simulations are in general, high resolution runs take a long time to run, even using relaxation instead of time stepping.



The reported values for Busse et al. in the following table are taken from Table 3 of Busse et al.

[59] (1994). The reported values for fieldstone are adimensionalised by means of a reference temperature (3700K), a reference lengthscale 2700km, and a reference time $L_z^2/\kappa \sim 7.29e+18$s.

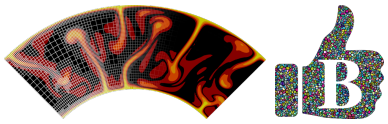
Mesh size	ASPECT Lz/24	$(Q_2 \times Q_1/Q_2)$ Lz/32	Lz/48	Busse et al [59] (best results)
Nu	3.5539	3.5447	3.5397	3.5374 ± 0.0005
v_{rms}	40.997	40.999	40.999	40.999 ± 0.004
$\langle T \rangle$ at $0.75 * Lz$	0.52148	0.52148	0.52148	0.52148 ± 0.00003
$w(0, 0, L_z/2)$	116.605	116.618	116.623	116.625 ± 0.030
$w(L_x, 0, L_z/2)$	-	-	-	-
$w(L_x, L_y, L_z/2)$	-	-	-	-
$w(0, L_y, L_z/2)$				40.500 ± 0.030
$T(0, 0, L_z/2)$	0.80126	0.80128	0.80129	0.80130 ± 0.00005
$T(L_x, 0, L_z/2)$	-	-	-	-
$T(L_x, L_y, L_z/2)$	-	-	-	-
$T(0, L_y, L_z/2)$				0.61876 ± 0.00005
$dTdz(0, 0, L_z)$	6.7679	6.7357	6.7189	6.7127 ± 0.0500
$dTdz(L_x, 0, L_z)$				1.5080 ± 0.0500
$dTdz(L_x, L_y, L_z)$	0.7237	0.7205	0.7174	0.7140 ± 0.0500
$dTdz(0, L_y, L_z)$				3.1740 ± 0.0500



Results obtained on 24x12x32 mesh

Stone 21

The annulus benchmark ($Q_2 \times Q_1$)



2D
annulus
Q2Q1
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
paraview_output

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_21`

This benchmark was developed in collaboration with Prof. E.G.P. Puckett.

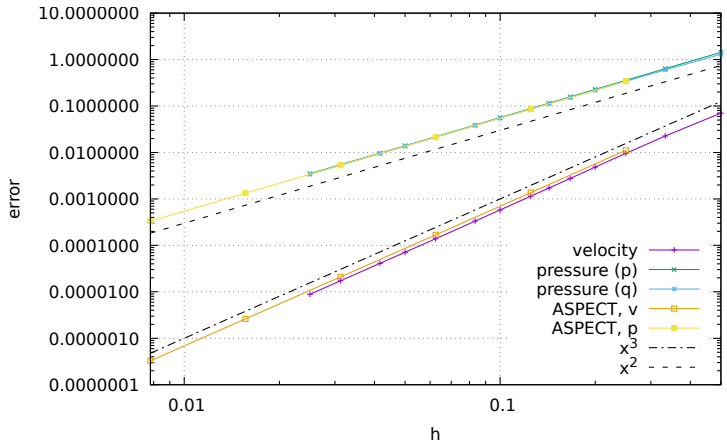
The benchmark is described fully in Section 19.18. The following results have been obtained with $k = 4$. Taylor-Hood $Q_2 \times Q_1$ elements are used with an isoparametric mapping.

For simplicity the layout of the points is borrowed from Stone 9 which showcases $Q_1 \times P_0$ elements. However the points are then 'grouped' to form Q_2 elements and Q_1 elements. The used parameters $nelr$ and $nelt$ are read in and then halved. Concretely, when the script is ran with $nelr = 10$ the real number of Q_2 elements in the radial direction is then 5. In this stone the number of elements in the tangential direction is fixed to $nelt = 12 * nelr$.

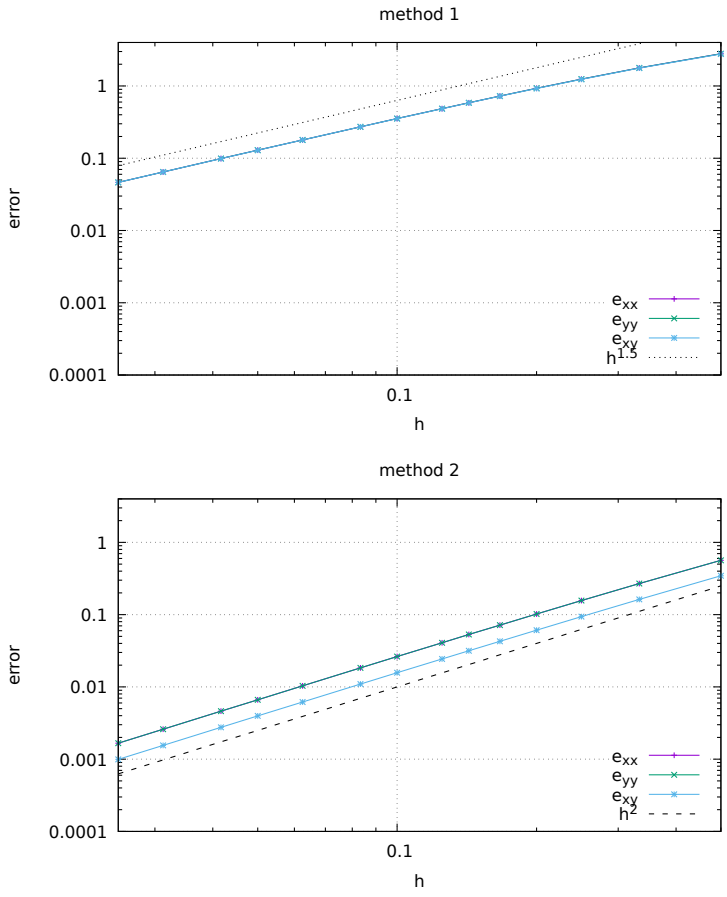
In this current version of the code, I have opted not to remove the pressure nullspace off the matrix and I simply carry out a simple post-processing which consists of removing the average pressure as measured on the inner boundary (or outer boundary – it yields the same result) from the pressure fields p and q (the projection of the Q_1 pressure onto the Q_2 mesh).

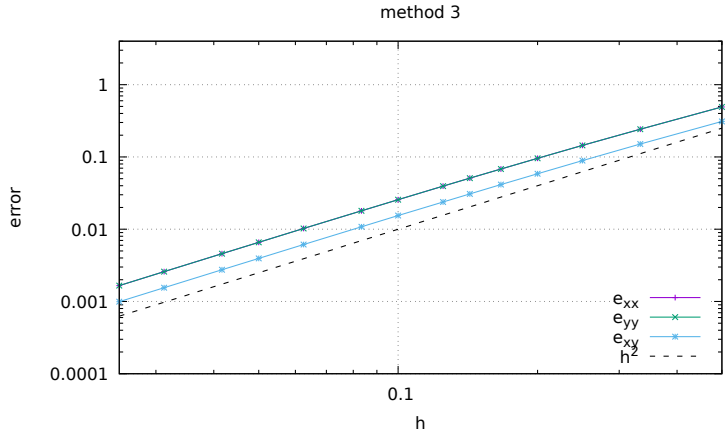
Just for a change I here compute the velocity gradient tensor first $\boldsymbol{L}(\vec{\mathrm{v}}) = \vec{\nabla}\vec{\mathrm{v}}$. I loop over elements. For each element I loop over its velocity nodes and use the Q_2 shape function derivatives to compute the four components $L_{xx}, L_{xy}, L_{yx}, L_{yy}$ at the node and add this value to the node (while keeping track of how many times a value has been added to the node). In the end I simply compute the average of the values on each node. From $\boldsymbol{L}(\vec{\mathrm{v}})$ I then compute $\dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}})$.

As expected we recover a cubic convergence for the velocity error and quadratic for the pressure. We also recover nearly identical results as those obtained with ASPECT (note that in this case $h = (R_2 - R_1)/nelr = 1/nelr$):

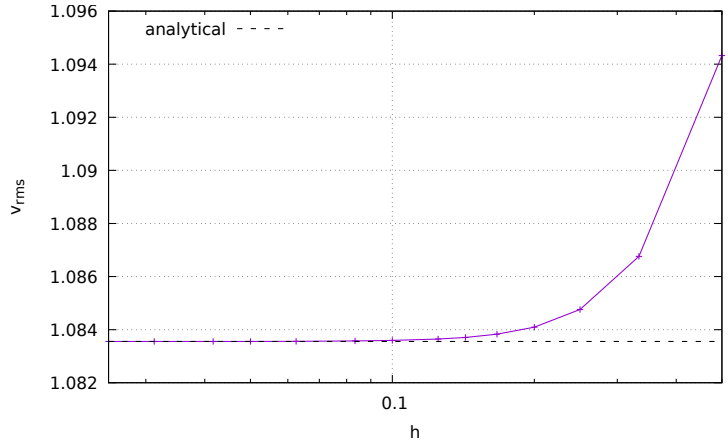


I also computed the error convergence of the strain rate components obtained with the three different methods presented in Section 15.28 and we recover a quadratic convergence for methods 2 and 3 while method 1 shows an exponent of 1.5:

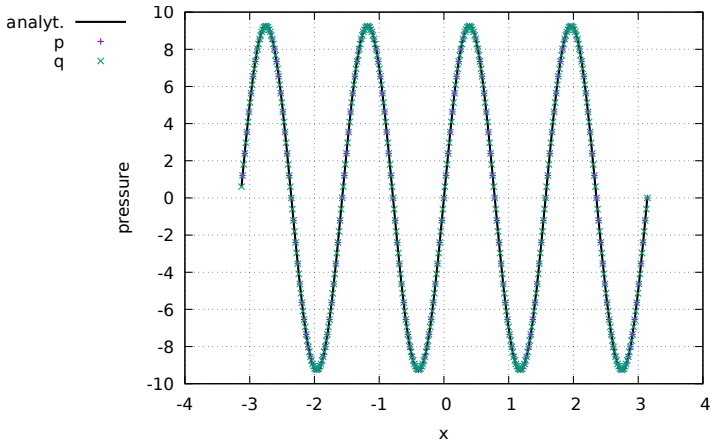


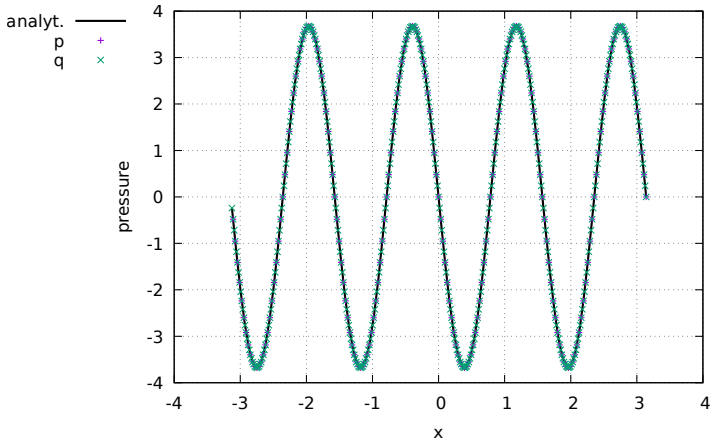


Also the root mean square velocity is logically found to converge to its expected analytical value.

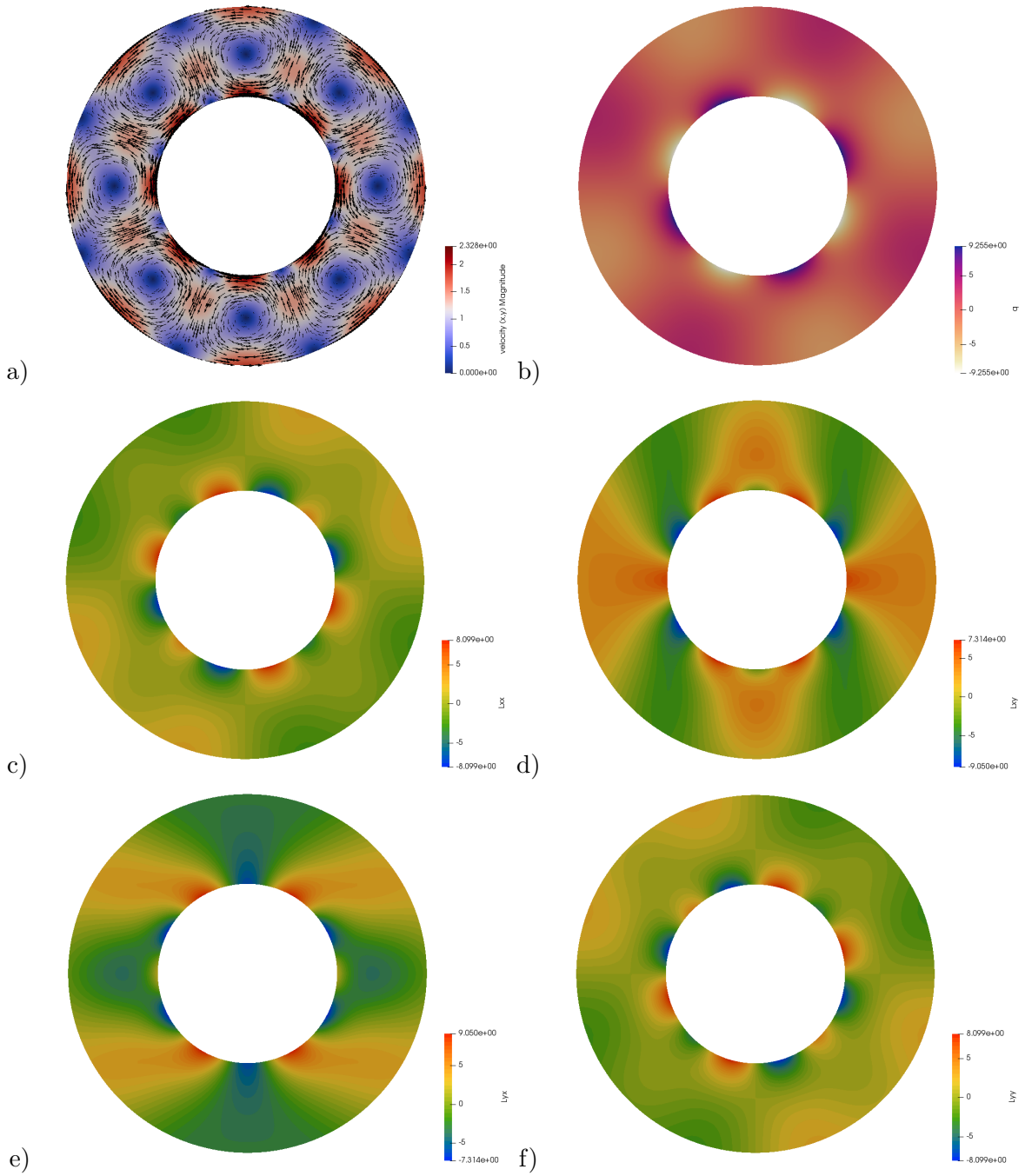


The pressure p and its projection onto the Q_2 grid at $r = R_1$ and $r = R_2$ are plotted here under:





Left: pressure p and q at $r = R_1$; Right: pressure p and q at $r = R_2$.



a) velocity field; b) pressure; c) L_{xx}; d) L_{xy}; e) L_{yx}; f) L_{yy}.

Stone 22

The stabilised $Q_1 \times Q_1$ element



2D
box
Q1Q1stab
sparse_matrix
analytical_benchmark
numerical_benchmark
error_convergence
stokes_equation
viscous_rheology
vrms
paraview_output
pressure_scaling
pressure_normalisation

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_22`

The original fortran version of this stone was co-developped with R. Hassani.

We wish to use $Q_1 \times Q_1$ element, which, unless stabilised, violates the LBB stability condition and therefore is unusable. Stabilisation can be of two types: least-squares [94, 323, 191, 25], or by means of an additional term in the weak form as first introduced in [93, 24], which is appealing since there is no explicit satabilisation parameter. It is further analysed in [242, 215, 160, 285, 140]. Note that an equal-order velocity-pressure formulation that does not exhibit spurious pressure modes (without stabilisaion) has been presented in [261].

This element corresponds to bilinear velocities, bilinear pressure (equal order interpolation for both velocity and pressure) which is very convenient in terms of data structures since all dofs are colocated.

In geodynamics, it is used in the Rhea code [303, 57] and in Gale [5]. It is also used in [210] in its stabilised form, in conjunction with AMR. This element is quickly discussed at page 217 of Volker John's book [171].

The stabilisation term $\mathbb{C}$ enters the Stokes matrix in the (2,2) position:

$$\left(\begin{array}{cc} \mathbb{K} & \mathbb{G} \\ \mathbb{G}^T & -\mathbb{C} \end{array} \right) \cdot \left(\begin{array}{c} \mathcal{V} \\ \mathcal{P} \end{array} \right) = \left(\begin{array}{c} f \\ h \end{array} \right)$$

The purpose of the $\mathbb{C}$ term is to stabilise the linear system. It is given by:

$$\mathbb{C}(p,q) = \sum_e \int_{\Omega_e} \frac{1}{\eta} (p - \Pi p)(q - \Pi q) d\Omega$$

where Π is the L^2-projection onto the space of element-wise constant functions:

$$\Pi p = \frac{1}{|\Omega_e|}\int_{\Omega_e} p d\Omega$$

Because of the stabilisation matrix $\mathbb{C}$, the numerical solution satisfies the incompressibility condition only approximately. Local mesh refinement helps to control these unwanted effects [56, 57]. Since $\mathbb{K}$ and $\mathbb{C}$ are symmetric matrices, the Stokes system is then an indefinite symmetric system. The Schur complement matrix $\mathbb{S}$ is then given by

$$\mathbb{S} = \mathbb{G}^T \cdot \mathbb{K}^{-1} \cdot \mathbb{G} + \mathbb{C}$$

One can further expand the above expression for the $\mathbb{C}$ term:

$$\begin{aligned}
\mathbb{C}(p,q) &= \sum_e \int_{\Omega_e} \frac{1}{\eta}(p-\Pi p)(q-\Pi q)d\Omega \\
&= \sum_e \int_{\Omega_e} \frac{1}{\eta}[pq-(\Pi p)q-(\Pi q)p+(\Pi p)(\Pi q)]d\Omega \\
&= \sum_e \frac{1}{\eta_e}\left[\int_{\Omega_e} pqd\Omega - \int_{\Omega_e}(\Pi p)qd\Omega - \int_{\Omega_e}(\Pi q)pd\Omega + \int_{\Omega_e}(\Pi p)(\Pi q)d\Omega\right] \\
&= \sum_e \frac{1}{\eta_e}\left[\int_{\Omega_e} pqd\Omega - (\Pi p)\int_{\Omega_e} qd\Omega - (\Pi q)\int_{\Omega_e} pd\Omega + (\Pi p)(\Pi q)\int_{\Omega_e} d\Omega\right] \\
&= \sum_e \frac{1}{\eta_e}\left[\int_{\Omega_e} pqd\Omega - (\Pi p)|\Omega_e|(\Pi q) - (\Pi q)|\Omega_e|(\Pi p) + (\Pi p)(\Pi q)|\Omega_e|\right] \\
&= \sum_e \frac{1}{\eta_e}\left[\int_{\Omega_e} pqd\Omega - |\Omega_e|(\Pi p)(\Pi q)\right] \qquad (22.1)
\end{aligned}$$

where we have used the fact that on each element Πp^h is constant. The left term will obviously yield a Q_1 mass matrix (scaled by the elemental viscosities). Note that this approach is not used in practice as we'll see hereafter.

The pressure inside an element is given by

$$p^h(\vec{x}) = \sum_k \mathcal{N}_k^p(\vec{x})p_k$$

so that

$$\Pi p^h = \frac{1}{|\Omega_e|}\int_{\Omega_e}\sum_k \mathcal{N}_k^p p_k d\Omega = \sum_k \left(\underbrace{\frac{1}{|\Omega_e|}\int_{\Omega_e}\mathcal{N}_k^p d\Omega}_{\tilde{\mathcal{N}}_k^p}\right)p_k \qquad (22.2)$$

and then

$$p^h - \Pi p^h = \sum_k \mathcal{N}_k^p(\vec{x})p_k - \sum_k \tilde{\mathcal{N}}_k^p p_k = \sum_k (\mathcal{N}_k^p(\vec{x}) - \tilde{\mathcal{N}}_k^p)p_k$$

The algorithm is straighforward and as follows: In the loop over elements, a) Compute the average of each shape function $N_k^p(\vec{x})$ over the element; b) Substract this average to the shape function; c) Build mass matrix with modified/offset shape functions (taking in account the viscosity).

In the case of rectangular elements of size (h_x, h_y), $\tilde{\mathcal{N}}_k^p$ simplifies even more:

$$\tilde{\mathcal{N}}_k^p = \frac{1}{|\Omega_e|}\int_{\Omega_e}\mathcal{N}_k^p(\vec{x})d\Omega = \frac{1}{h_x h_y}\frac{h_x h_y}{4}\int_{-1}^{+1}\int_{-1}^{+1}\mathcal{N}_k^p(r,s)drds = \frac{1}{4}\int_{-1}^{+1}\int_{-1}^{+1}\mathcal{N}_k^p(r,s)drds \qquad (22.3)$$

It is easy to show that the average of the Q_1 shape functions of over the reference element is 1, so that $\tilde{\mathcal{N}}_k^p = 1/4$. This explains why in the code we have:

```
Navrg = np.zeros(m,dtype=np.float64)
Navrg[0]=0.25
Navrg[1]=0.25
Navrg[2]=0.25
Navrg[3]=0.25
```

This also means that $\Pi p^h = (p_1 + p_2 + p_3 + p_4)/4$, i.e. the projected pressure is the mean of the vertex values. It follows, as shown on p.244 of [105] that the elemental $\mathbb{C}$ matrix is (omitting the viscosity term)

$$\mathbb{C}_{el} = \mathbb{M}_{el} - \vec{q}^T \vec{q} |\Omega_e| \qquad \vec{q} = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$$

The nullspace of $\mathbb{C}$ consists of constant vectors, i.e. $\vec{1} \in \text{null}(\mathbb{C})$ which means that the assembled stabilisation operator is consistent.

The elemental $\mathbb{C}_{el}$ matrix is then computed like a mass matrix, although with modified shape function vectors. Inside the loop over quadrature points, we do:

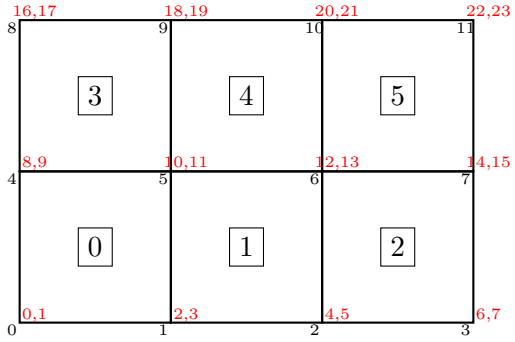
```
Nvect[0,0:m]=N[0:m]-Navrg[0:m]
C_el+=Nvect.T.dot(Nvect)*jcob*weightq/viscosity(xq,yq,case)
```

It is then assembled inside the big FEM matrix

```
for k1 in range(0,m):
    for k2 in range(0,m):
        C_mat[icon[k1,iel],icon[k2,iel]]+=C_el[k1,k2]
```

Non-zero pattern of the $\mathbb{G}$ matrix

Let us take a simple example: a 3x2 element grid.



The $\mathbb{K}$ matrix is of size $NfemV \times NfemV$ with $NfemV = ndofV \times nnp = 2 \times 12 = 24$. The $\mathbb{G}$ matrix is of size $NfemV \times NfemP$ with $NfemP = ndofP \times nnp = 1 \times 12 = 12$. The $\mathbb{C}$ matrix is of size $NfemP \times NfemP$.

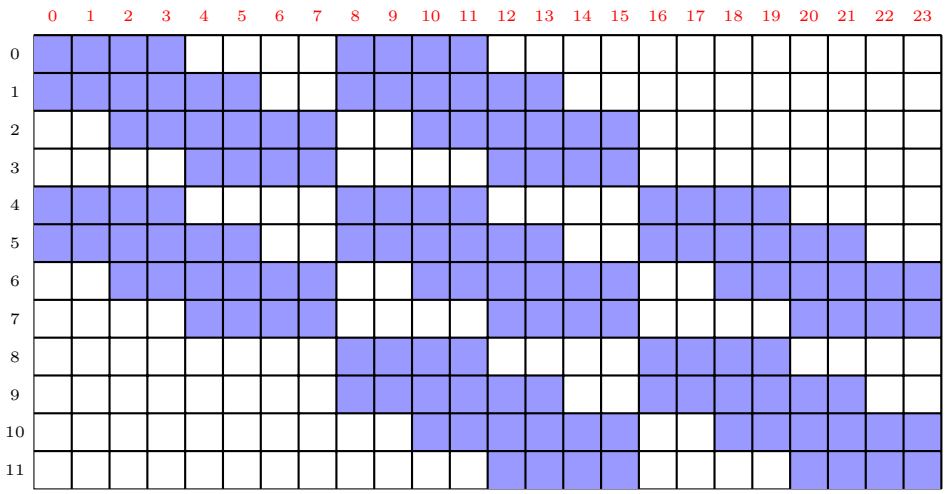
A corner pdof sees 4 vdofs, a side pdof sees 12 vdofs and an inside pdof sees 18 vdofs, so that the total number of nonzeros in $\mathbb{G}$ can be computed as follows:

$$NZ_{\mathbb{G}} = \underbrace{4}_{corners} + \underbrace{2(nnx-2)*12}_{2hor.sides} + \underbrace{2(nny-2)*12}_{2vert.sides} + \underbrace{(nnx-2)(nny-2)*18}_{insidenodes}$$

Concretely,

- pdof #0 sees vdofs 0,1,2,3,8,9,10,11
- pdof #1 sees vdofs 0,1,2,3,4,5,8,9,10,11,12,13
- pdof #5 sees vdofs 0,1,2,3,4,5,8,9,10,11,12,13,16,17,18,19,20,21

so that the $\mathbb{G}^T$ matrix non-zero structure then is as follows:

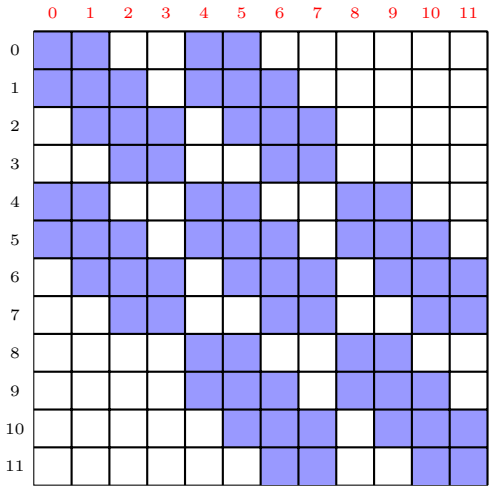


Non-zero pattern of the $\mathbb{C}$ matrix

Concretely,

- pdof #0 sees vdofs 0,1,4,5
- pdof #1 sees vdofs 0,1,2,4,5,6
- pdof #5 sees vdofs 0,1,2,4,5,6,8,9,10

so that the $\mathbb{C}$ matrix non-zero structure is as follows:



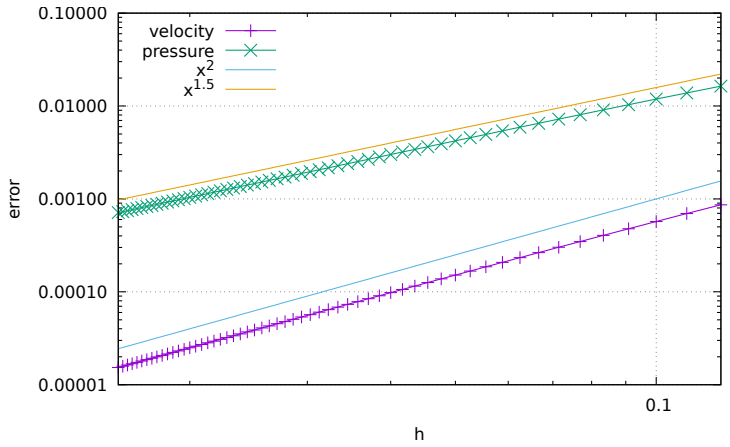
Constraining the pressure field to zero average

We impose $\int pdV = 0$ which means that the following constraint is added to the Stokes matrix:

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} & 0 \\ \mathbb{G}^T & \mathbb{C} & \mathbb{L} \\ 0 & \mathbb{L}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \mathcal{P} \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ h \\ 0 \end{pmatrix}$$

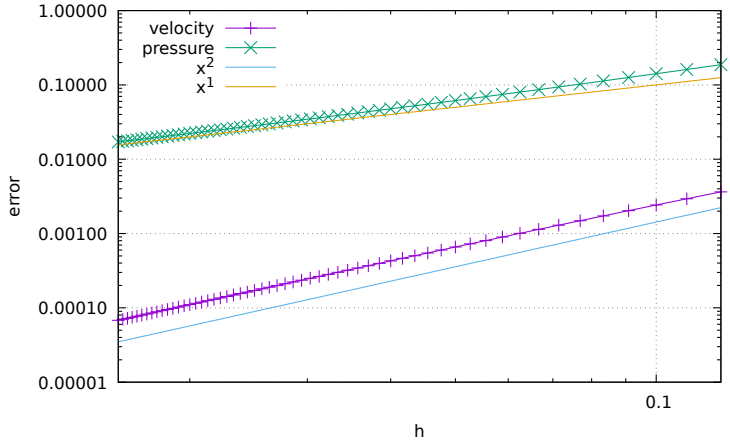
The Donea & Huerta benchmark (case 1)

As in [94] we solve the benchmark problem presented in section **??**.



The Dohrmann & Bochev benchmark (case 2)

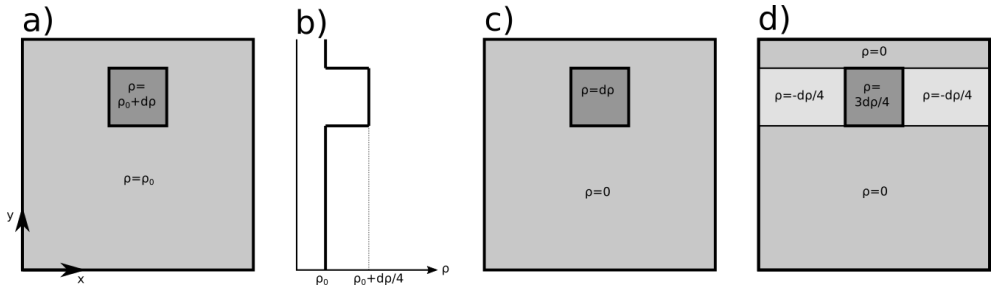
As in [93] we solve the benchmark problem presented in section **??**.

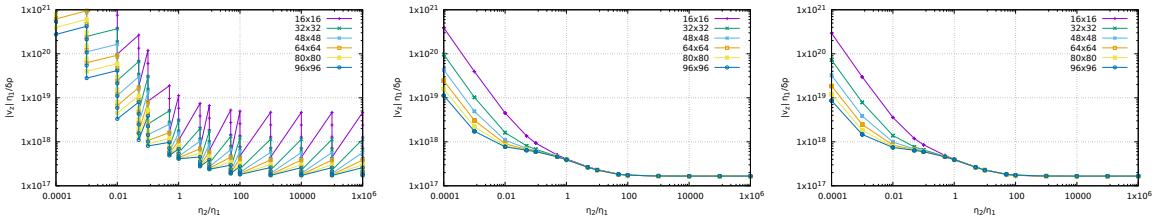


compare my rates with original paper!

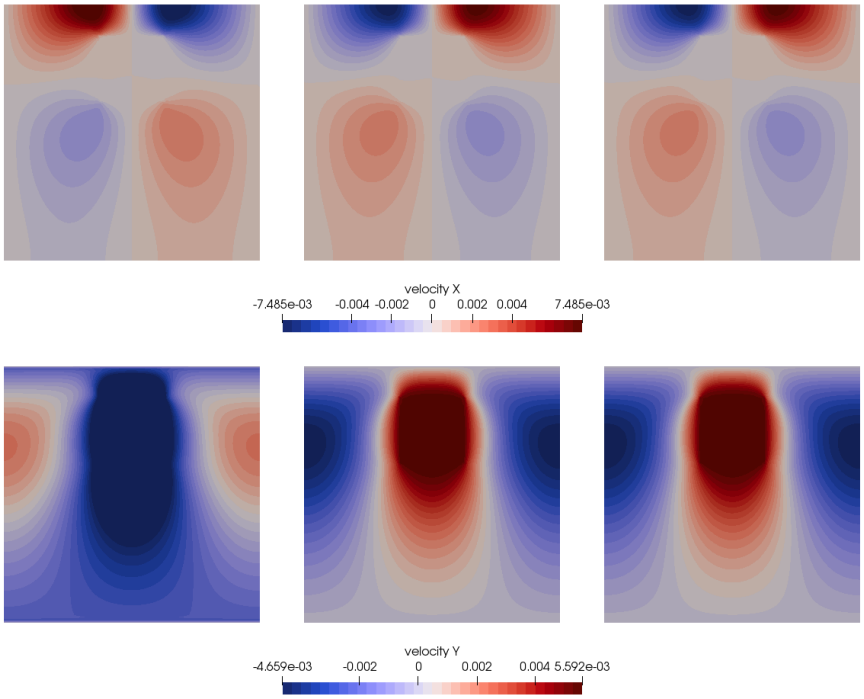
The sinking block experiment (case 3)

The setup is described in [328]. It consists of a two-dimensional 512×512km domain filled with a fluid (the "mantle") of density $\rho_1 = 3200\text{kg/m}^3$ and viscosity η_1. A square block of size 128×128km is placed in the domain and is centered at location (x_c, y_c)=(256km,384km) so as to insure that its sides align with cell boundaries at all resolutions. It is filled with a fluid of density $\rho_2 = \rho_1 + \delta\rho$ and viscosity η_2. The gravity vector points downwards with $|\vec{g}| = 10\text{m/s}^2$. Boundary conditions are free slip on all sides. Only one time step is carried out and we measure the velocity $|v_z|$ in the middle of the block.





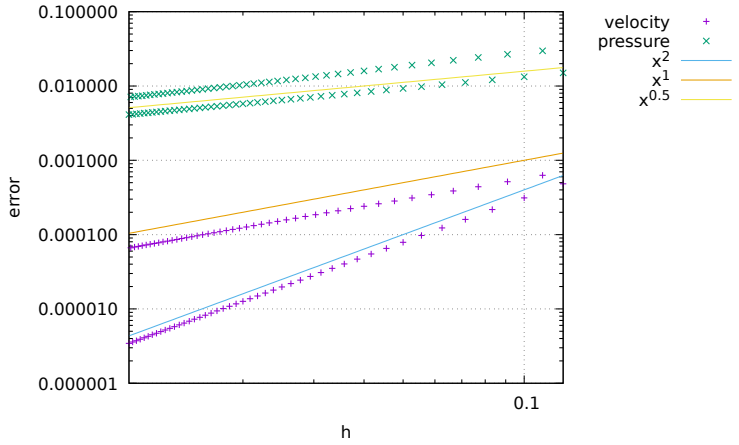
From left to right: subcase=1,2,3.



From left to right: subcase=1,2,3. Resolution 96x96, $\delta\rho = 32$, $\eta_2 = 10^{23}$

SolCx (case 4)

This benchmark is described in Section **??**.

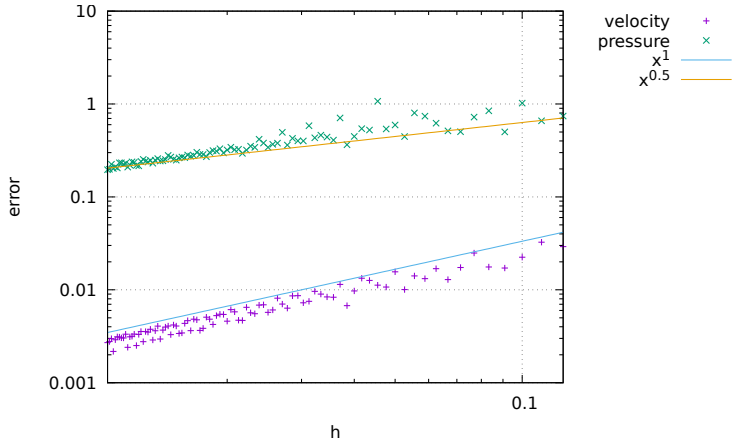


Resolutions 8x8 to 96x96. Even resolutions converge $\mathcal{O}(h^2)$ for velocity while even resolutions converge $\mathcal{O}(h^1)$. In both cases the

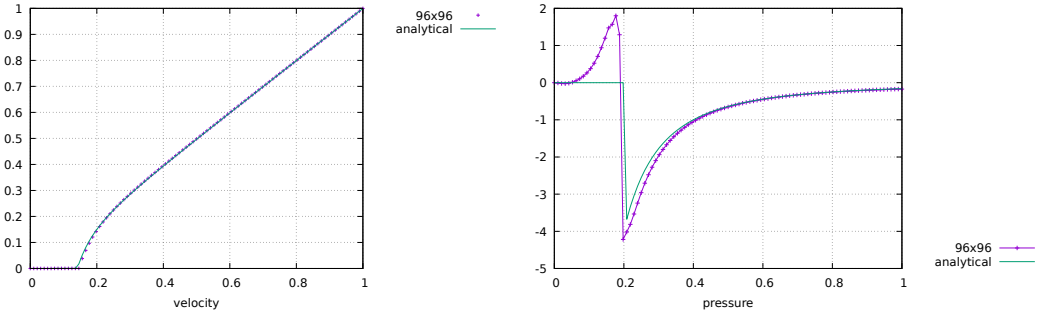
pressure converges $\mathcal{O}(h^{1/2})$.

SolVi (case 5)

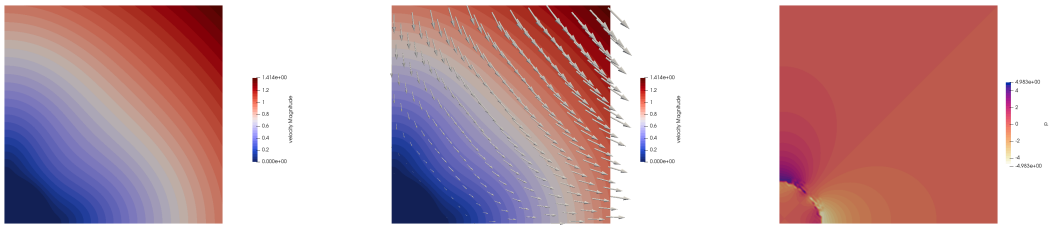
This benchmark is described in Section **??**.



Resolutions 8x8 to 96x96.



Velocity and pressure on the diagonal $y = x$, at resolution 96x96.



Stone 23

compressible flow (1) - analytical benchmark ($Q_1 \times P_0$)



2D
box
Q1P0
compressible
analytical_benchmark
error_convergence
stokes_equation
matplotlib
pressure_normalisation

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_23

This work is part of the MSc thesis of T. Weir (2018).

We first start with an isothermal Stokes flow, so that we disregard the heat transport equation and the equations we wish to solve are simply:

$$-\nabla \cdot \left[2\eta \left(\dot{\varepsilon}(\boldsymbol{v}) - \frac{1}{3}(\nabla \cdot \boldsymbol{v})\mathbf{1} \right) \right] + \nabla p = \rho \boldsymbol{g} \qquad \text{in } \Omega, \tag{23.1}$$

$$\nabla \cdot (\rho \boldsymbol{v}) = 0 \qquad \text{in } \Omega \tag{23.2}$$

The second equation can be rewritten $\nabla \cdot (\rho \boldsymbol{v}) = \rho \nabla \cdot \boldsymbol{v} + \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho = 0$ or,

$$\nabla \cdot \boldsymbol{v} + \frac{1}{\rho} \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho = 0$$

Note that this presupposes that the density is not zero anywhere in the domain.

We use a mixed formulation and therefore keep both velocity and pressure as unknowns. We end up having to solve the following system:

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} \\ \mathbb{G}^T + \mathbb{Z} & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \mathcal{P} \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix} \qquad \text{or,} \qquad \mathbb{A} \cdot X = rhs$$

Where $\mathbb{K}$ is the stiffness matrix, $\mathbb{G}$ is the discrete gradient operator, $\mathbb{G}^T$ is the discrete divergence operator, $\mathcal{V}$ the velocity vector, $\mathcal{P}$ the pressure vector. Note that the term $\mathbb{Z}\mathcal{V}$ derives from term $\boldsymbol{v} \cdot \boldsymbol{\nabla} \rho$ in the continuity equation.

Each block $\mathbb{K}$, $\mathbb{G}$, $\mathbb{Z}$ and vectors f and h are built separately in the code and assembled into the matrix $\mathbb{A}$ and vector rhs afterwards. $\mathbb{A}$ and rhs are then passed to the solver. We will see later that there are alternatives to solve this approach which do not require to build the full Stokes matrix $\mathbb{A}$.

Remark: the term $\mathbb{Z}\mathcal{V}$ is often put in the rhs (i.e. added to h) so that the matrix $\mathbb{A}$ retains the same structure as in the incompressible case. This is indeed how it is implemented in ASPECT. This however requires more work since the rhs depends on the solution and some form of iterations is needed.

In the case of a compressible flow the strain rate tensor and the deviatoric strain rate tensor are no more equal (since $\boldsymbol{\nabla}\cdot\boldsymbol{v}\neq 0$). The deviatoric strainrate tensor is given by[1]

$$\dot{\boldsymbol{\epsilon}}^d(\boldsymbol{v}) = \dot{\boldsymbol{\epsilon}}(\boldsymbol{v}) - \frac{1}{3}Tr(\dot{\boldsymbol{\epsilon}})\mathbf{1} = \dot{\boldsymbol{\epsilon}}(\boldsymbol{v}) - \frac{1}{3}(\boldsymbol{\nabla}\cdot\boldsymbol{v})\mathbf{1}$$

In that case:

$$\dot{\epsilon}^d_{xx} = \frac{\partial u}{\partial x} - \frac{1}{3}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right) = \frac{2}{3}\frac{\partial u}{\partial x} - \frac{1}{3}\frac{\partial v}{\partial y} \tag{23.3}$$

$$\dot{\epsilon}^d_{yy} = \frac{\partial v}{\partial y} - \frac{1}{3}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right) = -\frac{1}{3}\frac{\partial u}{\partial x} + \frac{2}{3}\frac{\partial v}{\partial y} \tag{23.4}$$

$$2\dot{\epsilon}^d_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{23.5}$$

and then

$$\dot{\boldsymbol{\epsilon}}^d(\boldsymbol{v}) = \begin{pmatrix} \frac{2}{3}\frac{\partial u}{\partial x} - \frac{1}{3}\frac{\partial v}{\partial y} & \frac{1}{2}\frac{\partial u}{\partial y} + \frac{1}{2}\frac{\partial v}{\partial x} \\ \\ \frac{1}{2}\frac{\partial u}{\partial y} + \frac{1}{2}\frac{\partial v}{\partial x} & -\frac{1}{3}\frac{\partial u}{\partial x} + \frac{2}{3}\frac{\partial v}{\partial y} \end{pmatrix}$$

From $\vec{\tau} = 2\eta\vec{\epsilon}^d$ we arrive at:

$$\begin{pmatrix} \tau_{xx} \\ \tau_{yy} \\ \tau_{xy} \end{pmatrix} = 2\eta \begin{pmatrix} \dot{\epsilon}^d_{xx} \\ \dot{\epsilon}^d_{yy} \\ \dot{\epsilon}^d_{xy} \end{pmatrix} = 2\eta \begin{pmatrix} 2/3 & -1/3 & 0 \\ -1/3 & 2/3 & 0 \\ 0 & 0 & 1/2 \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} \end{pmatrix} = \eta \begin{pmatrix} 4/3 & -2/3 & 0 \\ -2/3 & 4/3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} \end{pmatrix}$$

or,

$$\vec{\tau} = \boldsymbol{C}_\eta \boldsymbol{B} V$$

[1] See the ASPECT manual for a justification of the 3 value in the denominator in 2D and 3D.

In order to test our implementation we have created a few manufactured solutions:

- benchmark #1 (`ibench=1`)): Starting from a density profile of:

$$\rho(x,y) = xy \tag{23.6}$$

We derive a velocity given by:

$$v_x(x,y) = \frac{C_x}{x}, v_y(x,y) = \frac{C_y}{y} \tag{23.7}$$

With $g_x(x,y) = \frac{1}{x}$ and $g_y(x,y) = \frac{1}{y}$, this leads us to a pressure profile:

$$p = -\eta \left(\frac{4C_x}{3x^2} + \frac{4C_y}{3y^2} \right) + xy + C_0 \tag{23.8}$$

This gives us a strain rate:

$$\dot{\epsilon}_{xx} = \frac{-C_x}{x^2} \qquad \dot{\epsilon}_{yy} = \frac{-C_y}{y^2} \qquad \dot{\epsilon}_{xy} = 0$$

In what follows, we choose $\eta = 1$ and $C_x = C_y = 1$ and for a unit square domain $[1:2] \times [1:2]$ we compute C_0 so that the pressure is normalised to zero over the whole domain and obtain $C_0 = -1$.

- benchmark #2 (`ibench=2`): Starting from a density profile of:

$$\rho = \cos(x)\cos(y) \tag{23.9}$$

We derive a velocity given by:

$$v_x = \frac{C_x}{\cos(x)}, v_y = \frac{C_y}{\cos(y)} \tag{23.10}$$

With $g_x = \frac{1}{\cos(y)}$ and $g_y = \frac{1}{\cos(x)}$, this leads us to a pressure profile:

$$p = \eta \left(\frac{4C_x \sin(x)}{3\cos^2(x)} + \frac{4C_y \sin(y)}{3\cos^2(y)} \right) + (\sin(x) + \sin(y)) + C_0 \tag{23.11}$$

$$\dot{\epsilon}_{xx} = C_x \frac{\sin(x)}{\cos^2(x)} \qquad \dot{\epsilon}_{yy} = C_y \frac{\sin(y)}{\cos^2(y)} \qquad \dot{\epsilon}_{xy} = 0$$

We choose $\eta = 1$ and $C_x = C_y = 1$. The domain is the unit square $[0:1] \times [0:1]$ and we obtain C_0 as before and obtain

$$C_0 = 2 - 2\cos(1) + 8/3(\frac{1}{\cos(1)} - 1) \simeq 3.18823730$$

(thank you WolframAlpha)

- benchmark #3 (`ibench=3`)
- benchmark #4 (`ibench=4`)
- benchmark #5 (`ibench=5`)

ToDo:

- pbs with odd vs even number of elements
- q is 'fine' everywhere except in the corners - revisit pressure smoothing paper?
- redo A v d Berg benchmark (see Tom Weir thesis)

Stone 24

compressible flow (2) - convection box ($Q_1 \times P_0$)



2D
box
Q1P0
compressible
numerical_benchmark
stokes_equation
energy_equation
thermo_mechanical
time_stepping
viscous_rheology
buoyancy_driven
vrms
nusselt_number
rayleigh_number
paraview_output
steady_state

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_24`

This work is part of the MSc thesis of T. Weir (2018).

The physics

Let us start with some thermodynamics. Every material has an equation of state. The equilibrium thermodynamic state of any material can be constrained if any two state variables are specified. Examples of state variables include the pressure p and specific volume $\nu = 1/\rho$, as well as the temperature T.

After linearisation, the density depends on temperature and pressure as follows:

$$\rho(T,p) = \rho_0 \left((1 - \alpha(T - T_0) + \beta_T p \right)$$

where α is the coefficient of thermal expansion, also called thermal expansivity:

$$\alpha = -\frac{1}{\rho} \left(\frac{\partial \rho}{\partial T} \right)_p$$

α is the percentage increase in volume of a material per degree of temperature increase; the subscript p means that the pressure is held fixed.

β_T is the isothermal compressibility of the fluid, which is given by

$$\beta_T = \frac{1}{K} = \frac{1}{\rho}\left(\frac{\partial \rho}{\partial P}\right)_T$$

with K the bulk modulus. Values of $\beta_T = 10^{-12} - 10^{-11}$ Pa^{-1} are reasonable for Earth's mantle, with values decreasing by about a factor of 5 between the shallow lithosphere and core-mantle boundary. This is the percentage increase in density per unit change in pressure at constant temperature. Both the coefficient of thermal expansion and the isothermal compressibility can be obtained from the equation of state.

The full set of equations we wish to solve is given by

$$-\nabla \cdot \left[2\eta \dot{\boldsymbol{\epsilon}}^d(\boldsymbol{v})\right] + \nabla p = \rho_0 \left((1 - \alpha(T - T_0) + \beta_T p)\, \boldsymbol{g}\right) \qquad \text{in } \Omega \tag{24.1}$$

$$\nabla \cdot \boldsymbol{v} + \frac{1}{\rho} \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho = 0 \qquad \text{in } \Omega \tag{24.2}$$

$$\rho C_p \left(\frac{\partial T}{\partial t} + \boldsymbol{v} \cdot \nabla T\right) - \nabla \cdot k \nabla T = \rho H + 2\eta \dot{\boldsymbol{\epsilon}}^d : \dot{\boldsymbol{\epsilon}}^d + \alpha T \left(\frac{\partial p}{\partial t} + \boldsymbol{v} \cdot \nabla p\right) \qquad \text{in } \Omega, \tag{24.3}$$

Note that this presupposes that the density is not zero anywhere in the domain.

The numerics

We use a mixed formulation and therefore keep both velocity and pressure as unknowns. We end up having to solve the following system:

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} + \mathbb{W} \\ \mathbb{G}^T + \mathbb{Z} & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \mathcal{P} \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix} \qquad \text{or,} \qquad \mathbb{A} \cdot X = rhs$$

Where $\mathbb{K}$ is the stiffness matrix, $\mathbb{G}$ is the discrete gradient operator, $\mathbb{G}^T$ is the discrete divergence operator, $\mathcal{V}$ the velocity vector, $\mathcal{P}$ the pressure vector. Note that the term $\mathbb{Z}\mathcal{V}$ derives from term $\boldsymbol{v} \cdot \boldsymbol{\nabla}\rho$ in the continuity equation.

As perfectly explained in the step 32 of deal.ii[1], we need to scale the $\mathbb{G}$ term since it is many orders of magnitude smaller than $\mathbb{K}$, which introduces large inaccuracies in the solving process to the point that the solution is nonsensical. This scaling coefficient is η/L. After building the $\mathbb{G}$ block, it is then scaled as follows: $\mathbb{G}' = \frac{\eta}{L}\mathbb{G}$ so that we now solve

$$\begin{pmatrix} \mathbb{K} & \mathbb{G}' + \mathbb{W} \\ \mathbb{G}'^T + \mathbb{Z} & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathcal{V} \\ \mathcal{P}' \end{pmatrix} = \begin{pmatrix} f \\ h \end{pmatrix}$$

After the solve phase, we recover the real pressure with $\mathcal{P} = \frac{\eta}{L}\mathcal{P}'$.

adapt notes since I should scale $\mathbb{W}$ and $\mathbb{Z}$ too. h should be caled too !!!!!!!!!!!!!!

Each block $\mathbb{K}$, $\mathbb{G}$, $\mathbb{Z}$ and vectors f and h are built separately in the code and assembled into the matrix $\mathbb{A}$ and vector rhs afterwards. $\mathbb{A}$ and rhs are then passed to the solver. We will see later that there are alternatives to solve this approach which do not require to build the full Stokes matrix $\mathbb{A}$.

Remark 1: the terms $\mathbb{Z}\mathcal{V}$ and $\mathbb{W}\mathcal{P}$ are often put in the rhs (i.e. added to h) so that the matrix $\mathbb{A}$ retains the same structure as in the incompressible case. This is indeed how it is implemented in ASPECT, see also appendix A of Leng and Zhong [211] (2008). This however requires more work since the rhs depends on the solution and some form of iterations is needed.

Remark 2: Very often the adiabatic heating term $\alpha T\,(\boldsymbol{v} \cdot \nabla p)$ is simplified as follows: If you assume the vertical component of the gradient of the dynamic pressure to be small compared to the gradient of the total pressure (in other words, the gradient is dominated by the gradient of the hydrostatic pressure), then $-\rho \boldsymbol{g} \simeq \boldsymbol{\nabla} p$ and then $\alpha T\,(\boldsymbol{v} \cdot \nabla p) \simeq -\alpha \rho T \boldsymbol{v} \cdot \boldsymbol{g}$. We will however not be using this approximation in what follows.

[1] https://www.dealii.org/9.0.0/doxygen/deal.II/step_32.html

We have already established that

$$\vec{\tau} = \boldsymbol{C}_\eta \boldsymbol{B} V$$

The following measurements are carried out:

- The root mean square velocity (`vrms`):

$$v_{rms} = \sqrt{\frac{1}{V}\int_V v^2 dV}$$

- The average temperature (`Tavrg`):

$$< T >= \frac{1}{V}\int_V T dV$$

- The total mass (`mass`):

$$M = \int_V \rho dV$$

- The Nusselt number (`Nu`):

$$Nu = -\frac{1}{Lx}\frac{1}{\Delta T}\int_0^{L_x} \frac{\partial T(x, y = L_y)}{\partial y} dx$$

- The kinetic energy (`EK`):

$$E_K = \int_V \frac{1}{2}\rho v^2 dV$$

- The work done against gravity

$$< W >= -\int_V \rho g_y v_y dV$$

- The total viscous dissipation (`visc_diss`)

$$< \Phi >= \int \Phi dV = \frac{1}{V}\int 2\eta\dot{\boldsymbol{\epsilon}} : \dot{\boldsymbol{\epsilon}} dV$$

- The gravitational potential energy (`EG`)

$$E_G = \int_V \rho g_y (L_y - y) dV$$

- The internal thermal energy (`ET`)

$$E_T = \int_V \rho_{(0)} C_p T dV$$

Remark 3: Measuring the total mass can be misleading: indeed because $\rho = \rho_0(1 - \alpha T)$, then measuring the total mass amounts to measuring a constant minus the volume-integrated temperature, and there is no reason why the latter should be zero, so that there is no reason why the total mass should be zero...!

The experimental setup

The setup is as follows: the domain is $Lx = Ly = 3000$km. Free slip boundary conditions are imposed on all four sides. The initial temperature is given by:

$$T(x, y) = \left(\frac{L_y - y}{Ly} - 0.01\cos(\frac{\pi x}{L_x})\sin(\frac{\pi y}{Ly})\right)\Delta T + T_{surf}$$

with $\Delta T = 4000$K, $T_{surf} = T_0 = 273.15$K. The temperature is set to $\Delta T + T_{surf}$ at the bottom and T_{surf} at the top. We also set $k = 3$, $C_p = 1250$, $|g| = 10$, $\rho_0 = 3000$ and we keep the Rayleigh number Ra and dissipation number Di as input parameters:

$$Ra = \frac{\alpha g \Delta T L^3 \rho_0^2 C_p}{\eta k} \qquad Di = \frac{\alpha g L}{C_p}$$

From the second equation we get $\alpha = \frac{DiC_p}{gL}$, which we can insert in the first one:

$$Ra = \frac{DiC_p^2 \Delta T L^2 \rho_0^2}{\eta k} \qquad \text{or,} \qquad \eta = \frac{DiC_p^2 \Delta T L^2 \rho_0^2}{Ra\ k}$$

For instance, for $Ra = 10^4$ and $Di = 0.75$, we obtain $\alpha \simeq 3 \cdot 10^{-5}$ and $\eta \simeq 10^{25}$ which are quite reasonable values.

Scaling

Following [192], we non-dimensionalize the equations using the reference values for density ρ_r, thermal expansivity α_r, temperature contrast ΔT_r (`refTemp`), thermal conductivity k_r, heat capacity C_p, depth of the fluid layer L and viscosity η_r. The non-dimensionalization for velocity, u_r , pressure p_r and time, t_r become

$$u_r = \frac{k_r}{\rho_r C_p L} \quad (\texttt{refvel})$$

$$p_r = \frac{\eta_r k_r}{\rho_r C_p L^2} \quad (\texttt{refpress})$$

$$t_r = \frac{\rho_r C_p L^2}{k_r} \quad (\texttt{reftime})$$

In the case of the setup described hereabove, and when choosing $Ra = 10^4$ and $Di = 0.5$, we get:

```
alphaT 2.083333e-05
eta 8.437500e+24
reftime 1.125000e+19
refvel 2.666667e-13
refPress 7.500000e+05
```

Conservation of energy 1

under BA and EBA approximations

Following [211], we take the dot product of the momentum equation with the velocity $\boldsymbol{v}$ and integrate over the whole volume[2]:

$$\int_V [-\nabla \cdot \boldsymbol{\tau} + \boldsymbol{\nabla} p] \cdot \boldsymbol{v} dV = \int_V \rho \boldsymbol{g} \cdot \boldsymbol{v} dV$$

or,

$$-\int_V (\nabla \cdot \boldsymbol{\tau}) \cdot \boldsymbol{v} dV + \int_V \boldsymbol{\nabla} p \cdot \boldsymbol{v} dV = \int_V \rho \boldsymbol{g} \cdot \boldsymbol{v} dV$$

Let us look at each block separately:

$$-\int_V (\nabla \cdot \boldsymbol{\tau}) \cdot \boldsymbol{v} dV = -\int_S \boldsymbol{\tau} \underbrace{\boldsymbol{v} \cdot \boldsymbol{n}}_{=0\ (b.c.)} dS + \int_V \boldsymbol{\tau} : \boldsymbol{\nabla} \boldsymbol{v} dV = \int_V \boldsymbol{\tau} : \dot{\boldsymbol{\varepsilon}} dV = \int_V \Phi dV$$

which is the volume integral of the shear heating. Then,

$$\int_V \boldsymbol{\nabla} p \cdot \boldsymbol{v} dV = \int_S p \underbrace{\boldsymbol{v} \cdot \boldsymbol{n}}_{=0\ (b.c.)} dS - \int_V \underbrace{\boldsymbol{\nabla} \cdot \boldsymbol{v}}_{=0\ (incomp.)} p dV = 0$$

[2] Check: this is akin to looking at the power, force*velocity, says Arie

which is then zero in the case of an incompressible flow. And finally

$$\int_V \rho \boldsymbol{g} \cdot \boldsymbol{v} dV = W$$

which is the work against gravity.

Conclusion for an *incompressible* fluid: we should have

$$\int_V \Phi dV = \int_V \rho \boldsymbol{g} \cdot \boldsymbol{v} dV \tag{24.4}$$

This formula is hugely problematic: indeed, the term ρ in the rhs is the full density. We know that to the value of ρ_0 corresponds a lithostatic pressure gradient $p_L = \rho_0 g y$. In this case one can write $\rho = \rho_0 + \rho'$ and $p = p_L + p'$ so that we also have

$$\int_V \left[-\nabla \cdot \boldsymbol{\tau} + \boldsymbol{\nabla} p' \right] \cdot \boldsymbol{v} dV = \int_V \rho' \boldsymbol{g} \cdot \boldsymbol{v} dV$$

which will ultimately yield

$$\int_V \Phi dV = \int_V \rho' \boldsymbol{g} \cdot \boldsymbol{v} dV = \int_V (\rho - \rho_0) \boldsymbol{g} \cdot \boldsymbol{v} dV \tag{24.5}$$

Obviously Eqs.(24.4) and (24.5) cannot be true at the same time. The problem comes from the nature of the (E)BA approximation: $\rho = \rho_0$ in the mass conservation equation but it is not constant in the momentum conservation equation, which is of course inconsistent. Since the mass conservation equation is $\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0$ under this approximation then the term $\int_V \boldsymbol{\nabla} p \cdot \boldsymbol{v} dV$ is always zero for any pressure (full pressure p, or overpressure $p - p_L$), hence the paradox. This paradox will be lifted when a consistent set of equations will be used (compressible formulation). On a practical note, Eqs.(24.4) is not verified by the code, while (24.5) is.

In the end:

$$\boxed{\underbrace{\int_V \Phi dV}_{\texttt{visc_diss}} = \underbrace{\int_V (\rho - \rho_0) \boldsymbol{g} \cdot \boldsymbol{v} dV}_{\texttt{work_grav}}} \tag{24.6}$$

under no approximation at all

$$\int_V \boldsymbol{\nabla} p \cdot \boldsymbol{v} dV = \int_S p \underbrace{\boldsymbol{v} \cdot \boldsymbol{n}}_{=0 \ (b.c.)} dS - \int_V \boldsymbol{\nabla} \cdot \boldsymbol{v} \ p dV = 0 \tag{24.7}$$

$$= \int_V \frac{1}{\rho} \boldsymbol{v} \cdot \boldsymbol{\nabla} \rho \ p dV = 0 \tag{24.8}$$

$$\tag{24.9}$$

ToDo:see section 3 of [211] where this is carried out with the Adams-Williamson eos.

Conservation of energy 2

Also, following the Reynold's transport theorem [223],p210, we have for a property A (per unit mass)

$$\frac{d}{dt} \int_V A \rho dV = \int_V \frac{\partial}{\partial t} (A\rho) dV + \int_S A \rho \boldsymbol{v} \cdot \boldsymbol{n} dS$$

Let us apply to this to $A = C_p T$ and compute the time derivative of the internal energy:

$$\frac{d}{dt} \int_V \rho C_p T dV = \int_V \frac{\partial}{\partial t} (\rho C_p T) dV + \int_S A\rho \underbrace{\boldsymbol{v} \cdot \boldsymbol{n}}_{=0 \ (b.c.)} dS = \underbrace{\int_V C_p T \frac{\partial \rho}{\partial t} dV}_{I} + \underbrace{\int_V \rho C_p \frac{\partial T}{\partial t} dV}_{II} \tag{24.10}$$

In order to expand I, the mass conservation equation will be used, while the heat transport equation will be used for II:

$$I = \int_V C_p T \frac{\partial \rho}{\partial t} dV = -\int_V C_p T \boldsymbol{\nabla}\cdot(\rho\boldsymbol{v}) dV = -\int_V C_p T \rho \underbrace{\boldsymbol{v}\cdot\boldsymbol{n}}_{=0\ (b.c.)} dS + \int_V \rho C_p \boldsymbol{\nabla} T \cdot \boldsymbol{v} dV \tag{24.11}$$

$$II = \int_V \rho C_p \frac{\partial T}{\partial t} dV = \int_V \left[-\rho C_p \boldsymbol{v}\cdot\boldsymbol{\nabla} T + \boldsymbol{\nabla}\cdot k\boldsymbol{\nabla} T + \rho H + \Phi + \alpha T \left(\frac{\partial p}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla} p \right) \right] dV \tag{24.12}$$

$$= \int_V \left[-\rho C_p \boldsymbol{v}\cdot\boldsymbol{\nabla} T + \rho H + \Phi + \alpha T \left(\frac{\partial p}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla} p \right) \right] dV + \int_V \boldsymbol{\nabla}\cdot k\boldsymbol{\nabla} T dV \tag{24.13}$$

$$= \int_V \left[-\rho C_p \boldsymbol{v}\cdot\boldsymbol{\nabla} T + \rho H + \Phi + \alpha T \left(\frac{\partial p}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla} p \right) \right] dV + \int_S k\boldsymbol{\nabla} T \cdot \boldsymbol{n} dS \tag{24.14}$$

$$= \int_V \left[-\rho C_p \boldsymbol{v}\cdot\boldsymbol{\nabla} T + \rho H + \Phi + \alpha T \left(\frac{\partial p}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla} p \right) \right] dV - \int_S \boldsymbol{q}\cdot\boldsymbol{n} dS \tag{24.15}$$

Finally:

$$I + II = \frac{d}{dt} \underbrace{\int_V \rho C_p T dV}_{\texttt{ET}} = \int_V \left[\rho H + \Phi + \alpha T \left(\frac{\partial p}{\partial t} + \boldsymbol{v}\cdot\boldsymbol{\nabla} p \right) \right] dV - \int_S \boldsymbol{q}\cdot\boldsymbol{n} dS \tag{24.16}$$

$$= \int_V \rho H dV + \underbrace{\int_V \Phi dV}_{\texttt{visc_diss}} + \underbrace{\int_V \alpha T \frac{\partial p}{\partial t} dV}_{\texttt{extra}} + \underbrace{\int_V \alpha T \boldsymbol{v}\cdot\boldsymbol{\nabla} p dV}_{\texttt{adiab_heating}} - \underbrace{\int_S \boldsymbol{q}\cdot\boldsymbol{n} dS}_{\texttt{heatflux_boundary}} \tag{24.17}$$

This was of course needlessly complicated as the term $\partial\rho/\partial t$ is always taken to be zero, so that $I = 0$ automatically. The mass conservation equation is then simply $\boldsymbol{\nabla}\cdot(\rho\boldsymbol{v}) = 0$. Then it follows that

$$0 = \int_V C_p T \boldsymbol{\nabla}\cdot(\rho\boldsymbol{v}) dV = -\int_V C_p T \rho \underbrace{\boldsymbol{v}\cdot\boldsymbol{n}}_{=0\ (b.c.)} dS + \int_V \rho C_p \boldsymbol{\nabla} T \cdot \boldsymbol{v} dV \tag{24.18}$$

$$= \int_V \rho C_p \boldsymbol{\nabla} T \cdot \boldsymbol{v} dV \tag{24.19}$$

so that the same term in Eq.(24.15) vanishes too, and then Eq.(24.17) is always valid, although one should be careful when computing E_T in the BA and EBA cases as it should use ρ_0 and not ρ.

The problem of the onset of convection

[wiki] In geophysics, the Rayleigh number is of fundamental importance: it indicates the presence and strength of convection within a fluid body such as the Earth's mantle. The mantle is a solid that behaves as a fluid over geological time scales.

The Rayleigh number essentially is an indicator of the type of heat transport mechanism. At low Rayleigh numbers conduction processes dominate over convection ones. At high Rayleigh numbers it is the other way around. There is a so-called critical value of the number with delineates the transition from one regime to the other.

This problem has been studied and approached both theoretically and numerically [335, e.g.] and it was found that the critical Rayleigh number Ra_c is

$$Ra_c = (27/4)\pi^4 \simeq 657.5$$

in setups similar to ours.

VERY BIG PROBLEM

The temperature setup is built as follows: T_{surf} is prescribed at the top, $T_{surf} + \Delta T$ is prescribed at the bottom. The initial temperature profile is linear between these two values. In the case of BA, the actual value of T_{surf} is of no consequence. However, for the EBA the full temperature is present in the adiabatic heating term on the rhs of the hte, and the value of T_{surf} will therefore influence the solution greatly. This is very problematic as there is no real way to arrive at the surface temperature from the King paper. On top of this, the density uses a reference temperature T_0 which too will influence the solution without being present in the controlling Ra and Di numbers!!

In light thereof, it will be very difficult to recover the values of King et al for EBA!

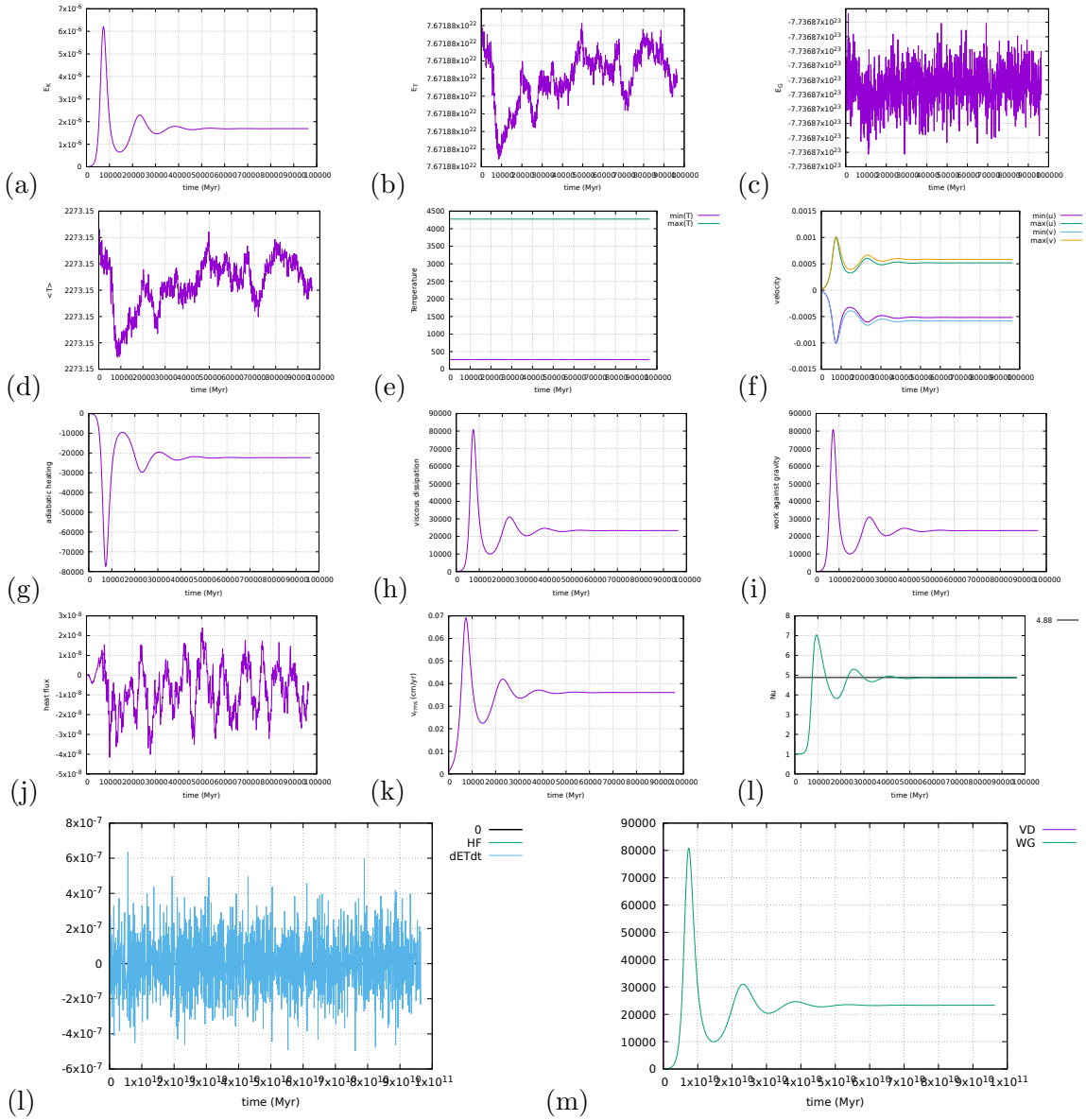
Relevant Literature: Bercovici, Schubert, and Glatzmaier [17], Ita and King [167], Tan and Gurnis [318], Leng and Zhong [211], King et al. [192], Leng and Zhong [210], Liu and Zhong [219], Ciskova, van den Berg, and Jacobs [74], Heister, Dannberg, Gassmöller, and Bangerth [151].

ToDo:

- heat flux is at the moment elemental, so Nusselt and heat flux on boundaries measurements not as accurate as could be.
- implement steady state detection
- do $Ra = 10^5$ and $Ra = 10^6$
- velocity average at surface
- non dimensional heat flux at corners [23]
- depth-dependent viscosity (case 2 of [23])

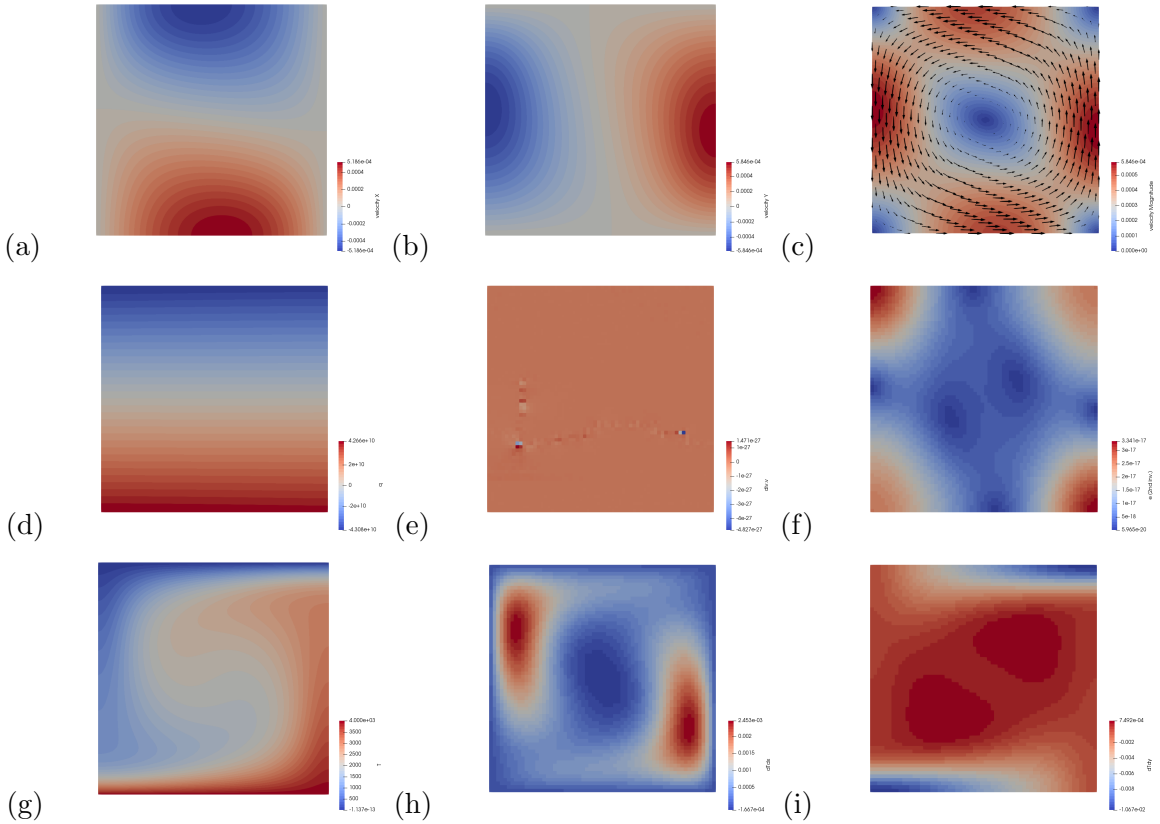
results - BA - $Ra = 10^4$

These results were obtained with a 64x64 resolution, and CFL number of 1. Steady state was reached after about 1250 timesteps.



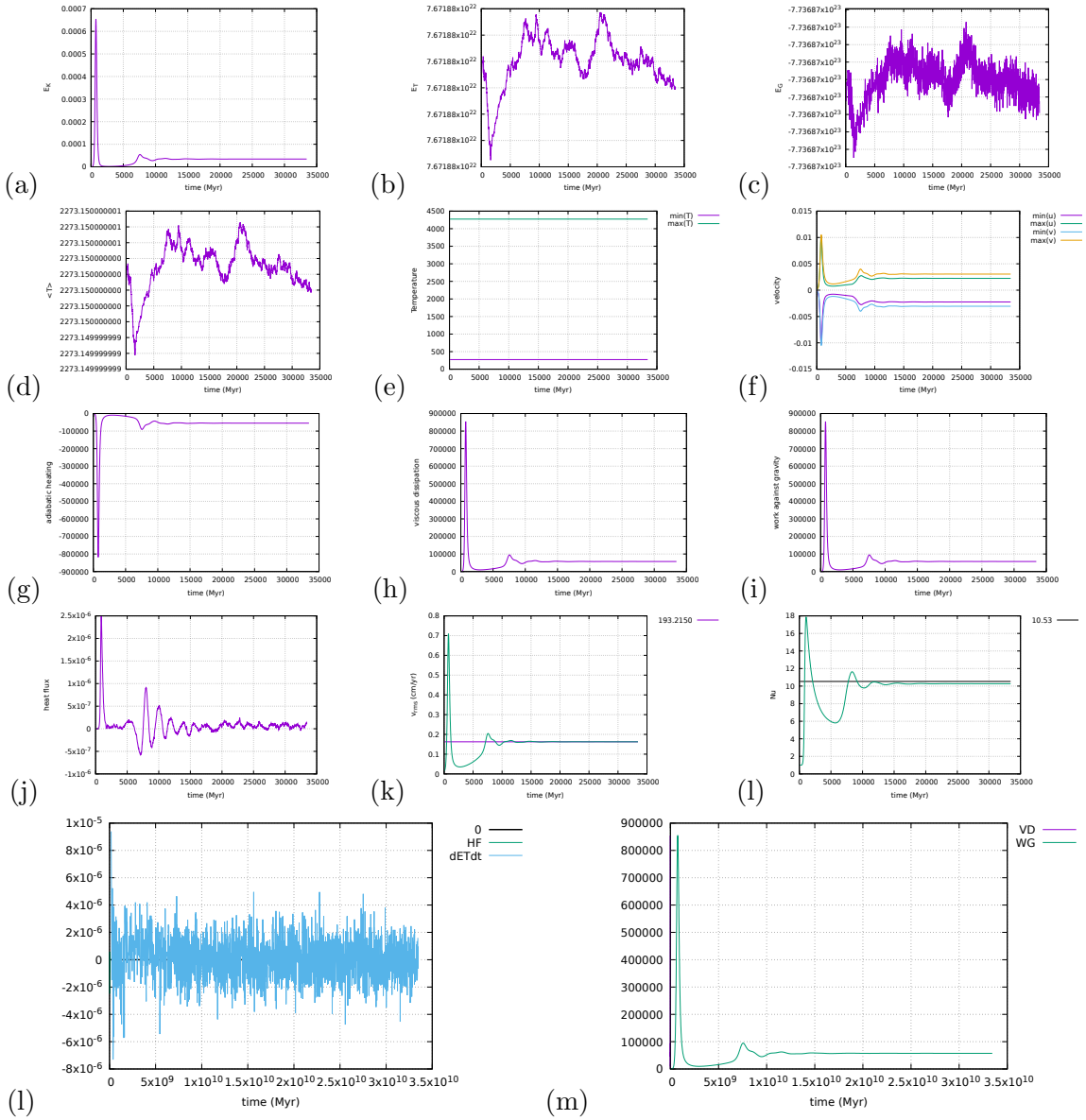
AH: adiabatic heating, VD: viscous dissipation, HF: heat flux, WG: work against gravity

Eq.(24.17) is verified by (l) and Eq.(24.6) is verified by (m).


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)

results - BA - $Ra = 10^5$

These results were obtained with a 64x64 resolution, and CFL number of 1. Steady state was reached after about 1250 timesteps.



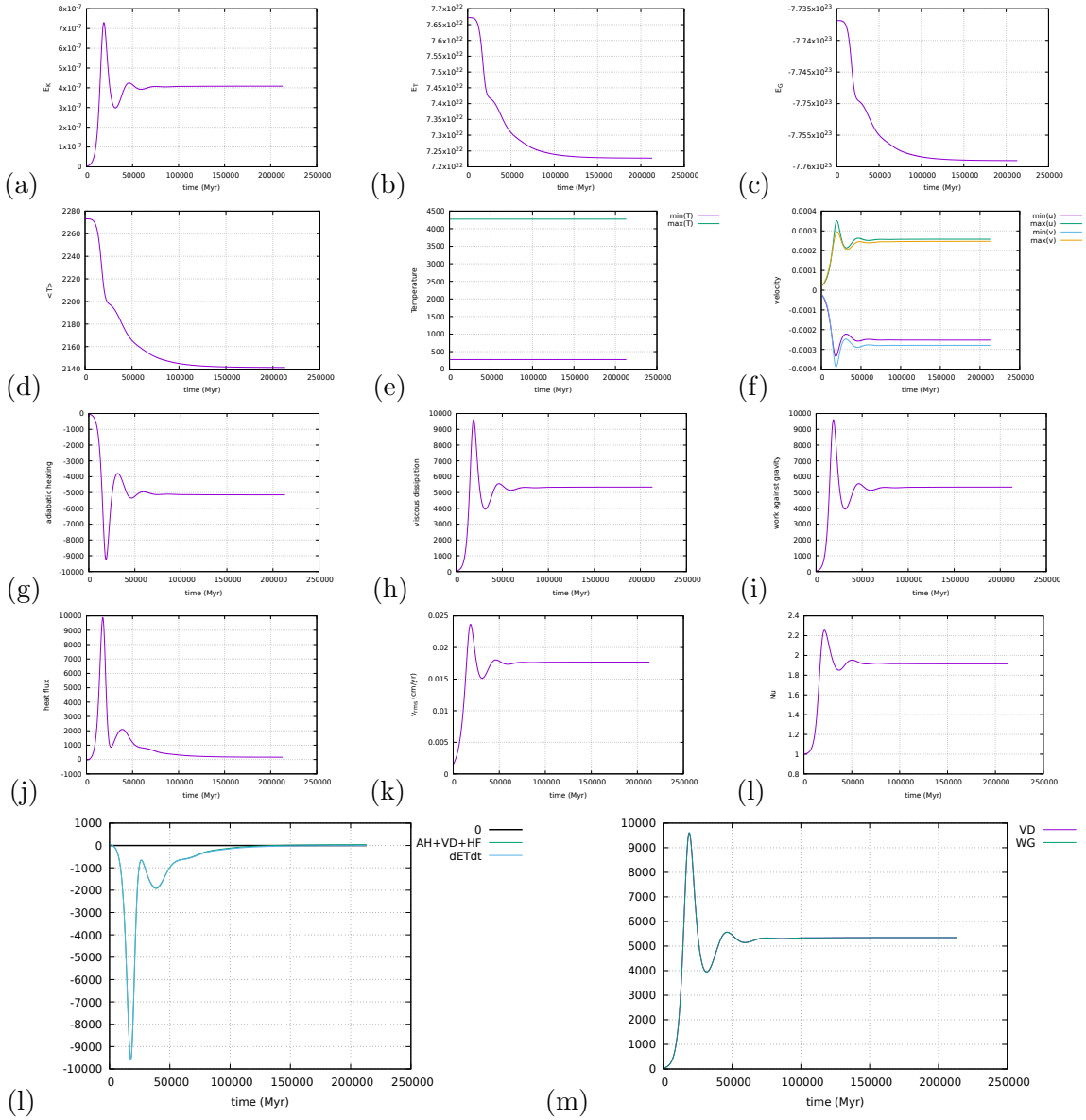
AH: adiabatic heating, VD: viscous dissipation, HF: heat flux, WG: work against gravity

Eq.(24.17) is verified by (l) and Eq.(24.6) is verified by (m).

results - BA - $Ra = 10^6$

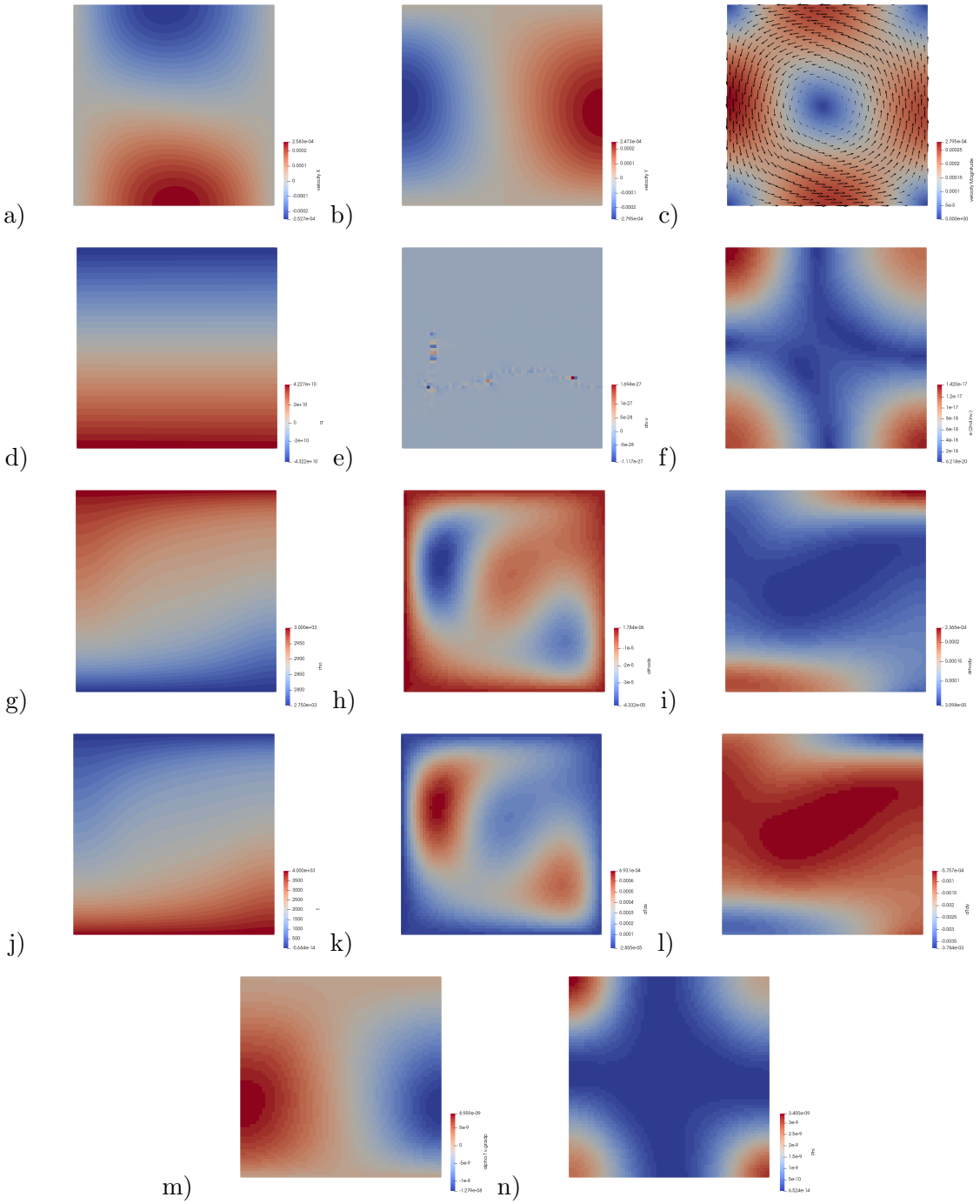
results - EBA - $Ra = 10^4$

These results were obtained with a 64x64 resolution, and CFL number of 1. Steady state was reached after about 2500 timesteps



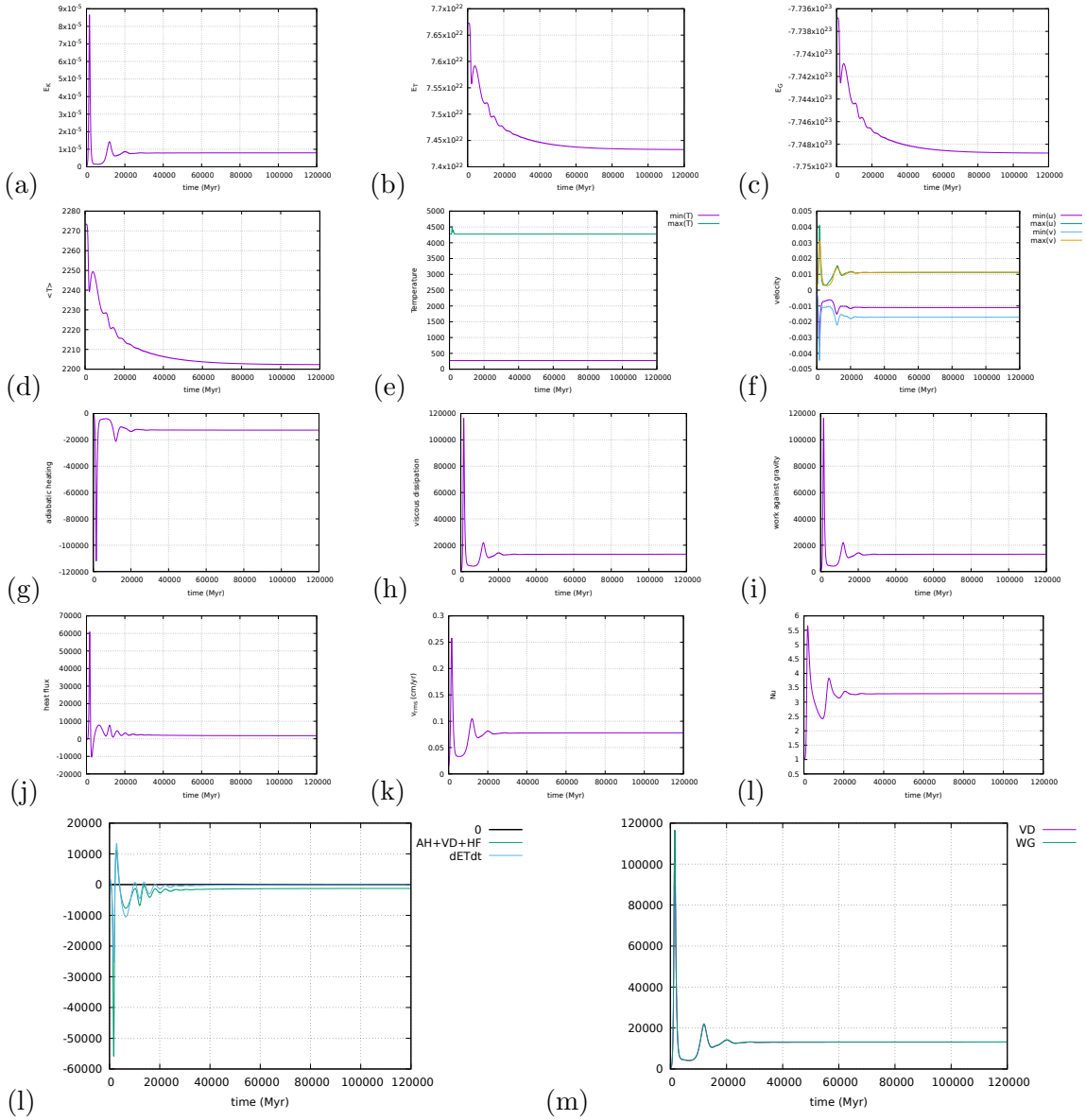
AH: adiabatic heating, VD: viscous dissipation, HF: heat flux, WG: work against gravity

Eq.(24.17) is verified by (l) and Eq.(24.6) is verified by (m).


a)
b)
c)
d)
e)
f)
g)
h)
i)
j)
k)
l)
m)
n)

results - EBA - $Ra = 10^5$

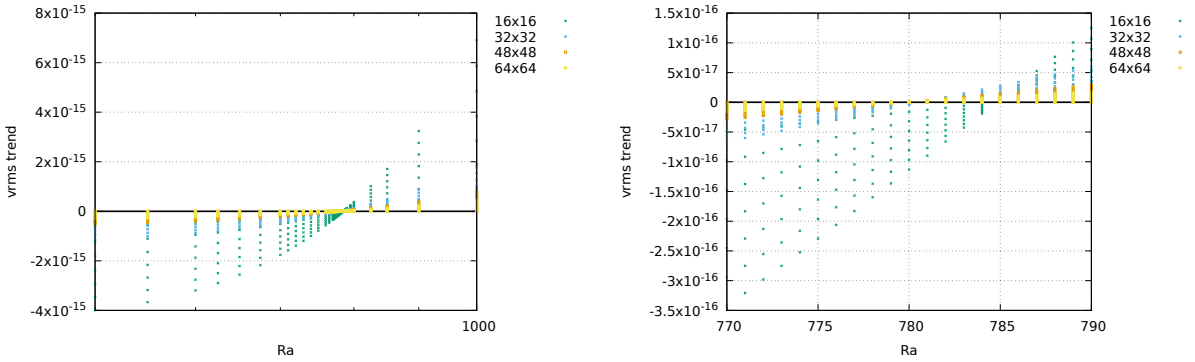
These results were obtained with a 64x64 resolution, and CFL number of 1. Simulation was stopped after about 4300 timesteps.



AH: adiabatic heating, VD: viscous dissipation, HF: heat flux, WG: work against gravity

Onset of convection

The code can be run for values of Ra between 500 and 1000, at various resolutions for the BA formulation. The value $v_{rms}(t) - v_{rms}(0)$ is plotted as a function of Ra and for the 10 first timesteps. If the v_{rms} is found to decrease, then the Rayleigh number is not high enough to allow for convection and the initial temperature perturbation relaxes by diffusion (and then $v_{rms}(t) - v_{rms}(0) < 0$. If the v_{rms} is found to increase, then $v_{rms}(t) - v_{rms}(0) > 0$ and the system is going to showcase convection. The zero value of $v_{rms}(t) - v_{rms}(0)$ gives us the critical Rayleigh number, which is found between 775 and 790.

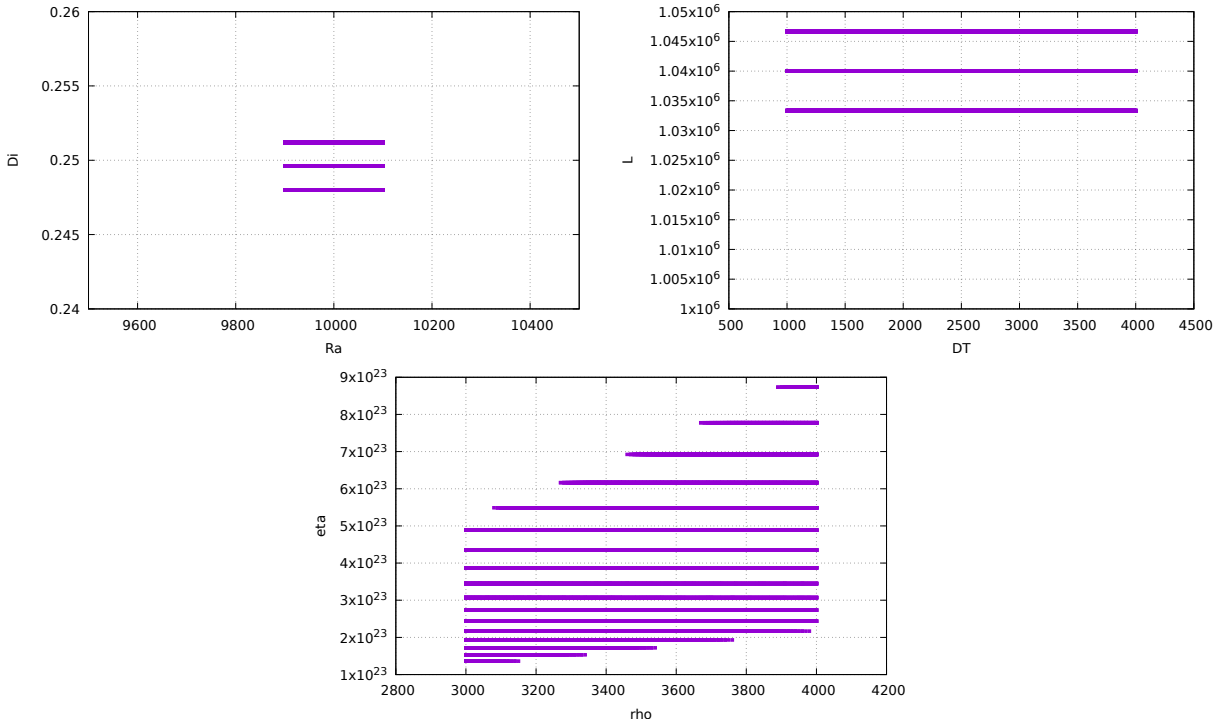


Appendix: Looking for the right combination of parameters for the King benchmark.

I run a quadruple do loop over L, ΔT, ρ_0 and η_0 between plausible values (see code targets.py) and write in a file only the combination which yields the required Rayleigh and Dissipation number values (down to 1% accuracy).

```
alpha=3e-5
g=10
hcapa=1250
hcond=3
DTmin=1000   ; DTmax=4000   ; DTnpts=251
Lmin=1e6     ; Lmax=3e6     ; Lnpts=251
rhomin=3000  ; rhomax=3500  ; rhonpts=41
etamin=19    ; etamax=25    ; etanpts=100
```

On the following plots the 'winning' combinations of these four parameters are shown:



We see that:

- the parameter L (being to the 3rd power in the Ra number) cannot vary too much. Although it is varied between 1000 and 3000km there seems to be a 'right' value at about 1040 km. (why?)
- viscosities are within 10^{23} and 10^{24} which are plausible values (although a bit high?).
- densities can be chosen freely between 3000 and 3500
- ΔT seems to be the most problematic value since it can range from 1000 to 4000K ...

Stone 25

Rayleigh-Taylor instability (1) - instantaneous ($Q_2 \times Q_1$, $Q_2 \times P_{-1}$)



2D
box
Q1P0
Q2Q1
numerical_benchmark
stokes_equation
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_normalisation

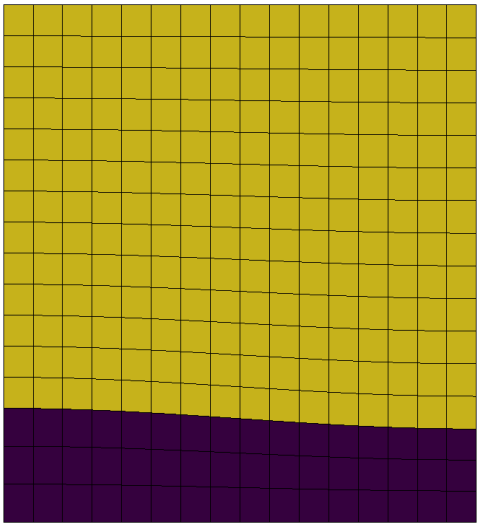
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_25`

This numerical experiment was first presented in van Keken, King, Schmeling, Christensen, Neumeister, and Doin [338] (1997). It consists of an isothermal Rayleigh-Taylor instability in a two-dimensional box of size $L_x = 0.9142$ and $L_y = 1$. Two Newtonian fluids are present in the system: the buoyant layer is placed at the bottom of the box and the interface between both fluids is given by $y(x) = 0.2 + 0.02 \cos\left(\frac{\pi x}{L_x}\right)$ The bottom fluid is parametrised by its mass density $\rho_1 = 1000$ and its viscosity $\eta_1 = 1, 10, 100$, while the layer above is parametrised by $\rho_2 = 1010$ and $\eta_2 = 100$.

No-slip boundary conditions are applied at the bottom and at the top of the box while free-slip boundary conditions are applied on the sides.

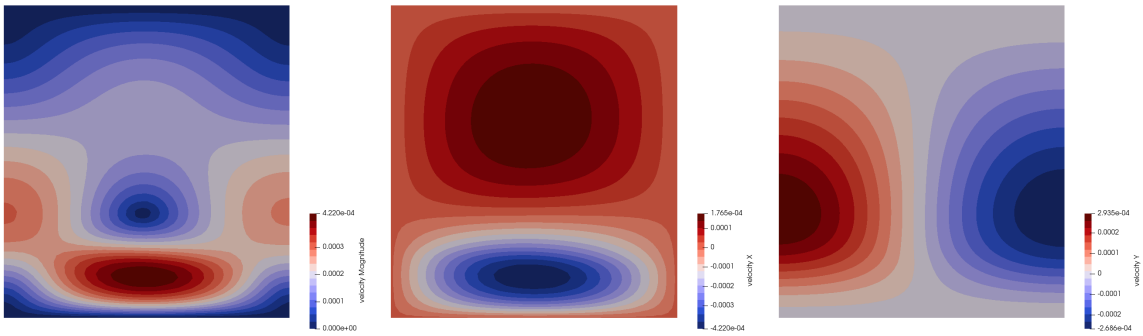
In the original benchmark the system is run over 2000 units of dimensionless time and the timing and position of various upwellings/downwellings is monitored. In this present experiment only the root mean square velocity is measured at $t = 0$: the code is indeed not yet foreseen of any algorithm capable of tracking deformation.

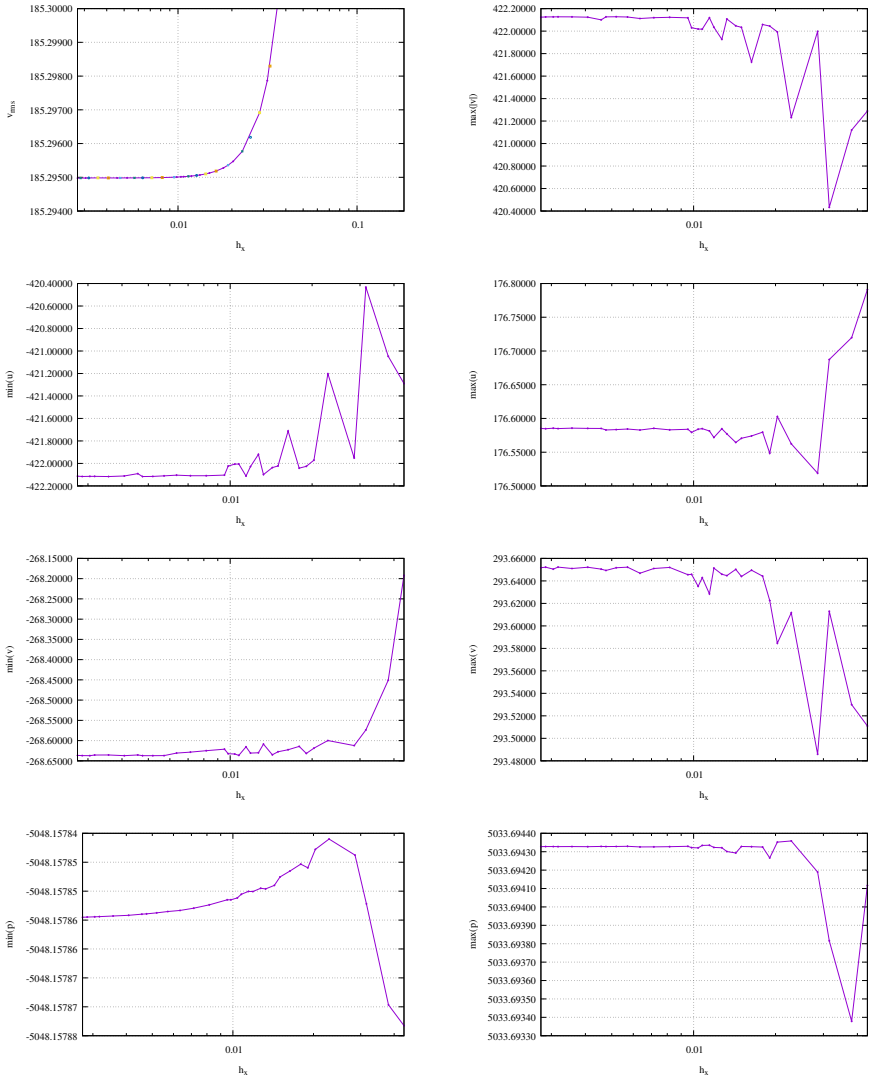
Another approach than the ones presented in the extensive literature which showcases results of this benchmark is taken. The mesh is initially fitted to the fluids interface and the resolution is progressively increased. This results in the following figure:



The code relies on Taylor-Hood eelements ($Q_2 \times Q_1$), see Section **??** and on $Q_2 \times P_{-1}$ elements, see Section **??**. It has been benchmarked agains the Donea & Huerta manufactured solution, see Section **??**.

Isoviscous case - $Q_2 \times Q_1$



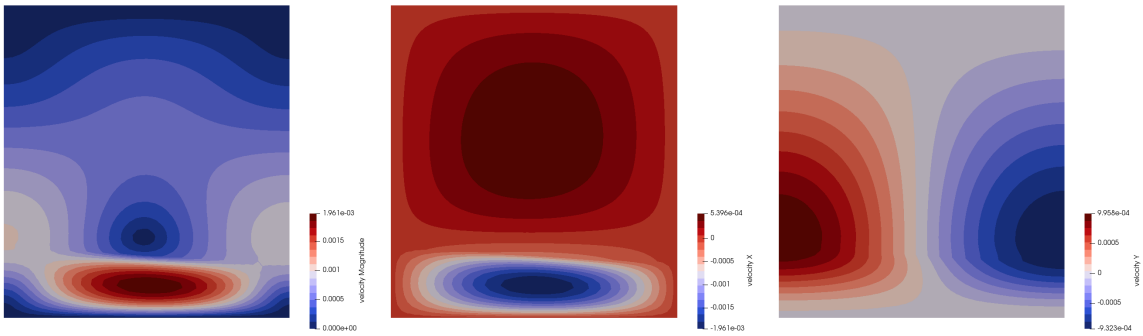


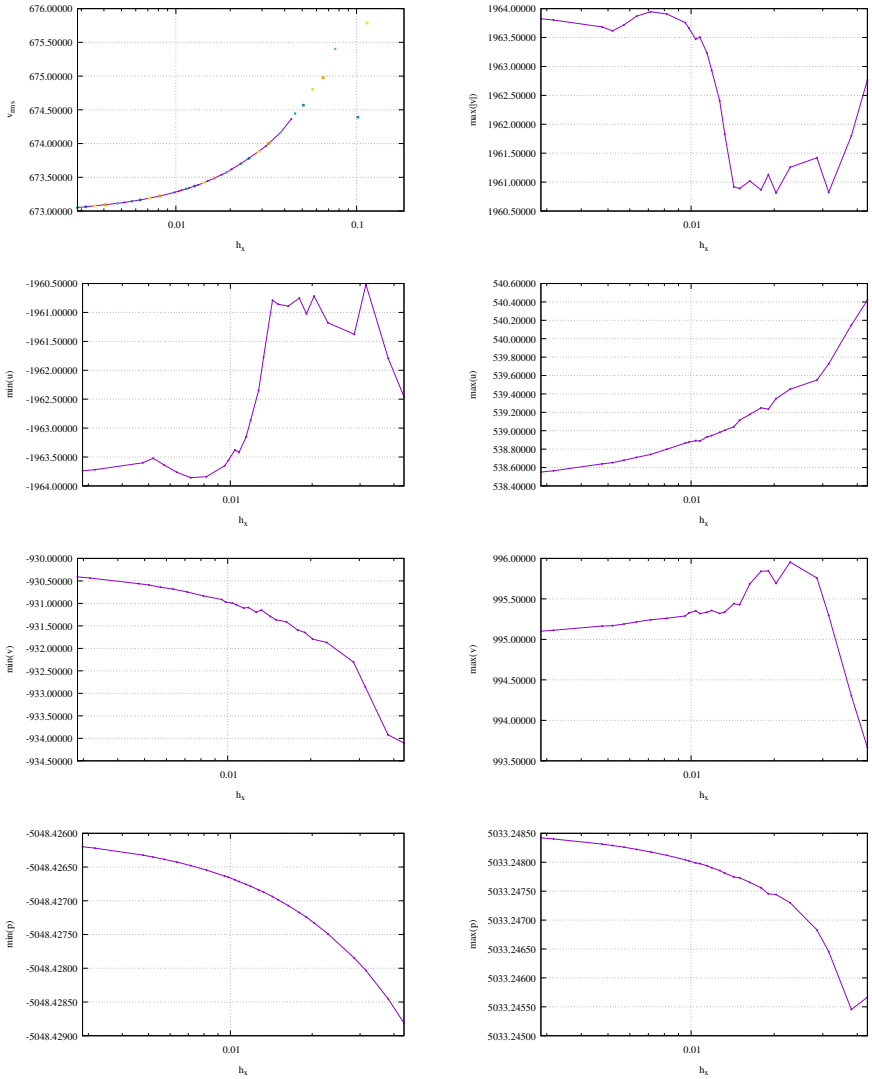
Results obtained with $Q_2 \times Q_1$ elements

nelx	hx	$\nu_{rms}(\times 10^{-6})$	$\nu^{\star}_{rms}(\times 10^{-6})$ (extrap.)	rate
25	0.036568	185.30019494838673	185.2949797	4.009763002
50	0.018284	185.29530341925154	185.2949798	4.0348654
100	0.009142	185.29499975286718	X	X
200	0.004571	185.29498097047734	X	X
8	0.114275	185.9669437569909	185.2954174	4.404188247
16	0.0571375	185.32713277686372	185.2949839	4.057065851
32	0.02856875	185.29691524980284	185.2949784	3.995213562
64	0.014284375	185.29509989938787	185.2949832	6.637356434
128	0.0071421875	185.29498605775253	X	X
256	0.00357109375	185.29498319303758	X	X

Isoviscous case - $Q_2 \times P_{-1}$ All things equal, this element gives me a headache because at low resolution the velocity field showcases a weird pattern which I cannot explain. It is immune to the number of quadrature points. Since the code was benchmarked against a manufactured solution for a regular and perturbed mesh, I don't understand what is going on.

Viscosity ratio = 10 - $Q_2 \times Q_1$

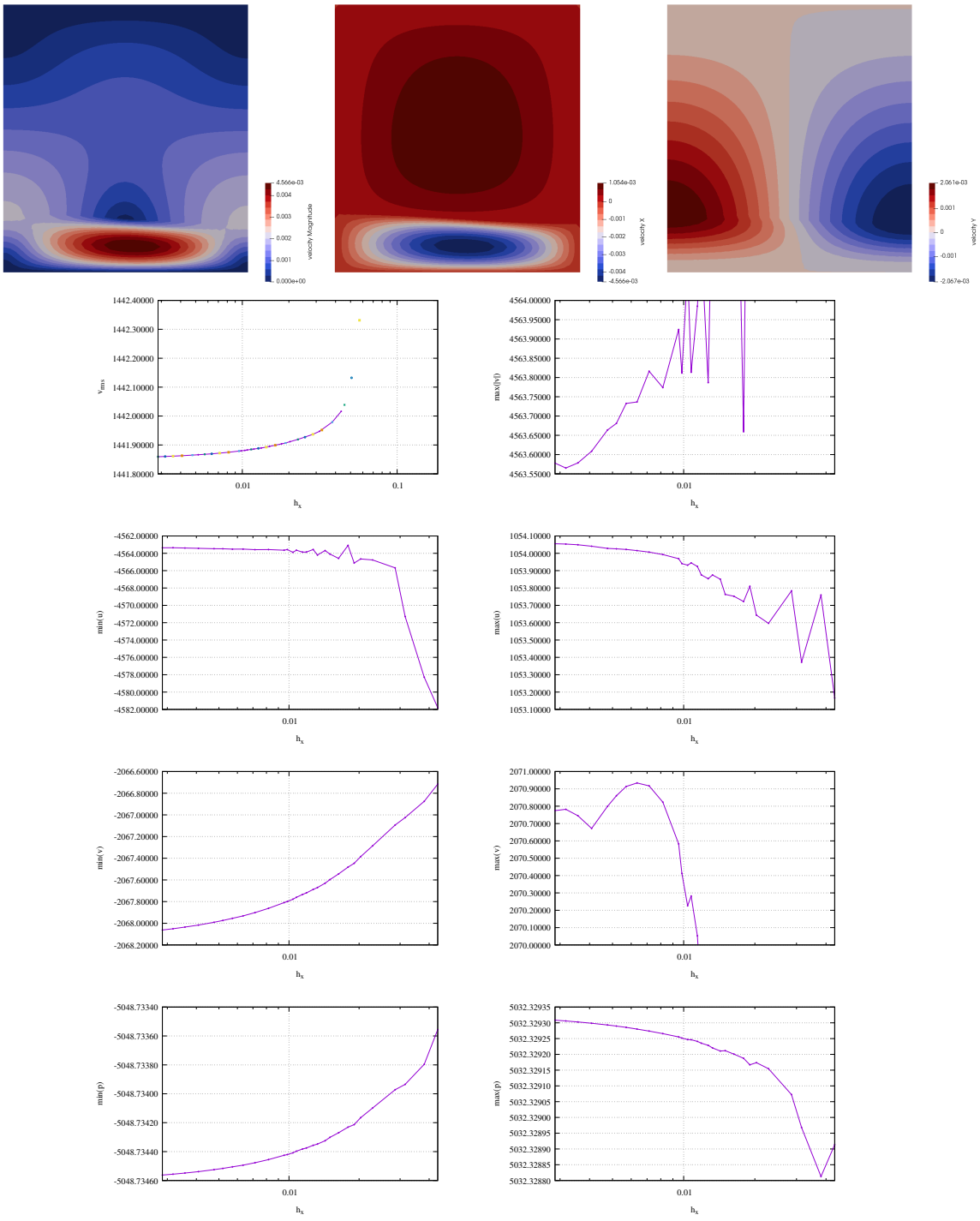




Results obtained with $Q_2 \times Q_1$ elements

nelx	hx	$\nu_{rms}(\times 10^{-6})$	$\nu^{\star}_{rms}(\times 10^{-6})$ (extrap.)	rate
8	0.114275	0.0006757876577772155	0.0006608591144	0.09867640232
16	0.0571375	0.0006748007227229576	0.0006729590166	1.001309003
32	0.02856875	0.000673879034543635	0.0006729654426	1.011510945
64	0.014284375	0.0006734186084028846	X	X
128	0.0071421875	0.0006731902248433064	X	X
256	0.00357109375	0.000673075657106445	X	X

Viscosity ratio = 100 - $Q_2 \times Q_1$



Results obtained with $Q_2 \times Q_1$ elements

$nelx$	h_x	$\nu_{rms}(\times 10^{-6})$	$\nu^{\star}_{rms}(\times 10^{-6})$ (extrap.)	rate
8	0.114275	0.0015073487507568553	0.001441934413	7.364696091
16	0.0571375	0.0014423313101926754	0.001441886911	3.154474001
32	0.02856875	0.0014419368207029728	0.0014418534	1.09262734
64	0.014284375	0.0014418925165763435	X	X
128	0.0071421875	0.0014418717420808874	X	X
256	0.00357109375	0.0014418612856352821	X	X

Stone 26

Slab detachment benchmark (1) - instantaneous ($Q_1 \times P_0$)



2D
box
Q1P0
sparse_matrix
numerical_benchmark
error_convergence
stokes_equation
nonlinear
viscous_rheology
plastic_rheology
buoyancy_driven
vrms
residual
paraview_output
pressure_scaling
pressure_normalisation

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_26`

Part of this stone was developed in collaboration with Thomas Sanders (BSc thesis).

As in Schmalholz [281] (2011), the computational domain is 1000 km × 660 km. No-slip boundary conditions are imposed on the sides of the system while free-slip boundary conditions are imposed at the top and bottom. Two materials are present in the domain: the lithosphere (mat.1) and the mantle (mat.2). The overriding plate (mat.1) is 80 km thick and is placed at the top of the domain. An already subducted slab (mat.1) of 250 km length hangs vertically under this plate. The mantle occupies the rest of the domain.

The mantle has a constant viscosity $\eta_0 = 10^{21}$ Pa s and a density $\rho = 3150$ kg/m^3. The slab has a density $\rho = 3300$ kg m^{-3} and is characterised by a power-law flow law so that its effective viscosity depends on the second invariant of the strainrate $\mathcal{I}_2$ as follows:

$$\begin{aligned}
\eta_{eff} &= \frac{1}{2} A^{-1/n_s} [\mathcal{I}_2(\dot{\varepsilon})]^{1/n_s-1} && (26.1)\\
&= \frac{1}{2} [(2 \times 4.75 \times 10^{11})^{-n_s}]^{-1/n_s} [\mathcal{I}_2(\dot{\varepsilon})]^{1/n_s-1} && (26.2)\\
&= 4.75 \times 10^{11} [\mathcal{I}_2(\dot{\varepsilon})]^{1/n_s-1} && (26.3)\\
&= \eta_0 [\mathcal{I}_2(\dot{\varepsilon})]^{1/n_s-1} && (26.4)
\end{aligned}$$

with $n_s = 4$ and $A = (2 \times 4.75 \times 10^{11})^{-n_s}$, or $\eta_0 = 4.75 \times 10^{11}$.

The mantle rheology can also be characterised by a power-law flow law with $n_m = 3$ and $A = (2 \times 4.54 \times 10^{10})^{-n_m}$.

In this example no material advection is implemented so we can only run the experiment for 1 time step and look at the process of nonlinear convergence and the final converged state. I have separated the experiments into four cases:

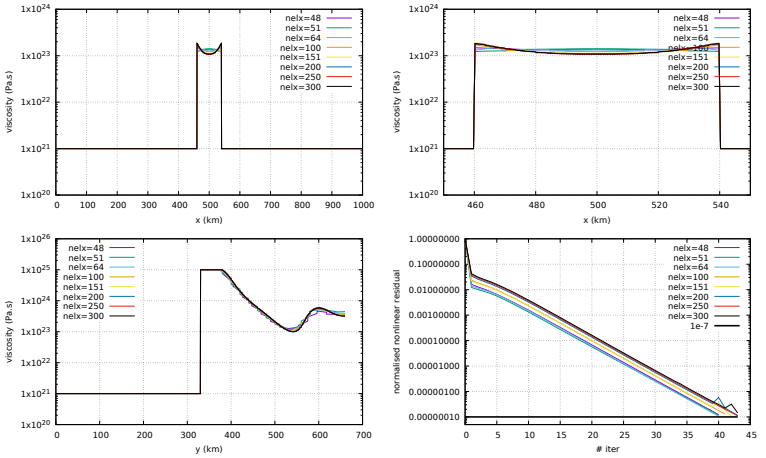
case number	lithosphere	mantle	top surface
1a	non linear	linear	free slip
1b	non linear	linear	open
2a	non linear	non linear	free slip
2b	non linear	non linear	open

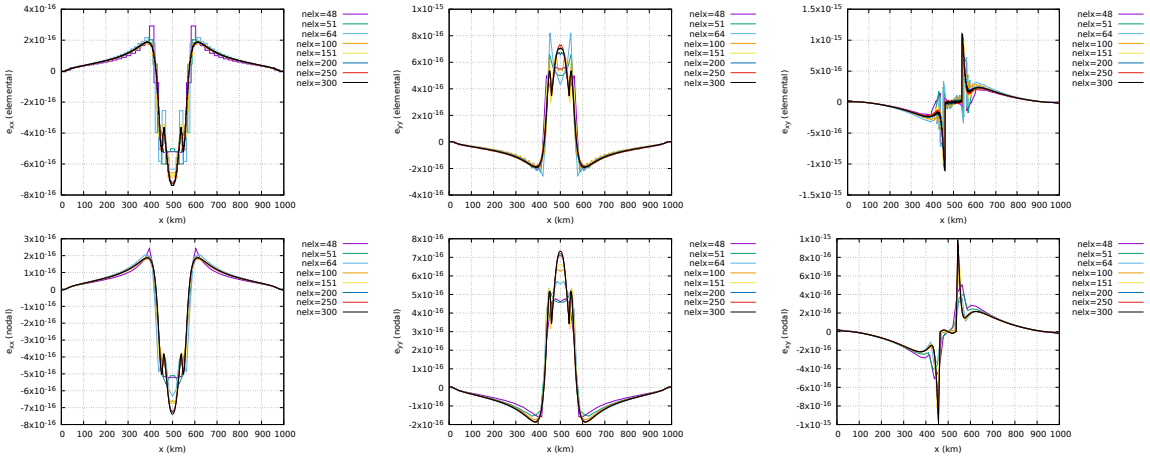
This experiment is implemented in ASPECT and is available with the code.

Measurements are carried out on a vertical line $x = L_x/2$ and a horizontal line $y = 550$ km.

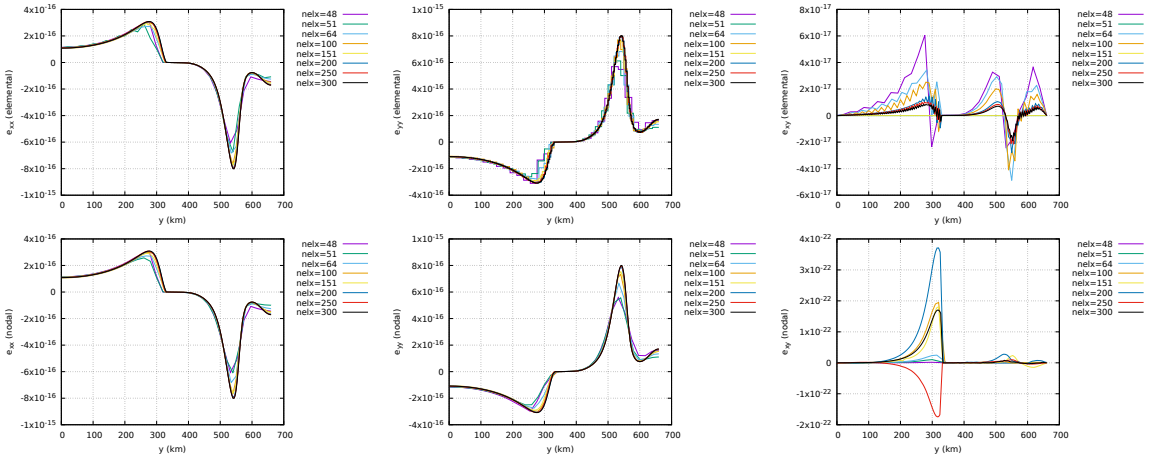
Relevant Literature: Bellas *et al.* (2018) [16].

Case 1a - linear mantle - free slip top surface



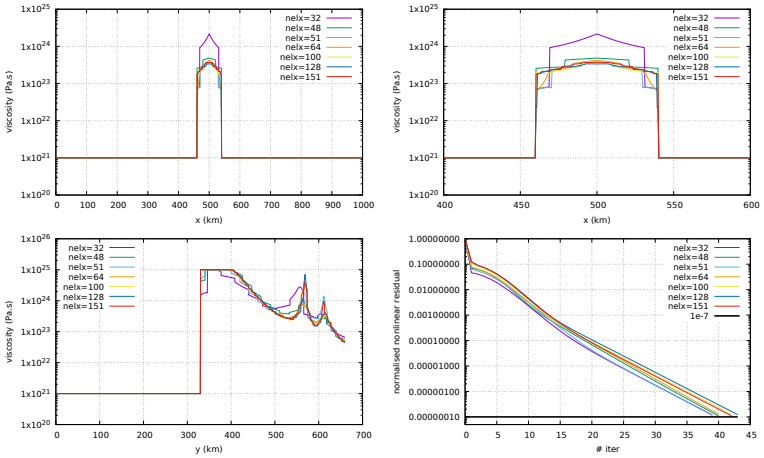


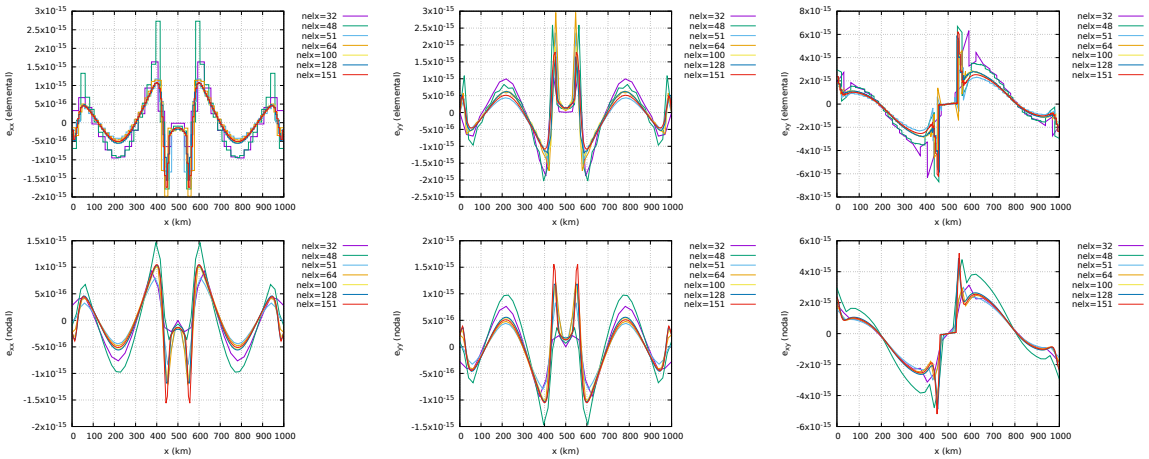
Along the horizontal line



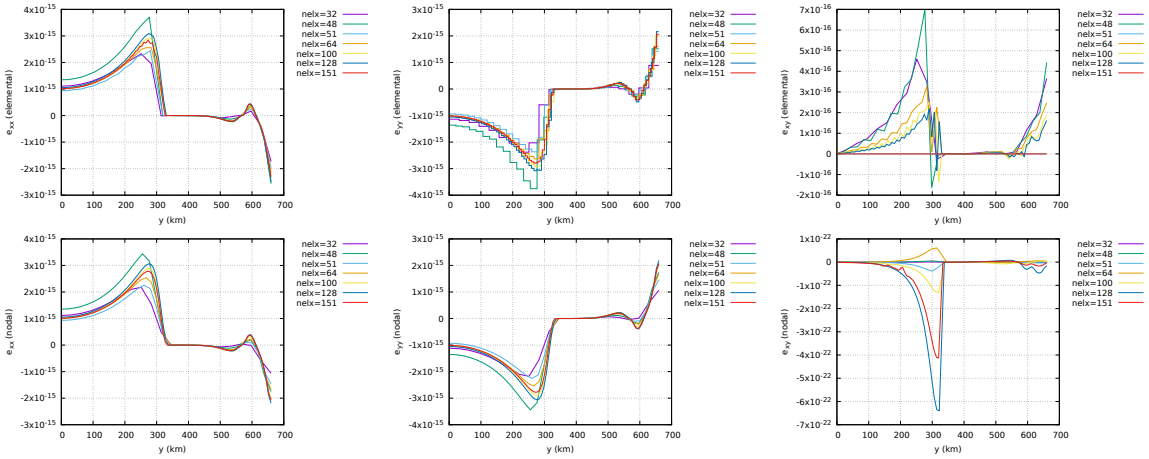
Along the vertical line

Case 1b - linear mantle - open top surface .



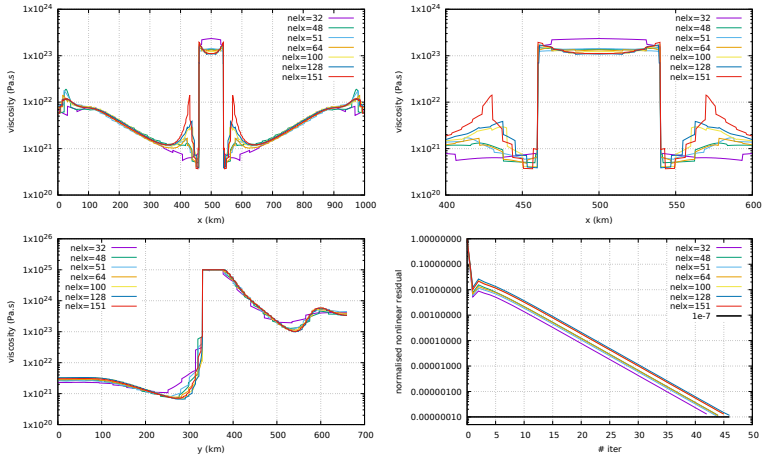


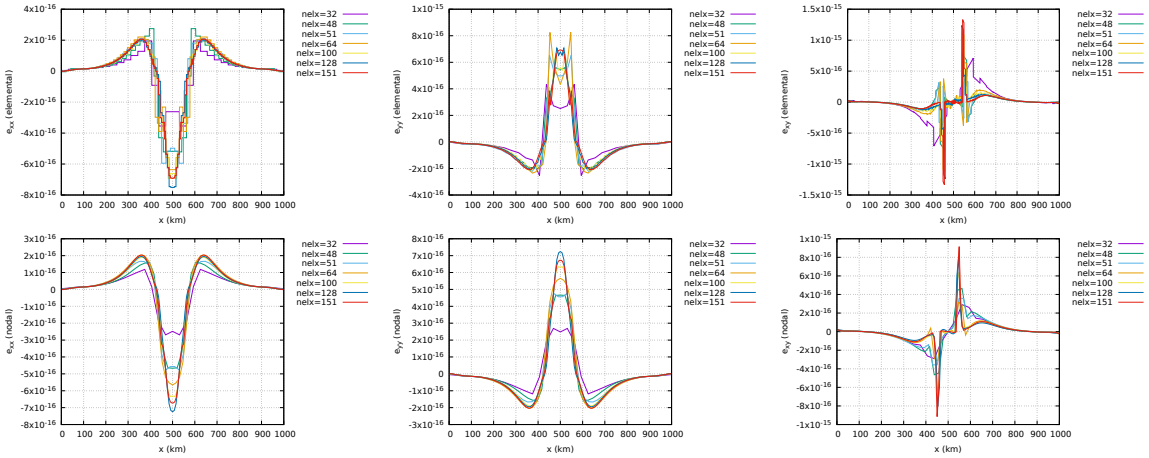
Along the horizontal line



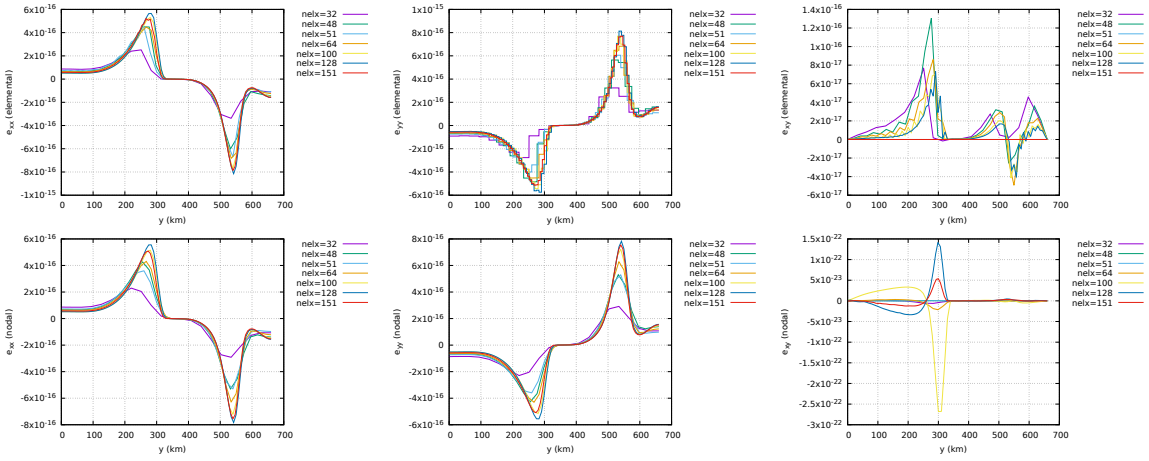
Along the vertical line

Case 2a - nonlinear mantle - free slip top surface



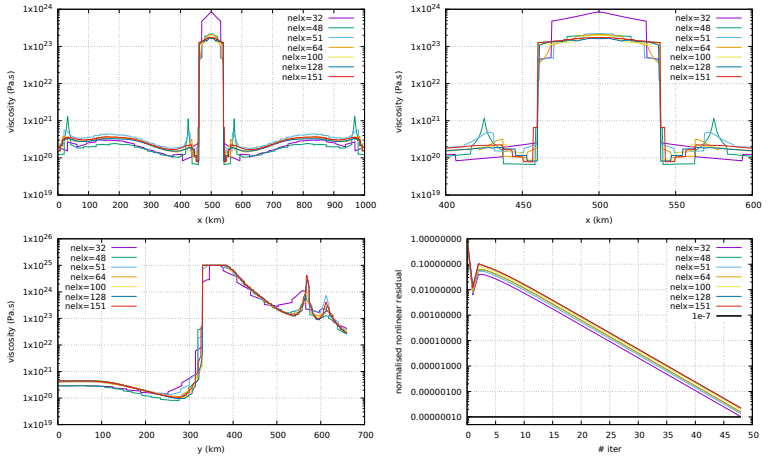


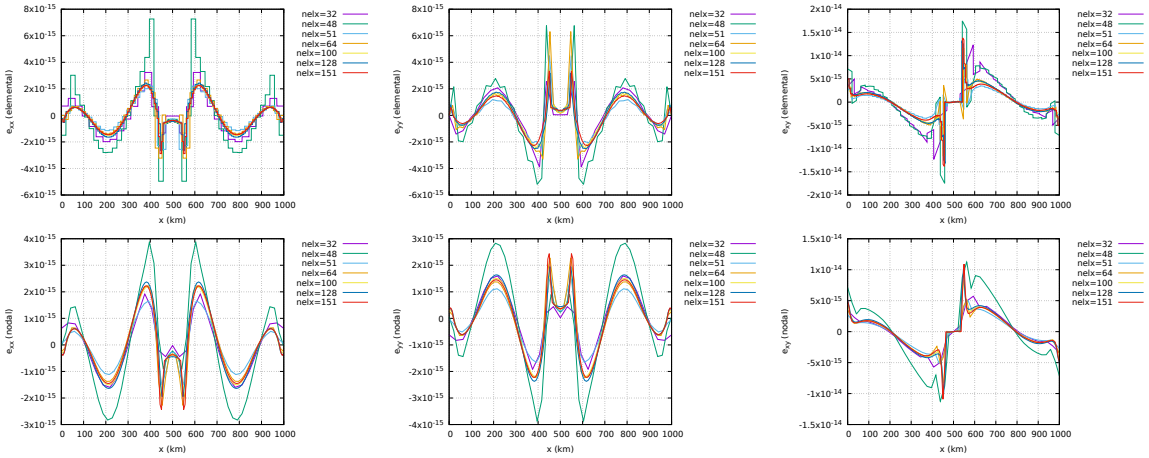
Along the horizontal line



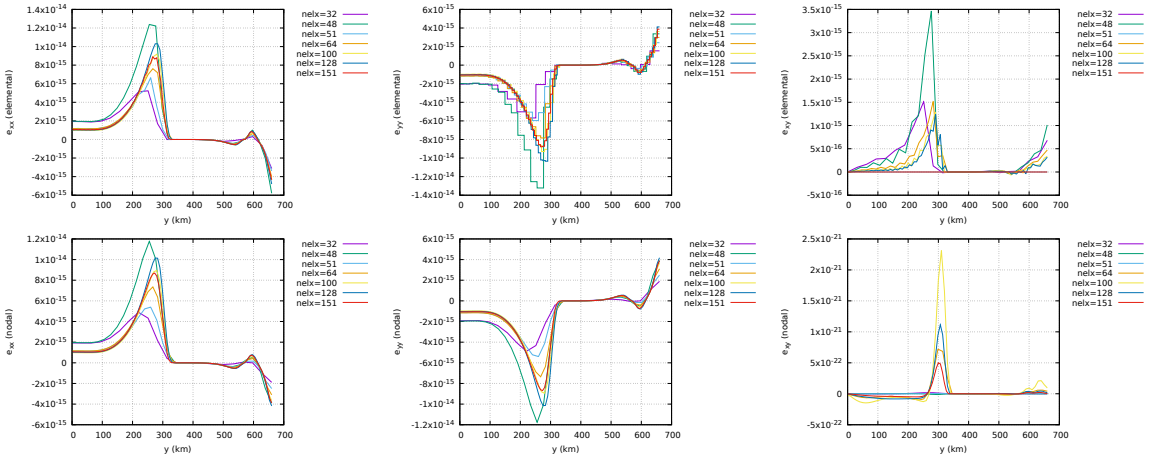
Along the vertical line

Case 2b - nonlinear mantle - open top surface



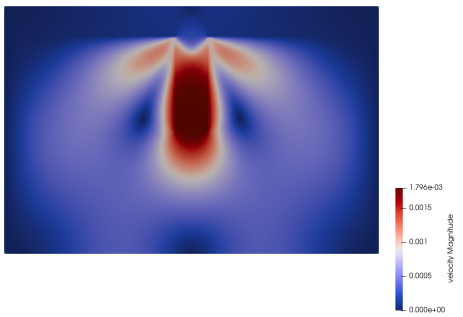


Along the horizontal line

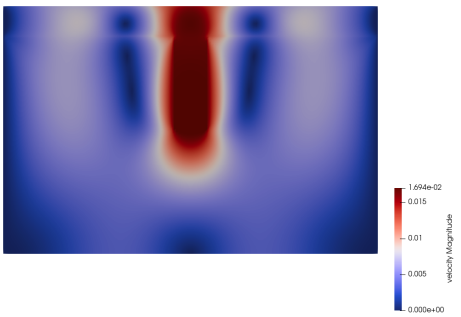


Along the vertical line

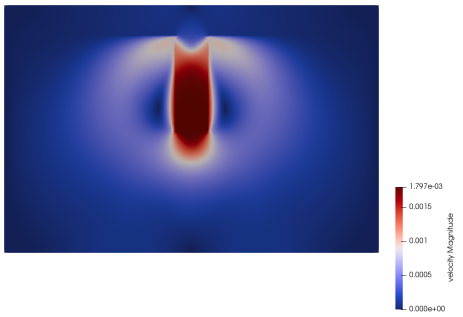
case 1a ↓



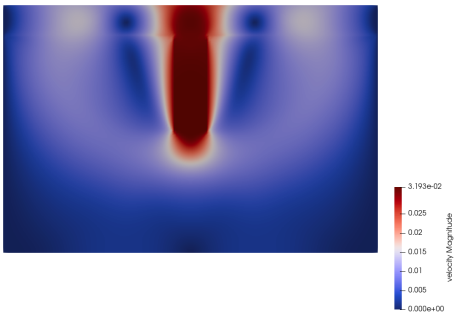
case 1b ↓



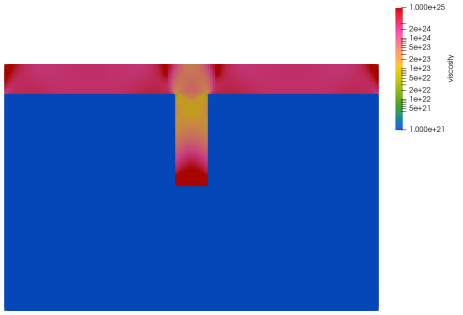
case 2a ↓



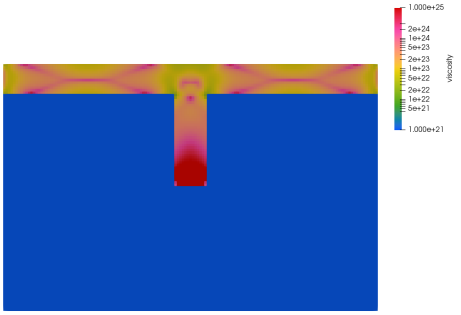
case 2b ↓



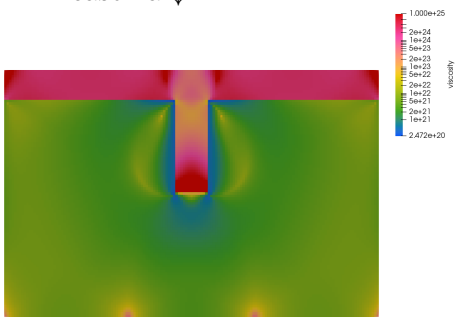
case 1a ↓



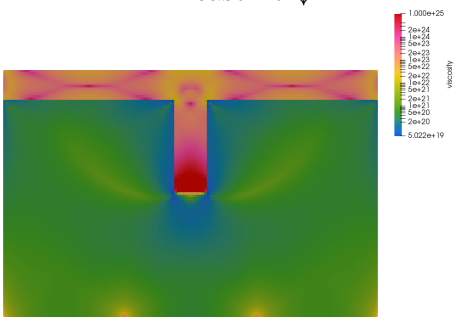
case 1b ↓



case 2a ↓



case 2b ↓



Stone 27

Consistent Boundary Flux ($Q_1 \times P_0$)



2D
box
Q1P0
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
buoyancy_driven
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_27`

In what follows we will be re-doing the numerical experiments presented in Zhong, Gurnis, and Hulbert [370] (1993).

The first benchmark showcases a unit square domain with free slip boundary conditions prescribed on all sides. The resolution is fixed to 64×64 $Q_1 \times P_0$ elements. The flow is isoviscous and the buoyancy force $\boldsymbol{f}$ is given by

$$\begin{aligned} f_x &= 0 \\ f_y &= \rho_0 \alpha T(x,y) \end{aligned}$$

with the temperature field given by

$$T(x,y) = \cos(kx)\delta(y-y_0)$$

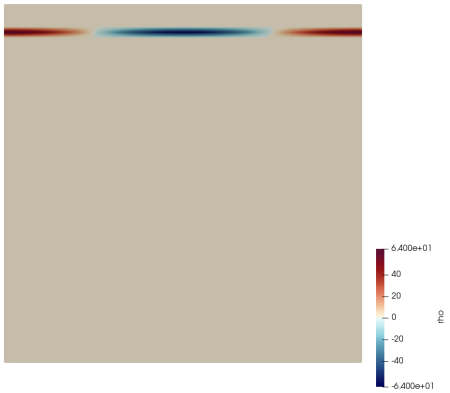
where $k = 2\pi/\lambda$ and λ is a wavelength, and y_0 represents the location of the buoyancy strip. We set $g_y = -1$ and prescribe $\rho(x,y) = \rho_0 \alpha \cos(kx)\delta(y-y_0)$ on the nodes of the mesh.

One can prove (Zhong *et al.* [370] and refs. therein) that there is an analytic solution for the surface stress σ_{zz} [1]

$$\frac{\sigma_{yy}}{\rho\alpha g h} = \frac{\cos(kx)}{\sinh^2(k)} \left[k(1-y_0)\sinh(k)\cosh(ky_0) - k\sinh(k(1-y_0)) + \sinh(k)\sinh(ky_0) \right]$$

We choose $\rho_0\alpha = 64$, $\eta = 1$ (note that in this case the normalising coefficient of the stress is exactly 1 (since $h = L_x/nelx = 1/64$) so it is not implemented in the code). $\lambda = 1$ is set to 1 and we explore $y_0 = \frac{63}{64}, \frac{62}{64}, \frac{59}{64}$ and $y_0 = 32/64$. Under these assumptions the density field for $y_0 = 59/64$ is:

[1] Note that in the paper the authors use $\rho\alpha g$ which does not have the dimensions of a stress

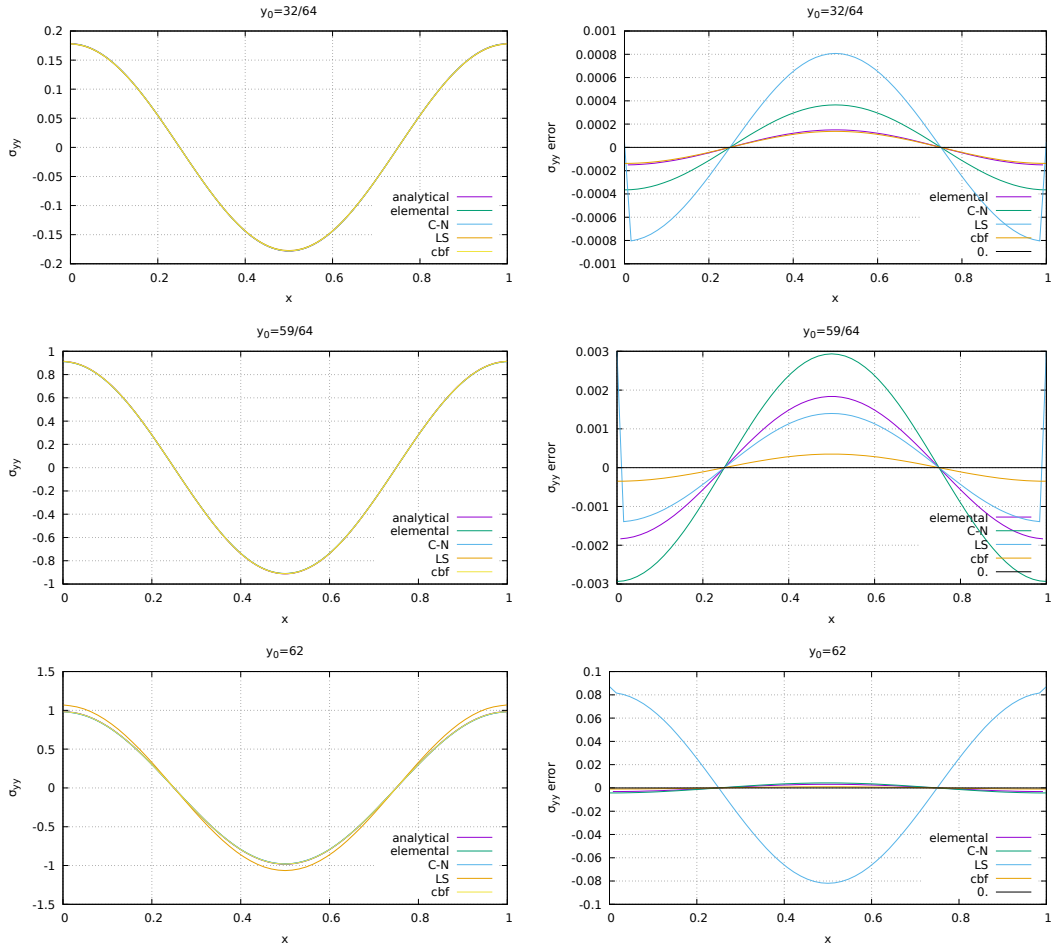


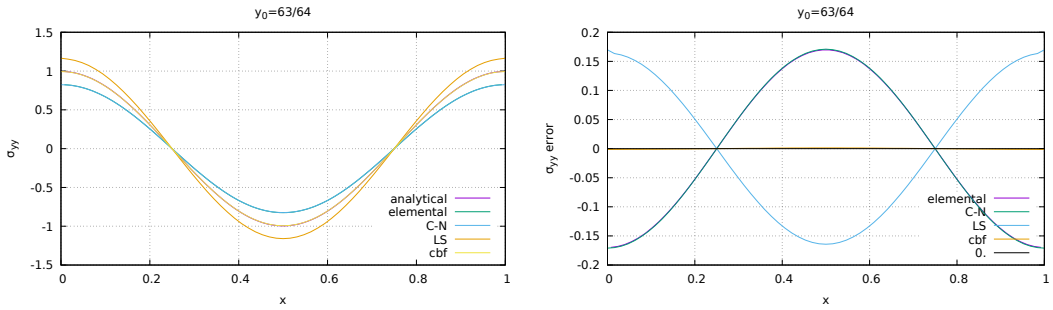
We can recover the stress at the boundary by computing the yy component of the stress tensor in the top row of elements:

$$\sigma_{yy} = -p + 2\eta\dot{\varepsilon}_{yy}(\vec{v})$$

Note that pressure is by definition elemental, and that strain rate components are then also computed in the middle of each element.

These elemental quantities can be projected onto the nodes (INSERT REFERENCE) by means of the C→N algorithm or a least square algorithm (LS).





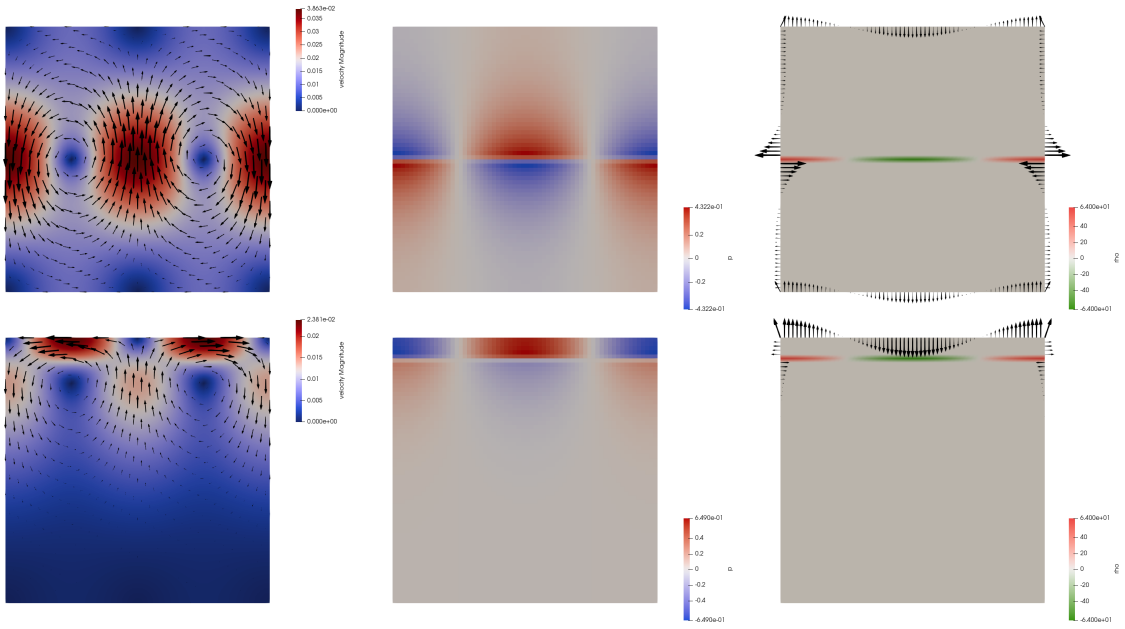
The consistent boundary flux (CBF) method allows us to compute traction vectors $\vec{t} = \boldsymbol{\sigma} \cdot \vec{n}$ on the boundary of the domain. On the top boundary, $\vec{n} = (0, 1)$ so that $\vec{t} = (\sigma_{xy}, \sigma_{yy})^T$ and t_y is the quantity we need to consider and compare to other results.

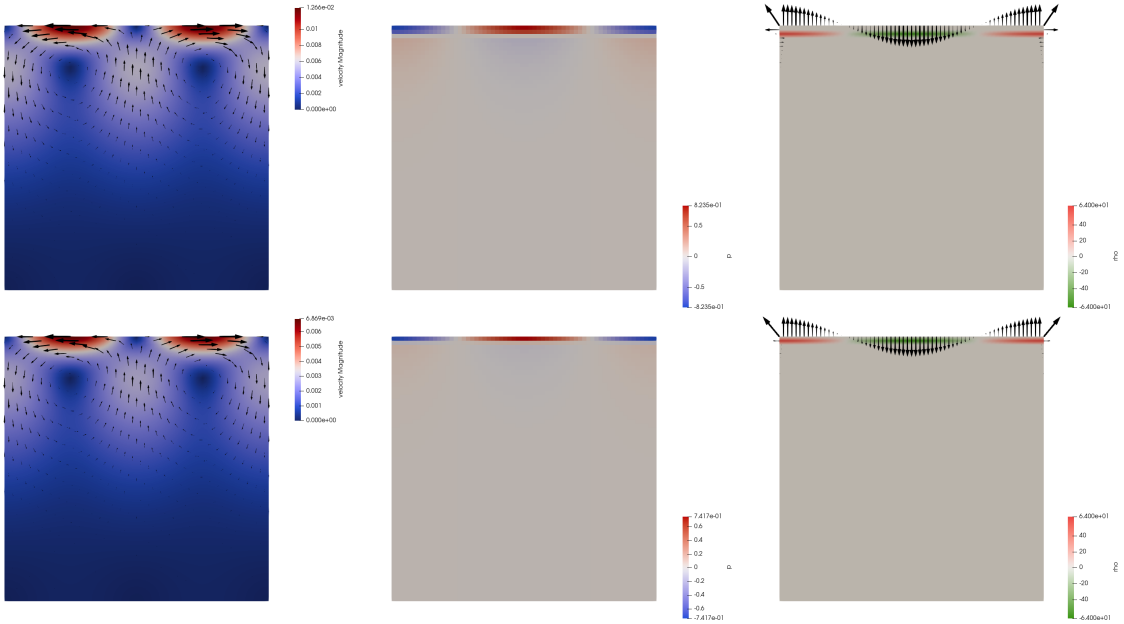
In the following table are shown the results presented in [370] alongside the results obtained with Fieldstone:

Method	$y_0 = 63/64$	$y_0 = 62/64$	$y_0 = 59/64$ [2]	$y_0 = 32/64$
Analytic solution	0.995476	0.983053	0.912506	0.178136
Pressure smoothing [370]	1.15974	1.06498	0.911109	n.a.
CBF [370]	0.994236	0.982116	0.912157	n.a.
fieldstone: elemental	0.824554 (-17.17 %)	0.978744 (-0.44%)	0.909574 (-0.32 %)	0.177771 (-0.20 %)
fieldstone: nodal (C→N)	0.824554 (-17.17 %)	0.978744 (-0.44%)	0.909574 (-0.32 %)	0.177771 (-0.20 %)
fieldstone: LS	1.165321 (17.06 %)	1.070105 (8.86%)	0.915496 (0.33 %)	0.178182 (0.03 %)
fieldstone: CBF	0.994236 (-0.13 %)	0.982116 (-0.10%)	0.912157 (-0.04 %)	0.177998 (-0.08 %)

We see that we recover the published results with the same exact accuracy, thereby validating our implementation. Also rather fascinating is the fact that the original paper carries out the whole study without showing any image of the 2D domain ever.

On the following figures are shown the velocity, pressure and traction fields for two cases $y_0 = 32/64$ and $y_0 = 63/64$.





Here lies the superiority of our approach over the one presented in the original article: our code computes all traction vectors on all boundaries at once.

explain how Medge is arrived at!

compare with ASPECT ??!

pressure average on surface instead of volume ?

Remark. *The original article on the CBF [370] uses a penalty-based formulation (see Section 10.7) so that they do not have to worry about pressure normalisation. However, since constant pressure fields lie in the nullspace of $\mathbb{G}$ the pressure normalisation constant does not play a role.*

As shown in Appendix C, the Q_1 mass matrix for the reference cell/element is given by:

$$\mathbb{M}^e = \frac{1}{3} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

This matrix needs to be multiplied by $h/2$ for an element of size h. Following the methodology presented in [370], one can also use the Gauss-Lobatto quadrature method to arrive at the mass matrix, and in this case the simple trapezoidal integration rule variant thereof (see Section 7.1.1). We then have:

$$\mathbb{M}_e = \int_{\Omega_e} \vec{N}^T \vec{N} dV = \int_{-1}^{+1} \vec{N}^T \vec{N} dr \tag{27.1}$$

on the reference element, with

$$\vec{N}^T = \begin{pmatrix} N_1(r) \\ N_2(r) \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1-r \\ 1+r \end{pmatrix}$$

We know compute the following integrals with the trapezoidal rule:

$$\int_{-1}^{+1} N_1(r) N_1(r) dr = (1--1) \frac{N_1(-1)N_1(-1) + N_1(+1)N_1(+1)}{2} = 1 \tag{27.2}$$

$$\int_{-1}^{+1} N_1(r) N_2(r) dr = (1--1) \frac{N_1(-1)N_2(-1) + N_1(+1)N_2(+1)}{2} = 0 \tag{27.3}$$

$$\int_{-1}^{+1} N_2(r) N_2(r) dr = (1--1) \frac{N_2(-1)N_2(-1) + N_2(+1)N_2(+1)}{2} = 1 \tag{27.4}$$

and finally

$$\mathbb{M}^e = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The resulting matrix is diagonal and it is simply the lumped version of the exact mass matrix.

Stone 28

convection 2D box - Tosi *et al.* , 2015 ($Q_1 \times P_0$)



2D
box
Q1P0
sparse_matrix
numerical_benchmark
stokes_equation
energy_equation
thermo_mechanical
time_stepping
nonlinear
viscous_rheology
plastic_rheology
buoyancy_driven
vrms
nusselt_number
rayleigh_number
residual
paraview_output
pressure_scaling
pressure_normalisation
steady_state

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_28`

This fieldstone was developed in collaboration with Rens Elbertsen.

This STONE is based on Tosi et al. [332] (2015). As for every STONE aiming at reproducing results off a publication I here include de abstract of the article:

Numerical simulations of thermal convection in the Earth's mantle often employ a pseudoplastic rheology in order to mimic the plate-like behavior of the lithosphere. Yet the benchmark tests available in the literature are largely based on simple linear rheologies in which the viscosity is either assumed to be constant or weakly dependent on temperature. Here we present a suite of simple tests based on nonlinear rheologies featuring temperature, pressure, and strain rate-dependent viscosity. Eleven different codes based on the finite volume, finite element, or spectral methods have been used to run five benchmark cases leading to stagnant lid, mobile lid, and periodic convection in a 2-D square box. For two of these cases, we also show resolution tests from all contributing codes. In addition, we present a bifurcation analysis, describing the transition from a mobile lid regime to a periodic regime, and from a periodic regime to a stagnant lid regime, as a function of the yield stress. At a resolution of around 100 cells or elements in both vertical and horizontal directions, all codes reproduce the required diagnostic quantities with a discrepancy of at most 3% in the presence of both linear and nonlinear rheologies. Furthermore, they consistently predict the critical value of the yield stress at which the transition between different regimes occurs. As the most recent mantle convection codes can handle a number of different geometries within a single solution framework, this benchmark will also prove useful when validating viscoplastic thermal convection simulations in such geometries.

The viscosity field η is calculated as the harmonic average between a linear part η_{lin} that depends on temperature only or on temperature and depth d , and a non-linear, plastic part η_{plast} dependent on the strain rate:

$$\eta(T, z, \dot{\boldsymbol{\epsilon}}) = 2\left(\frac{1}{\eta_{\text{lin}}(T, z)} + \frac{1}{\eta_{\text{plast}}(\dot{\boldsymbol{\epsilon}})}\right)^{-1}. \tag{28.1}$$

The linear part is given by the linearized Arrhenius law (the so-called Frank-Kamenetskii approximation [115]):

$$\eta_{\text{lin}}(T, z) = \exp(-\gamma_T T + \gamma_z z), \tag{28.2}$$

where $\gamma_T = \ln(\Delta\eta_T)$ and $\gamma_z = \ln(\Delta\eta_z)$ are parameters controlling the total viscosity contrast due to temperature ($\Delta\eta_T$) and pressure ($\Delta\eta_z$). The non-linear part is given by [334, 333]:

$$\eta_{\text{plast}}(\dot{\boldsymbol{\epsilon}}) = \eta^* + \frac{\sigma_Y}{\sqrt{\dot{\boldsymbol{\epsilon}} : \dot{\boldsymbol{\epsilon}}}}, \tag{28.3}$$

where η^* is a constant representing the effective viscosity at high stresses [305] and σ_Y is the yield stress, also assumed to be constant. In 2-D, the denominator in the second term of equation (28.3) is given explicitly by

$$\sqrt{\dot{\boldsymbol{\epsilon}} : \dot{\boldsymbol{\epsilon}}} = \sqrt{\dot{\epsilon}_{ij}\dot{\epsilon}_{ij}} = \sqrt{\left(\frac{\partial u_x}{\partial x}\right)^2 + \frac{1}{2}\left(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right)^2 + \left(\frac{\partial u_y}{\partial y}\right)^2}. \tag{28.4}$$

The viscoplastic flow law (equation 28.1) leads to linear viscous deformation at low stresses (equation (28.2)) and to plastic deformation for stresses that exceed σ_Y (equation (28.3)), with the decrease in viscosity limited by the choice of η^* [305].

In all cases that we present, the domain is a two-dimensional square box. The mechanical boundary conditions are for all boundaries free-slip with no flux across, i.e. $\tau_{xy} = \tau_{yx} = 0$ and $\boldsymbol{u} \cdot \boldsymbol{n} = 0$, where $\boldsymbol{n}$ denotes the outward normal to the boundary. Concerning the energy equation, the bottom and top boundaries are isothermal, with the temperature T set to 1 and 0, respectively, while side-walls are assumed to be insulating, i.e. $\partial T/\partial x = 0$. The initial distribution of the temperature field is prescribed as follows:

$$T(x, y) = (1 - y) + A\cos(\pi x)\sin(\pi y), \tag{28.5}$$

where $A = 0.01$ is the amplitude of the initial perturbation.

In the following Table, we list the benchmark cases according to the parameters used.

Case	Ra	$\Delta\eta_T$	$\Delta\eta_y$	η^*	σ_Y	Convective regime
1	10^2	10^5	1	–	–	Stagnant lid
2	10^2	10^5	1	10^{-3}	1	Mobile lid
3	10^2	10^5	10	–	–	Stagnant lid
4	10^2	10^5	10	10^{-3}	1	Mobile lid
5a	10^2	10^5	10	10^{-3}	4	Periodic
5b	10^2	10^5	10	10^{-3}	3 – 5	Mobile lid – Periodic – Stagnant lid

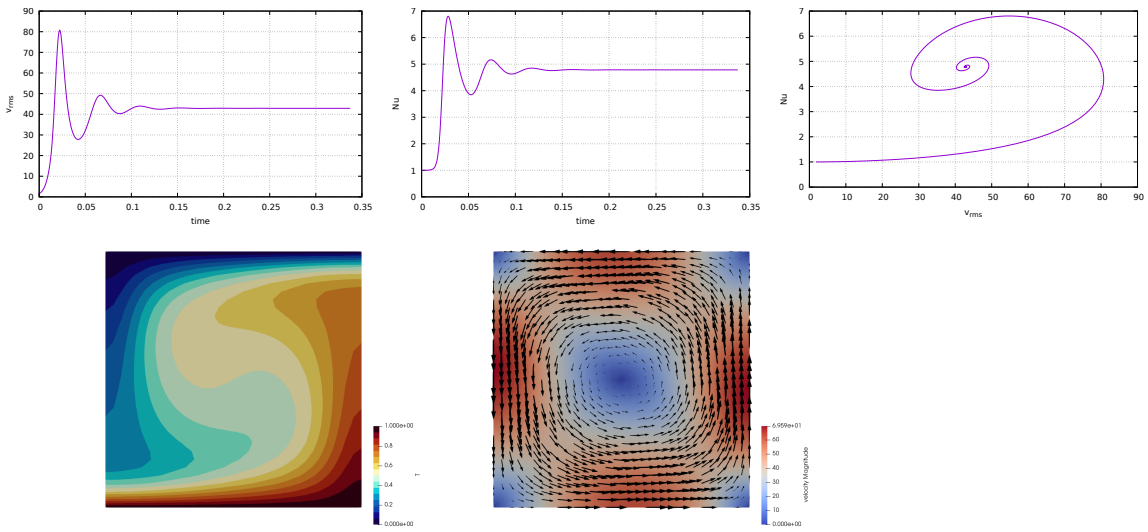
Benchmark cases and corresponding parameters.

In Cases 1 and 3 the viscosity is directly calculated from equation (28.2), while for Cases 2, 4, 5a, and 5b, we used equation (28.1). For a given mesh resolution, Case 5b requires running simulations with yield stress varying between 3 and 5

In all tests, the reference Rayleigh number is set at the surface ($y = 1$) to 10^2, and the viscosity contrast due to temperature $\Delta\eta_T$ is 10^5, implying therefore a maximum effective Rayleigh number of 10^7 for $T = 1$. Cases 3, 4, 5a, and 5b employ in addition a depth-dependent rheology with viscosity contrast $\Delta\eta_z = 10$. Cases 1 and 3 assume a linear viscous rheology that leads to a stagnant lid regime. Cases 2 and 4 assume a viscoplastic rheology that leads instead to a mobile lid regime. Case 5a also assumes a viscoplastic rheology but a higher yield stress, which ultimately causes the emergence of a strictly periodic regime. The setup of Case 5b is identical to that of Case 5a but the test consists in running several simulations using different yield stresses. Specifically, we varied σ_Y between 3 and 5 in increments of 0.1 in order to identify the values of the yield stress corresponding to the transition from mobile to periodic and from periodic to stagnant lid regime.

NOTE: no Crank-Nicolson scheme for temperature !!

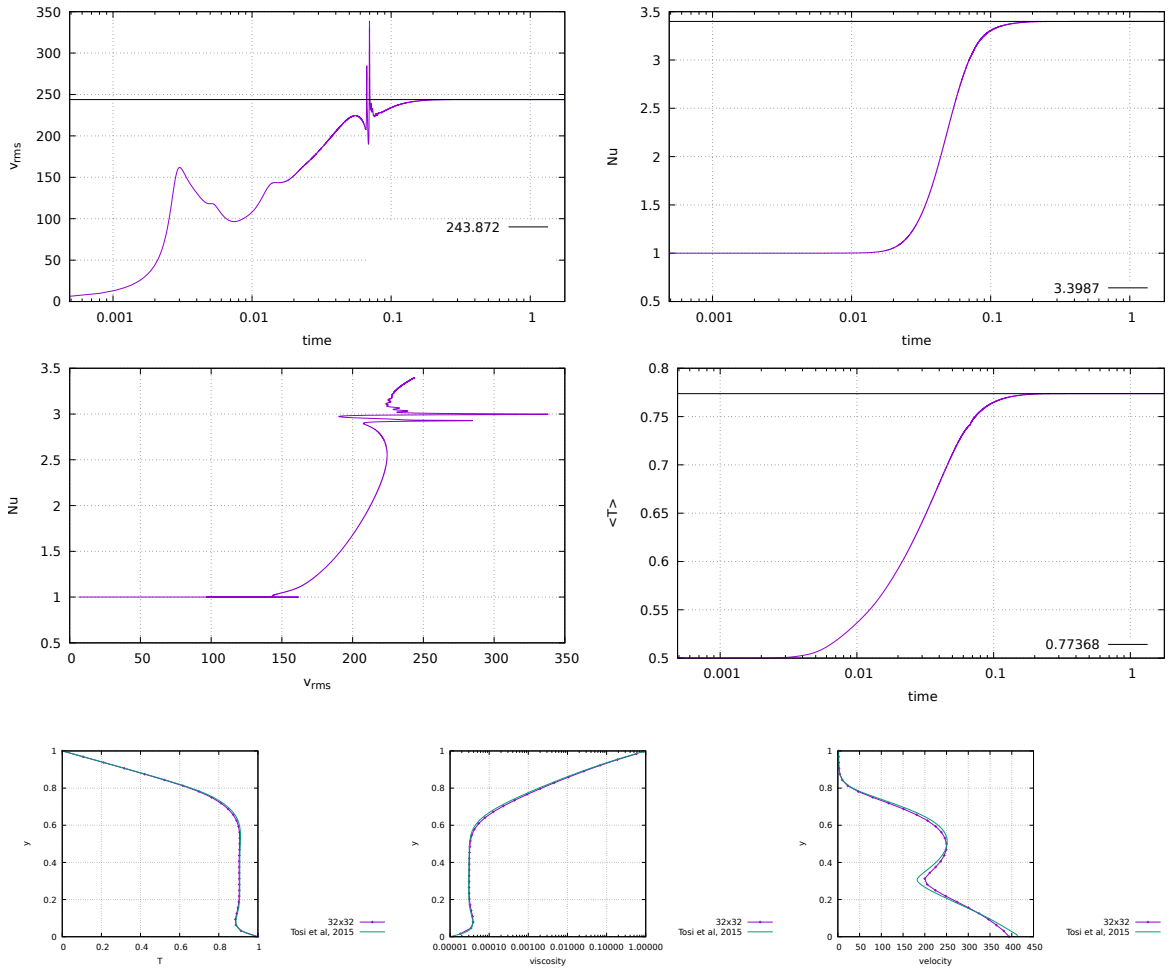
Case 0: Newtonian case, a la Blankenbach et al., 1989

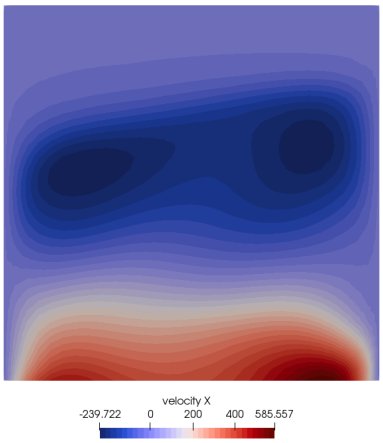


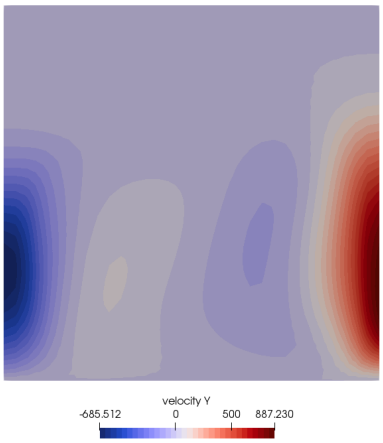
Case 1

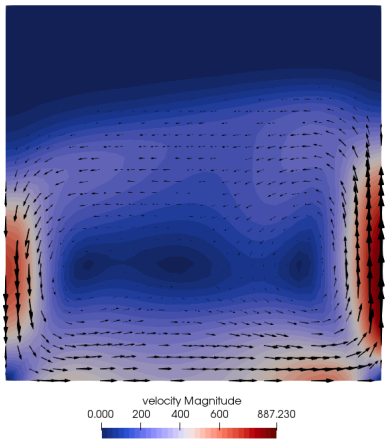
In this case $\eta^\star = 0$ and $\sigma_Y = 0$ so that η_{plast} can be discarded. The CFL number is set to 0.5 and the viscosity is given by $\eta(T, z, \dot{\boldsymbol{\epsilon}}) = \eta_{\rm lin}(T, z)$. And since $\Delta\eta_z = 1$ then $\gamma_z = 0$ so that $\eta_{\rm lin}(T, z) = \exp(-\gamma_T T)$

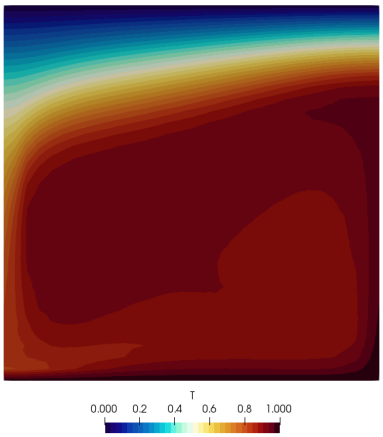
Code Resolution	YACC 100 × 100	Plaatjes 128×128*r*	CHIC 80 × 80	GAIA 100*r*×100*r*	StreamV 80 × 80	StagYY 128×128*r*	FEniCS 80 × 80	Fluidity 128 × 128	ELEFANT 100 × 100	ASPECT 64 × 64	MC3D 100 × 100
Case 1											
$\langle T \rangle$	0.7767	0.7759	0.7758	0.7759	0.776	0.776	0.7759	0.7758	0.7758	0.7768	0.779
$Nu_{\rm top}$	3.4298	3.4159	3.4260	3.4213	3.4091	3.419	3.5889	3.4253	3.4214	3.4305	3.3129
$Nu_{\rm bot}$	3.3143	3.4159	3.4259	3.4213	3.4091	3.419	3.4231	3.3795	3.313	3.4142	3.3139
$u_{\rm RMS}$	251.7997	249.54	249.2985	250.0738	252.0906	249.541	249.5730	248.9252	249.134	251.3069	296.6156
$u^{\rm surf}_{\rm RMS}$	1.8298	1.878	1.8999	1.8836	1.8823	1.8723	1.8698	1.8474	1.8642	1.8695	1.1114
$u^{\rm surf}_{\rm max}$	2.5516	2.618	2.6477	2.6254	2.64	2.6104	2.6066	2.5761	2.6119	2.6064	1.5329
$\langle W \rangle$	2.4583	2.369	2.431	2.4121	2.4071	2.4189	2.4246	2.4148	2.4316	2.4282	2.5548
$\langle \Phi \rangle$/Ra	2.4333	2.4119	2.4189	2.4165	2.392	2.4182	2.4246	2.4148	2.4276	2.4281	2.31916
δ	1.02%	1.78%	0.50%	0.18%	0.63%	0.03%	$< 0.01\%$	$< 0.01\%$	0.16%	$< 0.01\%$	9.22%

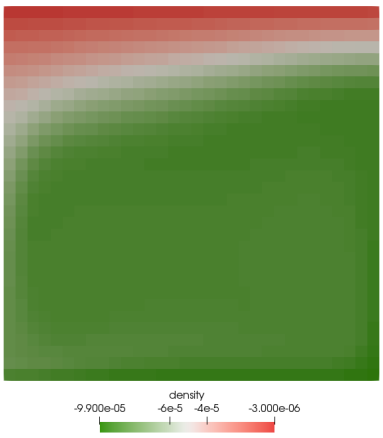


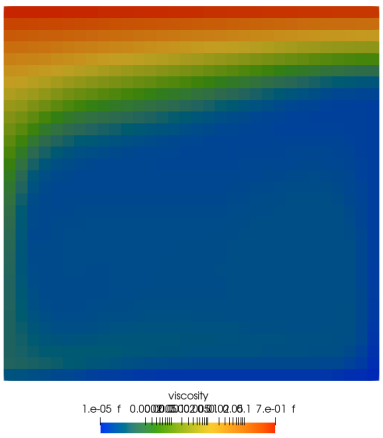

velocity X
-239.722 0 200 400 585.557


velocity Y
-685.512 0 500 887.230


velocity Magnitude
0.000 200 400 600 887.230

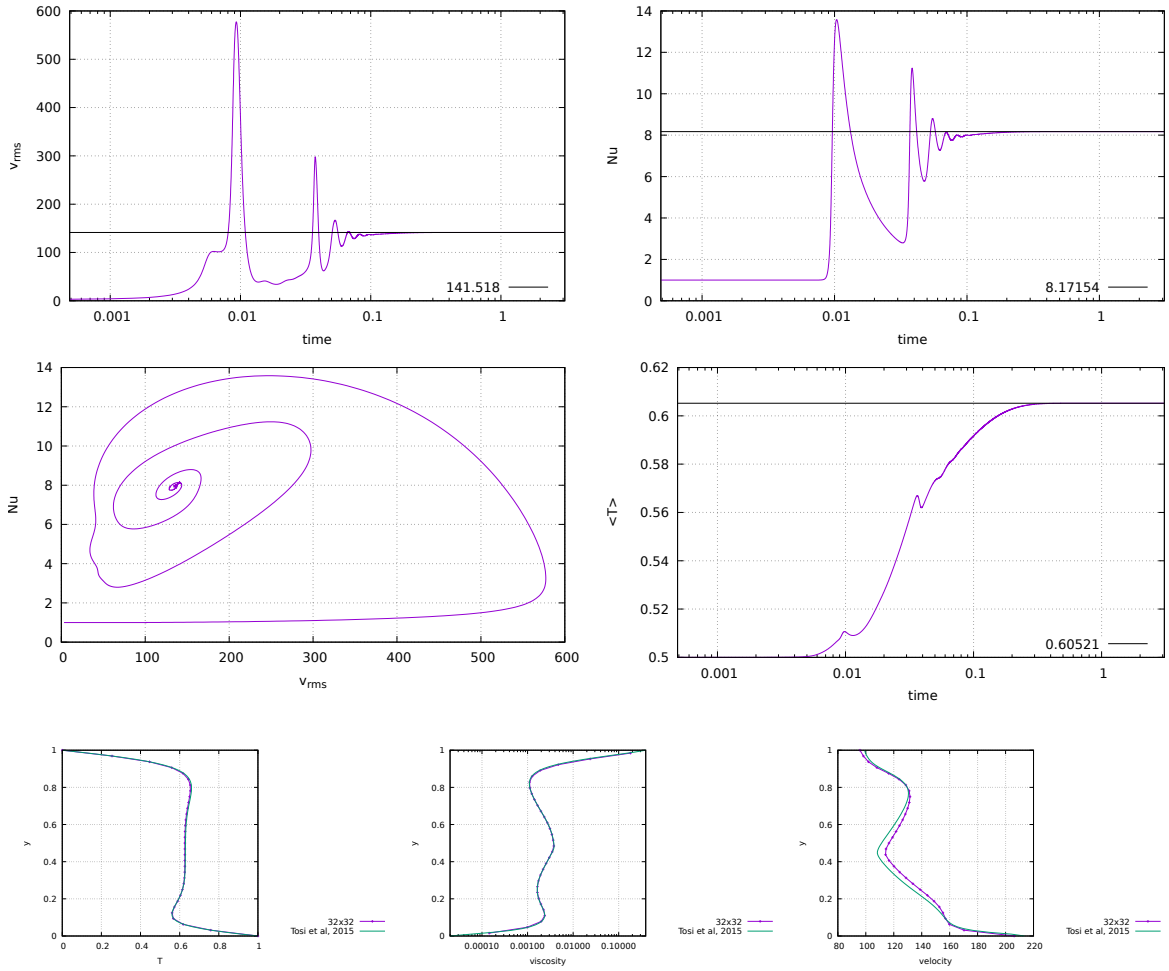

T
0.000 0.2 0.4 0.6 0.8 1.000

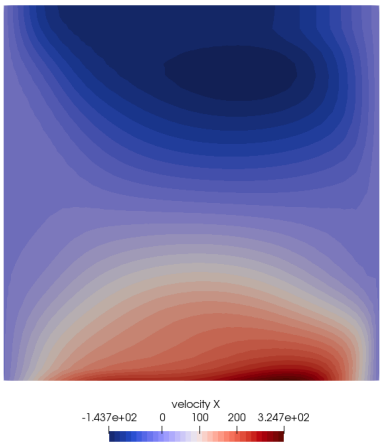

density
-9.900e-05 -6e-5 -4e-5 -3.000e-06

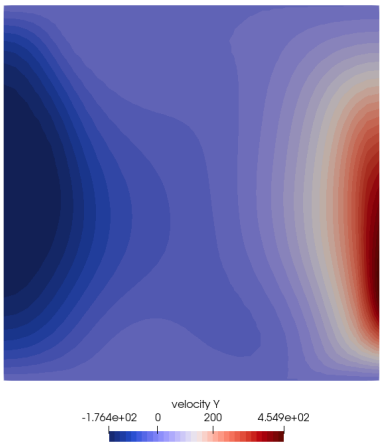

viscosity
1.e-05 f 7.e-01 f

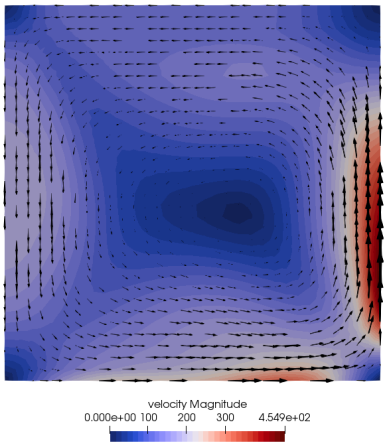
Case 2

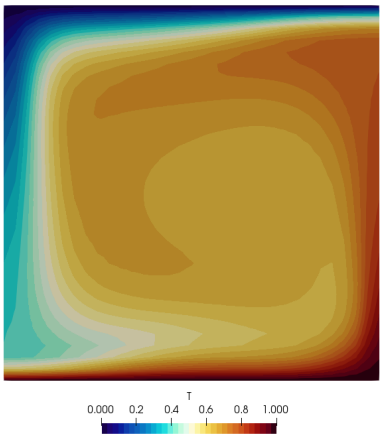
$\langle T \rangle$	0.5289	0.5276	0.5276	0.5289	0.5283	0.527304	0.527521	0.5274	0.5277	0.5278	0.5364
Nu_{top}	6.5572	6.6156	6.6074	6.5913	6.6356	6.61082	6.68224	6.6401	6.5912	6.6249	7.4376
Nu_{bot}	6.5243	6.6158	6.6073	6.5913	6.6356	6.61082	6.66899	6.6326	6.5834	6.6267	7.4376
u_{RMS}	79.6202	79.1358	79.0181	78.6652	79.4334	78.9903	79.0684	79.0318	79.1105	79.1996	98.8912
u_{RMS}^{surf}	75.4814	75.1727	75.0434	74.1719	74.8587	75.0606	75.0975	75.0827	74.7596	75.1903	93.5057
u_{max}^{surf}	89.2940	88.9715	88.8130	87.6118	89.1444	88.823	88.8753	88.85	88.9146	88.9848	123.046
η_{min}	1.9174×10^{-4}	1.9220×10^{-4}	1.9448×10^{-4}	1.9204×10^{-4}	1.9900×10^{-4}	1.9574×10^{-4}	1.9167×10^{-4}	1.9200×10^{-4}	1.9860×10^{-4}	1.9178×10^{-4}	1.93×10^{-4}
η_{max}	1.6773	1.9834	1.6508	1.9670	1.1800	1.6665	1.7446	1.8891	1.5200	1.8831	5.67×10^{-3}
$\langle W \rangle$	5.6512	5.6251	5.6076	5.5903	5.629	5.61012	5.61425	5.6136	5.6216	5.6235	6.751
$\langle \Phi \rangle$/Ra	5.6463	5.6174	5.6024	5.5434	5.6325	5.60152	5.61425	5.6136	5.6182	5.6235	7.786
δ	0.09%	0.14%	0.09%	0.84%	0.06%	0.15%	<0.01%	<0.01%	0.06%	<0.01%	13.29%

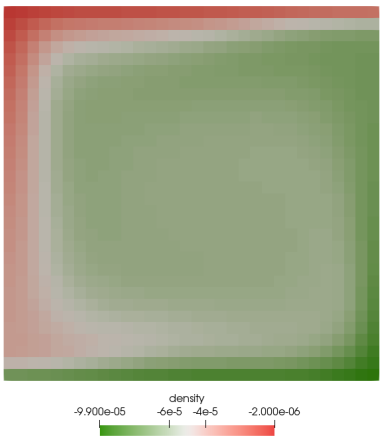


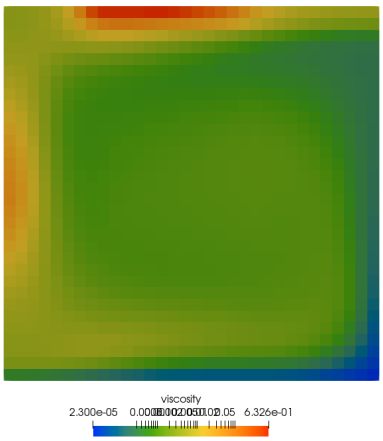

velocity X
-1.437e+02 0 100 200 3.247e+02


velocity Y
-1.764e+02 0 200 4.549e+02


velocity Magnitude
0.000e+00 100 200 300 4.549e+02

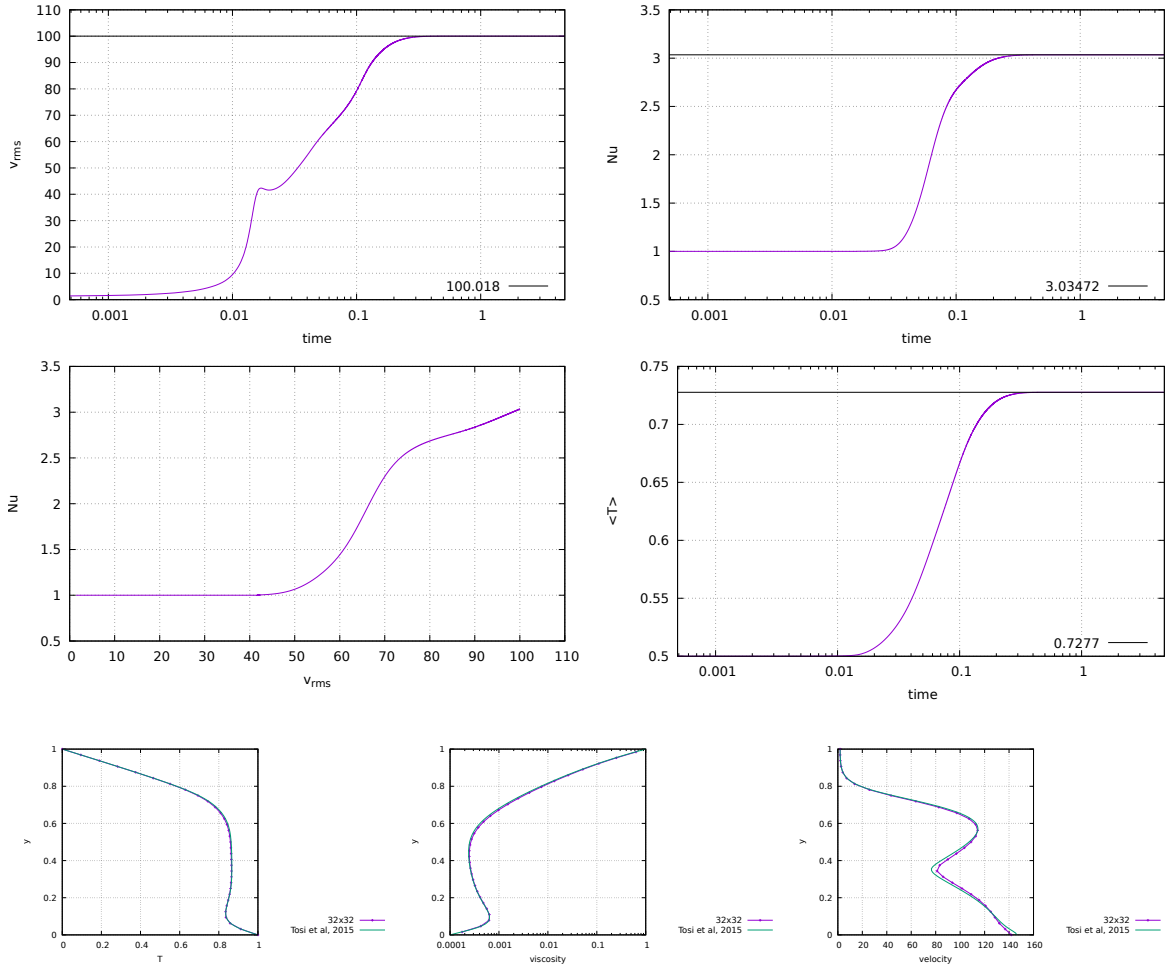

T
0.000 0.2 0.4 0.6 0.8 1.000

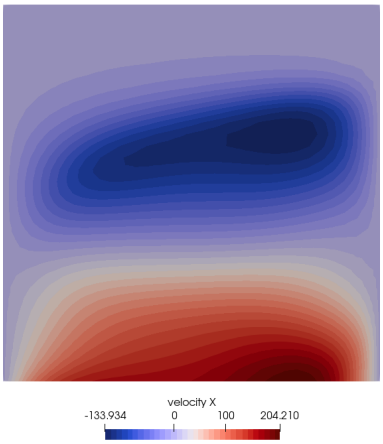

density
-9.900e-05 -6e-5 -4e-5 -2.000e-06

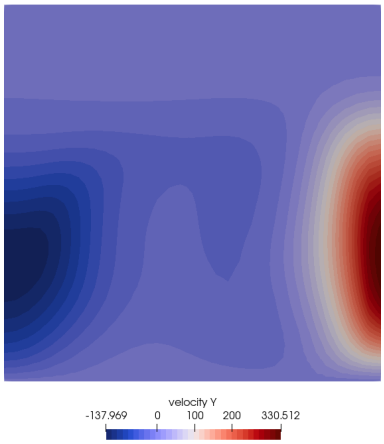

viscosity
2.300e-05 6.326e-01

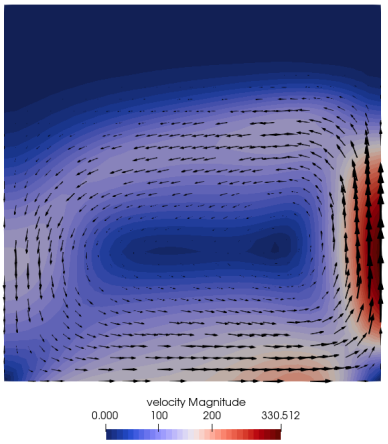
Case 3

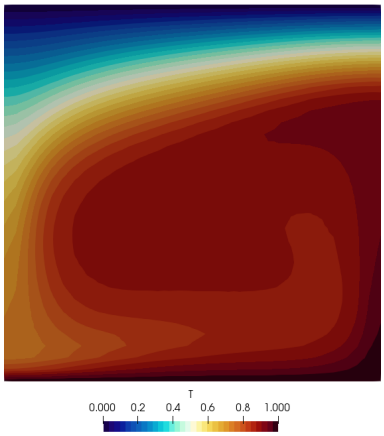
$\langle T \rangle$	0.7286	0.7275	0.7271	0.7272	0.7241	0.7274	0.727464	0.7275	0.7275	0.7278	0.7305
Nu_{top}	3.0374	3.0298	3.0324	3.0314	3.0253	3.03025	3.0918	3.0399	3.0347	3.0371	2.9311
Nu_{bot}	2.9628	3.0298	3.0323	3.0314	3.0253	3.03025	3.03487	3.0376	2.9908	3.0410	2.9311
u_{RMS}	100.9467	100.024	99.8701	99.9917	100.197	100.018	100.127	100.0396	100.1208	100.3368	111.6121
u^{surf}_{RMS}	2.0374	2.0785	2.0916	2.0835	2.0789	2.07299	2.07301	2.0569	2.0652	2.0727	1.356
u^{surf}_{max}	2.8458	2.9029	2.9201	2.9094	2.9207	2.89495	2.89501	2.873	2.9019	2.8946	1.8806
η_{min}	4.7907×10^{-5}	4.8140×10^{-5}	4.8014×10^{-5}	4.8047×10^{-5}	4.8400×10^{-5}	4.7951×10^{-5}	4.8081×10^{-5}	4.8000×10^{-5}	4.8080×10^{-5}	4.7972×10^{-5}	10^{-4}
η_{max}	1	0.9987	0.9857	0.9988	0.9010	0.9637	0.9999	1	0.9023	1	1
$\langle W \rangle$	2.0400	2.0028	2.0340	2.0269	2.0235	2.03002	2.03482	2.0298	2.0384	2.0362	2.056
$\langle \Phi \rangle$/Ra	2.0335	2.0277	2.0304	2.0286	2.0164	2.0302	2.03482	2.0298	2.037	2.0362	1.865
δ	0.32%	1.23%	0.18%	0.08%	0.35%	0.01%	<0.01%	<0.01%	0.07%	<0.01%	9.29%

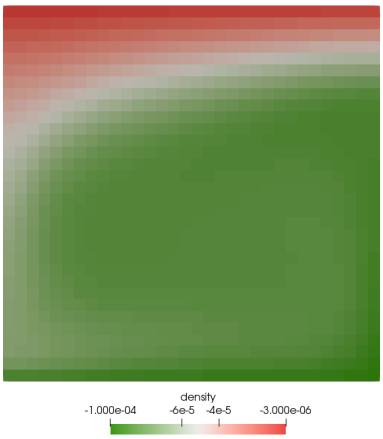


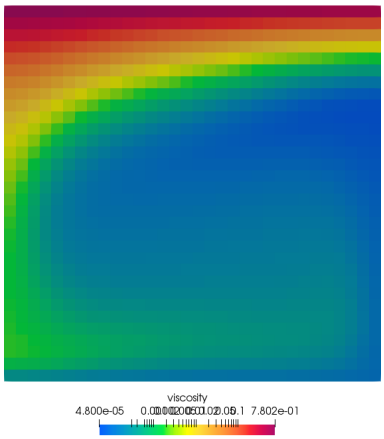

velocity X
-133.934
0
100
204.210


velocity Y
-137.969
0
100
200
330.512


velocity Magnitude
0.000
100
200
330.512

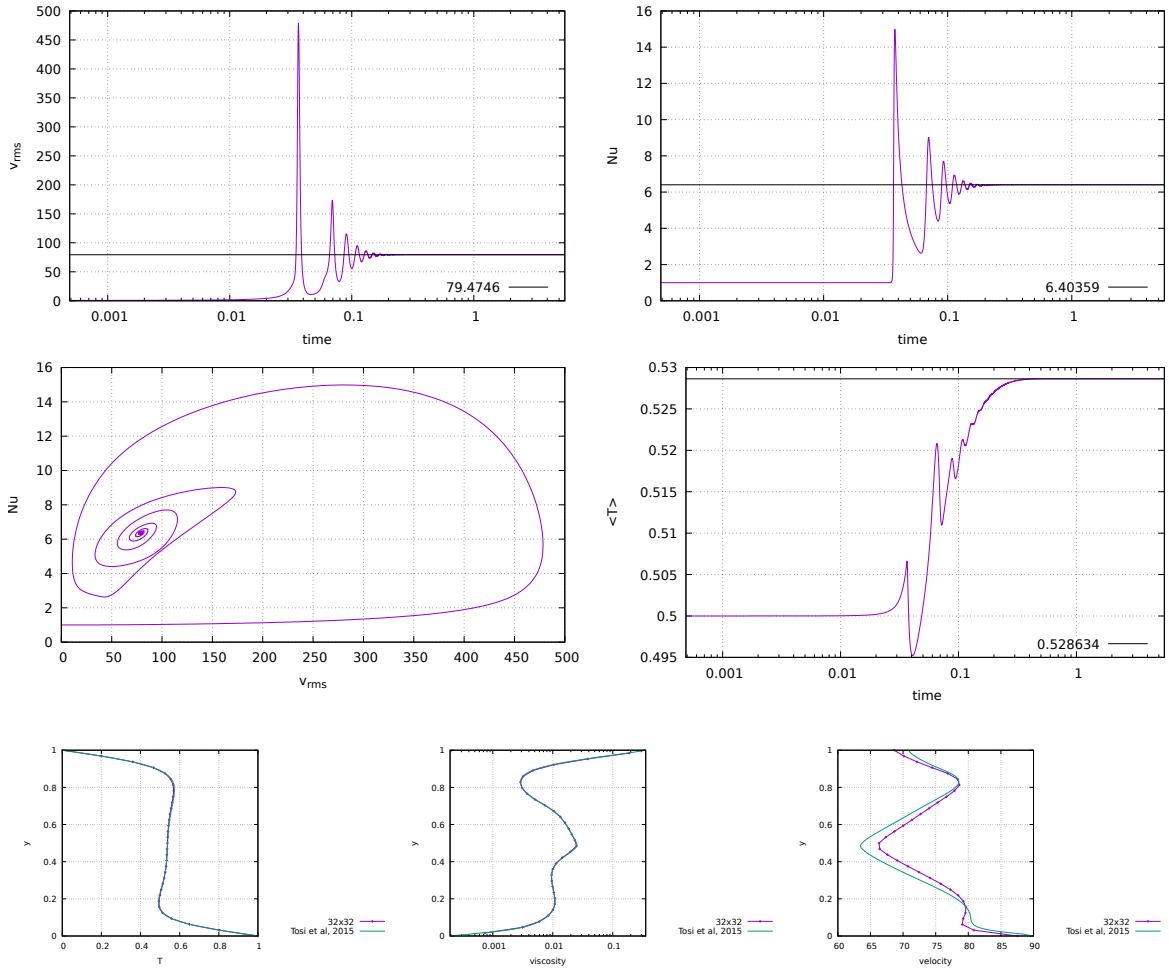

T
0.000
0.2
0.4
0.6
0.8
1.000

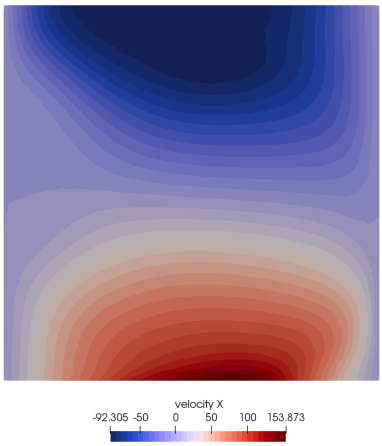

density
-1.000e-04
-6e-5
-4e-5
-3.000e-06

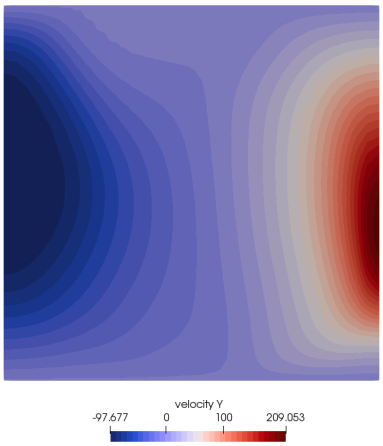

viscosity
4.800e-05
7.802e-01

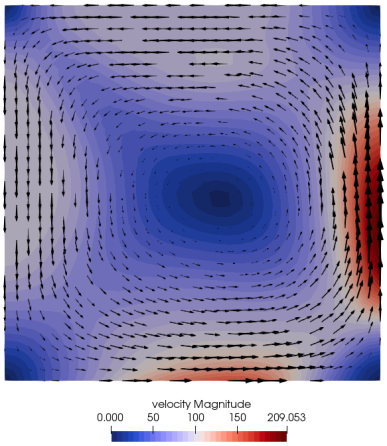
Case 4

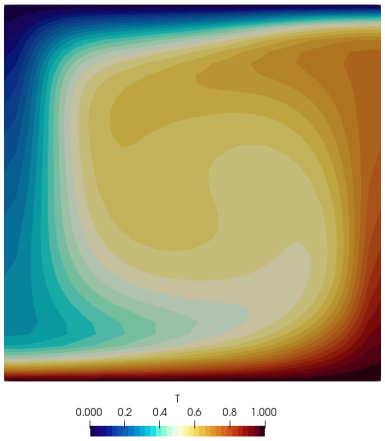
$\langle T \rangle$	0.5289	0.5276	0.5276	0.5289	0.5283	0.527304	0.527521	0.5274	0.5277	0.5278	0.5364
Nu_{top}	6.5572	6.6156	6.6074	6.5913	6.6356	6.61082	6.68224	6.6401	6.5912	6.6249	7.4376
Nu_{bot}	6.5243	6.6158	6.6073	6.5913	6.6356	6.61082	6.66899	6.6326	6.5834	6.6267	7.4376
u_{RMS}	79.6202	79.1358	79.0181	78.6652	79.4334	78.9903	79.0684	79.0318	79.1105	79.1996	98.8912
u_{RMS}^{surf}	75.4814	75.1727	75.0434	74.1719	74.8587	75.0606	75.0975	75.0827	74.7596	75.1903	93.5057
u_{max}^{surf}	89.2940	88.9715	88.8130	87.6118	89.1444	88.823	88.8753	88.85	88.9146	88.9848	123.046
η_{min}	1.9174×10^{-4}	1.9220×10^{-4}	1.9448×10^{-4}	1.9204×10^{-4}	1.9900×10^{-4}	1.9574×10^{-4}	1.9167×10^{-4}	1.9200×10^{-4}	1.9860×10^{-4}	1.9178×10^{-4}	1.93×10^{-4}
η_{max}	1.6773	1.9834	1.6508	1.9670	1.1800	1.6665	1.7446	1.8891	1.5200	1.8831	5.67×10^{-3}
$\langle W \rangle$	5.6512	5.6251	5.6076	5.5903	5.629	5.61012	5.61425	5.6136	5.6216	5.6235	6.751
$\langle \Phi \rangle$/Ra	5.6463	5.6174	5.6024	5.5434	5.6325	5.60152	5.61425	5.6136	5.6182	5.6235	7.786
δ	0.09%	0.14%	0.09%	0.84%	0.06%	0.15%	<0.01%	<0.01%	0.06%	<0.01%	13.29%

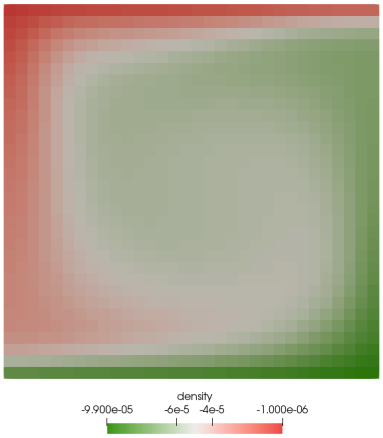


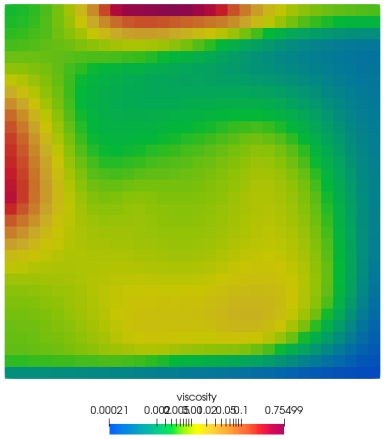

velocity X
-92.305 -50 0 50 100 153.873


velocity Y
-97.677 0 100 209.053

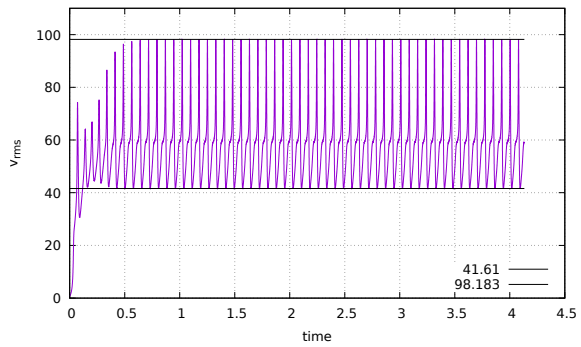

velocity Magnitude
0.000 50 100 150 209.053

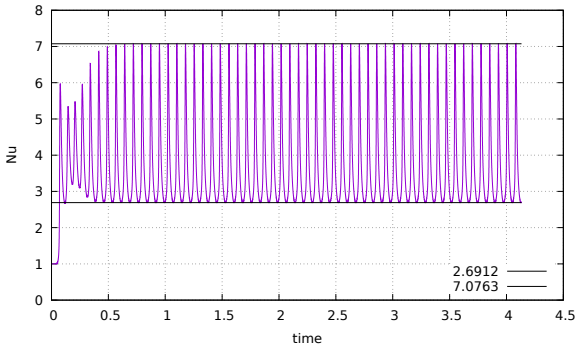

T
0.000 0.2 0.4 0.6 0.8 1.000

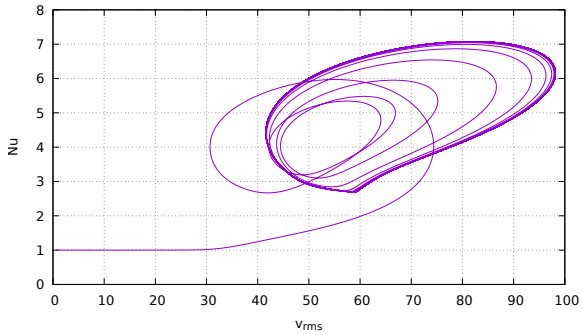

density
-9.900e-05 -6e-5 -4e-5 -1.000e-06

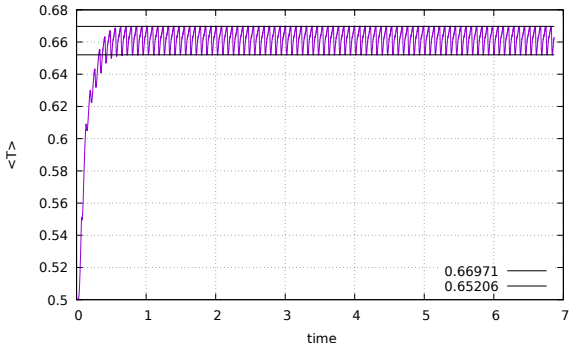

viscosity
0.00021 0.75499

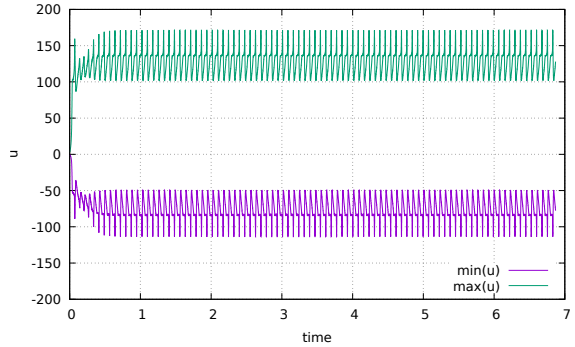
Case 5a - Periodic Solutions

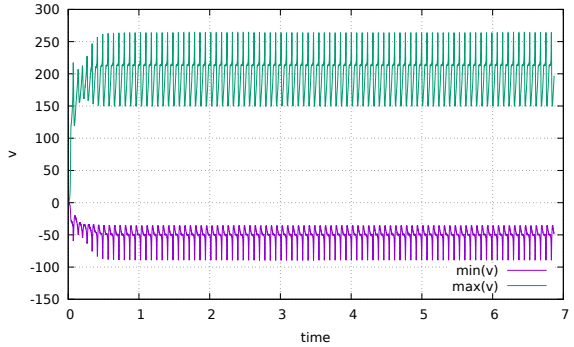


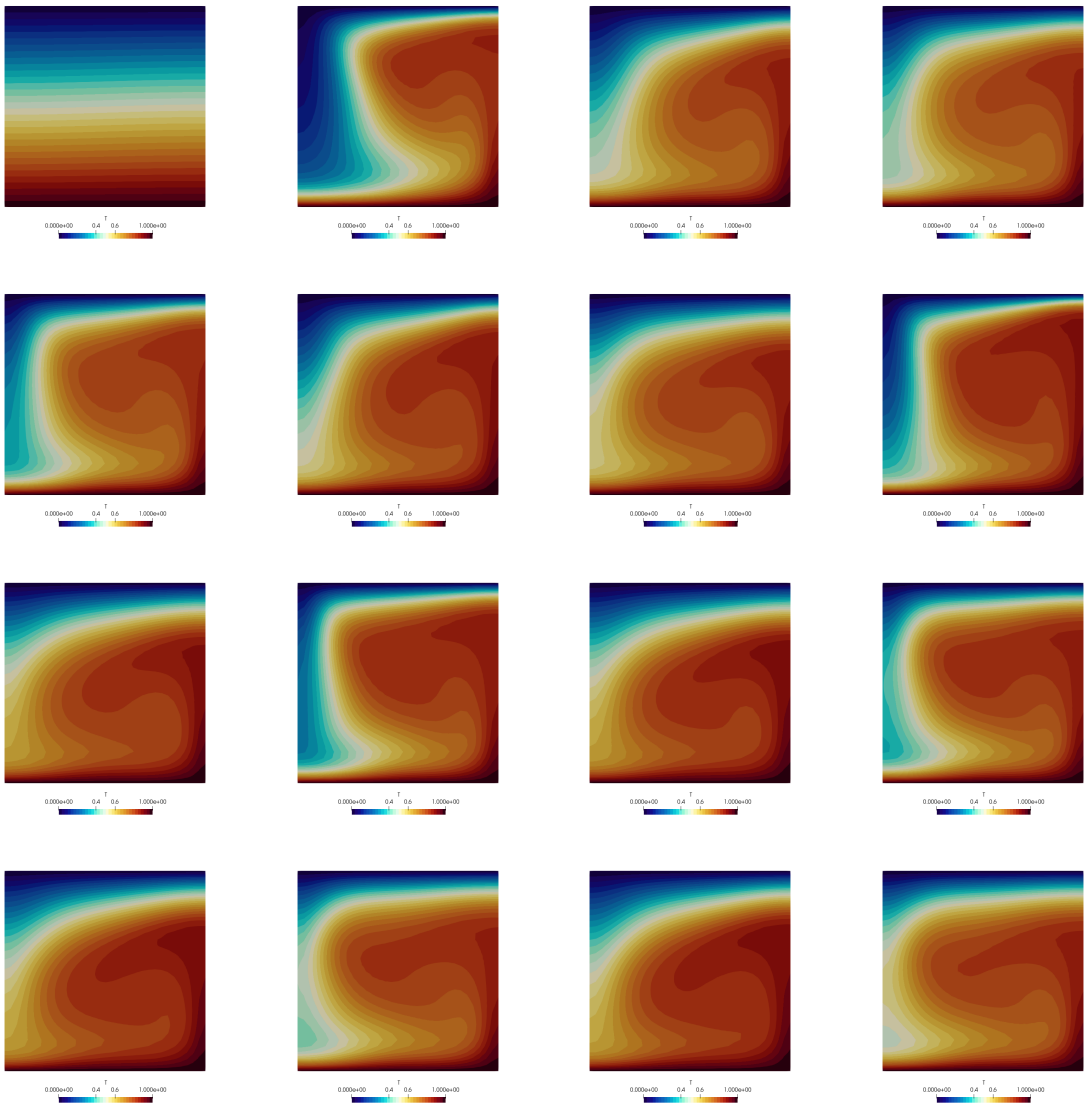












Temperature field evolution

Case 5b - Bifurcation Analysis

Case 5b is an extension of Case 5a. Here we varied the yield stress from $\sigma_Y = 3$ to 5 with the goal of identifying the critical values at which the system transitions from a steady, mobile lid regime (for low values of σ_Y), to a periodic regime first (for intermediate values of σ_Y), and to a steady, stagnant lid regime afterward (for high values of σ_Y).

Carry out all these runs and plot against paper data

Stone 29

open boundary conditions ($Q_1 \times P_0$)

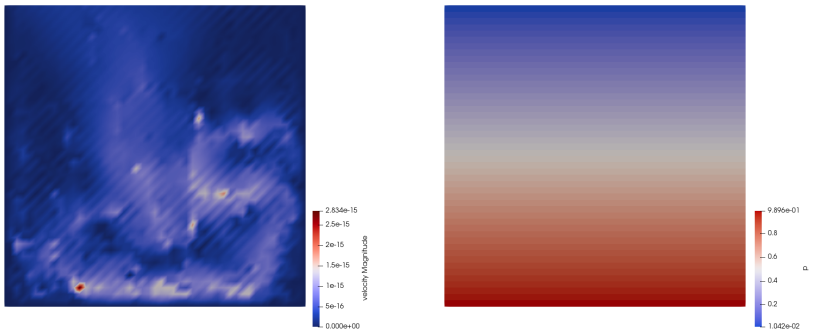
2D
box
Q1P0
sparse_matrix
numerical_benchmark
stokes_equation
viscous_rheology
buoyancy_driven
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_29`

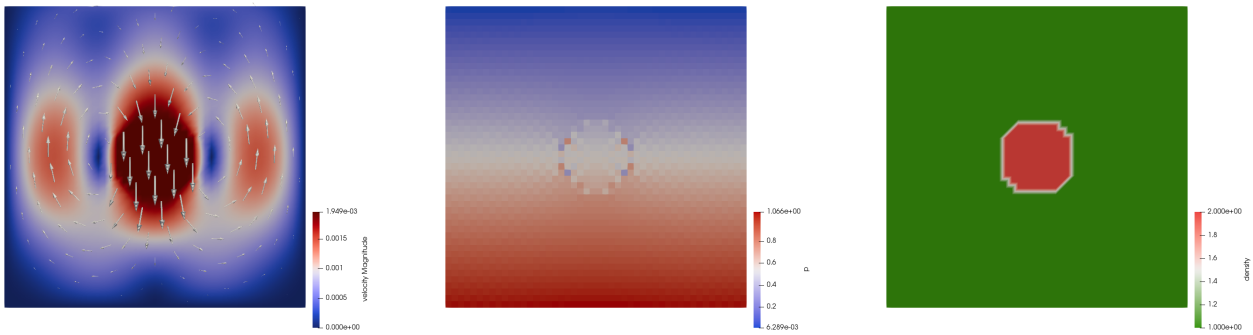
Relevant Literature[67]

In what follows we will investigate the use of the so-called open boundary conditions in the very simple context of a 2D Stokes sphere experiment.

We start with a domain without the sphere. Essentially, it is what people would call an aquarium. Free slip boundary conditions are prescribed on the sides and no-slip conditions at the bottom. The top surface is left free. The fluid has a density $\rho_0 = 1$ and a viscosity $\eta_0 = 1$. In the absence of any density difference in the domain there is no active buoyancy force so that we expect a zero velocity field and a lithostatic pressure field. This is indeed what we recover:



If we now implement a sphere parametrised by its density $\rho_s = \rho_0 + 1$, its viscosity $\eta_s = 10^3$ and its radius $R_s = 0.123$ in the middle of the domain, we recover a velocity field which logically shows the sphere falling downward and a symmetric return flow of the fluid on each side:

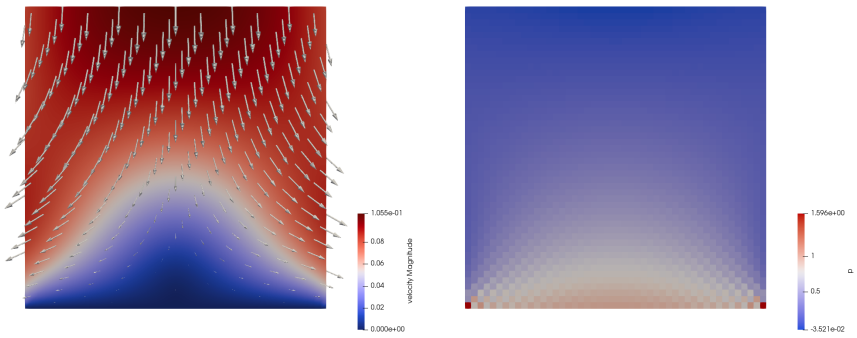


Unfortunately it has been documented that the presence of free-slip boundary conditions affects the evolution of subduction [67], even when these are placed rather far from the subduction zone. A proposed solution to this problem is the use of 'open boundary conditions' which are in fact stress boundary conditions. The main idea is to prescribe a stress on the lateral boundaries (instead of free slip) so that it balances out exactly the existing lithostatic pressure inside the domain along the side walls. Only pressure deviations with respect to the lithostatic are responsible for flow and such boundary conditions allow flow across the boundaries.

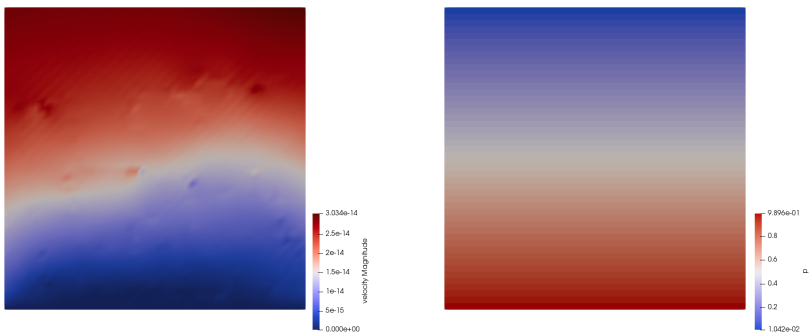
We therefore need the lithostatic pressure and compute it before hand (which is trivial in our case but can prove to be a bit more tedious in real life situations when for instance density varies in the domain as a function of temperature and/or pressure - see STONE 119).

```
plith = np.zeros(nnp,dtype=np.float64)
for i in range(0,nnp):
    plith[i]=(Ly-y[i])*rho0*abs(gy)
```

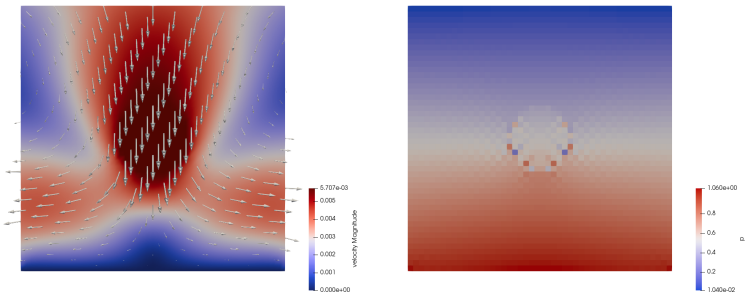
Let us start with a somewhat pathological case: even in the absence of the sphere, what happens when no boundary conditions are prescribed on the sides? The answer is simple: think about an aquarium without side walls, or a broken dam. The velocity field indeed shows a complete collapse of the fluid left and right of the bottom.



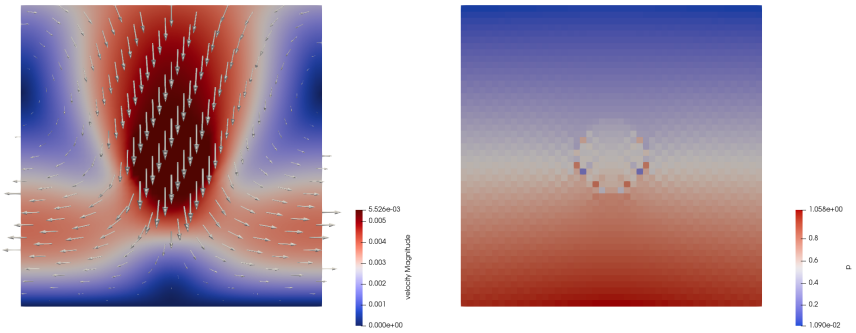
Let us then continue (still with no sphere) but let us now switch on the open boundary conditions. Since the side boundary conditions match the lithostatic pressure we expect no flow at all in the absence of any density perturbation in the system. This is indeed what is recovered:



Finally, let us reintroduce the sphere. This time flow is allowed through the left and right side boundaries:



Finally, although horizontal velocity Dirichlet boundary conditions and open boundary conditions are not compatible, the same is not true for the vertical component of the velocity: the open b.c. implementation acts on the horizontal velocity dofs only, so that one can fix the vertical component to zero, as is shown hereunder:



We indeed see that the in/outflow on the sides is perpendicular to the boundaries.

Turning now to the actual implementation, we see that it is quite trivial, since all element edges are vertical, and all have the same vertical dimension h_x. Since we use a Q_0 approximation for the pressure we need to prescribe a single pressure value in the middle of the element. Finally because of the sign of the normal vector projection onto the x-axis, we obtain:

```
if open_bc_left and x[icon[0,iel]]<eps: # left side
    pmid=0.5*(plith[icon[0,iel]]+plith[icon[3,iel]])
    f_el[0]+=0.5*hy*pmid
    f_el[6]+=0.5*hy*pmid
if open_bc_right and x[icon[1,iel]]>Lx-eps: # right side
    pmid=0.5*(plith[icon[1,iel]]+plith[icon[2,iel]])
    f_el[2]-=0.5*hy*pmid
    f_el[4]-=0.5*hy*pmid
```

These few lines of code are added after the elemental matrices and rhs are built, and before the application of other Dirichlet boundary conditions, and assembly.

Open boundary conditions are discussed in Section 10.15.

To Do, open questions, future work?

- For aquarium, look at convergence rate

Stone 30

conservative velocity interpolation



2D
box
analytical_benchmark
time_stepping
paraview_output
PIC

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_30`

In this STONE the Stokes equations are not solved and analytical velocity fields are used to advect particles. The CVI algorithm as introduced in Wang *et al.* (2015) [344] is implemented. Detailed derivations of the corrections and a thorough discussion of Wang *et al.* (2015) are presented in Section 14.2. Q_1 and Q_2 basis functions are used and in both cases the CVI algorithm can be toggled on/off. Particles can be distributed regularly or randomly at startup.

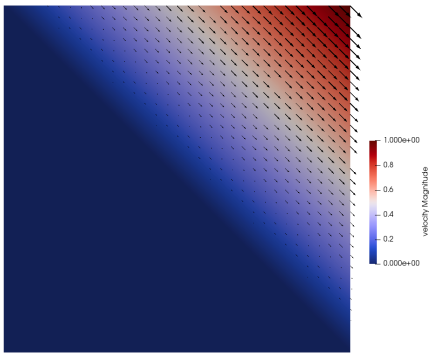
Five velocity fields can be prescribed on the mesh:

- the so-called Couette flow of Wang *et al.* (2015) [344]
- the SolCx solution (see Section 19, and other stones too)
- a flow created by means of a stream line function (see STONE 32)
- a convection cell
- Donea & Huerta manufactured solution, see Section 18.2

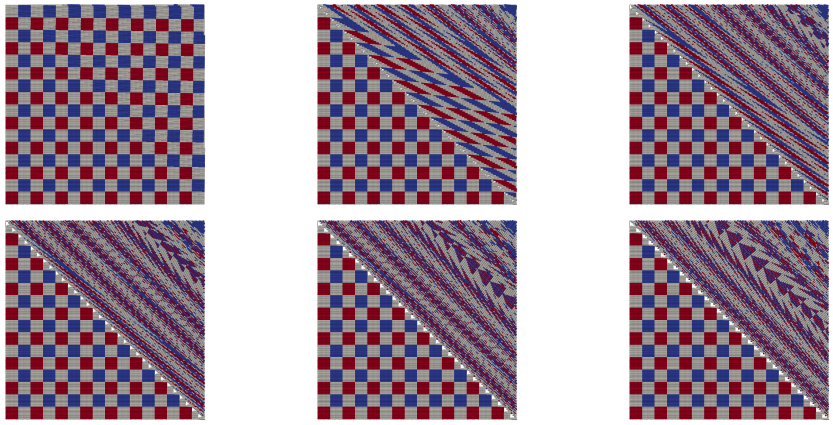
The array count is nel long and contains the number of markers present in each element. It is updated at every time step. Its min and max values are exported to an ascii file, as well as its standard deviation, computed with the std function of numpy defined as: "The standard deviation is the square root of the average of the squared deviations from the mean, i.e., std = sqrt(mean(x)), where x = abs(a - a.mean())**2."

Various Runge-Kutta algorithms (in space) are implemented, see Section 14.10 and are controlled with the RKorder parameter.

Couette flow



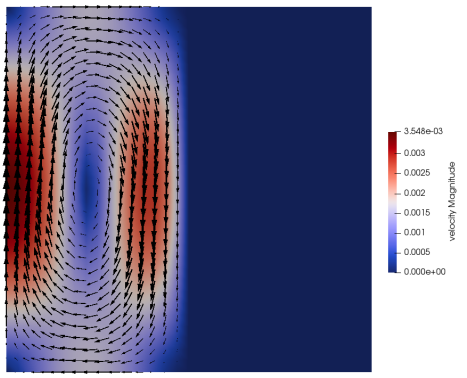
Velocity field - 32×32 Q_1 elements.



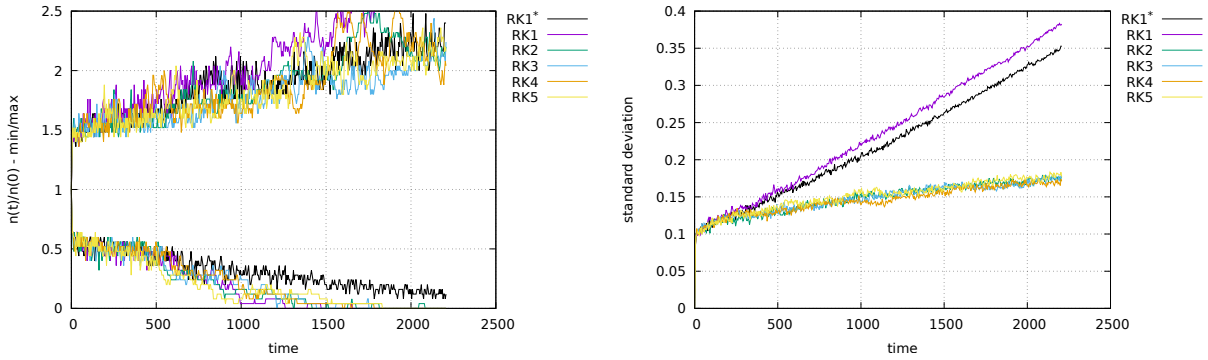
Due to the periodic boundary conditions, it only works with RKorder=1.

SolCx

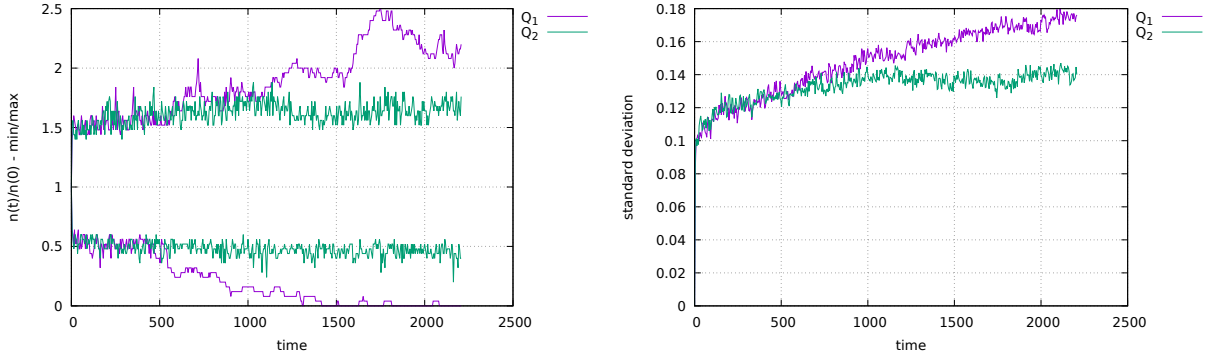
Reference model: 32×32 resolution, CFL nb=0.5, 5^2 particles randomly distributed inside each element, RKorder=2, Q_1 interpolation, no CVI. We look at the min/max number of particles per element as a function of time (normalised by the initial number). Total of 500 time steps.



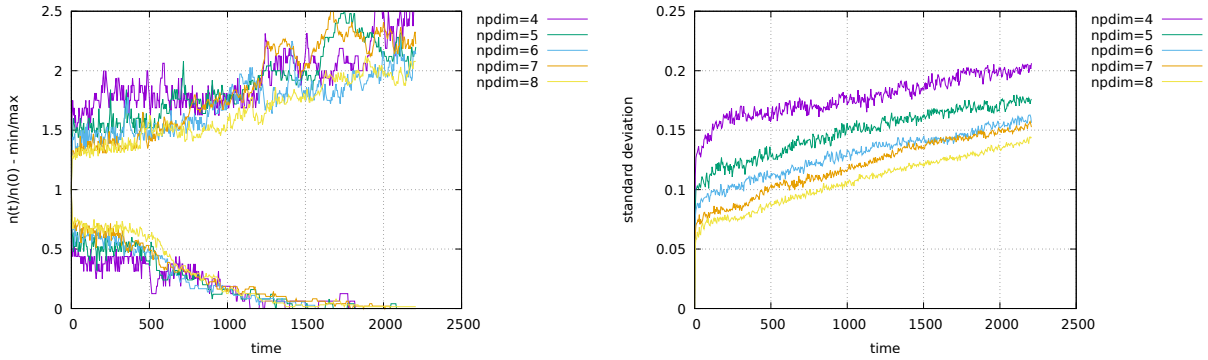
Velocity field - 32×32 Q_1 elements.



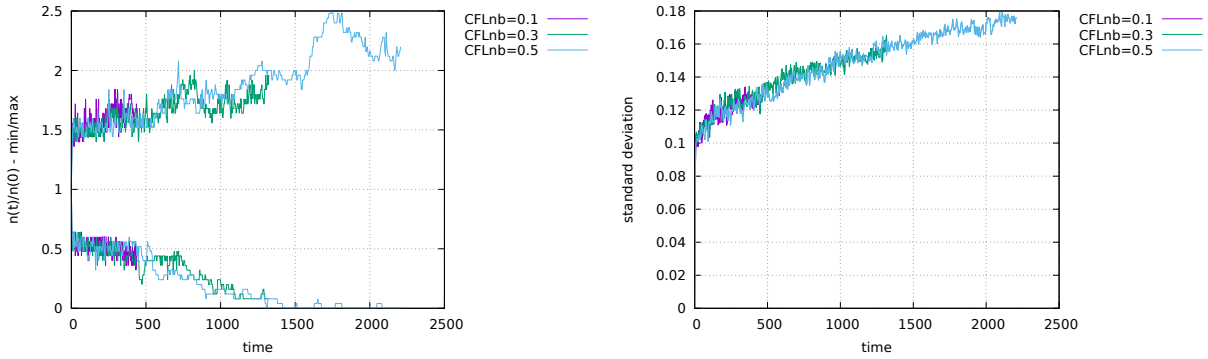
Ref model + varying RKorder (the star means that velocity is directly computed on the particle, no interpolation).



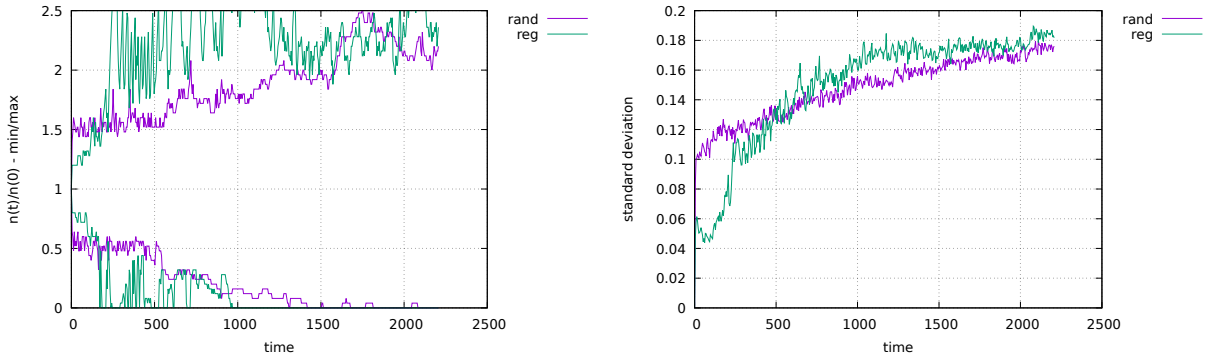
Ref model + Q_1 vs Q_2.



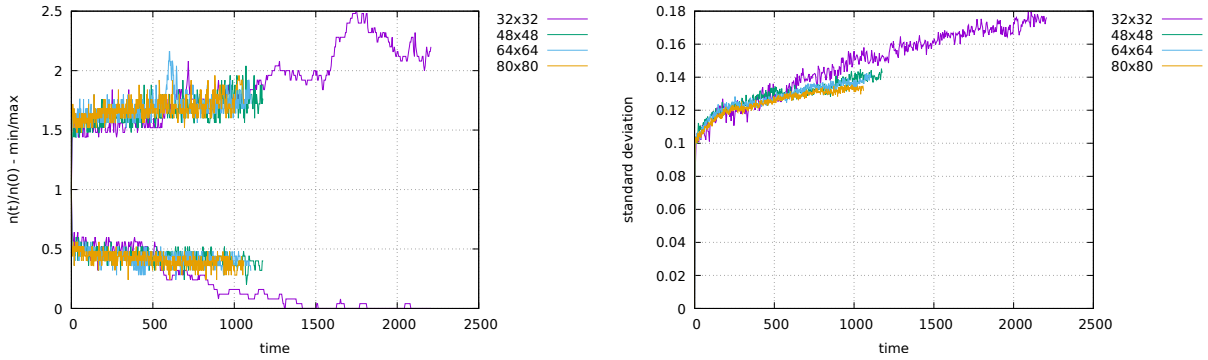
Ref model + initial nparticle per element between 4 and 8.



Ref model + CFL nb

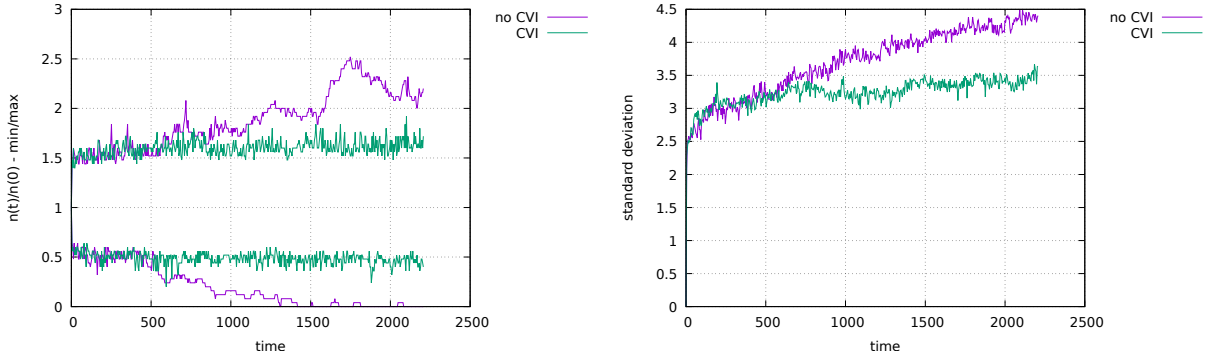


Ref model + random vs regular.



Ref model + different resolutions.

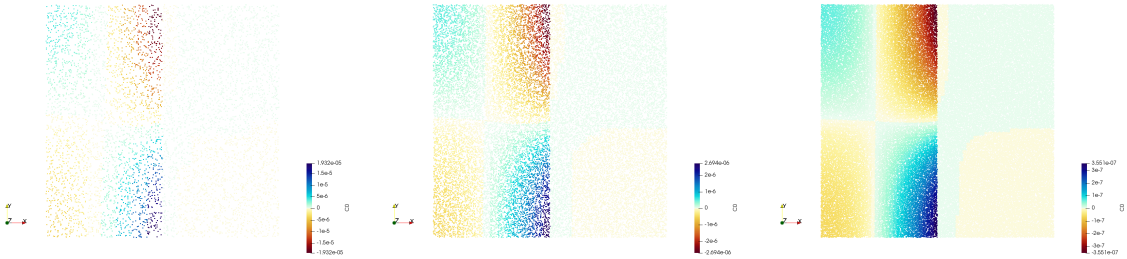
Conclusion: nothing really helps, it is a matter of time before an element is fully depleted of particles. A regular distribution of markers makes things worse. Only using Q_2 interpolation seems to really do something. Using a small CFL number will improve things, but at the expense of the calculation walltime.



Ref model + influence of CVI.

We find that CVI makes a substantial difference.

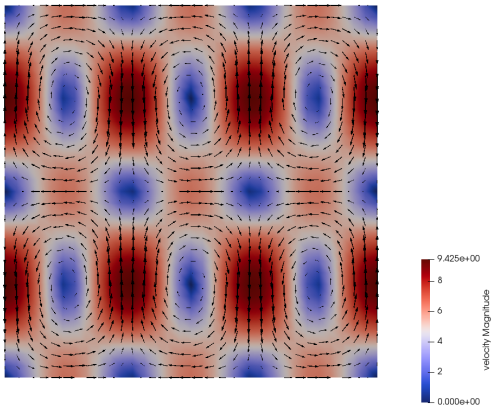
Coming back to the observations made in Section 14.2, we here show the C_0 quantity for three resolutions:



From left to right: 16x16, 32x32, 64x64.

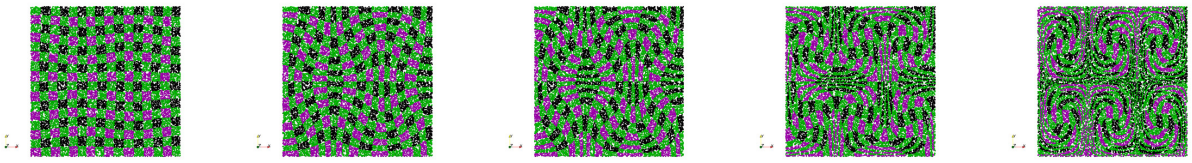
We find that it becomes smaller and smaller, most likely in $\mathcal{O}(h)$.

Streamline flow

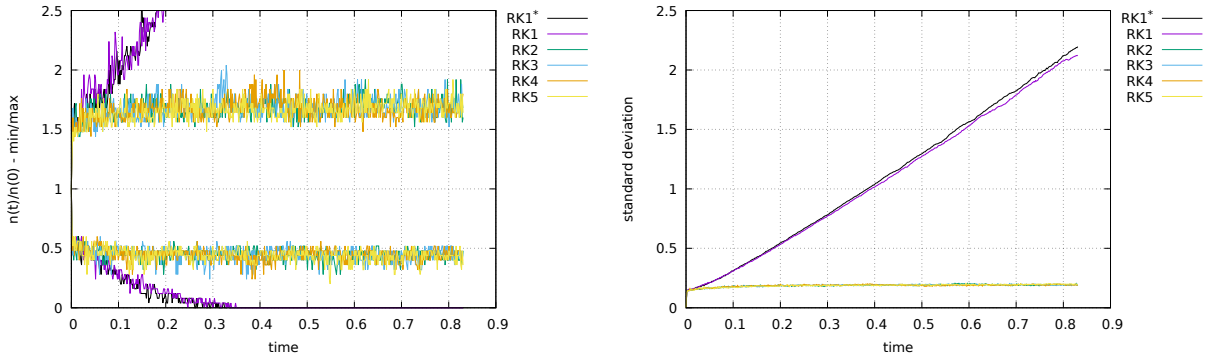


32x32 Q_1 mesh, 3x2 cells

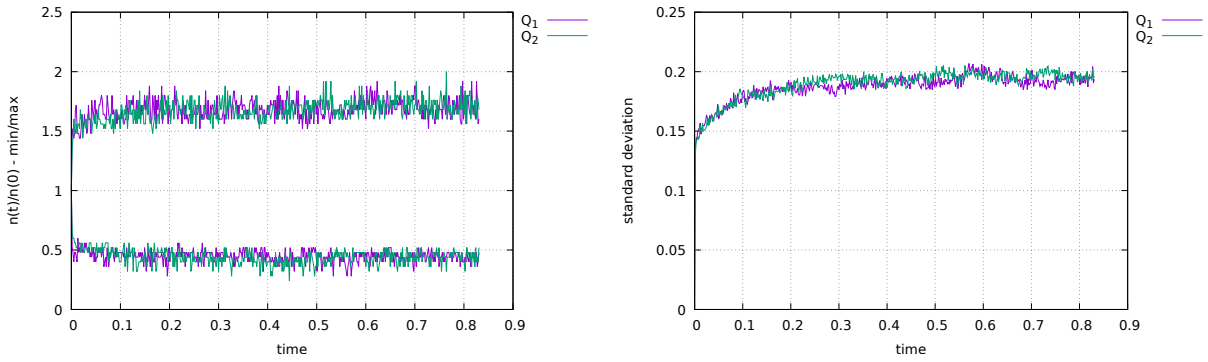
The reference model is as follows: 32x32 Q_1 elements, initial 5^2 particles per element placed at random, CFL number of 0.5, RK order of 2, no CVI. The `count` array of length `nel` is used to keep track of the number of particles per element. Its min/max values over time are recorded. The standard deviation of the `count` array is also computed at each time step, normalised by the initial number of particles in the element at $t = 0$ and plotted as a function of time.



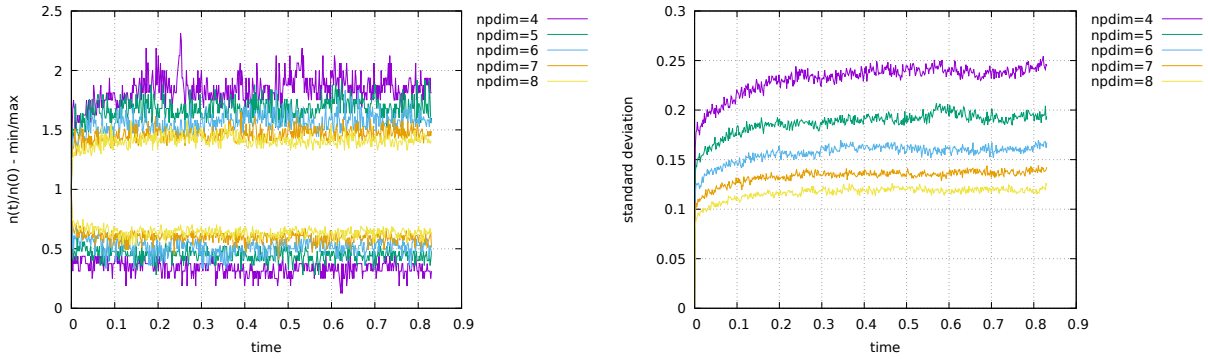
evolution of the swarm of markers.



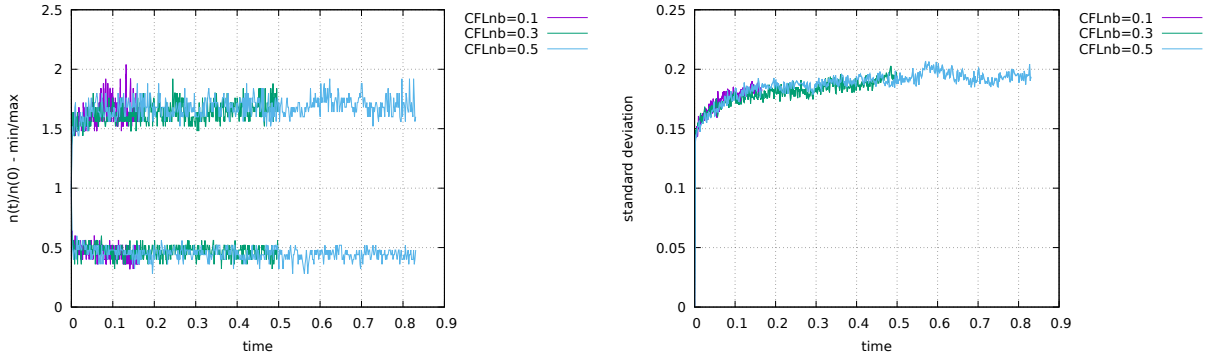
Ref model + varying RKorder (the star means that velocity is directly computed on the particle, no interpolation).



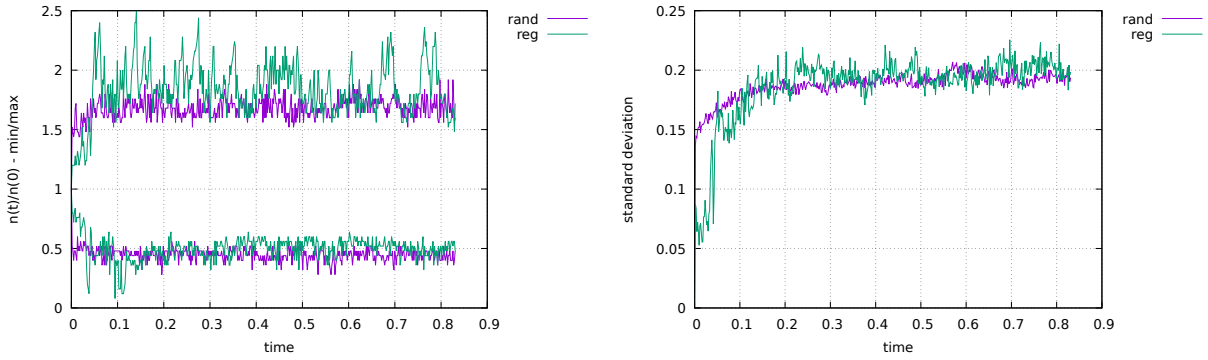
Ref model + Q_1 vs Q_2.



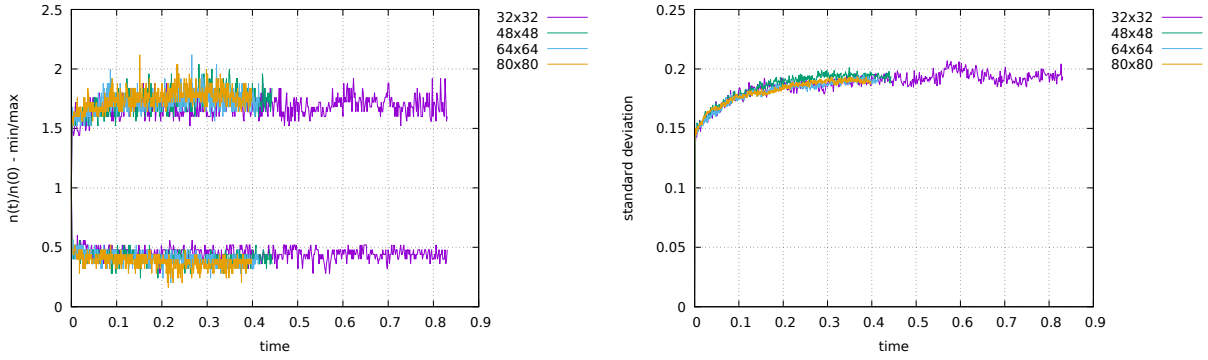
Ref model + initial nparticle per element between 4 and 8.



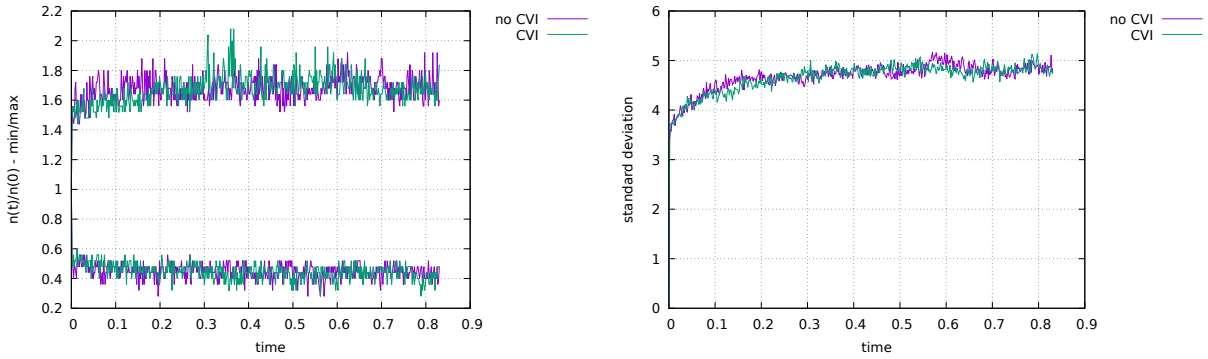
Ref model + CFL nb



Ref model + random vs regular.

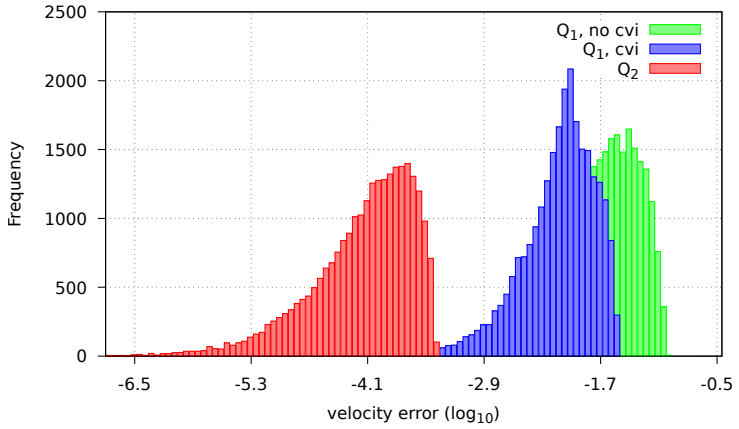


Ref model + different resolutions.



Ref model + influence of CVI.

In order to obtain the following figure no timestep is taken. The velocity is interpolated on the initial position of the markers (so is the CVI correction) and exported to a file along with the analytical value. The error for all markers is then presented in a histogram:



Distribution of velocity errors.

We see that using CVI improves the accuracy of the velocity field error, but is no match for the results obtained with quadratic elements.

Convection in square domain

In what follows I assume that the domain is the $[-1,1]\times[-1,1]$ square. I postulate

$$u(x,y) = (a_0+a_1x+a_2x^2+a_3x^3+a_4x^4)(b_0+b_1y+b_2y^2+b_3y^3) \quad (30.1)$$
$$v(x,y) = (c_0+c_1y+c_2y^2+c_3y^3+c_4y^4)(d_0+d_1x+d_2x^2+d_3x^3) \quad (30.2)$$

I wish to enforce no flux through the boundaries so I need $u(\pm1,y)=0$ and $v(x,\pm1)=0$, so I choose

$$u(x,y) = (2-x^2-x^4)(b_0+b_1y+b_2y^2+b_3y^3) \quad (30.3)$$
$$v(x,y) = (2-y^2-y^4)(d_0+d_1x+d_2x^2+d_3x^3) \quad (30.4)$$

The velocity divergence is then

$$\vec{\nabla}\cdot\vec{\mathrm{v}} = -2(x+2x^3)(b_0+b_1y+b_2y^2+b_3y^3) - 2(y+2y^3)(d_0+d_1x+d_2x^2+d_3x^3) \quad (30.5)$$

which yields:

$$b_1+d_1 = 0 \quad (30.6)$$
$$b_3+2d_1 = 0 \quad (30.7)$$
$$2b_1+d_3 = 0 \quad (30.8)$$
$$2b_3+2d_3 = 0 \quad (30.9)$$

4 equations for 4 unknowns but we can only express three parameters as a function of d_1:

$$b_1=-d_1 \qquad b_3=-2d_1 \qquad d_3=2d_1$$

I choose $d_1=1$ so finally

$$u(x,y) = (2-x^2-x^4)(-y-2y^3) \quad (30.10)$$
$$v(x,y) = (2-y^2-y^4)(x+2x^3) \quad (30.11)$$

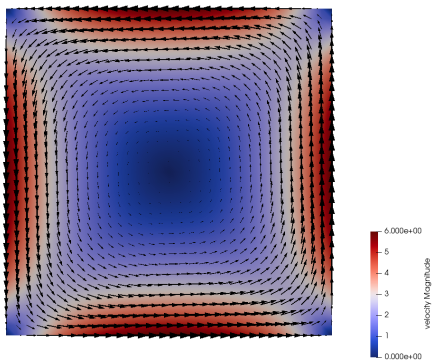
Let us quickly verify:

$$\vec{\nabla}\cdot\vec{\mathrm{v}} = (-2x-4x^3)(-y-2y^3)+(-2y-4y^3)(x+2x^3) = 2(-x-2x^3)(-y-2y^3)+2(-y-2y^3)(x+2x^3) = 0$$

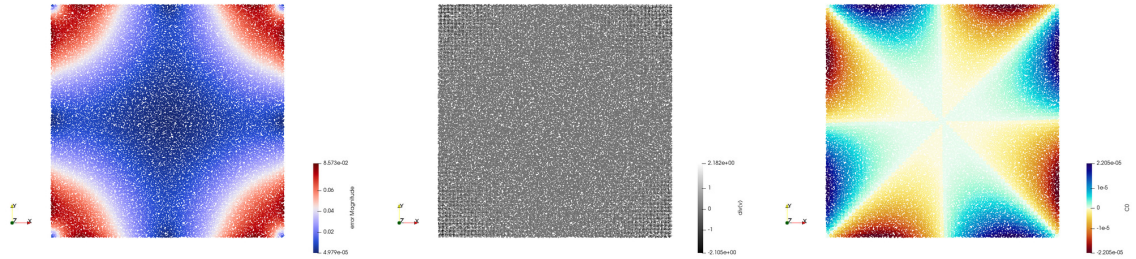
The computational domain remains the unit square, so we need to carry out a change of variables and the analytical velocity field inside the unit square is then

$$u(x,y) = (2-(2x-1)^2-(2x-1)^4)(-(2y-1)-2(2y-1)^3) \quad (30.12)$$
$$v(x,y) = (2-(2y-1)^2-(2y-1)^4)((2x-1)+2(2x-1)^3) \quad (30.13)$$

This velocity field is shown hereunder:

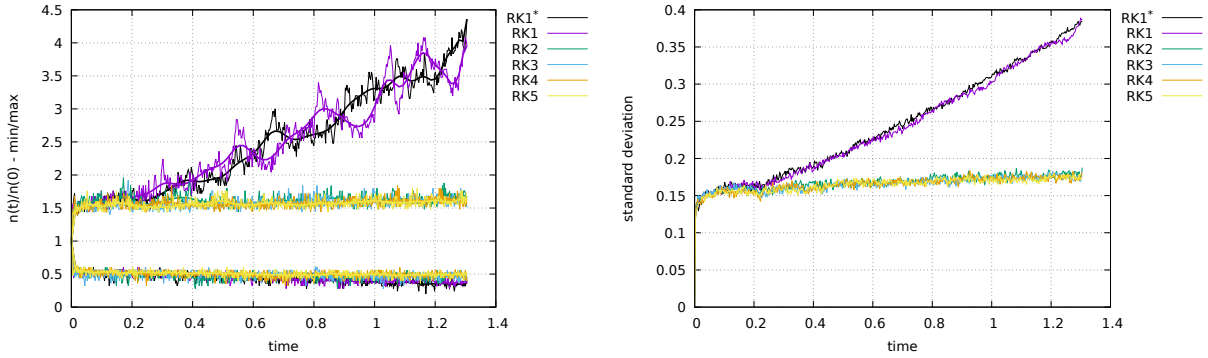


Velocity field.

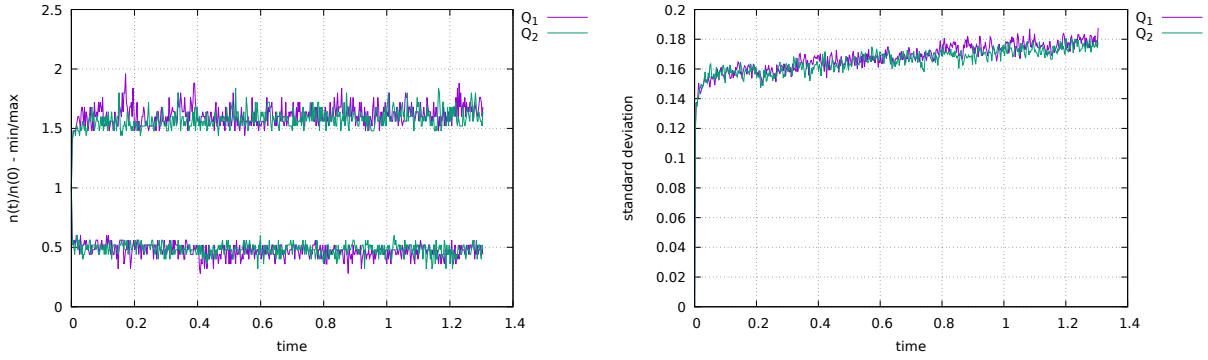


From left to right: velocity error (interpolated minus analytical), velocity divergence, C_0 value. 64x64 mesh.

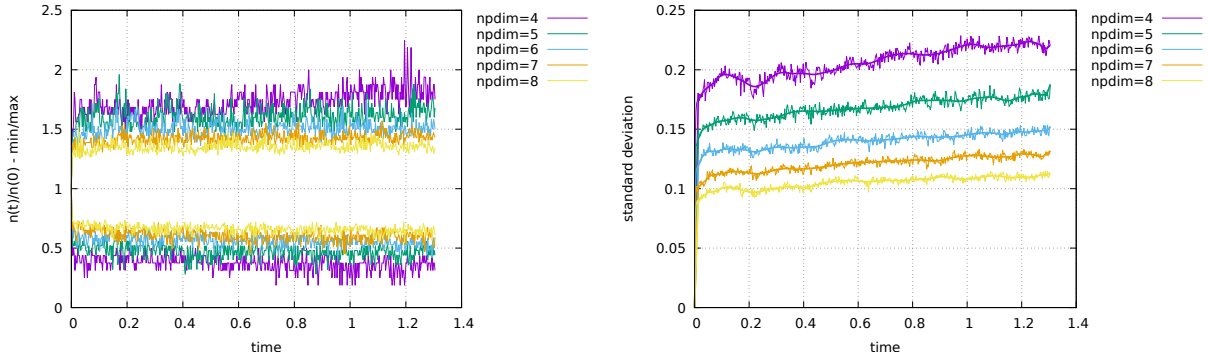
Note the difference between the amplitude of the velocity divergence and the C_0 term!



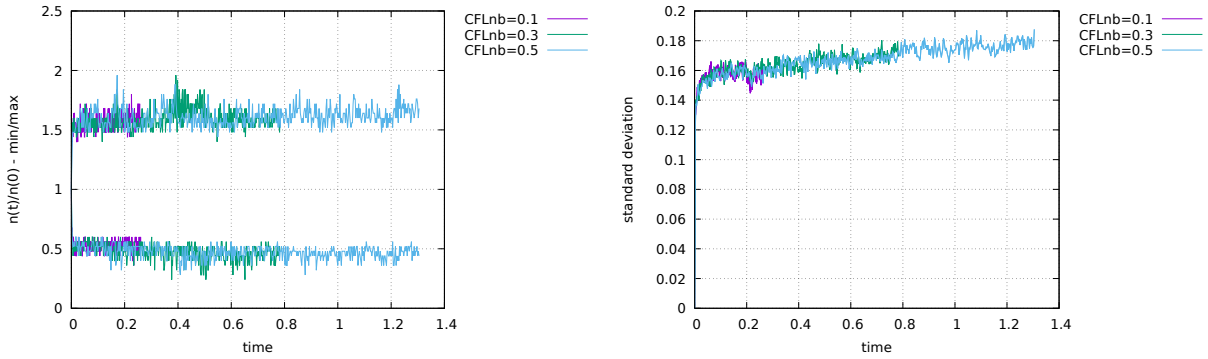
Ref model + varying RKorder (the star means that velocity is directly computed on the particle, no interpolation).



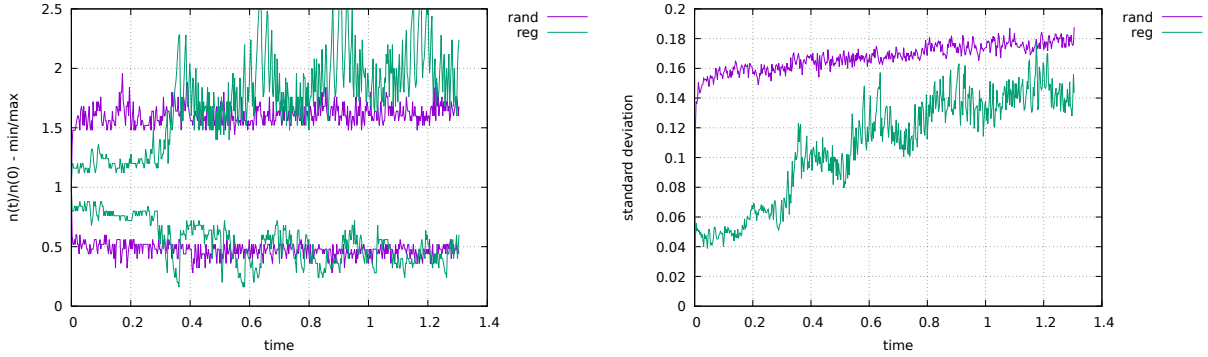
Ref model + Q_1 vs Q_2.



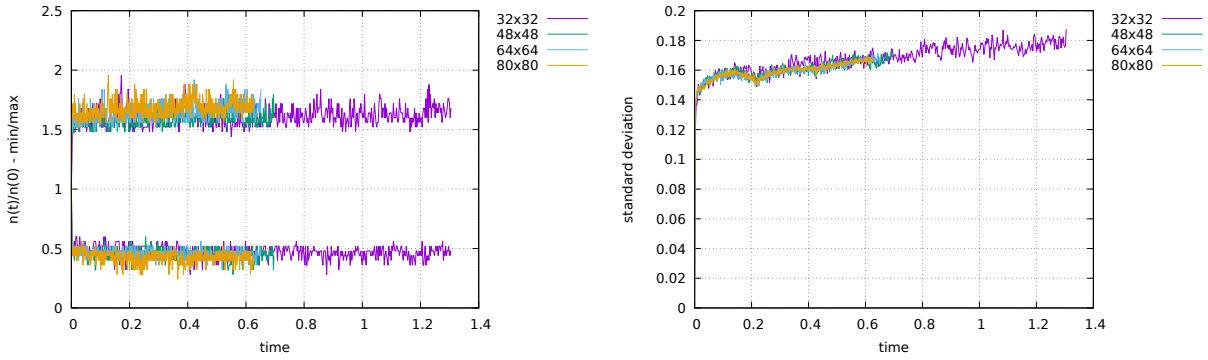
Ref model + initial nparticle per element between 4 and 8.



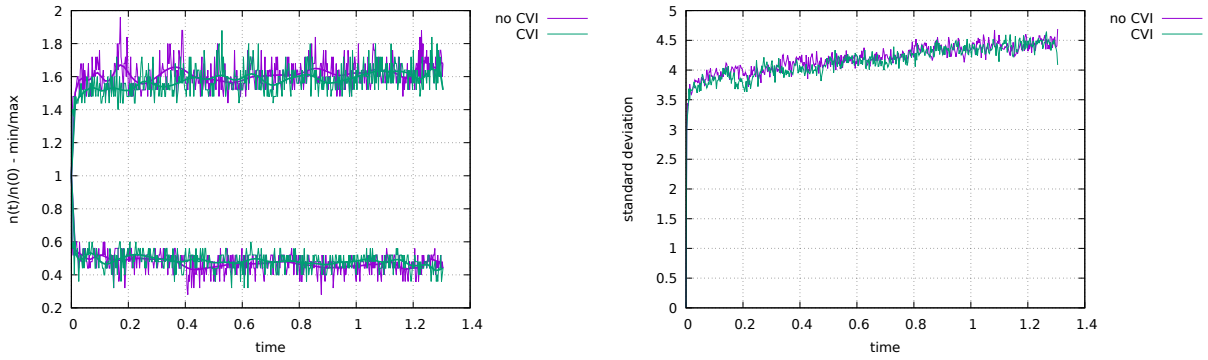
Ref model + CFL nb



Ref model + random vs regular.



Ref model + different resolutions.



Ref model + influence of CVI.

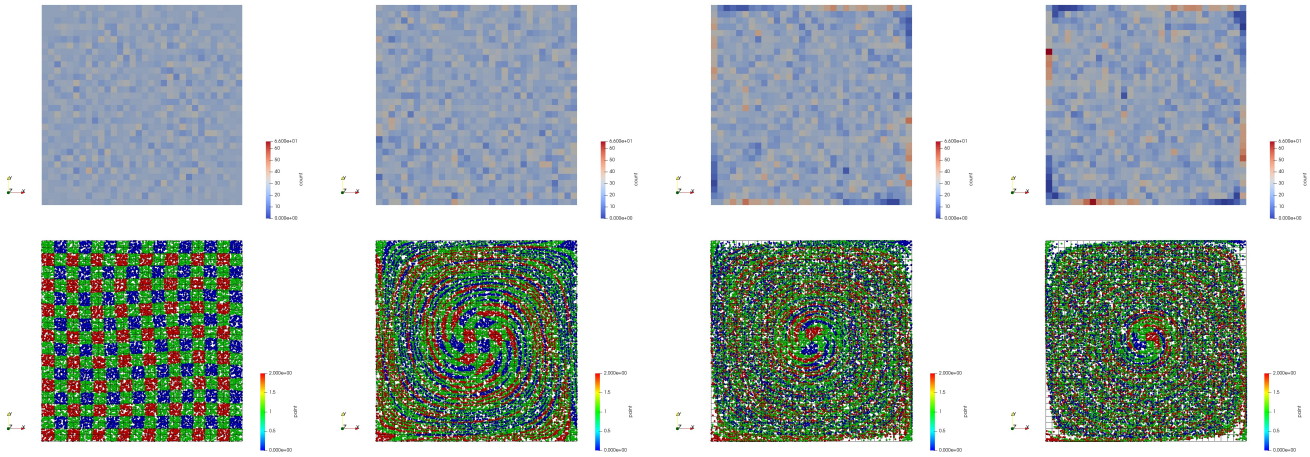
Despite similarities with SolCx (a convection cell in a rectangular domain), advection without CVI does not yield areas with too little/many markers. When CVI is used it does not alter results substantially.

Convection in square domain - Donea & Huerta velocity field

The domain is a unit square and the velocity field is the one of the Donea & Huerta manufactured solution, see Section 18.2:

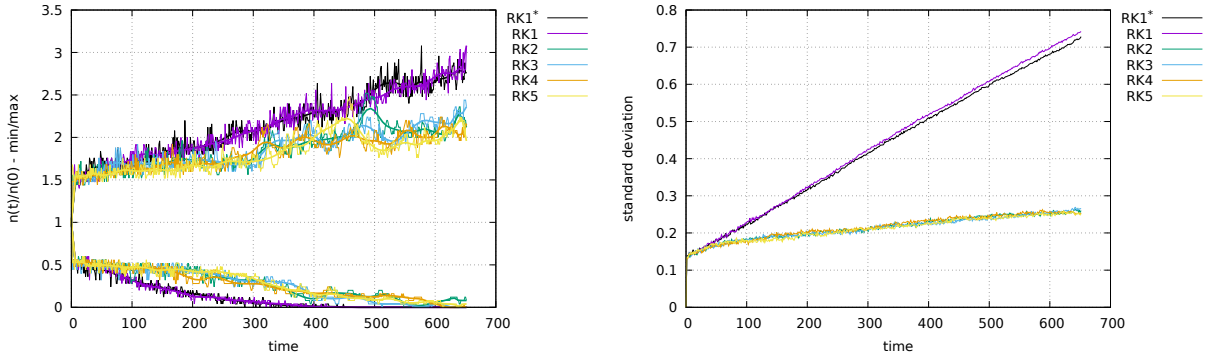
$$\begin{aligned} u(x,y) &= x^2(1-x)^2(2y-6y^2+4y^3) = x^2(1-x)^2 2y(1-3y+2y^2) \\ v(x,y) &= -y^2(1-y)^2(2x-6x^2+4x^3) = -y^2(1-y)^2 2x(1-3x+2x^2) \end{aligned} \tag{30.14}$$

After a few hundreds of timesteps we find that 4 zones in the domain progressively contain less and less markers:

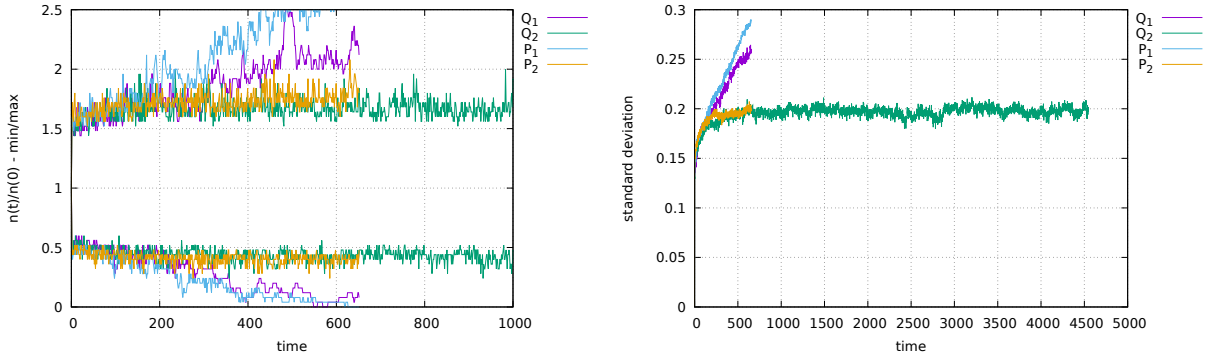


Reference model.

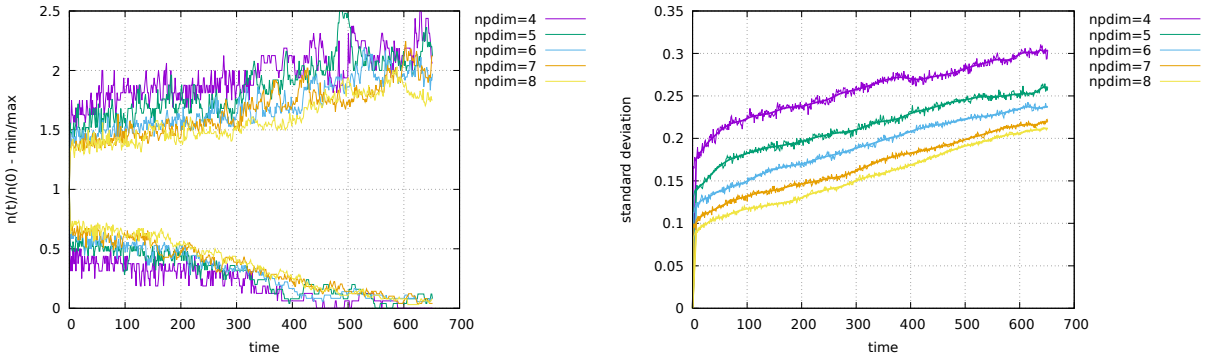
Looking at the marker count statistics we see that its minimum value reaches zero no matter what option is used, except for CVI:



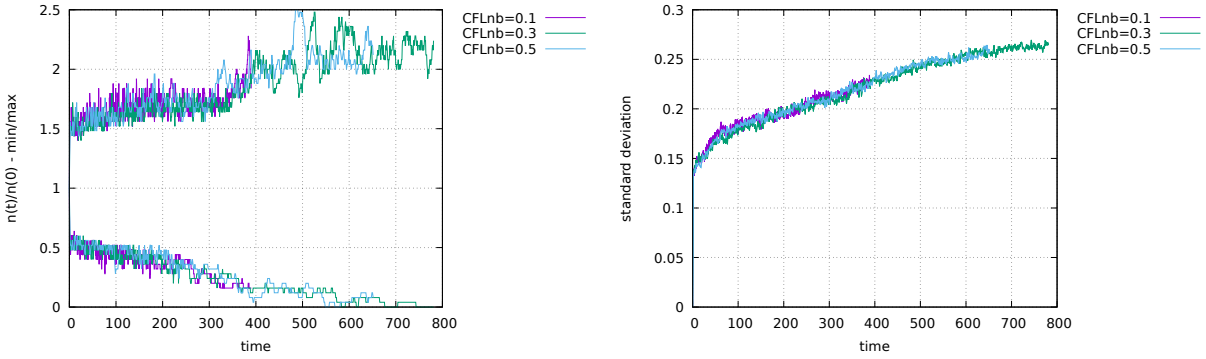
Ref model + varying RKorder (the star means that velocity is directly computed on the particle, no interpolation).



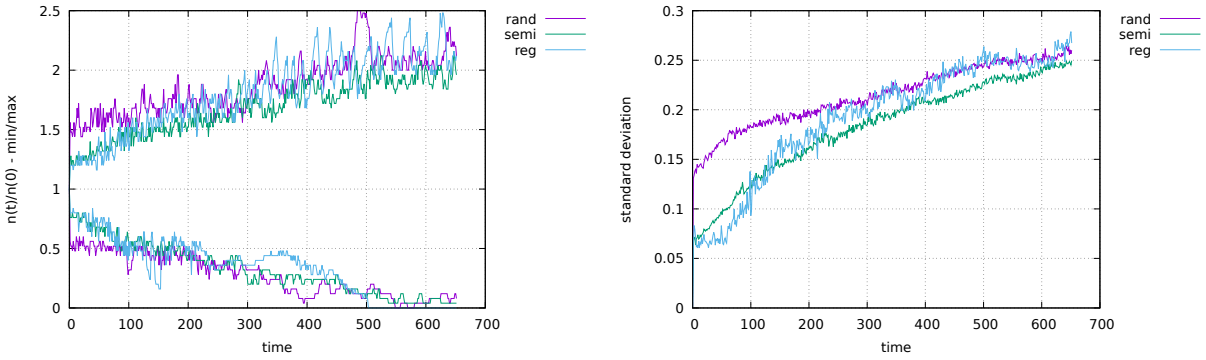
Ref model + Q_1, Q_2, P_1 and P_2.



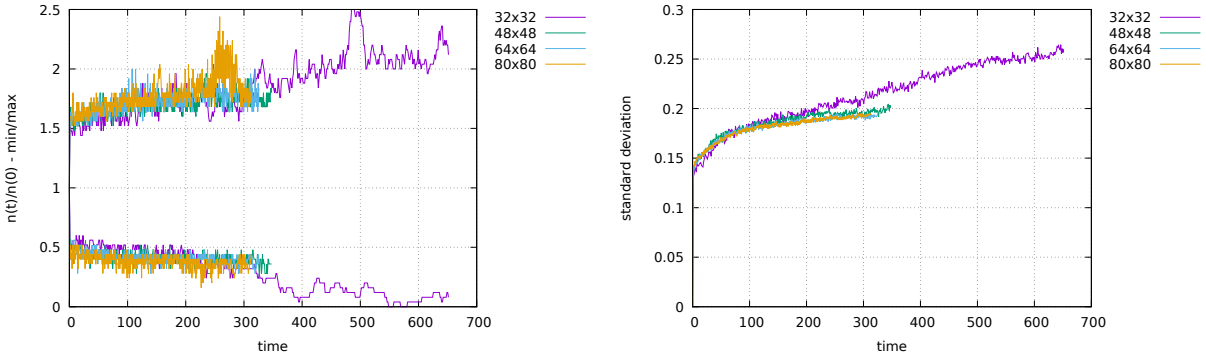
Ref model + initial nparticle per element between 4 and 8.



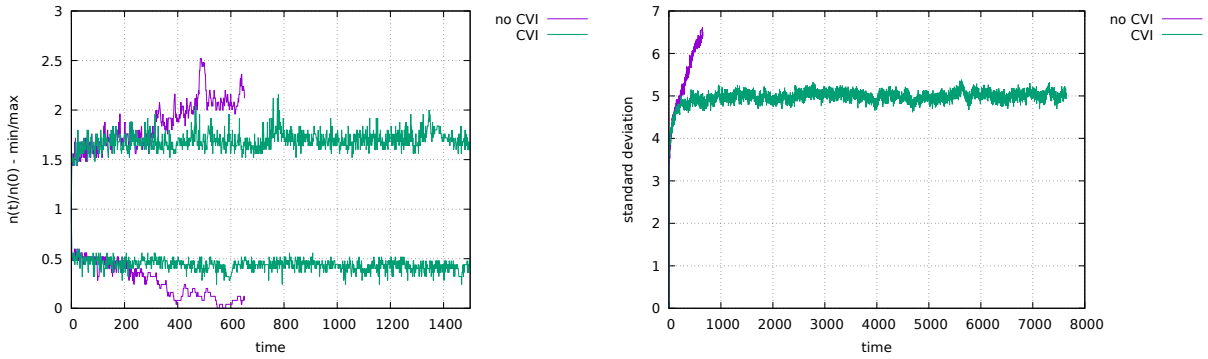
Ref model + CFL nb



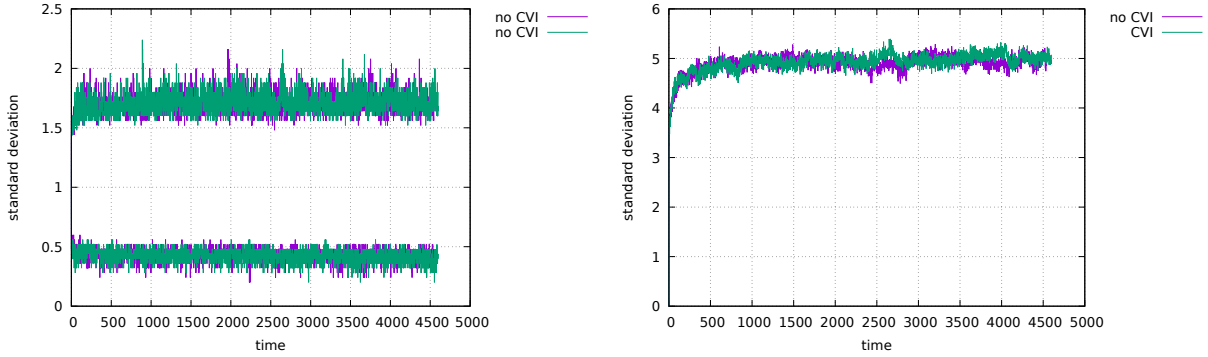
Ref model + random vs regular.



Ref model + different resolutions.



Ref model + influence of CVI.



Ref model + Q_2 + influence of CVI.

We can conclude that the cvi algorithm applied to Q_2 makes virtually no difference.

Stone 31

conservative velocity interpolation 3D

1D
2D
3D
axisymmetric
box
annulus
hollow_sphere
spherical_shell
chunk
disk
Q1P0
Q2Q1
Q2(8)Q1
Q2Pm1
Q3Q2
Q4Q3
MINI
P2pPm1
P2P1
Q1Q1stab
Q1P0RT
P1
P2
Q1
Q2
penalty
sparse_matrix
compressible
analytical_benchmark
numerical_benchmark
error_convergence
stokes_equation
energy_equation
thermo_mechanical
time_stepping
use_mesher
nonlinear
elastic_rheology
viscous_rheology
plastic_rheology
buoyancy_driven
vrms
nusselt_number
rayleigh_number
residual

paraview_output
stream_function
displacement_code
matplotlib
PIC
pressure_scaling
pressure_normalisation
pressure_bc
steady_state
free_surface
discontinuous_galerkin
schur_complement
defect_picard
gravity
Fortran
magnetostatics

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_31`

We shall use here several 3D velcoity fields:

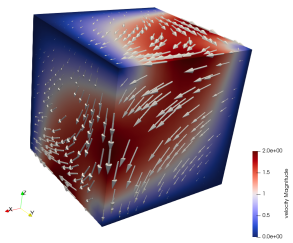
- vfield=1:

$$\vec{\mathbf{v}} = \begin{pmatrix} (y+z)(1-x^2) \\ (-x+z)(1-y^2) \\ (-x-y)(1-z^2) \end{pmatrix} \tag{31.1}$$

Let us compute all the strain rate tensor components:

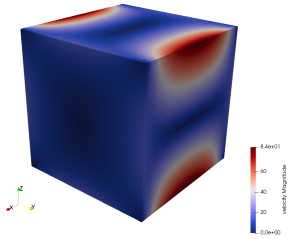
$$\begin{aligned} \dot{\epsilon}_{xx} &= -2x(y+z) \\ \dot{\epsilon}_{yy} &= -2y(-x+z) \\ \dot{\epsilon}_{zz} &= -2z(-x-y) \\ \dot{\epsilon}_{xy} &= \frac{1}{2}(y^2-x^2) \\ \dot{\epsilon}_{xz} &= \frac{1}{2}(z^2-x^2) \\ \dot{\epsilon}_{yz} &= \frac{1}{2}(y^2-y^2) \end{aligned}$$

One can easily verify that $\dot{\epsilon}_{xx}+\dot{\epsilon}_{yy}+\dot{\epsilon}_{zz}=0$.



- vfield=2

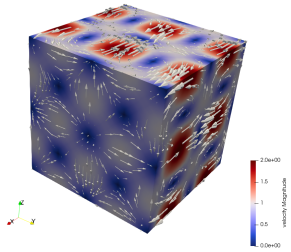
$$\vec{\mathbf{v}} = \begin{pmatrix} (2-x^2-x^4)*(3*y+3*y^3)*(z+2*z*3) \\ (2-y^2-y^4)*(x+2*x^3)*(-z-2*z^3) \\ (2-z^2-z^4)*(-x-2*x^3)*(2*y+y^3) \end{pmatrix} \tag{31.2}$$



- vfield=3:

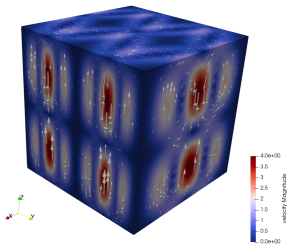
$$\vec{\mathrm{v}} = \begin{pmatrix} 2 * \sin(\pi * x) * \cos(\pi * y) * \cos(\pi * z) \\ -\sin(\pi * y) * \cos(\pi * x) * \cos(\pi * z) \\ -\sin(\pi * z) * \cos(\pi * x) * \cos(\pi * y) \end{pmatrix} \tag{31.3}$$

suspicious ... u(0,y,z)=0 ?!



- vfield=4 (from Braun, Thieulot, Fullsack, DeKool, and Huismans [36] (2008)):

$$\vec{\mathrm{v}} = \begin{pmatrix} \frac{1}{2}\sin(2\pi x)\cos(4\pi y)\cos(\pi z) + \sin(\pi x)\cos(2\pi y)\cos(\pi z) \\ \frac{1}{4}\cos(2\pi x)\sin(4\pi y)\cos(\pi z) + \frac{1}{2}\cos(\pi x)\sin(2\pi y)\cos(\pi z) \\ -2\cos(2\pi x)\cos(4\pi y)\sin(\pi z) - 2\cos(\pi x)\cos(2\pi y)\sin(\pi z) \end{pmatrix} \tag{31.4}$$



TODO:
- implement all strain rate terms for all velocity fields
- make script
-
highliht & correct inaccuracies of Wang et al
rederive the correction term for generic Q1 quadrilaterals and hexahedra
derive strain rate correction terms
derive correction for Q2 ?
do D&H test on all 3 sides of cube with regular distribution of markers. make sure stats are identical.
remove 'option' from code ?
importance of CFL ?

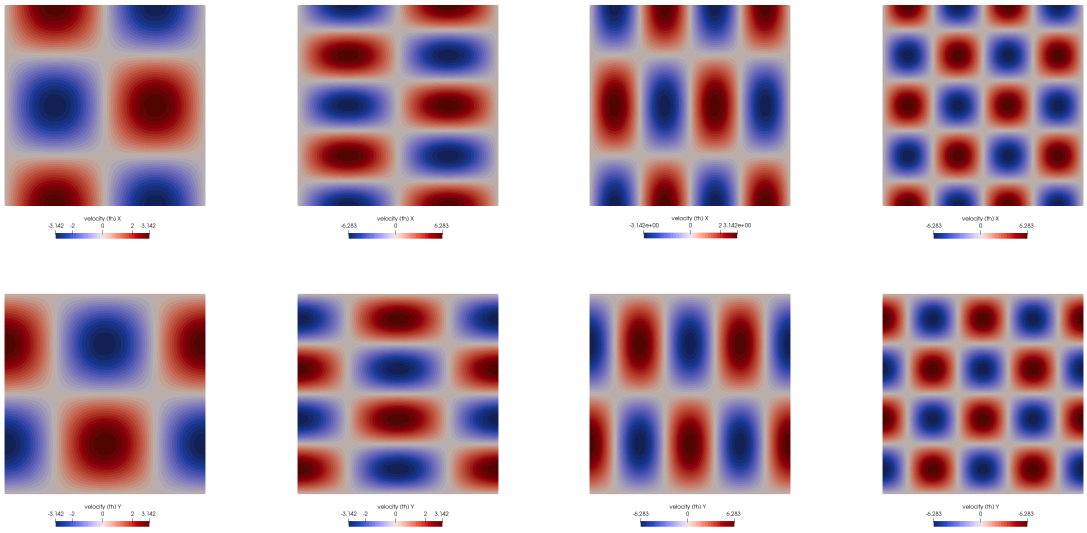
Stone 32

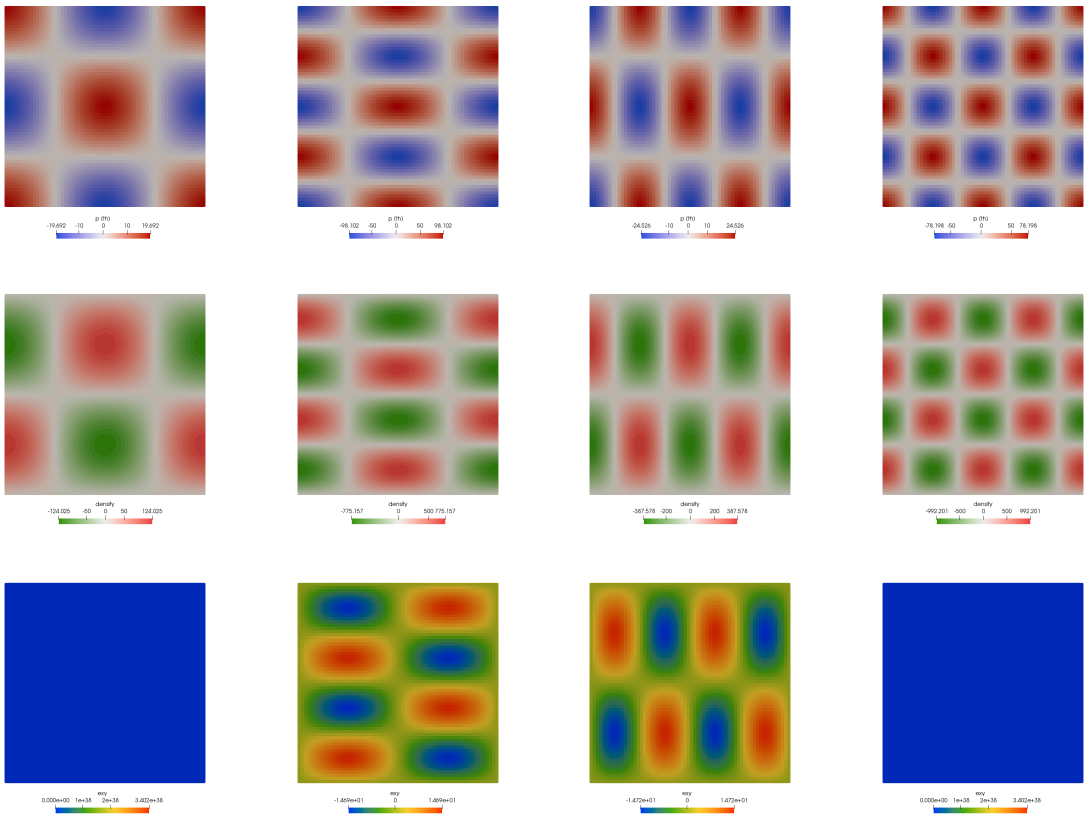
2D analytical sol. from stream function ($Q_1 \times P_0$)

2D
box
Q1P0
analytical_benchmark
error_convergence
stokes_equation
paraview_output
stream_function
pressure_normalisation

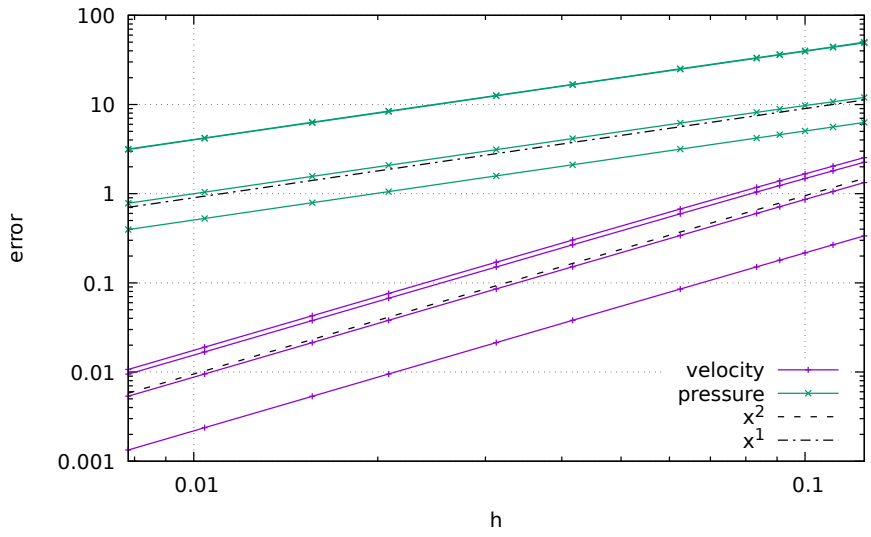
Psmall Code at
https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_32

The code uses a mixed formulation based on $Q_1 \times P_0$ elements. Volume normalisation of the pressure is turned on to insure $\int pdV = 0$. The benchmark is presented in Section 18.27.

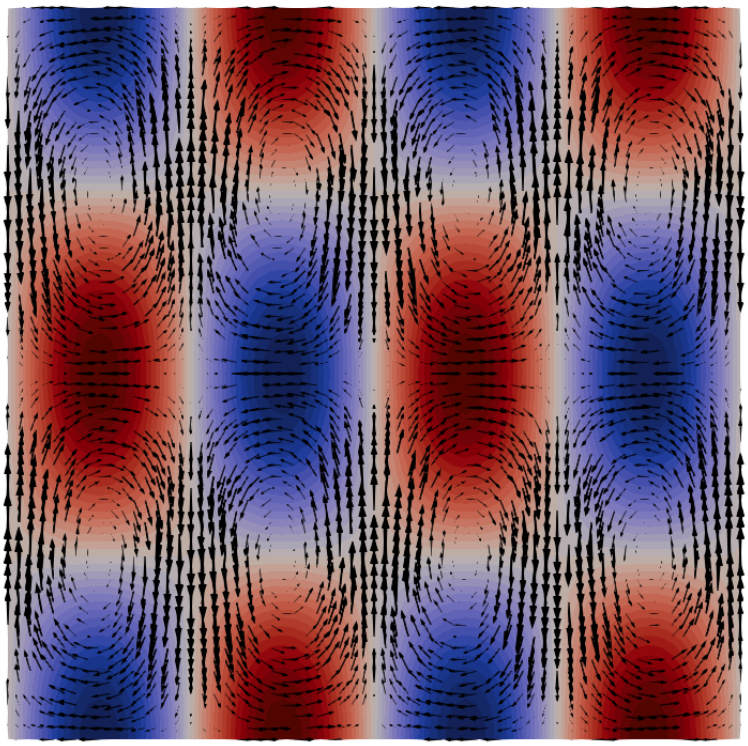




Top to bottom: Velocity components u and v, pressure p, density ρ and strain rate $\dot{\varepsilon}_{xy}$. From left to right: $(m, n) = (1, 1)$, $(m, n) = (1, 2)$, $(m, n) = (2, 1)$, $(m, n) = (2, 2)$



Errors for velocity and pressure for $(m, n) = (1, 1), (1, 2), (2, 1), (2, 2)$



Velocity arrows for $(m, n) = (2, 1)$

Stone 33

Convection in an annulus ($Q_1 \times P_0$)



2D
annulus
Q1P0
penalty
analytical_benchmark
stokes_equation
energy_equation
thermo_mechanical
time_stepping
nonlinear
viscous_rheology
buoyancy_driven
vrms
nusselt_number
rayleigh_number
residual
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_33`

This fieldstone was developed in collaboration with Rens Elbertsen.

This is based on the community benchmark for viscoplastic thermal convection in a 2D square box (Tosi et al (2015) [332]) as already carried out in STONE 28.

In this experiment the geometry is an annulus of inner radius $R_1 = 1.22$ and outer radius $R_2 = 2.22$. The rheology and buoyancy forces are identical to those of the box experiment. The initial temperature is now given by:

$$T(r,\theta) = T_c(r) + A\ s(1-s)\cos(N_0\theta) \qquad s = \frac{R_2 - r}{R_2 - R_1} \in [0,1]$$

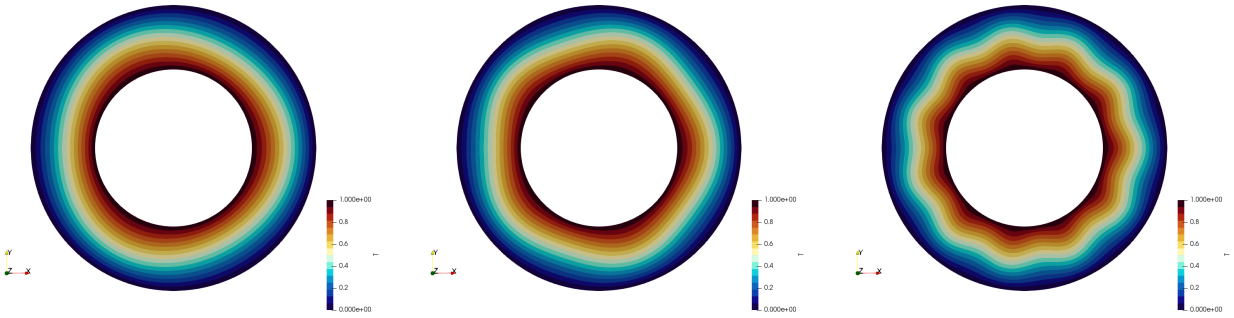
where s in the normalised depth, A is the amplitude of the perturbation and N_0 the number of lobes. In this equation $T_c(r)$ stands for the steady state purely conductive temperature solution which is obtained by solving the Laplace's equation in polar coordinates (all terms in θ are dropped because of radial symmetry) supplemented with two boundary conditions:

$$\Delta T_c = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right) = 0 \qquad T(r=R_1) = T_1 = 1 \qquad T(r=R_2) = T_2 = 0$$

We obtain

$$T_c(r) = \frac{\log(r/R_2)}{\log(R_1/R_2)}$$

Note that this profile differs from the straight line that is used in Tosi *et al.* (2015) [332] and in STONE 28.



Examples of initial temperature fields for $N_0 = 3, 5, 11$

Boundary conditions can be either no-slip or free-slip on both inner and outer boundary. However, when free-slip is used on both a velocity null space exists and must be filtered out: the solution contains an arbitrary rotation on top of the solution we are interested in. This additional velocity field can be problematic since velocity enters the advection-diffusion equation for temperature (and/or compositions) and it also essentially determines the time step value for a chosen mesh size (CFL condition).

Only the number of elements *nelr* in the radial direction can be chosen by the user since the number of elements in the tangential direction *nelt* is automatically determined such that elements at the surface of the annulus are approximately square.

About the nullspace For these reasons the nullspace must be removed from the obtained solution after every timestep. There are two types of nullspace removal: removing net angular momentum, and removing net rotations. The technique that we use here is simple:

1. pin the first node which is at position $(x = R_1, y = 0)$ by prescribing its vertical velocity to zero. This eliminates the nullspace completely.

2. compute the angular momentum and correct the velocity field so that it is zero as explained in section **??**.

   The corrected velocity field $\vec{\mathrm{v}}'$ is then given by

   $$\vec{\mathrm{v}}' = \vec{\mathrm{v}} - <\vec{\omega}> \times \vec{r}$$

   and it is easy to verify that

   $$\begin{aligned} <\vec{\omega}'> &= \frac{1}{V}\int_V \frac{\vec{r}\times\vec{\mathrm{v}}'}{r^2}dV \\ &= \frac{1}{V}\int_V \frac{\vec{r}\times(\vec{\mathrm{v}} - <\vec{\omega}> \times\vec{r})}{r^2}dV \\ &= \frac{1}{V}\int_V \frac{\vec{r}\times\vec{\mathrm{v}}}{r^2}dV - \frac{1}{V}\int_V \frac{\vec{r}\times <\vec{\omega}> \times\vec{r}}{r^2}dV \\ &= <\vec{\omega}> - <\vec{\omega}> \frac{1}{V}\int_V dV = 0 \end{aligned} \tag{33.1}$$

Measurements We calculate and monitor the following quantities:

- the average temperature $< T >$

  $$\langle T\rangle = \frac{\int_\Omega T d\Omega}{\int_\Omega d\Omega} = \frac{1}{V_\Omega}\int_\Omega T d\Omega \tag{33.2}$$

- the root mean square velocity v_{rms} as given by equation (3.125).

- the root mean square of the radial and tangential velocity components as given by equations (**??**) and (**??**).

- the heat transfer through both boundaries Q:

$$Q_{inner,outer} = \int_{\Gamma_{i,o}} \boldsymbol{q} \cdot \boldsymbol{n} \, d\Gamma \tag{33.3}$$

- the Nusselt number at both boundaries Nu as given by equations (**??**) and (**??**).
- the power spectrum of the temperature field:

$$PS_n(T) = \left| \int_{\Omega} T(r,\theta) e^{in\theta} d\Omega \right|^2 . \tag{33.4}$$

- the average viscosity $\langle \eta \rangle$ (see Eq. **??**)

Cases and rheologies In the following Table, we list the benchmark cases according to the parameters used.

Case	Ra	$\Delta\mu_T$	$\Delta\mu_y$	μ^*	σ_Y	Convective regime
1	10^2	10^5	1	–	–	Stagnant lid
2	10^2	10^5	1	10^{-3}	1	Mobile lid
3	10^2	10^5	10	–	–	Stagnant lid
4	10^2	10^5	10	10^{-3}	1	Mobile lid
5a	10^2	10^5	10	10^{-3}	4	Periodic
5b	10^2	10^5	10	10^{-3}	3 – 5	Mobile lid – Periodic – Stagnant lid

Benchmark cases and corresponding parameters.

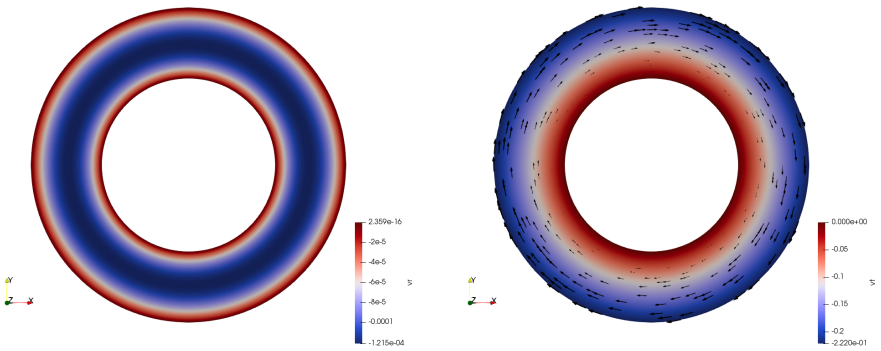
Steady state As explained in 20 we use a relaxation technique in combination with an energy equation stripped from its time derivative term. The use of this technique is controlled by the `find_ss` parameter.

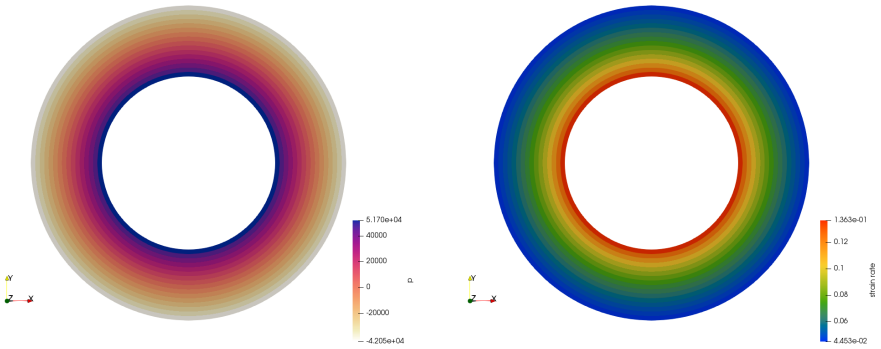
The termination criterion is based on the Nusselt number also.

```
if np.abs(Nu_boundary1-Nu_boundary1_old and \
   np.abs(Nu_boundary2-Nu_boundary2_old) <1.e-5:
   print("Nu converged to 1e-5")
   break
```

Spectral analysis/power spectrum CHECK [39] for power spectrum of Nusselt nb. check [49, 234, 253, 267, 288, 350, 263]

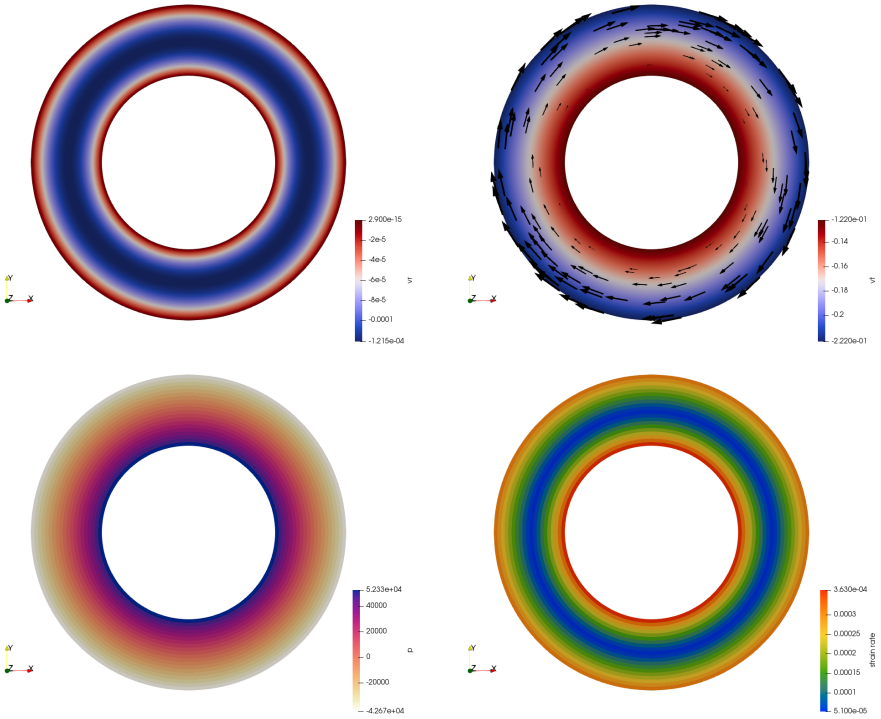
test 1 We here wish to test the boundary conditions. We set $\vec{\mathbf{v}} = (y, -x)$ on the outside boundary, and $\vec{\mathbf{v}} = (0, 0)$ on the inside boundary, set the temperature to zero, and run a single time step (note that the no net rotation/angular momentum algorithm must be turned off then).





The radial velocity is not zero (or machine precision zero) because of the use of the penalty method.

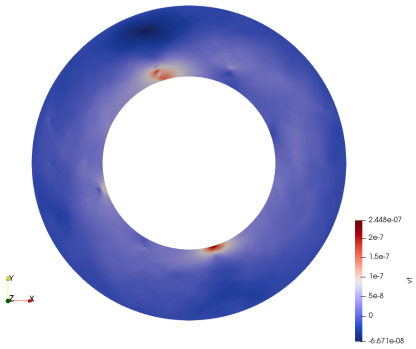
test 2a This is the same experiment as above but now the inner boundary is free slip. We then expect a constant angular velocity (measured by the code as -9.999999e-01) and also a velocity on the inner boundary equal (in magnitude) to $R_1 = 1.22$ which is indeed what we recover:



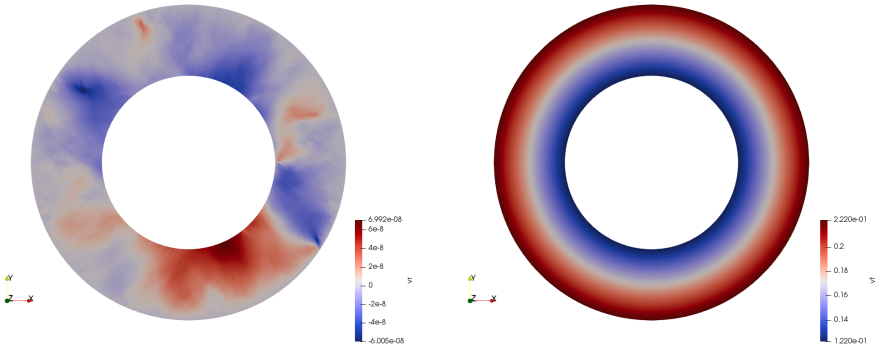
Unsurprisingly the pressure and radial velocity fields are unaffected by this change of boundary conditions.

test 3 Same experiment as exp.2, but boundary conditions are switched (free slip outside, prescribed inside). Results are not shown but the angular velocity has the right value, and so does the velocity field.

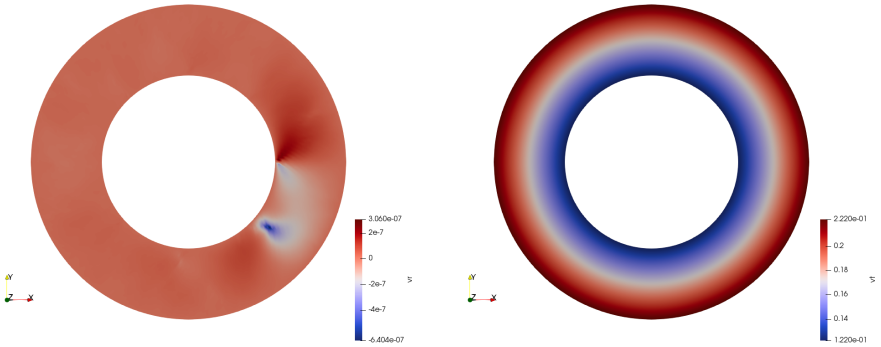
test 2b Same as exp.2 but now the net rotation algorithm is switched on. Since the boundary conditions are such that a constant angular rotation is expected, then the algorithm should yield a tangential velocity that is null, and this is indeed what we recover (at least 8 orders of magnitude smaller than before removal):



test 4 We now set free slip boundary conditions on the outside boundary and only a single node at position $(R_1, 0)$ is prescribed a velocity $(0, R_1)$. Gravity is set to zero and no net angular velocity removal is applied. We recover a (near) zero radial velocity component and a the expected angular velocity $9.999984e-01$.



test 4b Same as test 4, but now free slip boundary conditions are imposed on both the inner and outer boundaries at the exception of a single node as above. We expect identical results as in test 4 (same physics) and we indeed do recover the same v_t velocity field and a similarly negligible v_r field:



These tests are convincing: the free-slip and no-slip boundary conditions work as expected and the single node velocity b.c. to remove the nullspace in conjunction with free-slip also works. Let us now turn to resolution and time-discretisation.

Stone 34

2D plane strain elastic problems (Q_1)

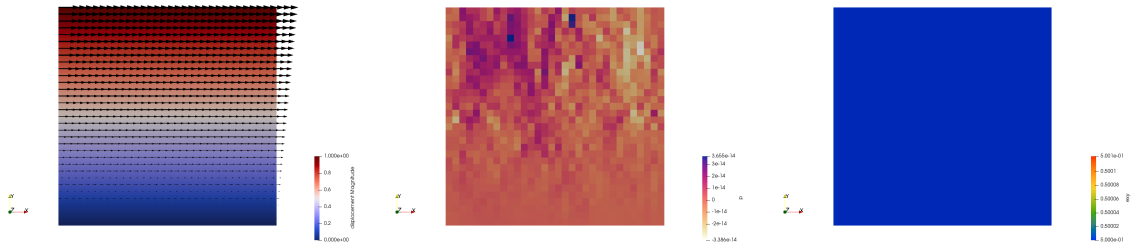


2D
box
Q1
sparse_matrix
compressible
analytical_benchmark
error_convergence
stokes_equation
elastic_rheology
paraview_output
displacement_code

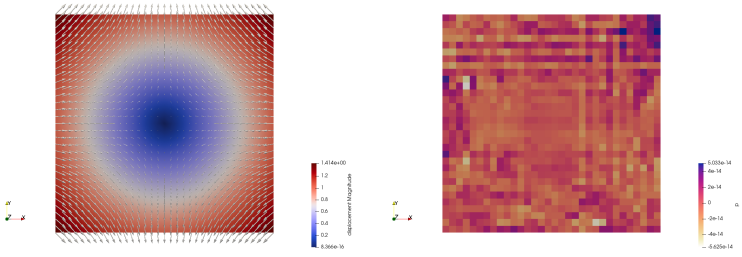
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_34`

Exp. 3 of this stone was developed in collaboration with Lukas van de Wiel.

Experiment 1: Simple shear



Experiment 2: Pure shear



Experiment 3: Aquarium

The setup is as follows: a 2D square of elastic material of size L is subjected to the following boundary conditions: free slip on the sides, no slip at the bottom and free at the top. It has a density ρ and is placed is a gravity field $\vec{g} = -g\vec{e}_y$. For an isotropic elastic medium the stress tensor is given by:

$$\boldsymbol{\sigma} = \lambda(\vec{\nabla} \cdot \vec{\mathrm{v}})\mathbf{1} + 2\mu\boldsymbol{\varepsilon}(\vec{\mathrm{v}})$$

where λ is the Lamé parameter and μ is the shear modulus. The displacement field is $\vec{\mathrm{v}} = (0, \mathrm{v}_y(y))$ because of symmetry reasons (we do not expect any of the dynamic quantities to depend on the x coordinate and also expect the horizontal displacement to be exactly zero). The velocity divergence is then $\vec{\nabla} \cdot \vec{\mathrm{v}} = \partial \mathrm{v}_y / \partial y$ and the strain tensor:

$$\boldsymbol{\varepsilon}(\vec{\mathrm{v}}) = \begin{pmatrix} 0 & 0 \\ 0 & \frac{\partial \mathrm{v}_y}{\partial y} \end{pmatrix}$$

so that the stress tensor is:

$$\boldsymbol{\sigma} = \begin{pmatrix} \lambda \frac{\partial \mathrm{v}_y}{\partial y} & 0 \\ 0 & (\lambda + 2\mu)\frac{\partial \mathrm{v}_y}{\partial y} \end{pmatrix}$$

$$\vec{\nabla} \cdot \boldsymbol{\sigma} = (\partial_x \quad \partial_y) \cdot \begin{pmatrix} \lambda \frac{\partial \mathrm{v}_y}{\partial y} & 0 \\ 0 & (\lambda + 2\mu)\frac{\partial \mathrm{v}_y}{\partial y} \end{pmatrix} = \begin{pmatrix} 0 \\ (\lambda + 2\mu)\frac{\partial^2 \mathrm{v}_y}{\partial y^2} \end{pmatrix} = \begin{pmatrix} 0 \\ \rho g \end{pmatrix}$$

so that the vertical displacement is then given by:

$$\mathrm{v}_y(y) = \frac{1}{2}\frac{\rho g}{\lambda + 2\mu} y^2 + \alpha y + \beta$$

where α and β are two integration constants. We need now to use the two boundary conditions: the first one states that the displacement is zero at the bottom, i.e. $u_y(y = 0) = 0$ which immediately implies $\beta = 0$. The second states that the stress at the top is zero (free surface), which implies that $\partial u_y / \partial y(y = L) = 0$ which allows us to compute α. Finally:

$$u_y(y) = \frac{\rho g}{\lambda + 2\mu}\left(\frac{y^2}{2} - Ly\right)$$

The pressure is given by

$$p = -(\lambda + \frac{2}{3}\mu)\vec{\nabla} \cdot \vec{\mathrm{v}} = (\lambda + \frac{2}{3}\mu)\frac{\rho g}{\lambda + 2\mu}(L - y) = \frac{\lambda + \frac{2}{3}\mu}{\lambda + 2\mu}\rho g(L - y) = \frac{1 + \frac{2\mu}{3\lambda}}{1 + 2\mu/\lambda}\rho g(L - y)$$

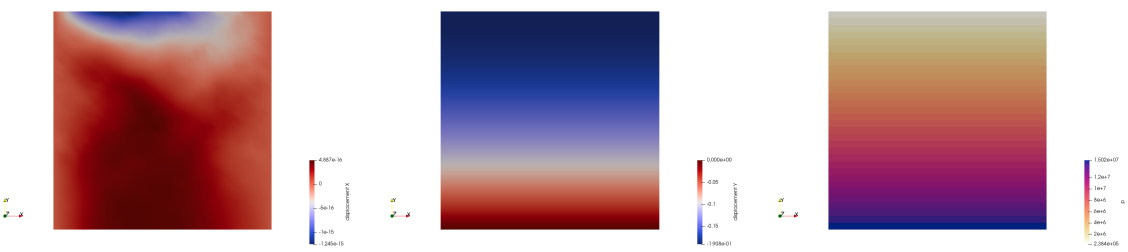
In the incompressible limit, the poisson ratio is $\nu \sim 0.5$. Materials are characterised by a finite Young's modulus E, which is related to ν and λ:

$$\lambda = \frac{E\nu}{(1 + \nu)(1 - 2\nu)} \qquad \mu = \frac{E}{2(1 + \nu)}$$

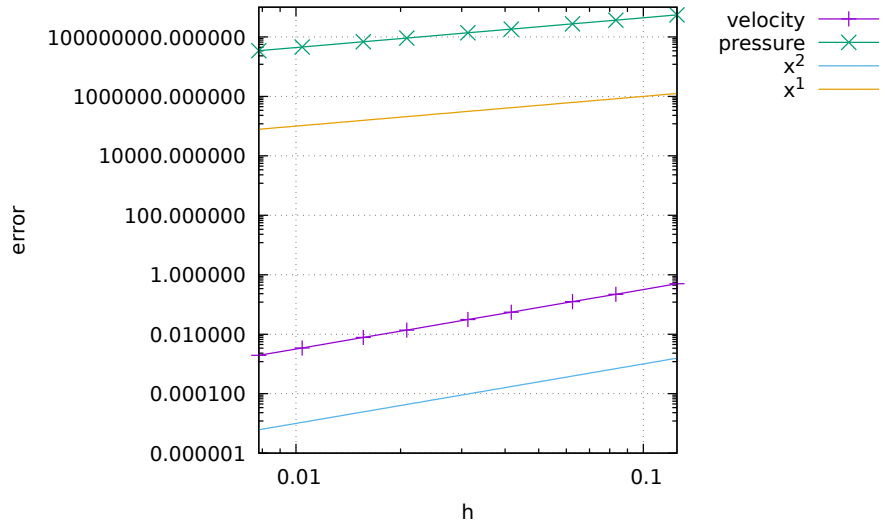
It is then clear that for incompressible parameters λ becomes infinite while μ remains finite. In that case the pressure then logically converges to the well known formula:

$$p = \rho g(L - y)$$

In what follows we set $L = 1000\mathrm{m}$, $\rho = 2800$, $\nu = 0.25$, $E = 6 \cdot 10^{10}$, $g = 9.81$.

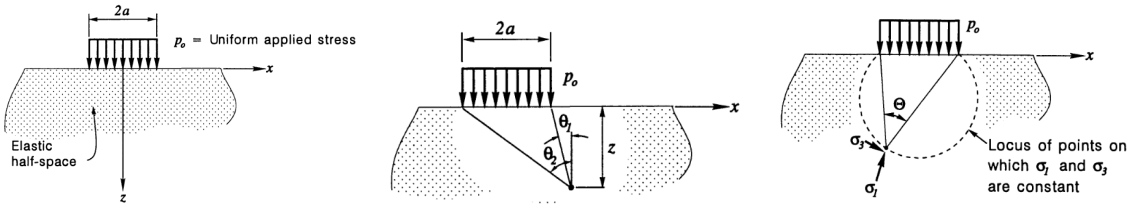






Experiment 4: Strip load

The domain is 3×2 km. Free slip boundary conditions are applied on the sides and bottom boundaries. Traction boundaries corresponding to a pressure of 100 MPa are prescribed on the top boundaries for $|x - L_x/2| < a$ with $a = 50$ m. Material parameters are identical to those of experiment 3.



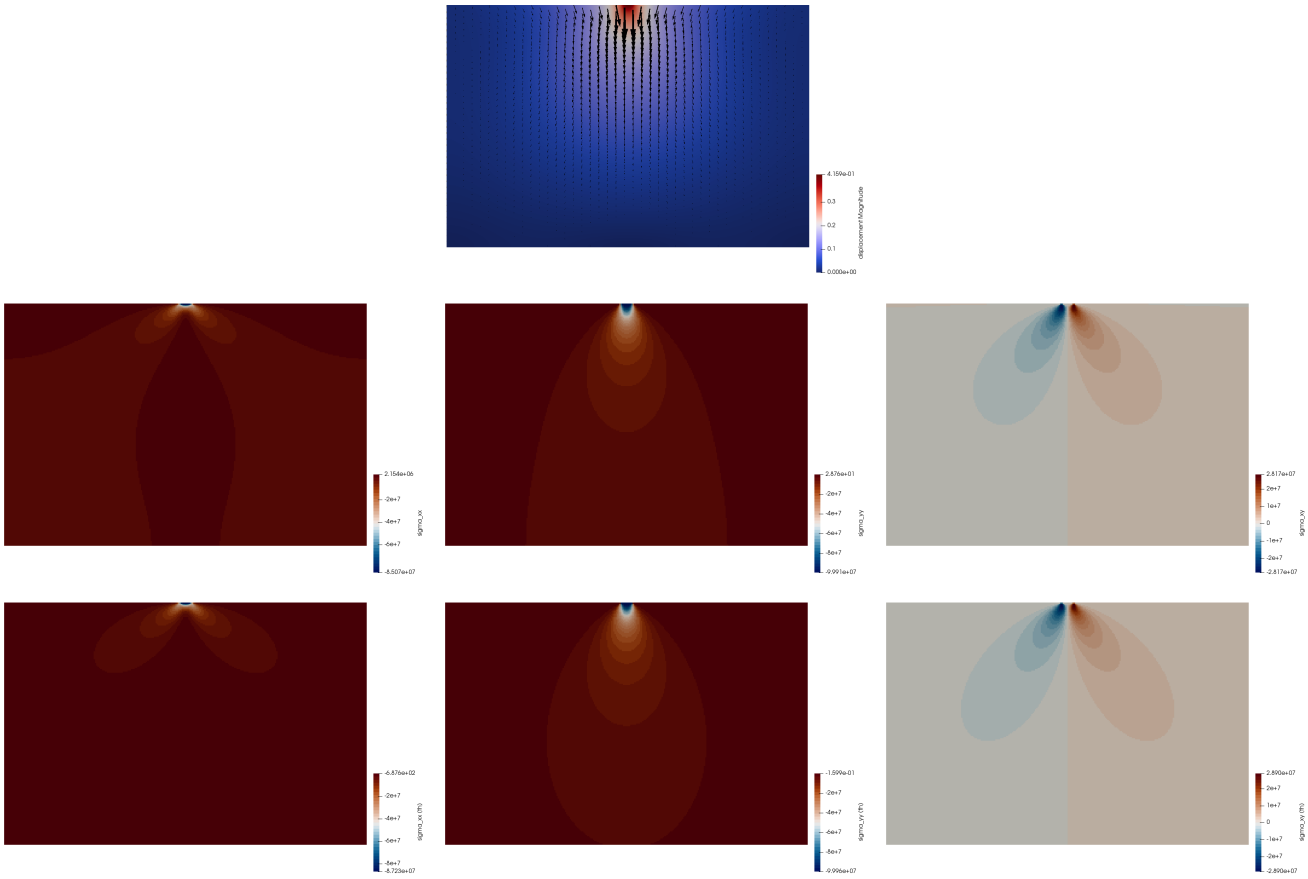
The analytical solution is given in terms of the stress tensor. From Davis and Selvadurai [85](section 4.6) we have the components of the stress tensor:

$$\sigma_{xx} = \frac{p_0}{\pi}\left[\theta - \frac{1}{2}\sin 2\theta\right]_{\theta_1}^{\theta_2} \qquad \sigma_{zz} = \frac{p_0}{\pi}\left[\theta + \frac{1}{2}\sin 2\theta\right]_{\theta_1}^{\theta_2} \qquad \sigma_{xz} = \frac{p_0}{\pi}\left[\sin^2\theta\right]_{\theta_1}^{\theta_2}$$

or,

$$\begin{aligned}
\sigma_{xx} &= \frac{p_0}{\pi}\left[\theta - \frac{1}{2}\sin 2\theta\right]_{\theta_1}^{\theta_2} = \Theta - \frac{1}{2}(\sin 2\theta_2 - \sin 2\theta_1) \\
\sigma_{zz} &= \frac{p_0}{\pi}\left[\theta + \frac{1}{2}\sin 2\theta\right]_{\theta_1}^{\theta_2} = \Theta + \frac{1}{2}(\sin 2\theta_2 - \sin 2\theta_1) \\
\sigma_{xz} &= \frac{p_0}{\pi}\left[\sin^2\theta\right]_{\theta_1}^{\theta_2} = \sin^2\theta_2 - \sin^2\theta_1
\end{aligned}$$

with $\Theta = \theta_2 - \theta_1$.



Results obtained on 300x200 mesh. Top row is recovered stresses. Bottom row is analytical solution.

Stone 35

analytical sol. in annulus from stream function ($Q_1 \times P_0$)



2D
annulus
Q1P0
penalty
analytical_benchmark
error_convergence
stokes_equation
paraview_output
stream_function

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_35`

The domain is an annulus with inner radius R_1 and outer radius R_2. Gravity is radial and has norm 1. Boundary conditions are no-slip on the inner and outer boundaries. The velocity, pressure and density fields have been derived in Section 19.19:

$$\mathrm{v}_r(r,\theta) = -\frac{1}{r}(ar^4+br^3+cr^2+dr+e)k\sin(k\theta) \tag{35.1}$$

$$\mathrm{v}_\theta(r,\theta) = -(4ar^3+3br^2+2cr+d)\cos(k\theta) \tag{35.2}$$

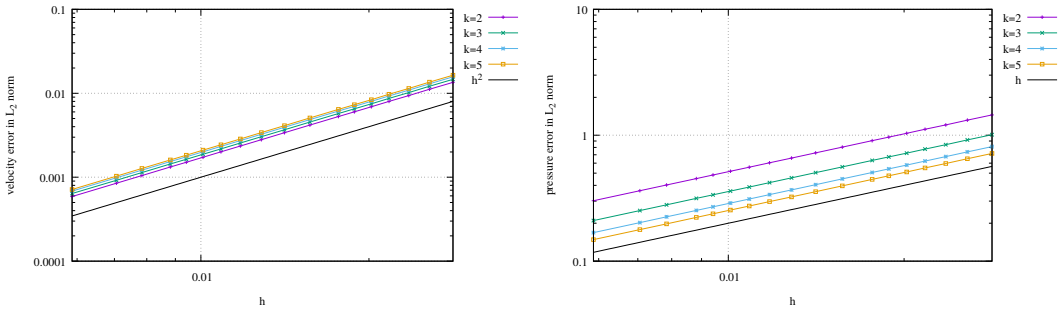
$$p(r,\theta) = \frac{1}{k}\sin(k\theta)\left[2(k^2-16)ar^2+(k^2-9)br+(1-k^2)\frac{d}{r}-2k^2\frac{e}{r^2}\right] \tag{35.3}$$

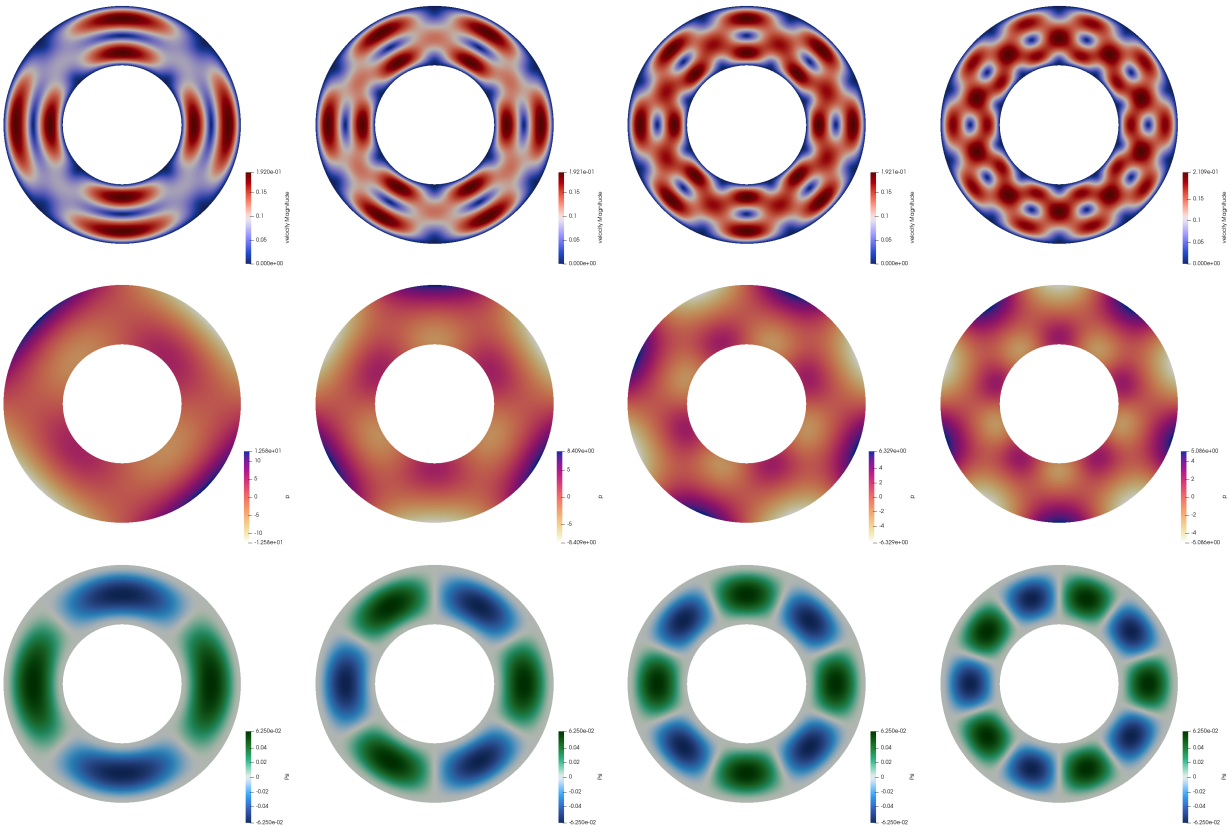
$$\rho(r,\theta) = \frac{\sin(k\theta)}{k}\frac{Aar^4+Bbr^3+Ccr^2+Ddr+Ee}{r^3} \tag{35.4}$$

with

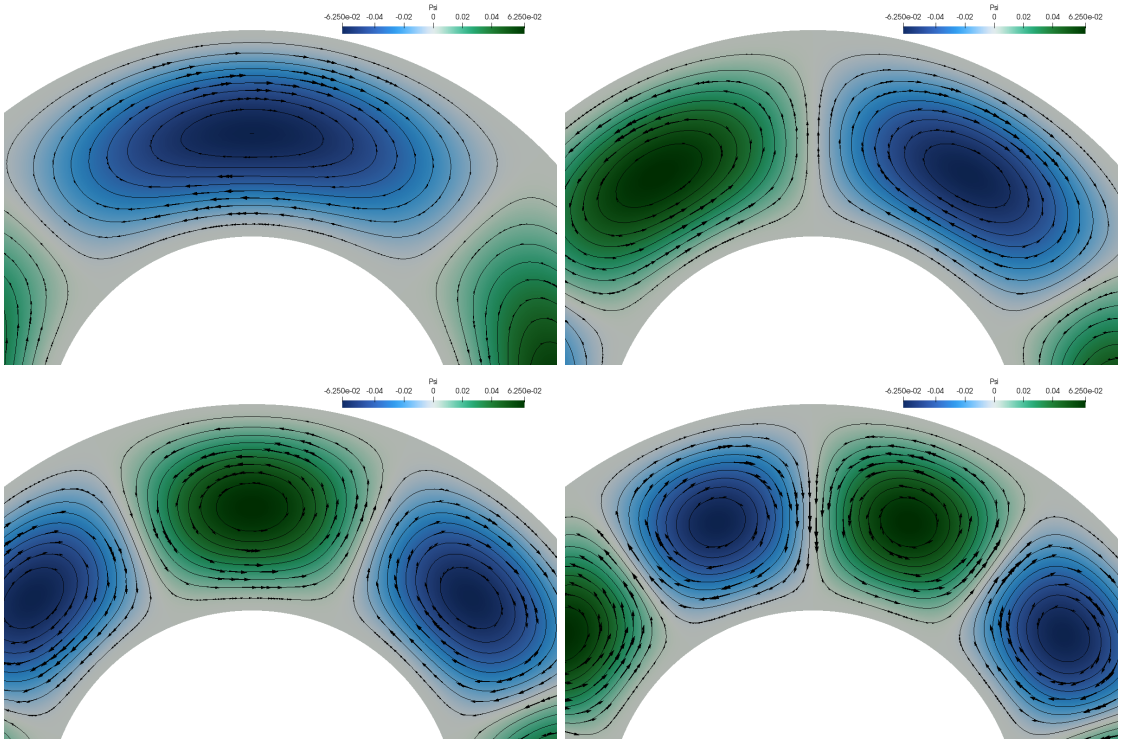
$$
\begin{aligned}
a &= 1 \\
b &= -2R_1 - 2R_2 \\
c &= R_1^2 + R_2^2 + 4R_1R_2 \\
d &= -2R_1R_2^2 - 2R_1^2R_2 \\
e &= R_1^2R_2^2 \\
A &= (k^2-4)(k^2-16) \\
B &= k^4 - 10k^2 + 9 \\
C &= k^2(k^2-4) \\
D &= (k^2-1)(k^2-1) \\
E &= k^2(k^2-4)
\end{aligned}
$$

I have computed the velocity and pressure discretisation errors for $k = 3, 4, 5$ and have recovered the expected rates for this type of element.





From left to right: $k = 2, 3, 4, 5$



Isocontours of the stream function Ψ. Velocity vectors are of course tangent to these.

Stone 36

the annulus geometry elastic aquarium (Q_1)



2D
annulus
Q1
sparse_matrix
compressible
analytical_benchmark
error_convergence
stokes_equation
elastic_rheology
paraview_output
displacement_code

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_36`

This stone was developed in collaboration with Lukas van de Wiel.

The domain is an annulus with inner radius R_1 and outer radius R_2. It is filled with a single elastic material characterised by a Young's modulus E and a Poisson ratio ν, a density ρ_0. The gravity $\vec{g} = -g_0 \vec{e}_r$ is pointing towards the center of the domain.

The problem at hand is axisymmetric so that the tangential component of the displacement vector v_θ is assumed to be zero as well as all terms containing ∂_θ. The components of the strain tensor $\boldsymbol{\varepsilon}(\vec{\mathbf{v}})$ are

$$\varepsilon_{rr} = \frac{\partial v_r}{\partial r} \tag{36.1}$$

$$\varepsilon_{\theta\theta} = \frac{v_r}{r} + \frac{1}{r}\frac{\partial v_\theta}{\partial \theta} = \frac{v_r}{r} \tag{36.2}$$

$$\varepsilon_{r\theta} = \frac{1}{2}\left(\frac{\partial v_\theta}{\partial r} - \frac{v_\theta}{r} + \frac{1}{r}\frac{\partial v_r}{\partial \theta} \right) = 0 \tag{36.3}$$

so that the tensor simply is

$$\boldsymbol{\varepsilon}(\vec{\mathbf{v}}) = \begin{pmatrix} \varepsilon_{rr} & \varepsilon_{r\theta} \\ \varepsilon_{r\theta} & \varepsilon_{\theta\theta} \end{pmatrix} = \begin{pmatrix} \frac{\partial v_r}{\partial r} & 0 \\ 0 & \frac{v_r}{r} \end{pmatrix} \tag{36.4}$$

The pressure is

$$p = -\lambda \vec{\nabla} \cdot \vec{\mathbf{v}} = -\lambda \left(\frac{1}{r}\frac{\partial (r v_r)}{\partial r} \right) \tag{36.5}$$

and finally the stress tensor:

$$\boldsymbol{\sigma} = -p\mathbf{1} + 2\mu\boldsymbol{\varepsilon} = \begin{pmatrix} \lambda\frac{1}{r}\frac{\partial(rv_r)}{\partial r} + 2\mu\frac{\partial v_r}{\partial r} & 0 \\ 0 & \lambda\frac{1}{r}\frac{\partial(rv_r)}{\partial r} + 2\mu\frac{v_r}{r} \end{pmatrix} \tag{36.6}$$

The divergence of the stress tensor is given by [287]:

$$\vec{\nabla}\cdot\boldsymbol{\sigma} = \begin{pmatrix} \frac{\partial\sigma_{rr}}{\partial r} + \frac{\sigma_{rr}-\sigma_{\theta\theta}}{r} + \frac{1}{r}\frac{\partial\sigma_{\theta r}}{\partial\theta} \\ \\ \frac{\partial\sigma_{r\theta}}{\partial r} + \frac{1}{r}\frac{\sigma_{\theta\theta}}{\partial\theta} + \frac{\sigma_{r\theta}+\sigma_{\theta r}}{r} \end{pmatrix} \tag{36.7}$$

Given the diagonal nature of the stress tensor this simplifies to (also remember that $\partial_\theta = 0$):

$$\vec{\nabla}\cdot\boldsymbol{\sigma} = \begin{pmatrix} \frac{\partial\sigma_{rr}}{\partial r} + \frac{\sigma_{rr}-\sigma_{\theta\theta}}{r} \\ \\ 0 \end{pmatrix} \tag{36.8}$$

Focusing on the r-component of the stress divergence:

$$\begin{aligned}
(\vec{\nabla}\cdot\boldsymbol{\sigma})_r &= \frac{\partial\sigma_{rr}}{\partial r} + \frac{\sigma_{rr}-\sigma_{\theta\theta}}{r} && (36.9)\\
&= \frac{\partial}{\partial r}\left[\lambda\frac{1}{r}\frac{\partial(rv_r)}{\partial r} + 2\mu\frac{\partial v_r}{\partial r}\right] + \frac{1}{r}\left[\lambda\frac{1}{r}\frac{\partial(rv_r)}{\partial r} + 2\mu\frac{\partial v_r}{\partial r} - \lambda\frac{1}{r}\frac{\partial(rv_r)}{\partial r} - 2\mu\frac{v_r}{r}\right] && (36.10)\\
&= \lambda\frac{\partial}{\partial r}\frac{1}{r}\frac{\partial(rv_r)}{\partial r} + 2\mu\frac{\partial^2 v_r}{\partial r^2} + \lambda\frac{1}{r^2}\frac{\partial(rv_r)}{\partial r} + \frac{2\mu}{r}\frac{\partial v_r}{\partial r} - \lambda\frac{1}{r^2}\frac{\partial(rv_r)}{\partial r} - \frac{2\mu v_r}{r^2} && (36.11)\\
&= \lambda(-\frac{v_r}{r^2} + \frac{1}{r}\frac{\partial v_r}{\partial r} + \frac{\partial^2 v_r}{\partial r^2}) + 2\mu\frac{\partial^2 v_r}{\partial r^2} + \frac{2\mu}{r}\frac{\partial v_r}{\partial r} - \frac{2\mu v_r}{r^2} && (36.12)\\
&= -(2\mu+\lambda)\frac{v_r}{r^2} + (2\mu+\lambda)\frac{1}{r}\frac{\partial v_r}{\partial r} + (2\mu+\lambda)\frac{\partial^2 v_r}{\partial r^2} && (36.13)
\end{aligned}$$

So the momentum conservation in the r direction is

$$(\vec{\nabla}\cdot\boldsymbol{\sigma} + \rho_0\vec{g})_r = -(2\mu+\lambda)\frac{v_r}{r^2} + (2\mu+\lambda)\frac{1}{r}\frac{\partial v_r}{\partial r} + (2\mu+\lambda)\frac{\partial^2 v_r}{\partial r^2} - \rho_0 g_0 = 0 \tag{36.14}$$

or,

$$\boxed{\frac{\partial^2 v_r}{\partial r^2} + \frac{1}{r}\frac{\partial v_r}{\partial r} - \frac{v_r}{r^2} = \frac{\rho_0 g_0}{\lambda+2\mu}} \tag{36.15}$$

We now look at the boundary conditions. On the inner boundary we prescribe $v_r(r = R_1) = 0$ while free surface boundary conditions are prescribed on the outer boundary, i.e. $\boldsymbol{\sigma}\cdot\vec{n} = \vec{0}$ (i.e. there is no force applied on the surface).

The general form of the solution can then be obtained:

$$v_r(r) = C_1 r^2 + C_2 r + \frac{C_3}{r} \tag{36.16}$$

with

$$C_1 = \frac{\rho_0 g_0}{3(\lambda+2\mu)} \qquad C_2 = -C_1 R_1 - \frac{C_3}{R_1^2} \qquad C_3 = \frac{k_1+k_2}{(R_1^2+R_2^2)(2\mu+\lambda) + (R_2^2-R_1^2)\lambda} \tag{36.17}$$

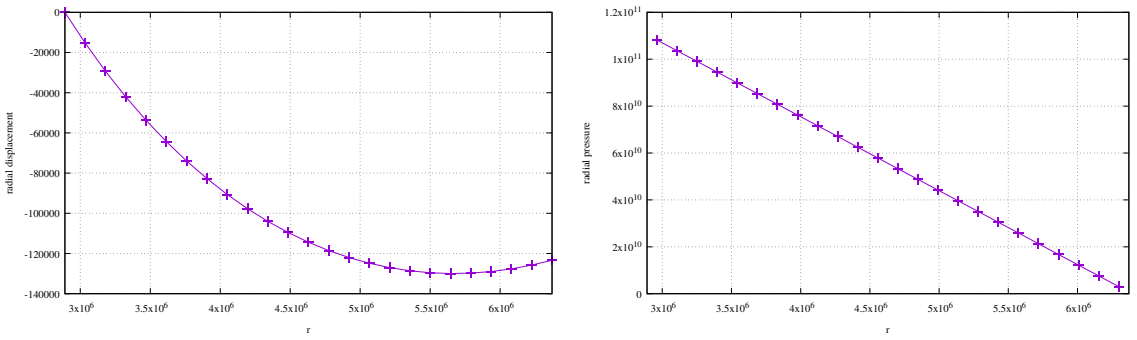
and

$$k_1 = (2\mu+\lambda)C_1(2R_1^2R_2^3 - R_1^3R_2^2) \qquad k_2 = \lambda C_1(R_1^2R_2^3 - R_1^3R_2^2) \tag{36.18}$$

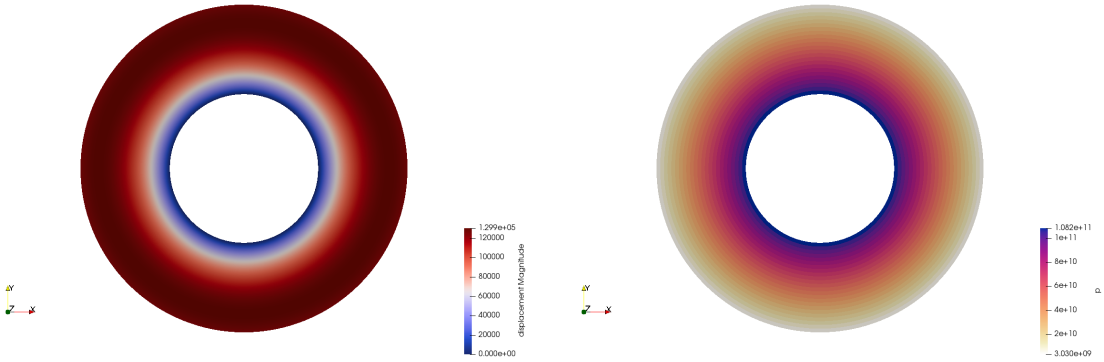
Pressure can then be computed as follows:

$$p = -\lambda\vec{\nabla}\cdot\vec{\mathbf{v}} = -\lambda\left(\frac{1}{r}\frac{\partial(rv_r)}{\partial r}\right) = -\lambda\left(\frac{1}{r}(3C_1r^2 + 2C_2r)\right) = -\lambda(3C_1r + 2C_2) \tag{36.19}$$

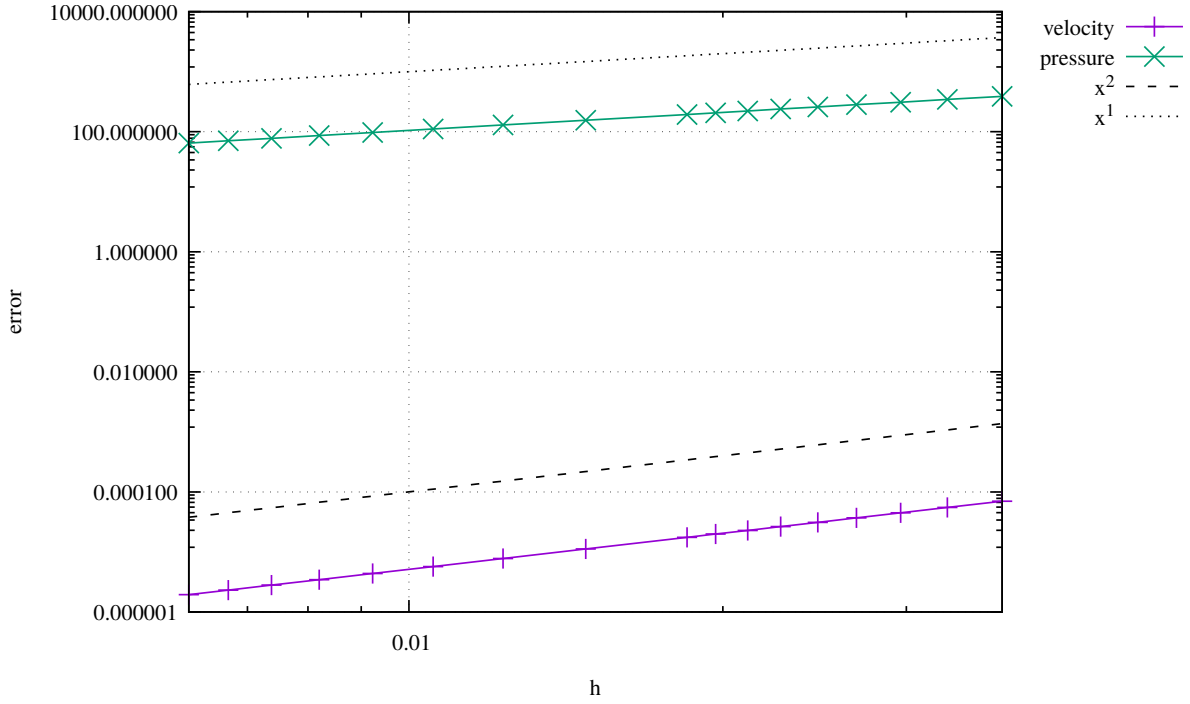
We choose $R_1 = 2890$km, $R_2 = 6371$km, $g_0 = 9.81\text{ms}^{-2}$, $\rho_0 = 3300$, $E = 6\cdot 10^{10}$, $\nu = 0.49$.



radial profiles of the displacement and pressure fields



displacement and pressure fields in the domain



Stone 37

marker advection and population control

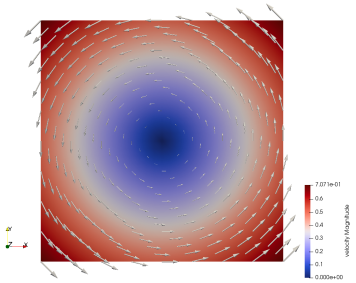
PIC

2D
box
analytical_benchmark
time_stepping
paraview_output
PIC

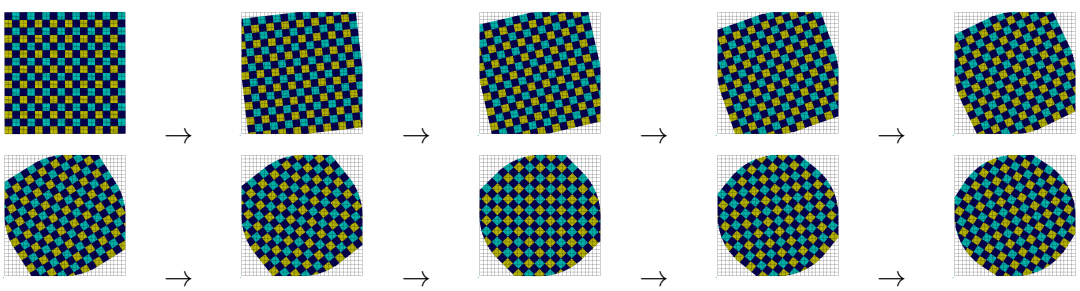
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_37`

The domain is a unit square. The Stokes equations are not solved, the velocity is prescribed everywhere in the domain as follows:

$$\begin{aligned} u(x,y) &= -(y-0.5) \\ v(x,y) &= (x-0.5) \end{aligned}$$



If `RKorder` is zero, then velocity is computed on the particle itself via the equation above. If `RKorder` is 1,2,3,4,5 then the corresponding Runge-Kutta algorithm (in space) is used and basis functions are used to interpolate the velocity from the nodes onto the particles.. Particles can be placed on a regular grid or randomly inside each element at the beginning. The timestep is controlled via a CFL condition. When particles are advected outside the domain, they are arbitrarily placed at location (-0.0123,-0.0123).



I will some day add a population control algorithm to replenish elements with not enough particles.

Stone 38

convection in a 700km box with T b.c. at the top ($Q_1 \times P_0$-penalty)



2D
box
Q1P0
penalty
sparse_matrix
numerical_benchmark
stokes_equation
energy_equation
thermo_mechanical
time_stepping
buoyancy_driven
vrms
nusselt_number
rayleigh_number
paraview_output
steady_state

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_38`

This STONE is based on McKenzie, Roberts, and Weiss [233] (1974) published in J. Fluid Mech. As for every STONE aiming at reproducing results off a publication I here include de abstract of the article:

> Plate tectonics provides a remarkably accurate kinematic description of the motion of the earth's crust but a fully dynamical theory requires a n under- standing of convection in the mantle. Thus the properties of plates and of the mantle must be related to a systematic study of convection. This paper reviews both the geophysical information and the fluid dynamics of convection in a Boussinesq fluid of infinite Prandtl number. Numerical experiments have been carried out on several simple two-dimensional models, in which convection is driven by imposed horizontal temperature gradients or else by heating either internally or from below. The results are presented and analysed in terms of simple physical models. Although the computations are highly idealized and omit variation of viscosity and other major features of mantle convection, they can be related to geophysical measurements. I n particular, the external gravity field depends on changes in surface elevation; this suggests an observational means of investigating convection in the upper mantle.

The domain is a 2D square of size L. The fluid is incompressible, and the Boussinesq approximation is used. Boundary conditions are free slip on all sides. $T = 0$ is prescribed at the bottom and $T(x) = T_0 \cos(\pi x/L)$ at the top. The two sides are insulated.

Following Sato & Thompson (1976) [279], parameters are $\kappa = 1.5 \cdot 10^{-6}$ m^2 s^{-1}, $\rho_0 = 3700$ kg m^{-3}, $\alpha = 2 \cdot 10^{-5}$ K^{-1}, $\nu = \eta/\rho_0 = 2 \cdot 10^{17}$ m^2 s, $g = 10$ m s^{-2}, $C_p = 1200$ J kg^{-1} K^{-1}, $T_0 = 0.1, 1, 10$ K.

We then deduce that $k = \kappa \rho_0 C_p = 6.66$. The Rayleigh number is given by

$$\mathsf{Ra} = \frac{\rho_0 g \alpha T_0 L^3}{\kappa \eta} = \frac{g \alpha T_0 L^3}{\kappa \nu}$$

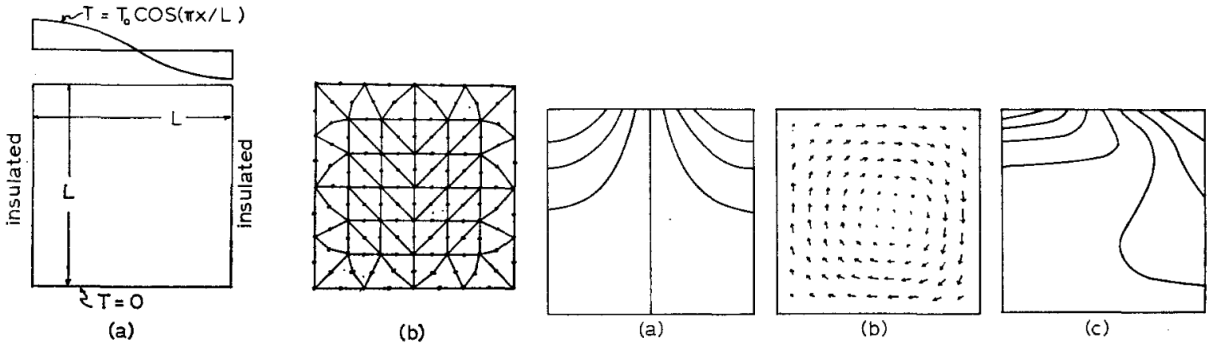
In the paper three values $\mathsf{Ra} = 45.8, 158, 4580$ are used corresponding to $T_0 = 0.1, 1, 10$. However there is a problem since then

$$L = \left(\frac{\mathsf{Ra} \kappa \eta}{\rho_0 g \alpha} T_0 \right)^{1/3} = \left(\frac{458 \cdot 1.5e-6 \cdot 7.4e20}{3700 \cdot 10 \cdot 2e-5} 1 \right)^{1/3} = (6.87e17)^{1/3} \simeq 882.361\text{km}$$

and the vertical dimension of the box is given in [233] as 700km. We then find that there is effectively a factor 2 missing in the Rayleigh number in order to reconcile the above parameters with the desired Rayleigh numbers... Given how very plausible κ, C_p, α and ρ_0 are, I choose to decrease the viscosity ν by a factor two. We then have

$$\{45.733, 457.33, 4573.3\} = \frac{\rho_0 g \alpha L^3}{\kappa \eta} \{0.1, 1, 10\}$$

Still not sure how/why in the paper (see caption of Fig. 4 of [233]) they report Rayleigh numbers 45.8, 458, 4580...?



Left: (a) Boundary conditions; (b) Mesh layout - the authors use $P_2 \times P_1$ elements;

Right: Isotherms and velocity field: (a) Steady-state conduction; velocity field; (c) Steady-state isotherms R = 4580.

The code is based on STONE 03, and because velocities are very small for low Rayleigh numbers reduced densities are used, i.e. the buoyancy force in the Stokes equation is given by $-\alpha(T - T_0)\vec{g}$. Steady state is reached when the velocity and temperature fields do not change by more than $tol = 10^{-6}$.

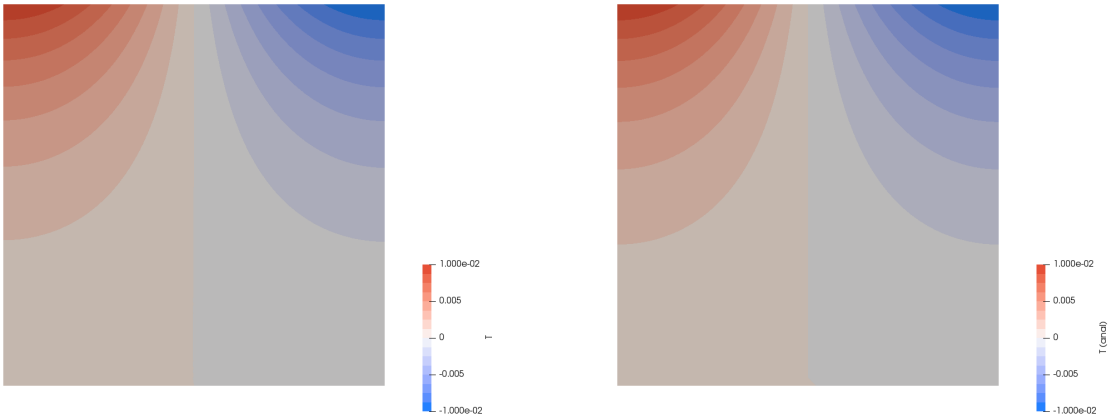
Note that what is called Nu in the code is in fact $Q = \int_0^{L_x} q_y dx$.

I have also run the very same experiments with ASPECT at mesh uniform level 5 (and found that level 6 did not change anything to the results). The prm file is hereby joined to this repository.

As explained in McKenzie *et al.* (1974), for $\mathsf{Ra} << 1$ advection of heat is negligible and T is therefore harmonic:

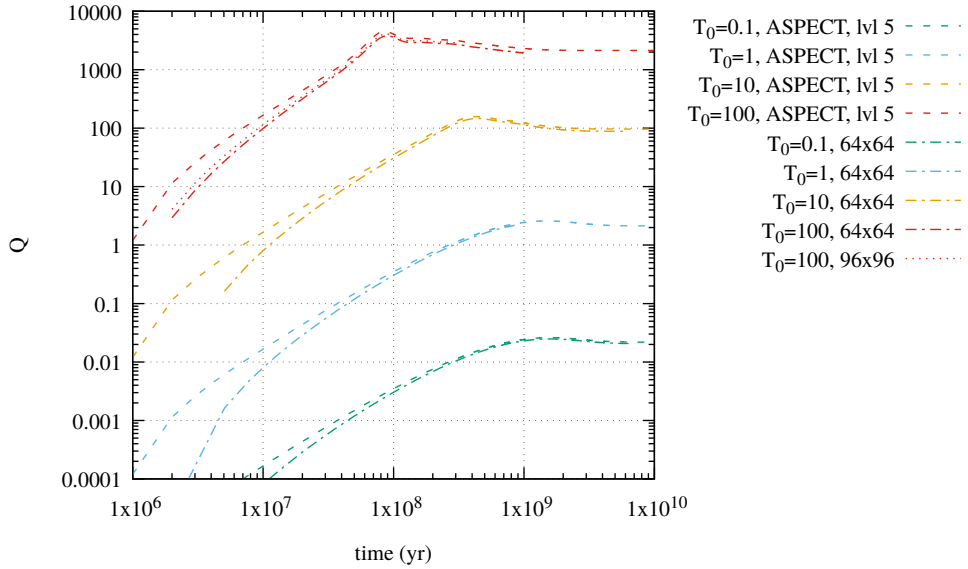
$$T(x, y) = \Delta T \cos(\pi x/L) \frac{\sinh \pi z/L}{\sinh \pi L_y/L_x}$$

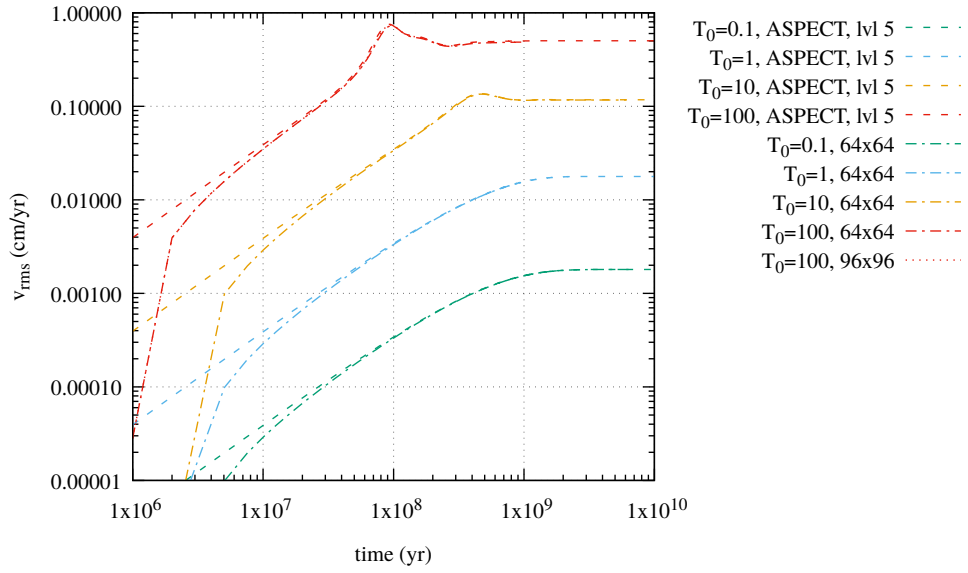
When $T_0 = 0.01$, only heat diffusion is present and we recover the analytical solution presented in McKenzie *et al.* :



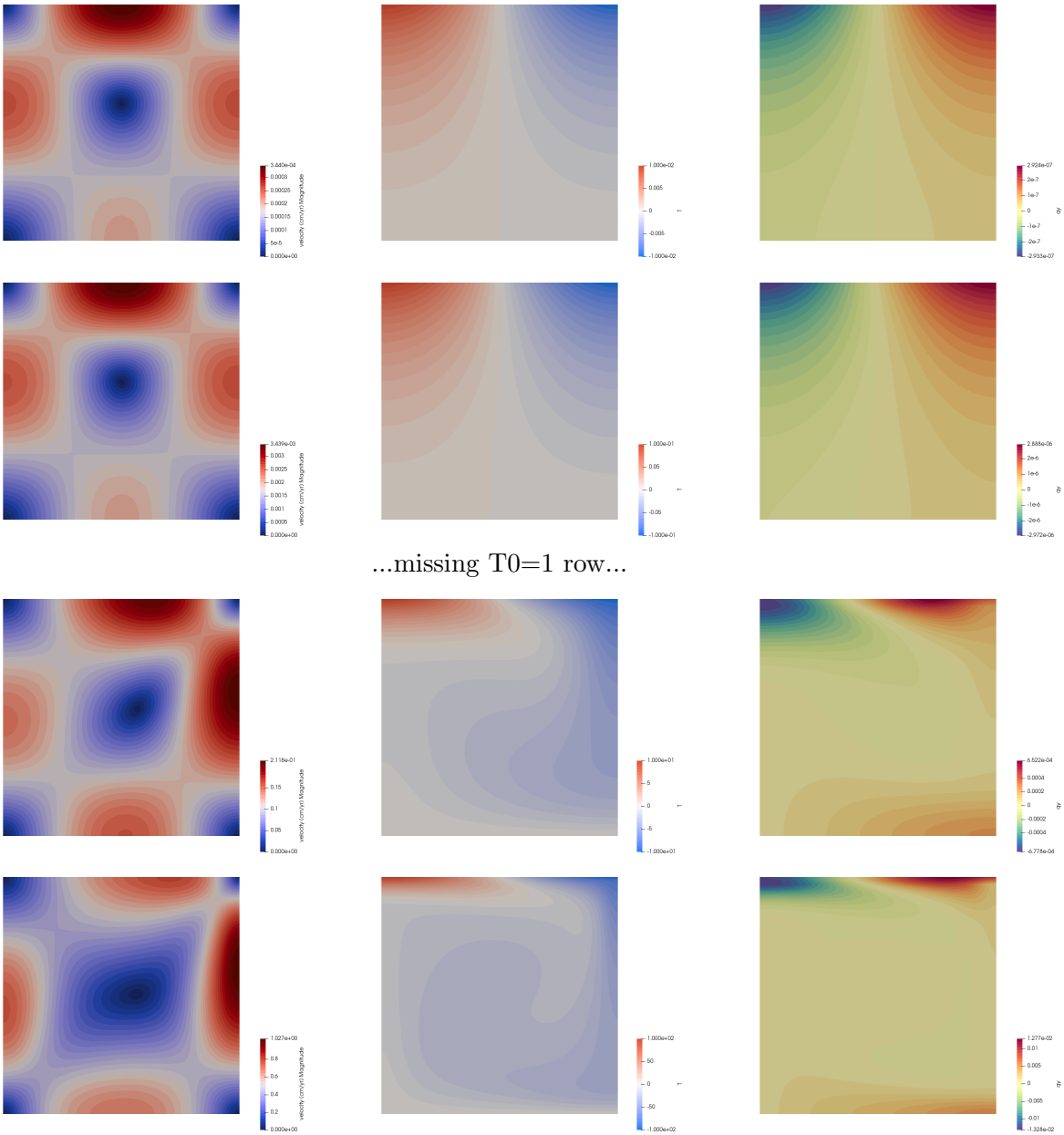
Left: computed temperature, 48 × 48 resolution; Right: analytical solution.

Steady-state study





Heat flux on top boundary (top) and root mean square velocity (bottom) as measured with STONE 38 and ASPECT on 64 × 64 mesh.



...missing T0=1 row...

Obtained with STONE 38 on 64×64 mesh. Top to bottom: $T_0 = 0.01, 0.1, 1, 10, 100$

TODO: boundary conditions should be done on elemental matrices.

Stone 39

The dilation rate approach of Choi and Petersen [69] ($Q_2 \times Q_1$)



2D
box
Q2Q1
sparse_matrix
numerical_benchmark
stokes_equation
nonlinear
viscous_rheology
plastic_rheology
residual
paraview_output
pressure_scaling

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_39`

Choi and Petersen [69] (2015) offer a solution to the problem of the angle of shear bands in geodynamic models. The underlying idea is based on simple modifications brought to existing incompressible flow codes. Note that the codes featured in that paper also implemented elastic behaviour but this can be easily switched off by setting $Z = 1$ in their equations.

Their plasticity implementation starts with a modification of the continuity equation:

$$\vec{\nabla} \cdot \vec{\mathrm{v}} = R = 2 \sin\psi \, \dot{\varepsilon}_p$$

where R is the dilation rate, ψ is the dilation angle and $\dot{\varepsilon}_p$ is the square root of the second invariant of the plastic strain rate tensor.

Under this assumption, the deviatoric strain rate tensor is given by

$$\dot{\boldsymbol{\varepsilon}}^d(\vec{\mathrm{v}}) = \dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}}) - \frac{1}{3} Tr[\dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}})] \mathbf{1} = \dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}}) - \frac{1}{3} \vec{\nabla} \cdot \vec{\mathrm{v}} \, \mathbf{1} = \dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}}) - \frac{1}{3} R \, \mathbf{1} \tag{39.1}$$

Turning now to the momentum conservation equation:

$$\begin{aligned} -\vec{\nabla} p + \vec{\nabla} \cdot \boldsymbol{\tau} &= -\vec{\nabla} p + \vec{\nabla} \cdot (2\eta \dot{\boldsymbol{\varepsilon}}^d(\vec{\mathrm{v}})) \\ &= -\vec{\nabla} p + \vec{\nabla} \cdot \left[2\eta \left(\dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}}) - \frac{1}{3} R \, \mathbf{1} \right) \right] \\ &= -\vec{\nabla} p + \vec{\nabla} \cdot (2\eta \dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}})) - \frac{2}{3} \vec{\nabla}(\eta R) \end{aligned} \tag{39.2}$$

The last term is then an addition to the right hand side of the momentum equation and its weak form is as follows:

$$\vec{f}' = \int_\Omega N_v \frac{2}{3}\vec{\nabla}(\eta R) dV = \frac{4}{3}\sin\psi \int_\Omega N_v \vec{\nabla}(\eta \dot{\boldsymbol{\varepsilon}}_p) dV \tag{39.3}$$

This formulation proves to be problematic since in order to compute the gradient, we would need the viscosity and the plastic strain rate on the mesh nodes and both these quantities are effectively computed on the quadrature points. One option could be to project those quadrature values onto the nodes, which may introduce interpolation errors/artefacts and/or smoothing. Another option is to resort to integration by parts:

$$\int_\Omega N_v \vec{\nabla}(\eta \dot{\boldsymbol{\varepsilon}}_p) dV = [N_v \eta \dot{\boldsymbol{\varepsilon}}_p]_\Gamma - \int_\Omega \vec{\nabla} N_v (\eta \dot{\boldsymbol{\varepsilon}}_p) dV \tag{39.4}$$

The last term is now trivial to compute since the shape function derivatives, the viscosity and the plastic strain rate are known at the quadrature points. Remains the surface term. We will neglect it for now to simplify our implementation and note that a) it will not directly affect what happens inside the domain, b) it could be somewhat important when shear bands intersect with the free surface. Then

$$\vec{f}' = -\frac{4}{3}\sin\psi \int_\Omega \vec{\nabla} N_v (\eta \dot{\varepsilon}_p) dV = -\frac{2}{3}\int_\Omega \vec{\nabla} N_v (\eta R) dV \tag{39.5}$$

This is what is implemented in the code.

```
for i in range(0,mV):
    f_el[ndofV*i+0]-=2./3.*dNNNVdx[i]*jcob*weightq*etaq[counter]*dilation_rate
    f_el[ndofV*i+1]-=2./3.*dNNNVdy[i]*jcob*weightq*etaq[counter]*dilation_rate
    h_el[:]-=NNNP[:]*dilation_rate*weightq*jcob
```

In practice we compute the strain rate components at the quadrature points, knowing that the strain rate is not deviatoric there.

We call the approach above method 1.

Although the authors do indicate that they add a term in each rhs, it is not very clear how they deal with the implementation issue above. We then propose an alternative: instead of explicitely removing the deviatoric part of the strain rate as in (39.2) and replace the trace of the tensor by R, one could leave the term inside the matrix, thereby using a compressible form of the viscous block of the Stokes matrix. We will recover the same converged solution as before, but the path to convergence will be different that the first approach. In what follows, we denote the original approach by Choi & Petersen 'method 1' and the latter 'method 2'. The $\boldsymbol{C}$ matrix is then

$$\boldsymbol{C}_{method\ 1} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{C}_{method\ 2} = \begin{pmatrix} 4/3 & -2/3 & 0 \\ -2/3 & 4/3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

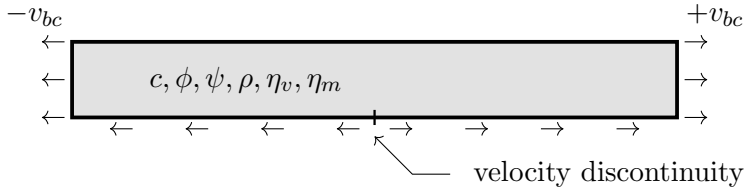
Note that method 2 bypasses the discussion above about the integration by parts and the surface term.

My feeling is that method 1 stems from a FDM vision (i.e. from the paper). Method 2 is elegant, simple.

Remark. *The rhs of the continuity equation is formulated as a function of the (effective) plastic strain rate so that in the case of multiple deformation mechanisms one should be careful about this term.*

Benchmark #1 - The simple brick

The 2D Cartesian domain filled with a single rigid-plastic material characterised by a cohesion $c = 10\text{MPa}$, an angle of friction ϕ, a dilation angle ψ and a density $\rho = 2800\text{kg/m}^3$. Gravity is set to $|g_y| = 10\text{m s}^{-2}$.



Extensional boundary conditions are as follows:

- left boundary: $u = -\mathrm{v}_{bc}$, v free;
- right boundary: $u = +\mathrm{v}_{bc}$, v free;
- bottom boundary: $v = 0$, $u = -\mathrm{v}_{bc}$ for $x < L_x/2$, $u = +\mathrm{v}_{bc}$ for $x > L_x/2$, and $u = 0$ if $x = L_x/2$;
- top boundary: zero traction.

For compressional boundary conditions the signs of all horizontal velocities should be reversed. The nonlinear tolerance is set to $\text{tol} = 10^{-6}$. Nonlinear iterations stop when maximum of the normalised nonlinear residual reaches the desired tolerance. v_{bc} is chosen so that the background strain rate is $\dot{\varepsilon}_{bckgr} = 10^{-15}\text{s}^{-1}$, i.e. $\mathrm{v}_{bc} = \dot{\varepsilon}_{bckgr} L_x/2$.

Following Choi & Petersen [69], we run the experiment with an associative ($\phi = \psi$) plasticity and a non associative one ($\psi = 0$, i.e. $R = 0$). This second approach is essentially what many codes do (i.e. $\vec{\nabla} \cdot \vec{\mathrm{v}} = 0$). We can also run the model with $\phi = \psi = 0°$ so that we recover the von Mises plasticity yield criterion which is independent of pressure.

The velocity, pressure, strain rate, dilation rate, and velocity divergence are shown hereunder both in extension and compression. Note that the domain has been substantially extended in the horizontal direction because of the wide shear band angles obtained in compression. It is now 80km long and 10km high (aspect ratio 8:1).

Three angles are mechanically stable (e.g. [178]):

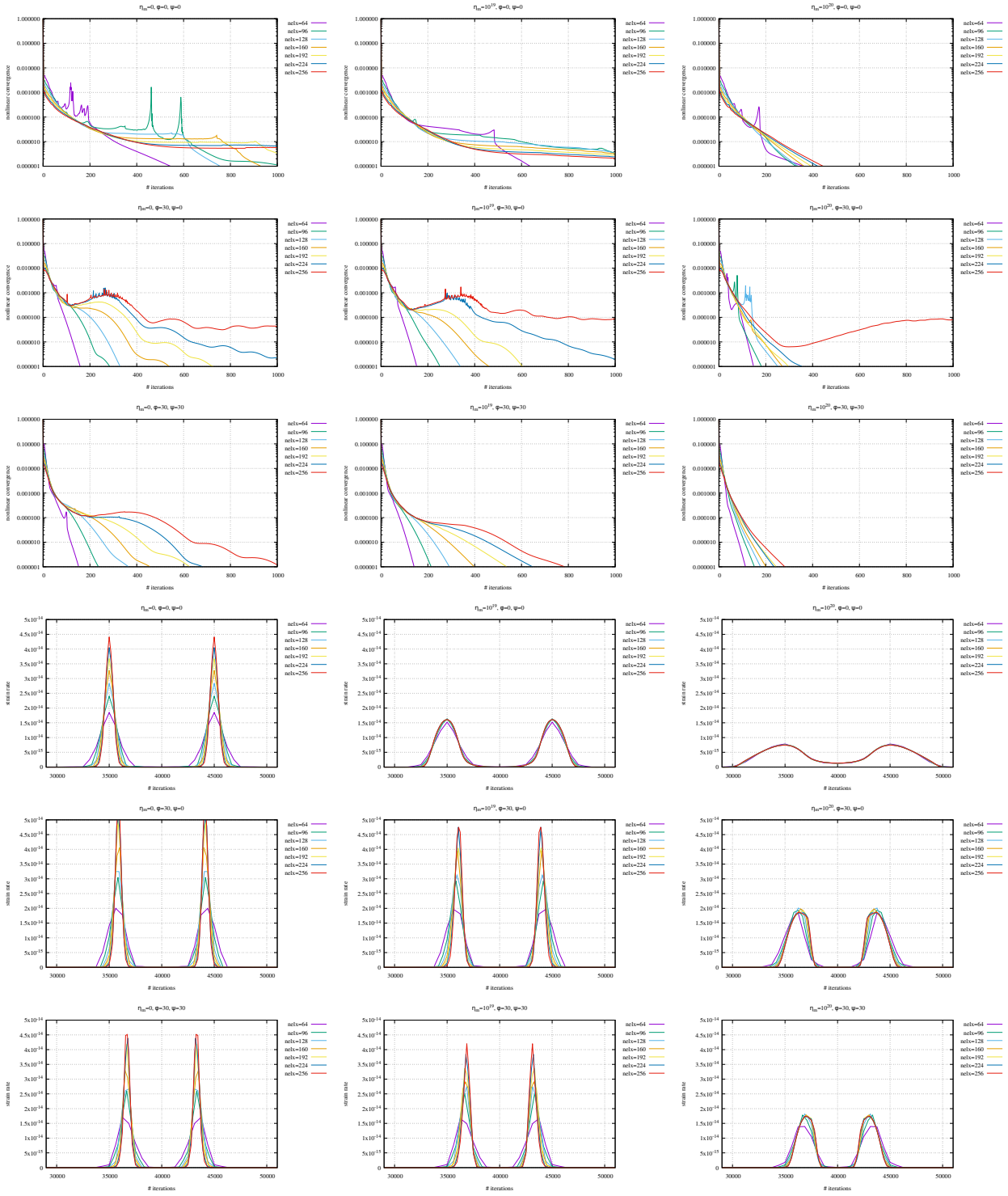
$$\theta = \frac{\pi}{4} \pm \frac{\psi}{2} \qquad \text{Roscoe angle}$$

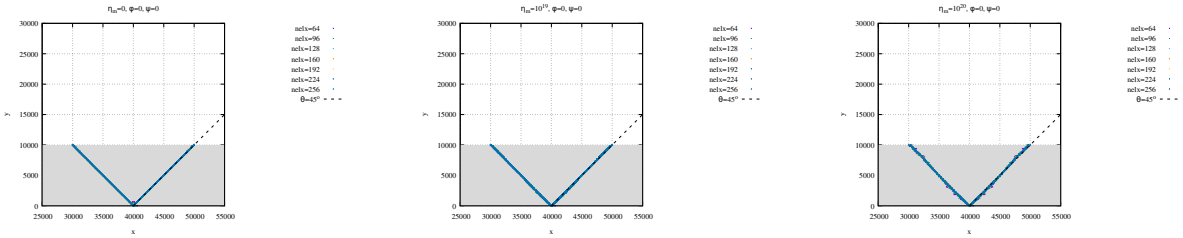
$$\theta = \frac{\pi}{4} \pm \frac{\phi}{2} \qquad \text{Coulomb angle}$$

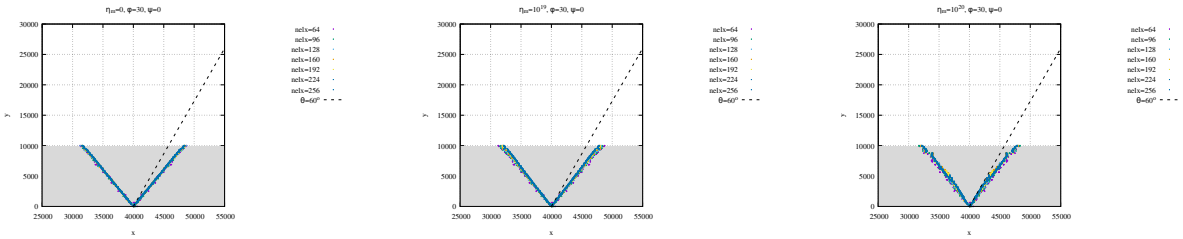
$$\theta = \frac{\pi}{4} \pm \frac{\phi + \phi}{4} \qquad \text{Arthur angle}$$

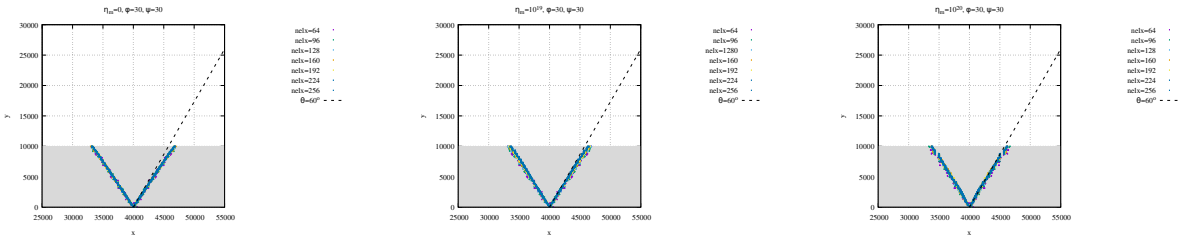
In the case of associative plasticity, $\phi = \psi$, so that all three angles are the same. Per row of elements, nodes or quadrature points, and per half of the domain (left and right) we find the location with the highest strain-rate and record their coordinates.

Extension

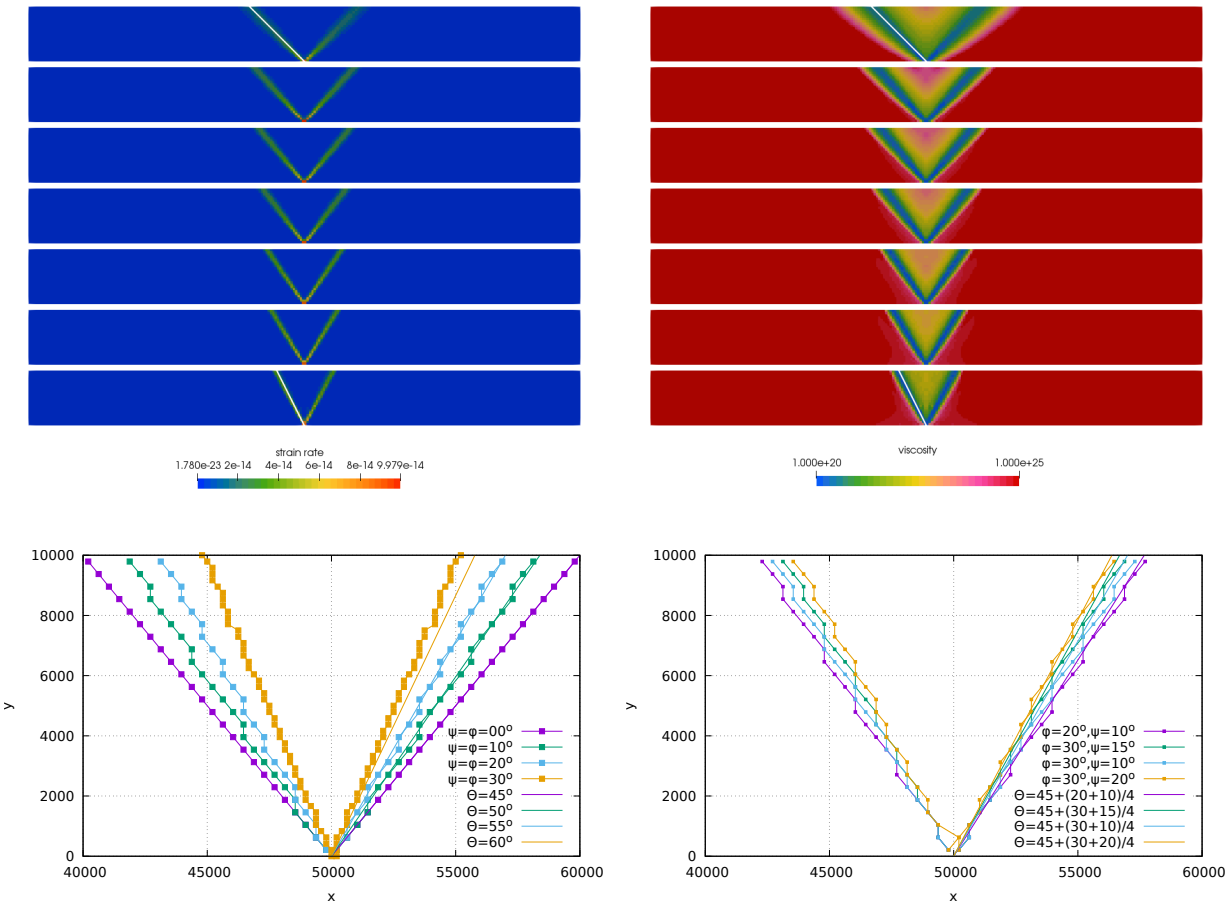

η_{lm}=0, φ=0, ψ=0
η_{lm}=10^{10}, φ=0, ψ=0
η_{lm}=10^{20}, φ=0, ψ=0
η_{lm}=0, φ=30, ψ=0
η_{lm}=10^{10}, φ=30, ψ=0
η_{lm}=10^{20}, φ=30, ψ=0
η_{lm}=0, φ=30, ψ=30
η_{lm}=10^{10}, φ=30, ψ=30
η_{lm}=10^{20}, φ=30, ψ=30
nelx=64
nelx=96
nelx=128
nelx=160
nelx=192
nelx=224
nelx=256
nonlinear convergence
strain rate
iterations


η_{int}=0, φ=0, ψ=0
$\eta_{int}=10^{19}$, φ=0, ψ=0
$\eta_{int}=10^{20}$, φ=0, ψ=0
nelx=64
nelx=96
nelx=128
nelx=160
nelx=192
nelx=224
nelx=256
θ=45°
x
y


η_{int}=0, φ=30, ψ=0
$\eta_{int}=10^{19}$, φ=30, ψ=0
$\eta_{int}=10^{20}$, φ=30, ψ=0
nelx=64
nelx=96
nelx=128
nelx=160
nelx=192
nelx=224
nelx=256
θ=60°
x
y


η_{int}=0, φ=30, ψ=30
$\eta_{int}=10^{19}$, φ=30, ψ=30
$\eta_{int}=10^{20}$, φ=30, ψ=30
nelx=64
nelx=96
nelx=1280
nelx=128
nelx=160
nelx=192
nelx=224
nelx=256
θ=60°
x
y

One can also run the extension model for $\phi = \psi = 0, 5, 10, 15, 20, 25, 30°$:



Results obtained on a 240x24 grid, max 50 nl iterations.

Note that benchmarking this in not easy. One solution Timo and I found was to add a velocity field $\underline{\vec{v}} = (x, y, z)$ (with $\vec{\nabla} \cdot \underline{\vec{v}} = 3$) to an existing analytical problem, e.g. the Burstedde benchamrk.

Benchmark #2 - Spiegelman et al (2016) brick

The setup is *similar* to the one in Spiegelman et al (2016) [302]. It is a 2D Cartesian domain filled with an isoviscous layer at the bottom and a visco-plastic material on top, as shown here:

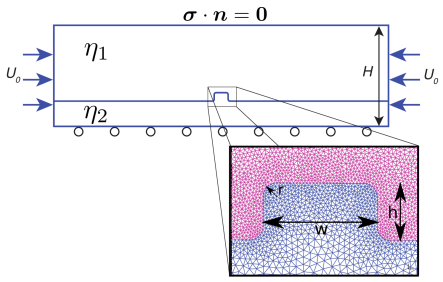


Figure 1. Geometry and boundary conditions for the simple compressional notch test. Inset shows close-up view of the rounded notch along with the typical unstructured mesh resolution.

Table 1. Parameters Used in the Compressional Notch Test

Name	Description	Values
η_1	background viscosity of upper viscoplastic layer	10^{23}–5×10^{24} Pa s
η_2	Viscosity of the lower layer and notch (isoviscous)	10^{21} Pa s
η_0	Reference viscosity for scaling	10^{22} Pa s
H	Domain depth (and scale length)	30 km
H_1	Upper layer depth	$0.75H$
H_2	Lower layer depth	$0.25H$
L	Domain width	$4H$
w	Notch width	$1/6H$
h	Notch height	$1/12H$
r	Radius of curvature of notch corners	$0.02H$
U_0	Horizontal velocity (compression)	0.0025–0.0125 m/yr
C	Cohesion	10^8 Pa
φ	Friction angle	0–30 °
ρ	Density (both layers)	2700 kg m^{-3}
g	Acceleration due to gravity	9.81 m s^{-2}

The domain has dimensions $L_x = 128$km, $L_y = Lx/4$. The lower layer depth is $L_y/4$. The notch has dimensions $w = 4$km and $h = 2$km.

In what follows the nonlinear tolerance is set to 10^{-6}. Due to a lack of resolution, I do not implement the rounded edges of the seed. U_0 is set to $25\text{mm}\,\text{yr}^{-1}$ and the background viscosity of the brittle layer is set to $\eta_v = 10^{24}\text{Pa}\,\text{s}$.

Note that the pressure colour bars in Fig(6) of [302] are most likely not correct at all:

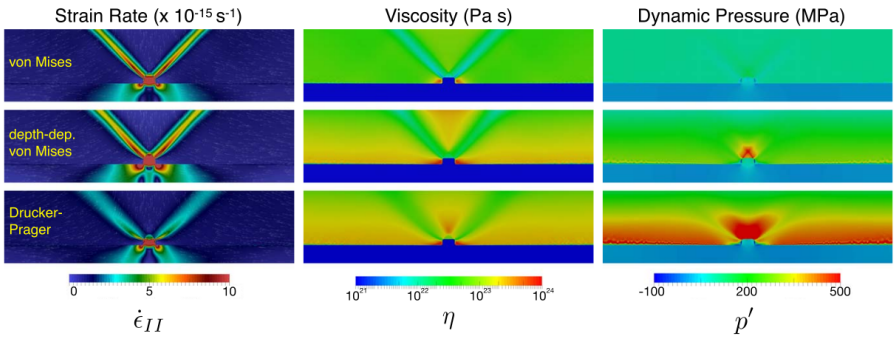
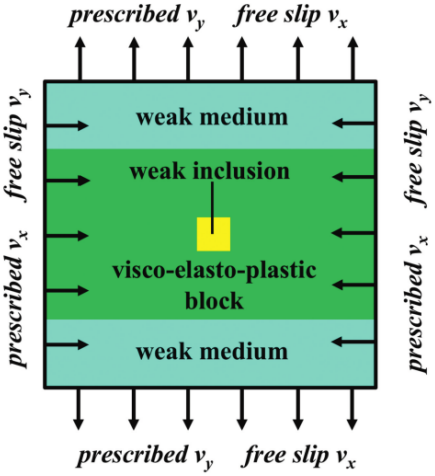


Fig.(6) from Spiegelman et al (2016) [302].

Benchmark #4: Shortening of a visco-plastic bloc - part 1

This benchmark originates in the book of Gerya. In the first edition the domain is 1000x1000km, while it is 100x100km in the second. We keep the second edition as a guideline. The benchmark is actually elasto-visco-plastic but in this stone we neglect the elastic deformation. The boundary conditions are shown here:

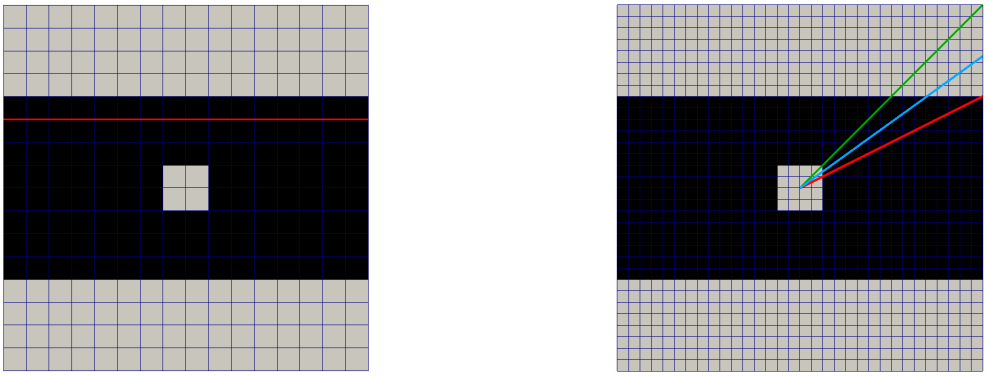


Taken from [129].

The velocity on the boundaries is $\mathrm{v}_{bc} = 5 \cdot 10^{-9}$ m s^{-1} so that the background strain rate is $2\mathrm{v}_{bc}/L = 10^{-13}$s^{-1}. The weak medium and the weak inclusion have a viscosity $\eta = 10^{17}$Pa s while the block is visco-plastic, with $\eta = 10^{23}$Pa s, $c = 10^{8}$Pa and $\phi = 37°$. Pressure is set to zero in the top right corner and later post-processed so as to insure a zero volume average.

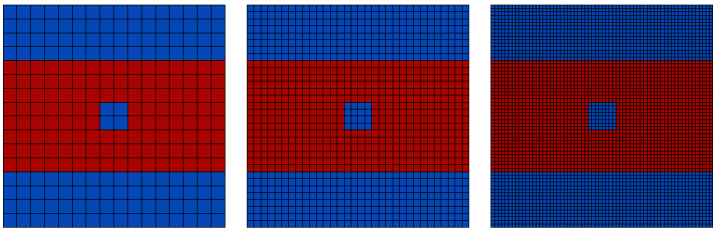
In the case of von Mises ($\phi = 0$) we expect the shear bands at 45°. When $\phi > 0$, as mentioned above we expect:

$$\frac{\pi}{4} - \frac{\phi}{2} = 26.5° \qquad \frac{\pi}{4} - \frac{\phi}{4} = 35.75°$$



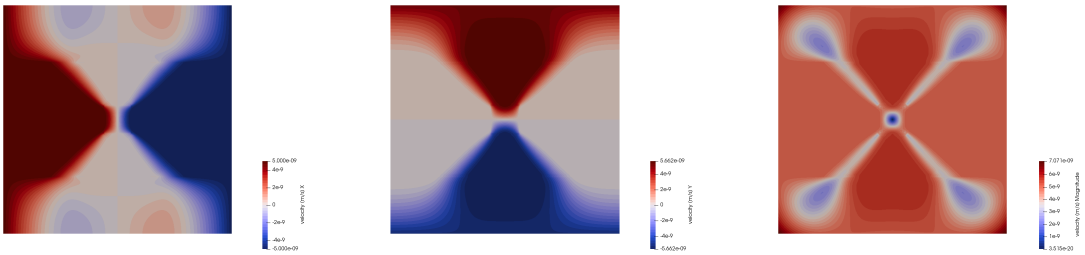
a) Line on which strain rate is measured. b) possible shear band angles: green is 45°, blue is 35.75° and red is 26.5°.

I have adapted slightly the dimensions inside the domain such that resolutions 16x16, 32x32, etc... all showcase elements whose sides exactly align with the material interfaces. The size of the inclusion is $L/8 = 12.5$km and the block has a thickness of 50km:

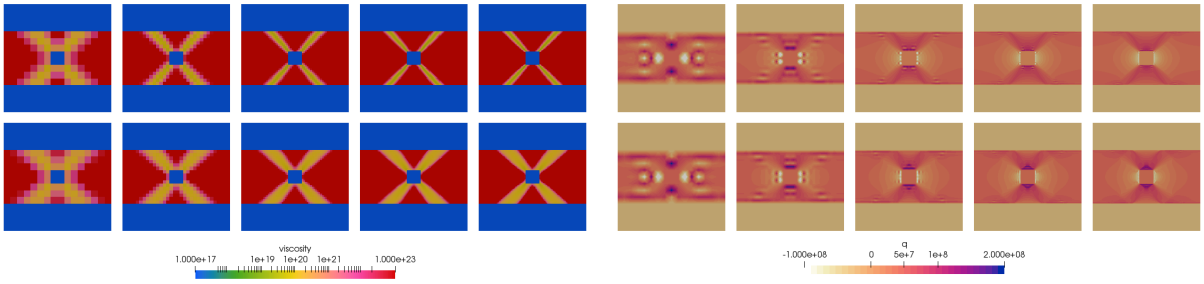


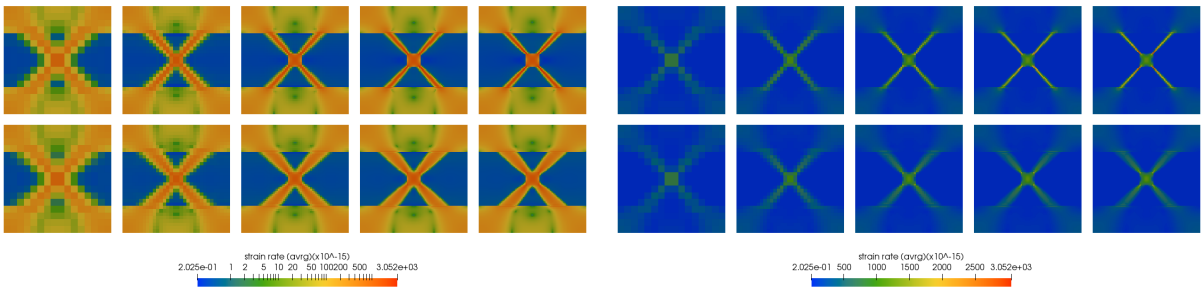
Material geometry for resolutions 16x16, 32x32 and 64x64.

von Mises (no dilation rate):

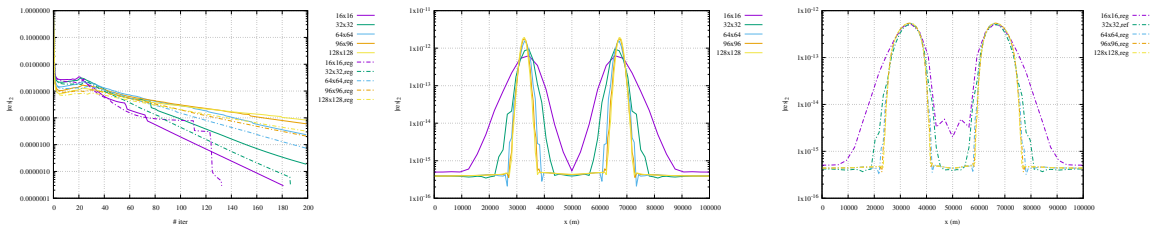


Velocity field for 128x128, $\eta_m = 1e19$



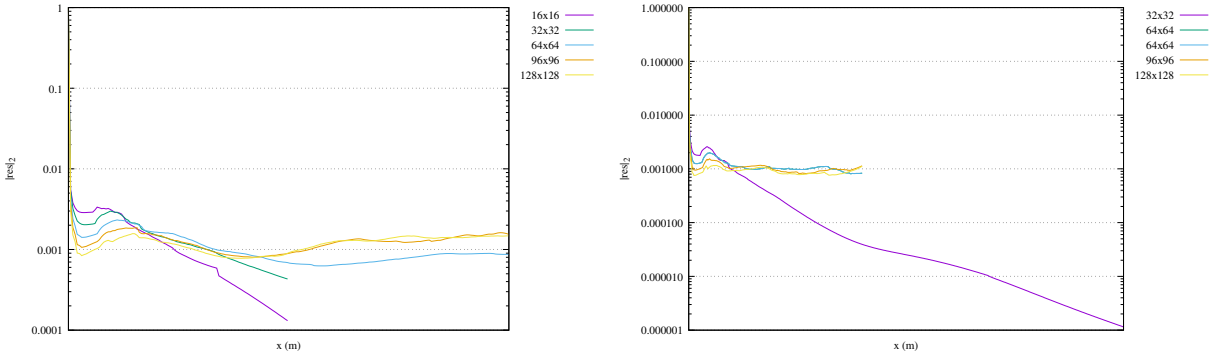


From left to right: resolutions are 16x16, 32x32, 64x64 and 96x96. 100 nonlinear Picard iterations.

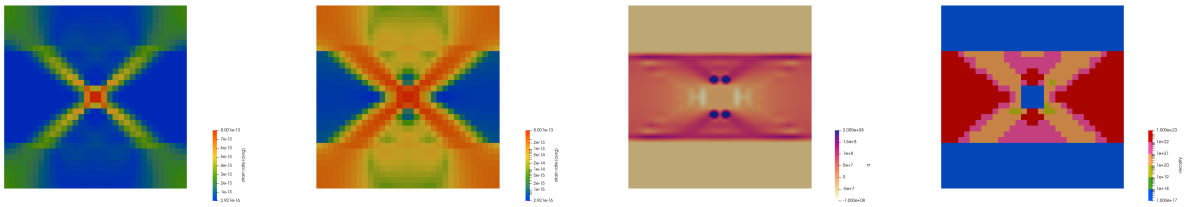


$\eta_m = 1e19$

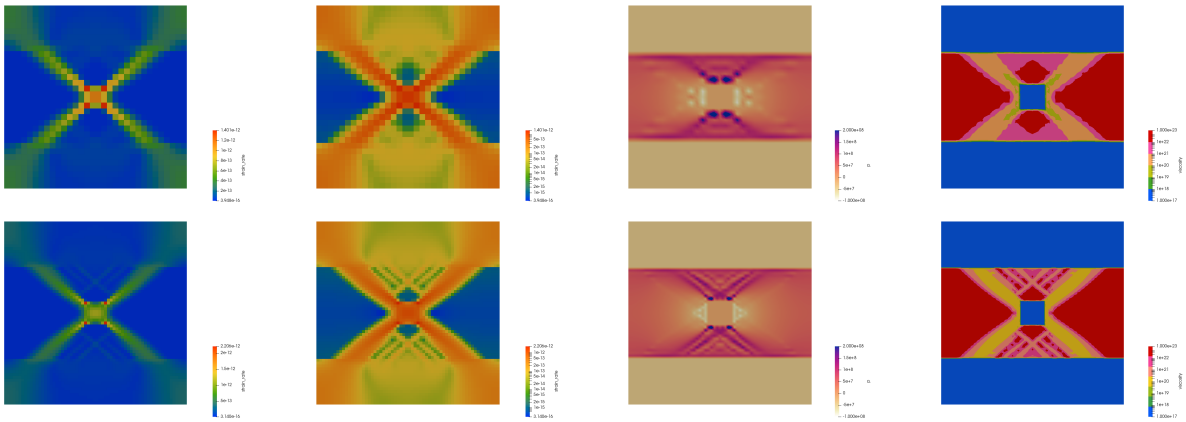
viscous damper, no dilation rate, Drucker-Prager, phi=37:



Left: $\eta_m = 0$, right: $\eta_m = 10^{19}$



Fieldstone 32x32

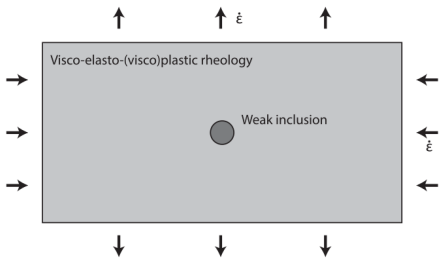


ASPECT 32x32, 64x64

To explore:
- associative and non-associative drucker-prage
- visco-plastic vs visco-viscoplastic

Benchmark #5: Shortening of a visco-plastic bloc - part 2

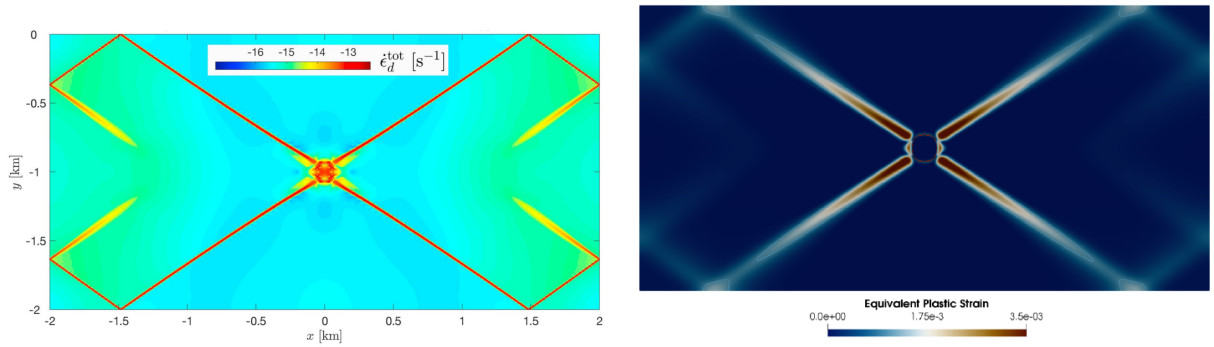
The setup originates in Duretz et al (2018) [98] and is also carried out in Jacquey & Cacace (2020) [168]. We here again neglect the elastic deformation. The domain is 4x2 km. There is a linear viscous inclusion in the middle of radius 100m and viscosity $\eta_{inc} = 10^{17}$Pa s. The material around is characterised by $\eta_v = 10^{24}$pascals, $c = 3e7$pascal, $\phi = 30°$, $psi = 10°$. Boundary conditions are as follows:



In the papers they prescribe $\Delta\varepsilon = 5 \cdot 10^{-5}$ over a time step of $\Delta t = 10^{10}$second, i.e. the background strain rate is $\dot{\varepsilon} = 5 \cdot 10^{-15}\text{s}^{-1}$. Given L_x and L_y we can easily arrive at the velocity values to be prescribed:

$$|u_{bc}| = \dot{\varepsilon}_{bc} L_x/2 \qquad |v_{bc}| = \dot{\varepsilon}_{bc} L_y/2$$

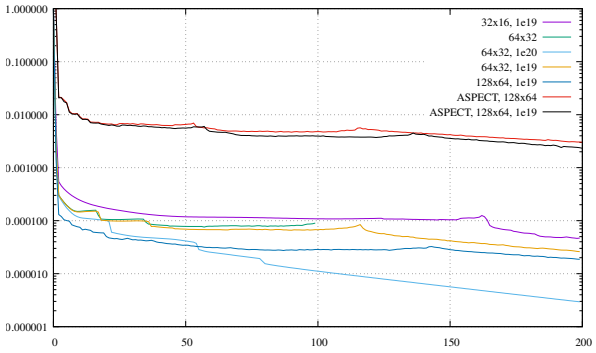
The inclusion is circular, i.e. no amount of mesh refinement will be such that the mesh edges align with it, as opposed to the previous experiment.



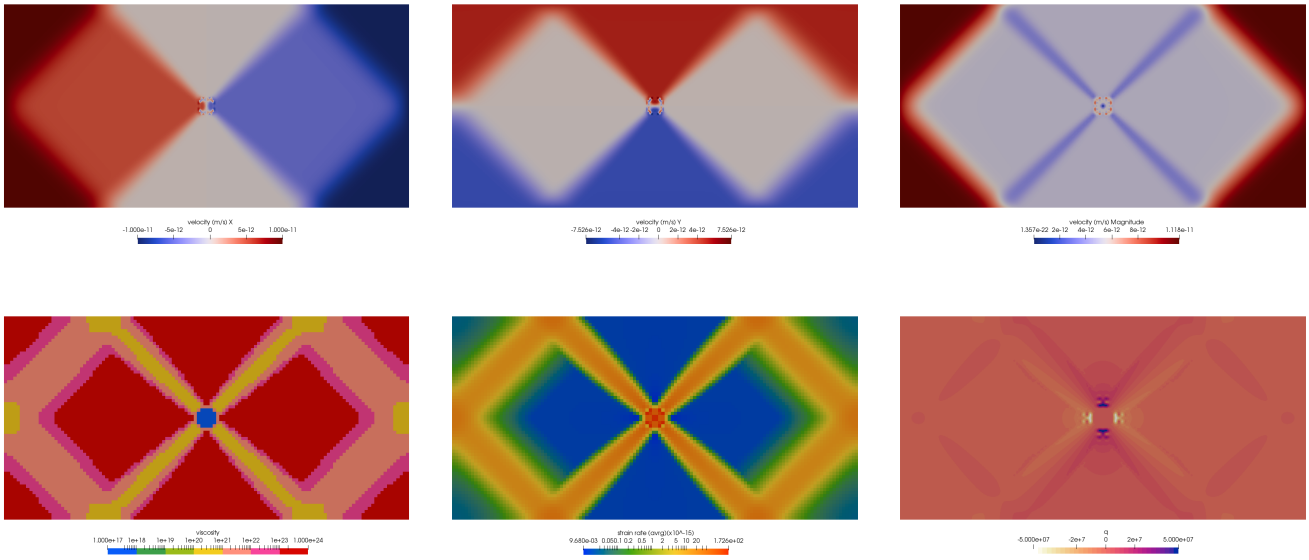
Left: taken from Duretz et al (2018) [98]; Right: taken from Jacquey & Cacace (2020) [168]. Note that these simulations have been run with elasto-visco-plastic rheologies for a certain amount of time so that the propagation of the shear bands is not final/complete, as opposed to the converged visco-plastic solutions shown hereafter.

As before, pressure is set to zero in the top right corner, and further post processed to insure that its volume average is zero.

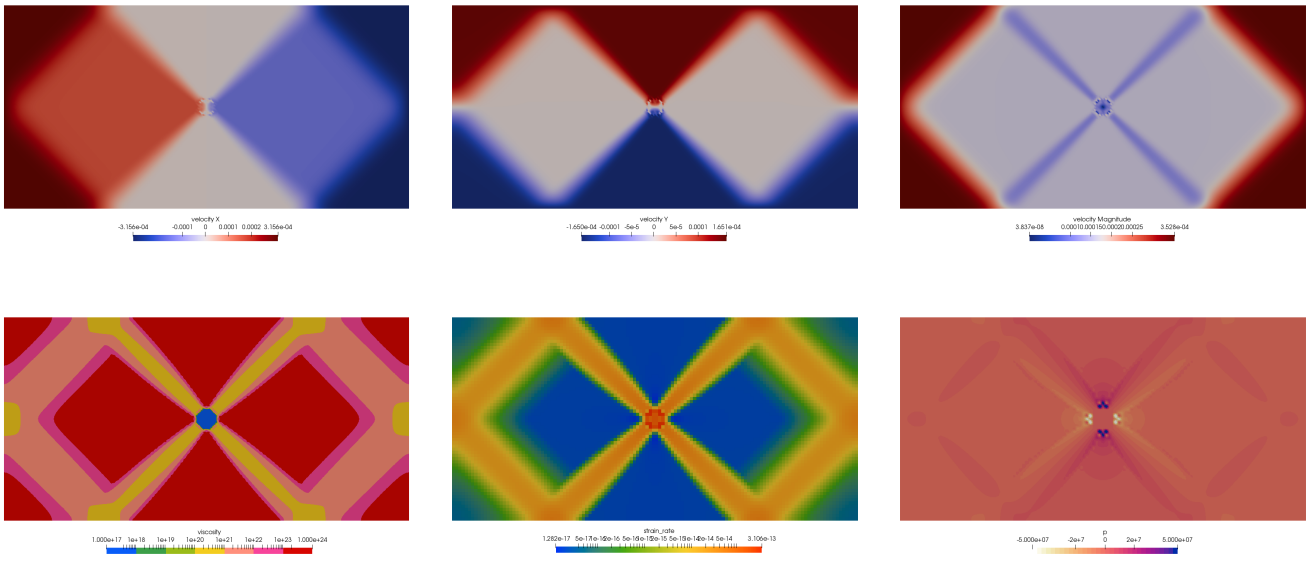
von Mises rheology



Nonlinear convergence, fieldstone and aspect, 128x64, 200 nonlinear iterations, $\eta_m = 10^{19}$.

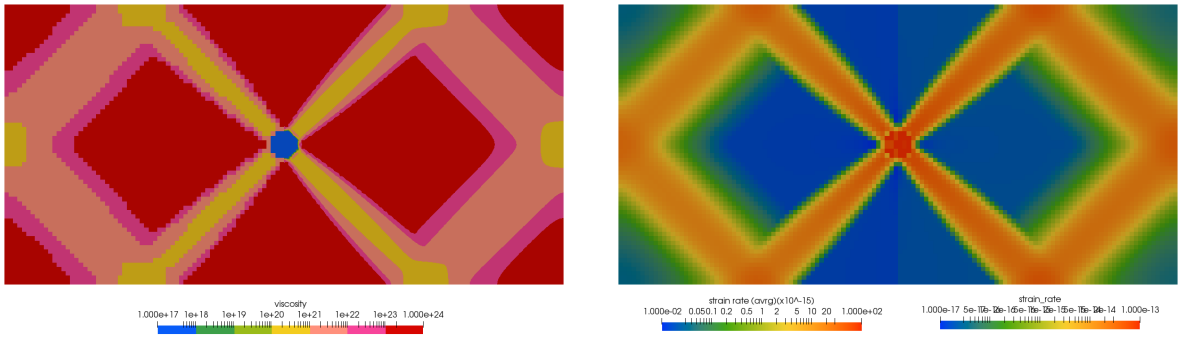


128x64, 200 nonlinear iterations, $\eta_m = 10^{19}$

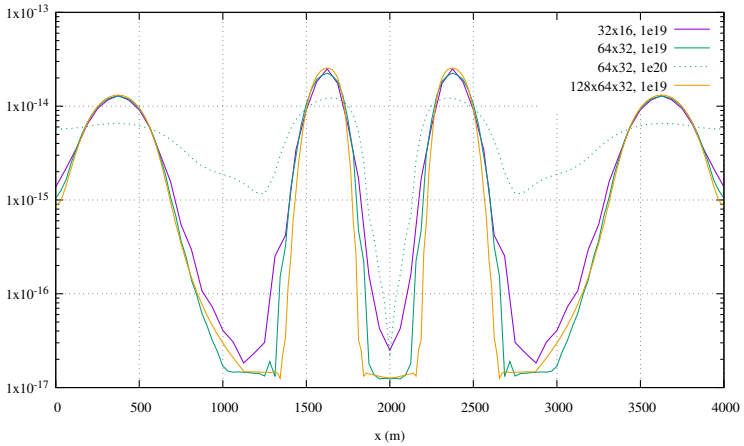


ASPECT: 128x64, 200 nonlinear iterations, $\eta_m = 10^{19}$

In order to help the comparison, I plot the effective viscosity and the strain rate for both next to one another:

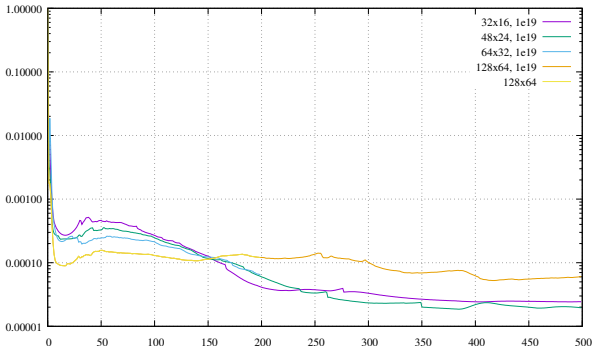


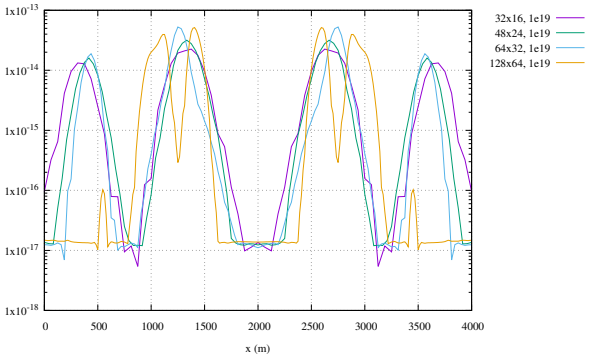
Left half is fieldstone, Right half is aspect.

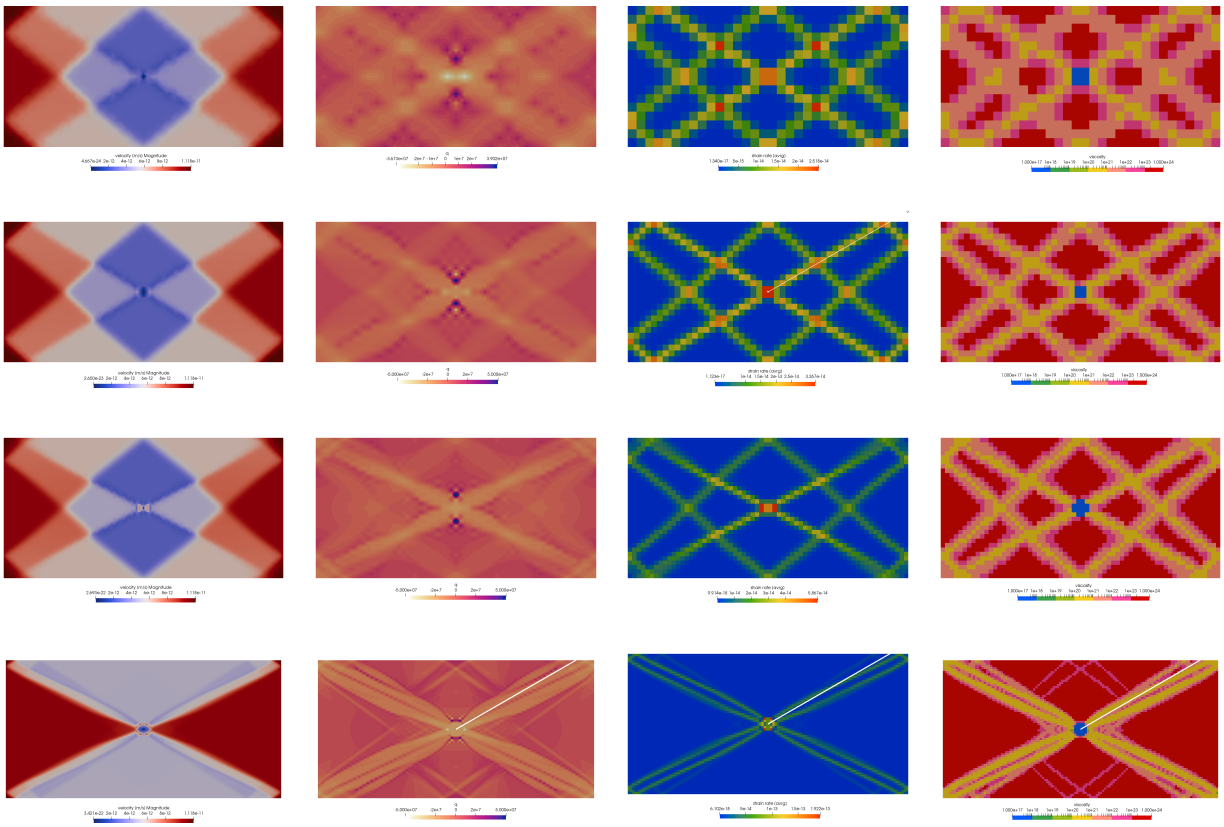


Strain rate cross section at $y = 11L_y/16$.

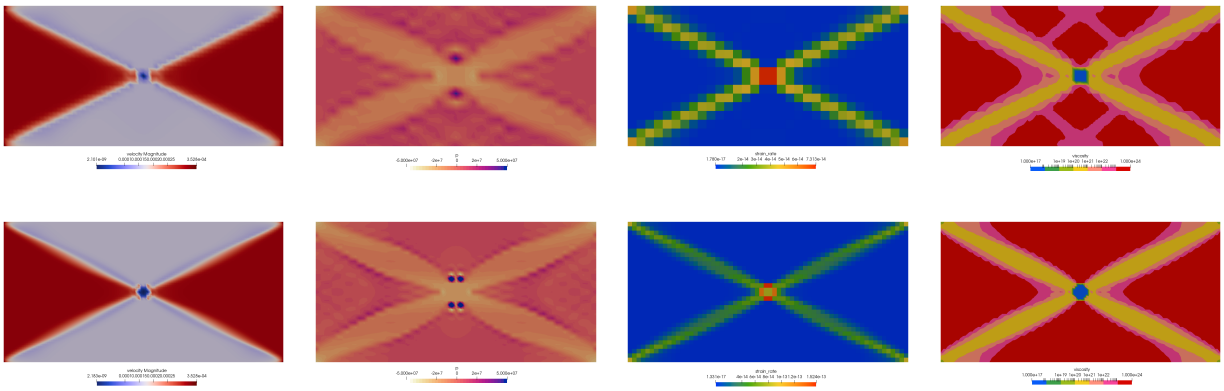
Drucker-Prager rheology ($\phi = 30°$)







Fielstone. Top to bottom: 32x16, 48x24, 64x32, 128x64, etam=1e19, phi=30



ASPECT, 64x32, etam=1e19, phi=30

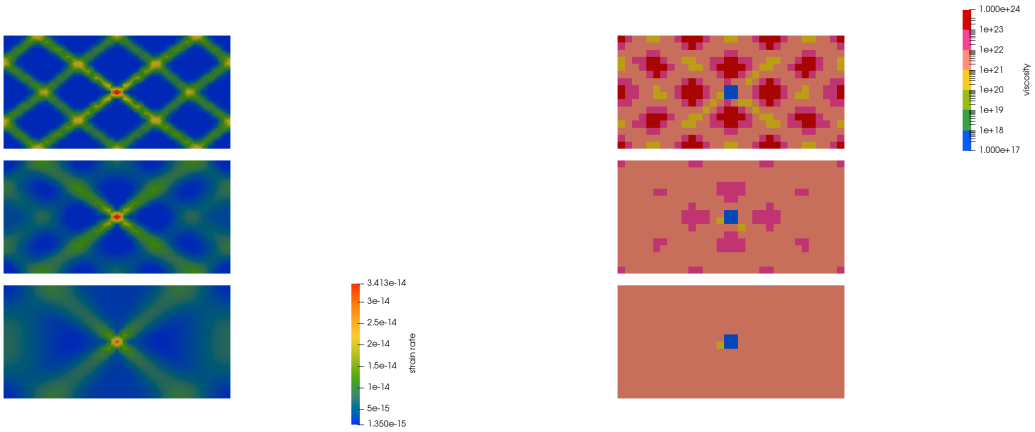
Note that ASPECT does not use the proper partitioning of strain rates, and uses total strain rate in plastic element. This could explain the difference?

A very different approach: diffusion plasticity

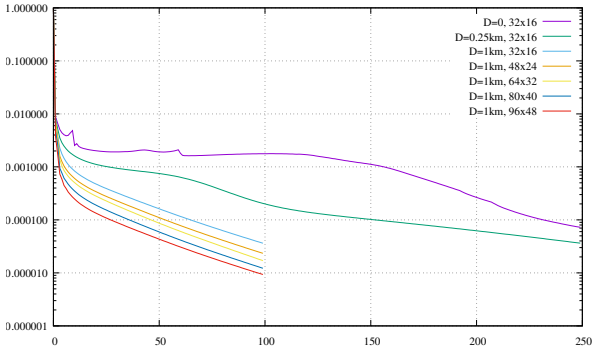
The idea is to solve a diffusion equation for each component of the strain rate tensor. In order to do so, we need nodal values of this tensor and these values must then also be used in the rheology: the shape functions are used to interpolate these values onto the quadrature points (as opposed to the standard way of using the shape function derivatives to interpolate the velocity).

After each nonlinear iteration the nodal fields $\dot{\varepsilon}_{xx}$, $\dot{\varepsilon}_{yy}$ and $\dot{\varepsilon}_{xy}$ are diffused. The diffusion coefficient D value is not straightforward to choose. Also, boundary conditions are unknown so we resort to doing nothing, i.e. zero Neumann boundary conditions.

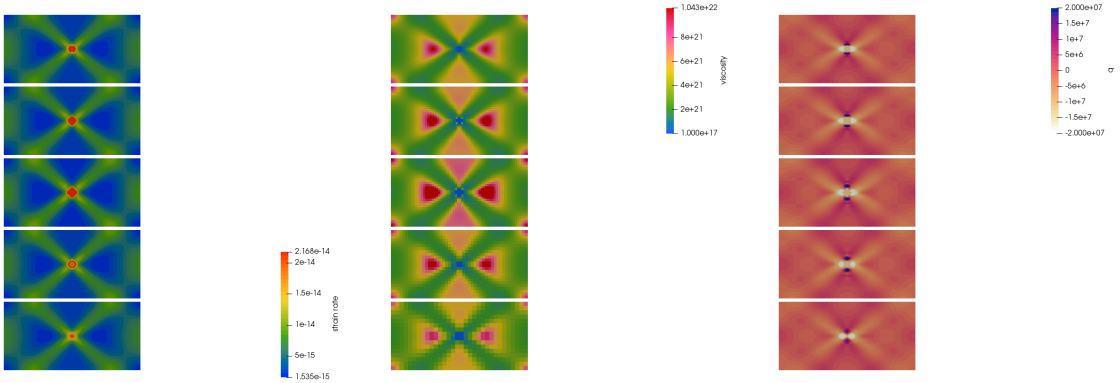
Note that in this case we have $\eta_m = 0$ and that it is tTriggered in the code by *use_srn_diff=True*.



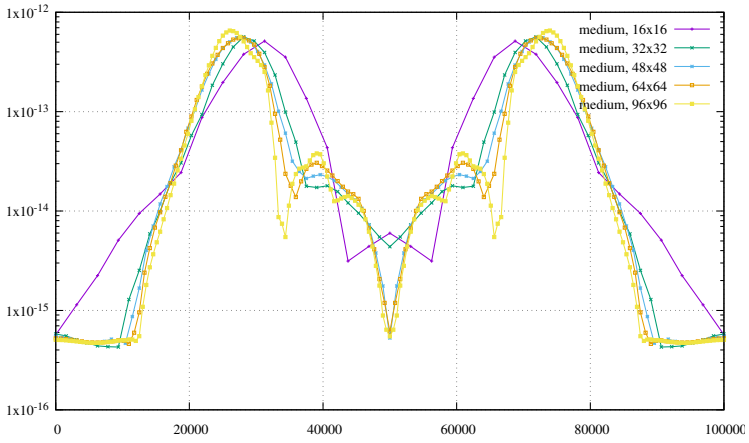
resolution 32x16, from top to bottom $D = 0, 2h_x, 8h_x$.



The diffusion helps with convergence. Rather importantly, it requires the use of nodal strainrate in the rheology, as opposed to before where strain rate at the quadrature point was computed using the derivatives of the shape functions.



D=1km, 32x16, 48x24, 64x32, 80x40, 96x48



Strain rate cross section at $y = 11L_y/16$.

Stone 40

Instantaneous Rayleigh-Taylor instability ($Q_2 \times Q_1$)

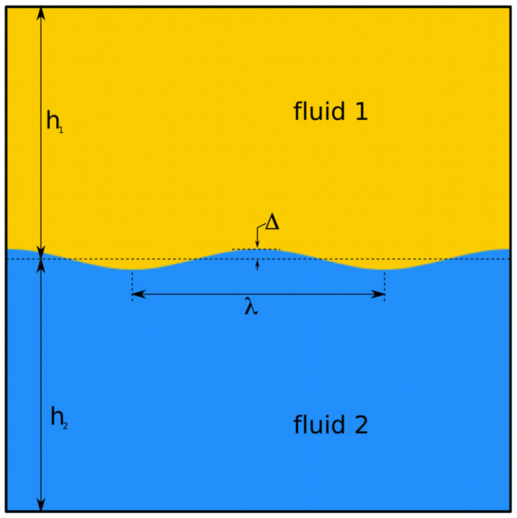


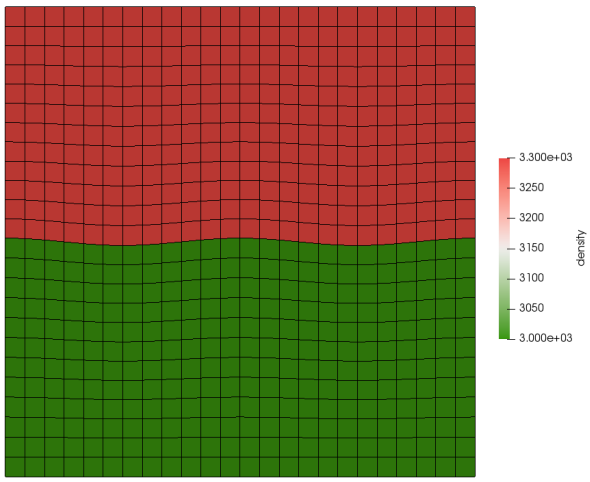
2D
box
Q2Q1
analytical_benchmark
error_convergence
stokes_equation
viscous_rheology
buoyancy_driven
paraview_output
pressure_scaling
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_40`

This benchmark is carried out in Deubelbeiss and Kaus [88] (2008), Gerya [128] (2010) and Thieulot [326] (2011) and is based on the analytical solution by Ramberg [257] (1968). It consists of a buoyancy-driven two-layer system driven. The domain is a square of size 512×512km. Free slip are imposed on the sides while no-slip boundary conditions are imposed on the top and the bottom of the box.

Fluid 1 (ρ_1, η_1) of thickness h_1 overlays fluid 2 (ρ_2, η_2) of thickness h_2 (with $h_1 + h_2 = L_y$). An initial sinusoidal disturbance of the interface between these layers is introduced and is characterised by an amplitude Δ and a wavelength $\lambda = L_x/2$ as shown here:

a) 

b) 

a) Setup of the experiment, taken from [326]; b) grid setup for 24×24 grid.

Under this condition, the velocity of the diapiric growth v_y is given by the relation

$$\frac{v_y}{\Delta} = -K\frac{\rho_1 - \rho_2}{2\eta_2}h_2 g$$

with the dimensionless growth factor K being

$$K = \frac{-d_{12}}{c_{11}j_{22} - d_{12}i_{21}}$$

and

$$c_{11} = \frac{\eta_1 2\phi_1^2}{\eta_2(\cosh 2\phi_1 - 1 - 2\phi_1^2)} - \frac{2\phi_2^2}{\cosh 2\phi_2 - 1 - 2\phi_2^2} \tag{40.1}$$

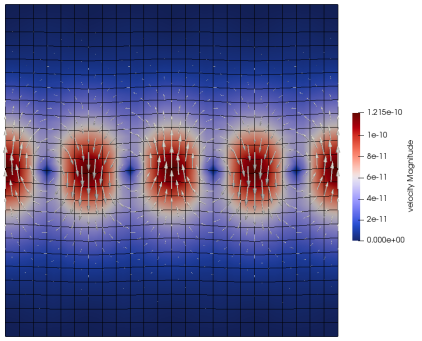
$$d_{12} = \frac{\eta_1(\sinh 2\phi_1 - 2\phi_1)}{\eta_2(\cosh 2\phi_1 - 1 - 2\phi_1^2)} + \frac{\sinh 2\phi_2 - 2\phi_2}{\cosh 2\phi_2 - 1 - 2\phi_2^2} \tag{40.2}$$

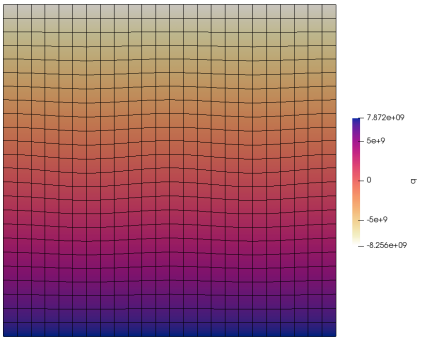
$$i_{21} = \frac{\eta_1\phi_2(\sinh 2\phi_1 + 2\phi_1)}{\eta_2(\cosh 2\phi_1 - 1 - 2\phi_1^2)} + \frac{\phi_2(\sinh 2\phi_2 + 2\phi_2)}{\cosh 2\phi_2 - 1 - 2\phi_2^2} \tag{40.3}$$

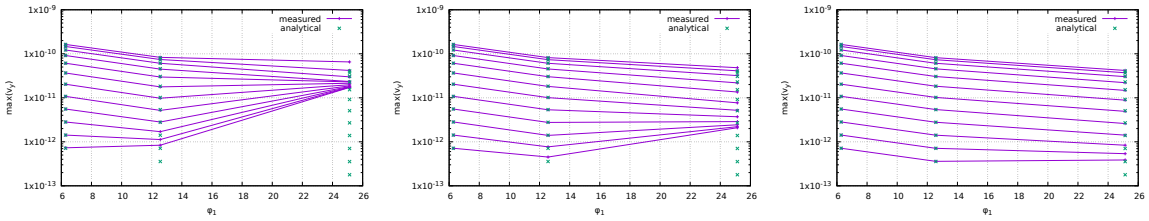
$$j_{22} = \frac{\eta_1 2\phi_1^2\phi_2}{\eta_2(\cosh 2\phi_1 - 1 - 2\phi_1^2)} - \frac{2\phi_2^3}{\cosh 2\phi_2 - 1 - 2\phi_2^2} \tag{40.4}$$

$$\phi_1 = \frac{2\pi h_1}{\lambda} \tag{40.5}$$

$$\phi_2 = \frac{2\pi h_2}{\lambda} \tag{40.6}$$







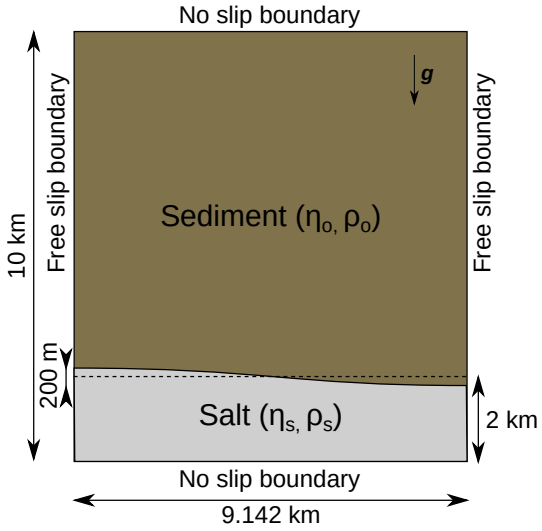
Left: 24×24 elements, middle: 32×32; right: 48×48. We see that increasing resolution yields more accurate results in the cases of short wavelength perturbations (points on the right on the plots).

Note that in [326] I fixed $\lambda = L_x/2$ and varied L_x. Here I keep L_x fixed and vary $\lambda = L_x/2, L_x, 4, L_x/8$. Each line corresponds to a different value of the viscosity η_2.

Stone 41

Stokes flow and structural restoration ($Q_2 \times Q_1$)

Setup of the experiment It originates in Schuh-Senlis *et al.* [289] and is a scaled-up version of the Rayleigh-Taylor experiment of van Keken et al (1997) [338]. In the original experiment the domain is 0.9142×1, viscosity is $\eta = 100$, densities are 1000 and 1010 (i.e. $\delta\rho = 10$) and the gravity is $|\vec{g}| = 10$. In our case the domain is 10,000 times larger, i.e. 9142×10000m, our viscosity is $\eta = 10^{19}$, our densities are 2150 and 2600, i.e. $\delta\rho = 450$, and we also set $|\vec{g}| = 10$.



Boundary conditions are no slip on the top and bottom, free slip on the sides.

In the original paper, length=1, viscosity=100, gravity =10, density contrast =10 and time unit =1, so the dimensionless quantity

$$\frac{\eta}{\delta\rho \cdot g\ length \cdot time} = \frac{100}{10 \cdot 10 \cdot 1 \cdot 1} = 1$$

If we now run a geometrically similar experiment with length=10km, viscosity=10^{19}, gravity=10, density contrast=450 and time unit = t then we should also have

$$\frac{\eta}{\delta\rho \cdot g\ length \cdot time} = \frac{10^{19}}{450 \cdot 10 \cdot 10^4 \cdot t} = 1$$

i.e.,

$$t = \frac{10^{14}}{450} \simeq 2.222 \cdot 10^{11}\text{s}$$

This means that in order to plot our results against those in van Keken et al, their results must be scaled: times must be multiplied by t and velocities divided by $t/length = 2.222 \cdot 10^7$m/s

The initial mass of the system is:

$$M(t=0) = L_x \times 2km \times \rho_s + Lx \times 8km \times \rho_0 = 9142 \cdot (2000 \cdot 2150 + 8000 \cdot 2600) \simeq 2.294642 \cdot 10^{11}\text{kg}$$

The code It is based on stable $Q_2 \times Q_1$ elements for the Stokes equations and the heat transport equation need not be solved. It relies on the Particle-in-Cell method (see Section 14.1) for material advection. At startup `nmarker_per_dim**2` particles are regularly placed in each element. There are in total:

```
nmarker=nmarker_per_element*nel
```

Markers track material number, i.e. 1 or 2:

```
for im in range (0,nmarker):
    if swarm_y[im]>salt_thickness+amplitude*np.cos(np.pi*swarm_x[im]/Lx):
       swarm_mat[im]=2
    else:
       swarm_mat[im]=1
```

Marker density is interpolated onto the Q1 mesh by means of the Q1 shape functions:

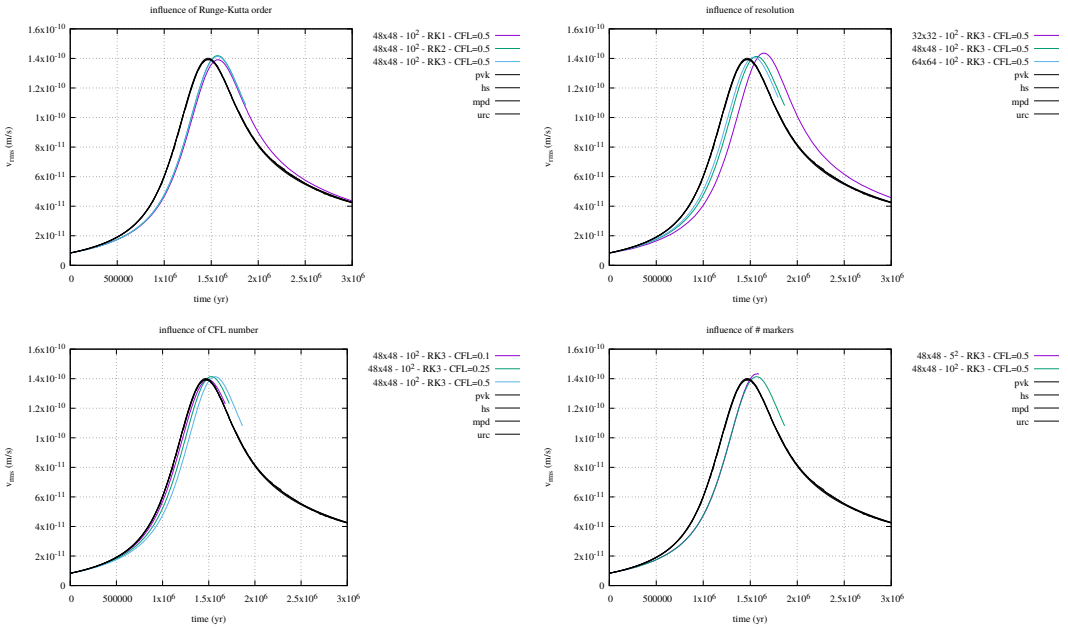
```
for im in range(0,nmarker):
    ielx=int(swarm_x[im]/Lx*nelx)
    iely=int(swarm_y[im]/Ly*nely)
    iel=nelx*(iely)+ielx
    N1=0.25*(1-swarm_r[im])*(1-swarm_s[im])
    N2=0.25*(1+swarm_r[im])*(1-swarm_s[im])
    N3=0.25*(1+swarm_r[im])*(1+swarm_s[im])
    N4=0.25*(1-swarm_r[im])*(1+swarm_s[im])
    rho_nodal[iconP[0,iel]]+=rho_mat[swarm_mat[im]-1]*N1
    rho_nodal[iconP[1,iel]]+=rho_mat[swarm_mat[im]-1]*N2
    rho_nodal[iconP[2,iel]]+=rho_mat[swarm_mat[im]-1]*N3
    rho_nodal[iconP[3,iel]]+=rho_mat[swarm_mat[im]-1]*N4
    rho_nodal_counter[iconP[0,iel]]+=N1
    rho_nodal_counter[iconP[1,iel]]+=N2
    rho_nodal_counter[iconP[2,iel]]+=N3
    rho_nodal_counter[iconP[3,iel]]+=N4
rho_nodal/=rho_nodal_counter
```

This is a marker-centric approach, which is identical to the somewhat more instinctive node-centric approach which itself is similar to the meshless methods approach (think of the SPH method kernel but with a square support). Density is later on interpolated onto the quadrature points with the Q_1 shape functions again to avoid the problems highlighted in Section 15.27. Viscosity is averaged per element by means of an arithmetic, geometric or harmonic mean.

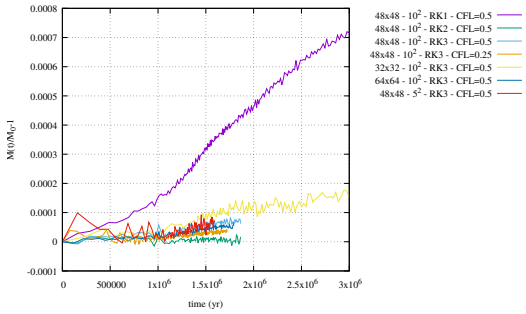
Particles are advected by means of a (space) Runge-Kutta 1st/2nd/3rd order algorithm (see Section 14.10). The time step δt is set by means of the CFL criterion (see Section 15.9).

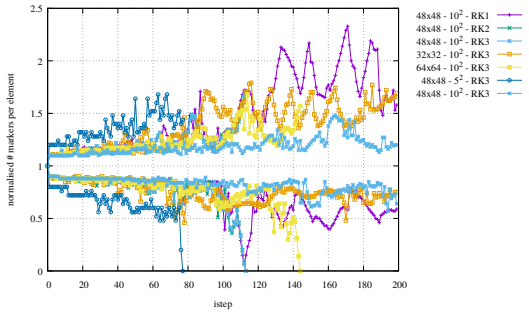
The root mean square velocity and total mass of the system are computed every time step.

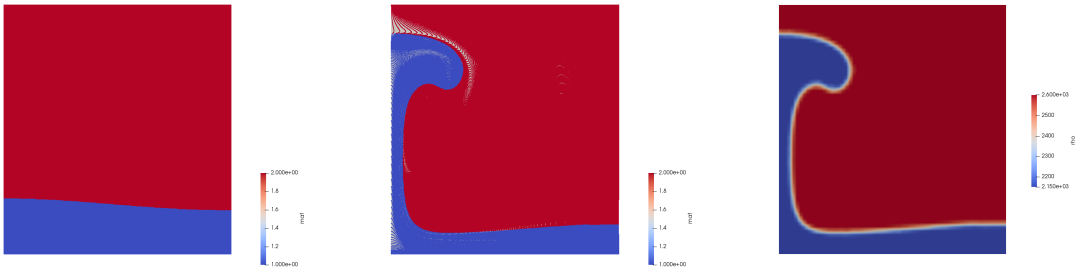
Simulations end when the maximum number of time steps is reached or when an element does not contain any particle.



Root mean square velocity as a function of time. Black curves are those in van Keken et al (1997). Letters stand for authors initials.







64x64 simulation right before crash

Stokes sphere under deformable free surface

The setup is described in Section 19.25. Densities are projected on the nodes or elements with an arithmetic averaging while viscosities are projected onto elements (`avrg`> 0) or onto Q_1nodes (`avrg`< 0). There are 7 averaging schemes implemented:

`avrg`=-1 first-order basis function weighed arithmetic average onto nodes

`avrg`=-2 first-order basis function weighed geometric average onto nodes

`avrg`=-3 first-order basis function weighed harmonic average onto nodes

`avrg`=+1 elemental piecewise constant arithmetic average

`avrg`=+2 elemental piecewise constant geometric average

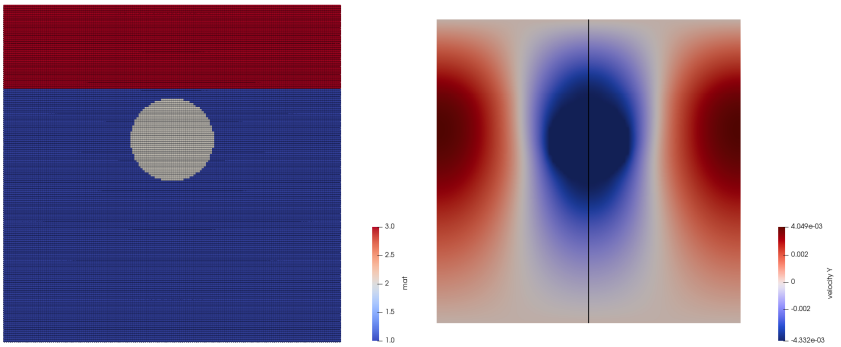
`avrg`=+3 elemental piecewise constant harmonic average

`avrg`=+4 elemental first-order least square projection

Let us recall that the Least Squares methodology is presented for example in Thielmann *et al.* [324] and in Section 14.1: the density or viscosity values carried by the markers are represented by a linear function in a 2D space inside each element, i.e. $\tilde{\eta}(\vec{r}) = a + bx + cy$ and $\tilde{\rho} = d + ex + fy$.

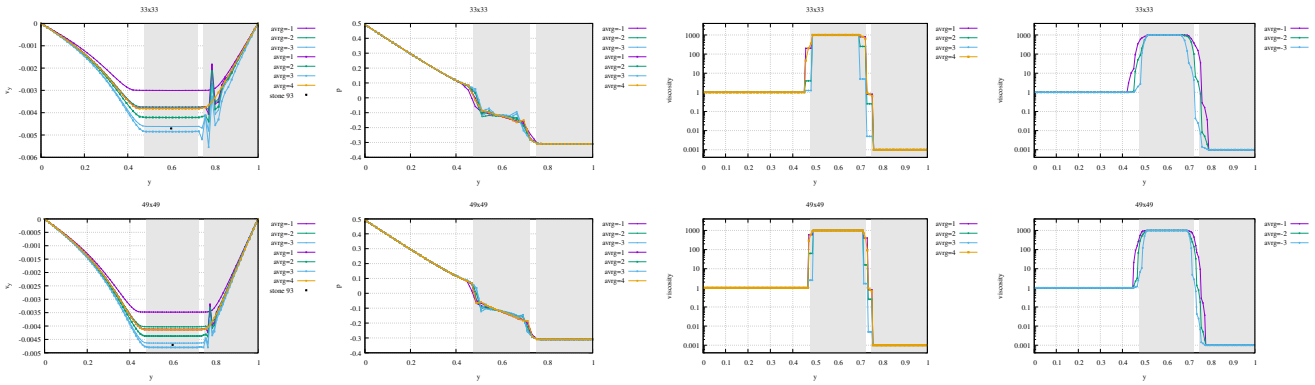
When `avrg`< 0 values are projected onto quadrature points by means of Q_1 basis functions using the values at the corners of the element. When `avrg`$= 1, 2, 3$ all quadrature points receive the same elemental value. When `avrg`$= 4$ the coordinates of each quadrature points are used in order to estimate the value from the coefficients $\{a, b, c\}$ and $\{d, e, f\}$ of the (limited) least square.

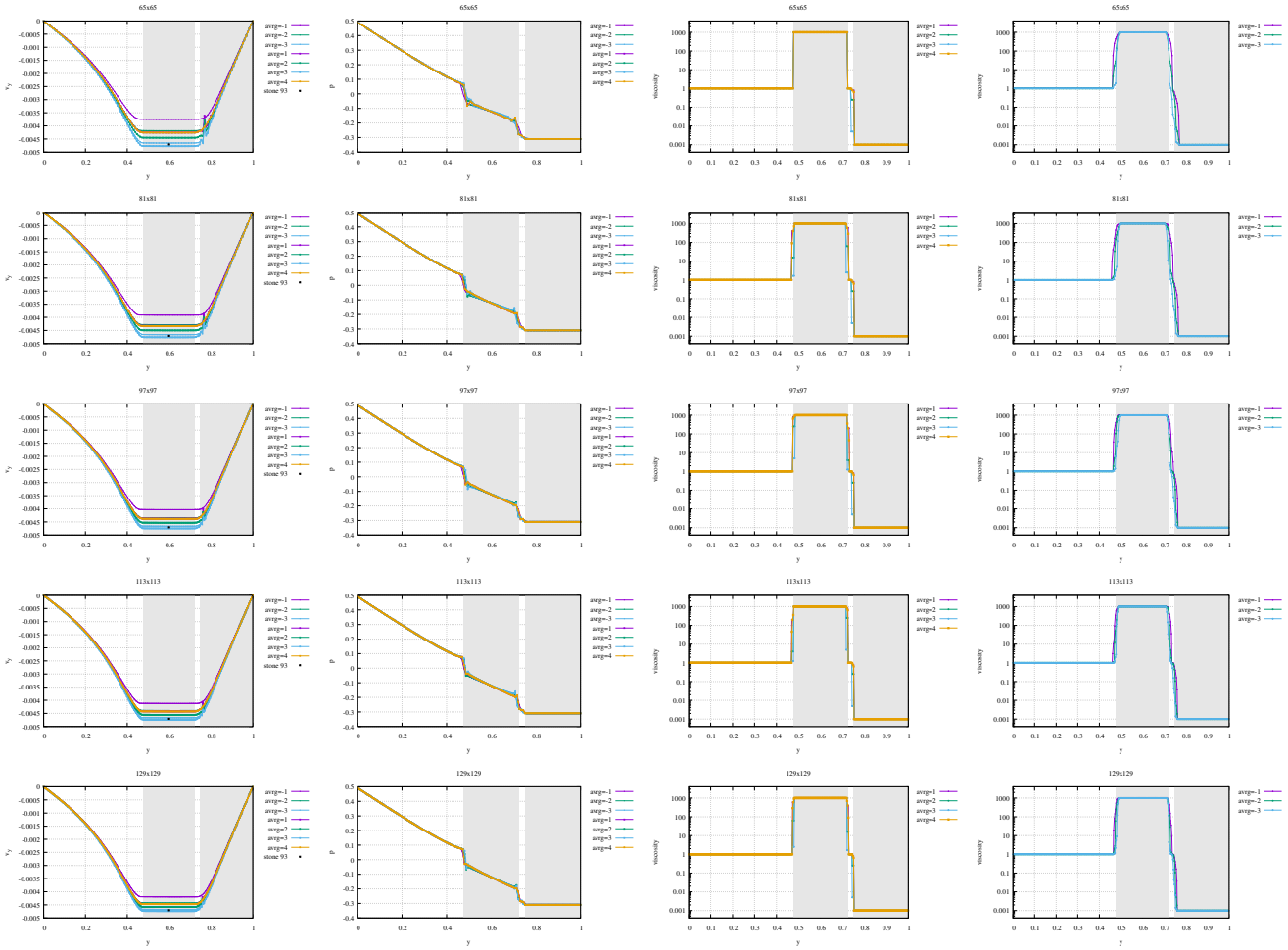
Given the current limitations of this simple code I can only run experiments with limited resolutions. Also since I later wish to plot the density and viscosity values carried by the quadrature points on a vertical line passing through the middle of the sphere, I focus on odd numbers of elements in the horizontal direction and investigate 33×33, 49×49, 65×65, 81×81, and 97×97.



Left: markers on 33x33 grid, 5^2 markers per element. Right: vertical component v of the velocity as obtained on the 81×81 mesh using least squares + limiter projection.

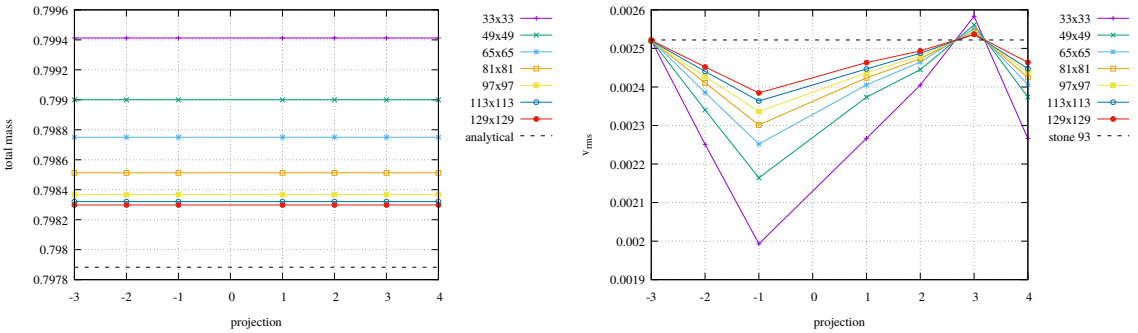
The black line indicates the location of profile measurements.





1st column: vertical velocity profile; 2nd column: vertical pressure profile; 3rd column: viscosity on quadrature points in log scale for `avrg`> 0 (elemental approach); 4th column: viscosity on quadrature points in log scale for `avrg`< 0 (nodal approach); 5^2 markers per element. The black dot indicates the velocity in the middle of the sphere obtained with STONE 93 using Crouzeix-Raviart triangular elements.

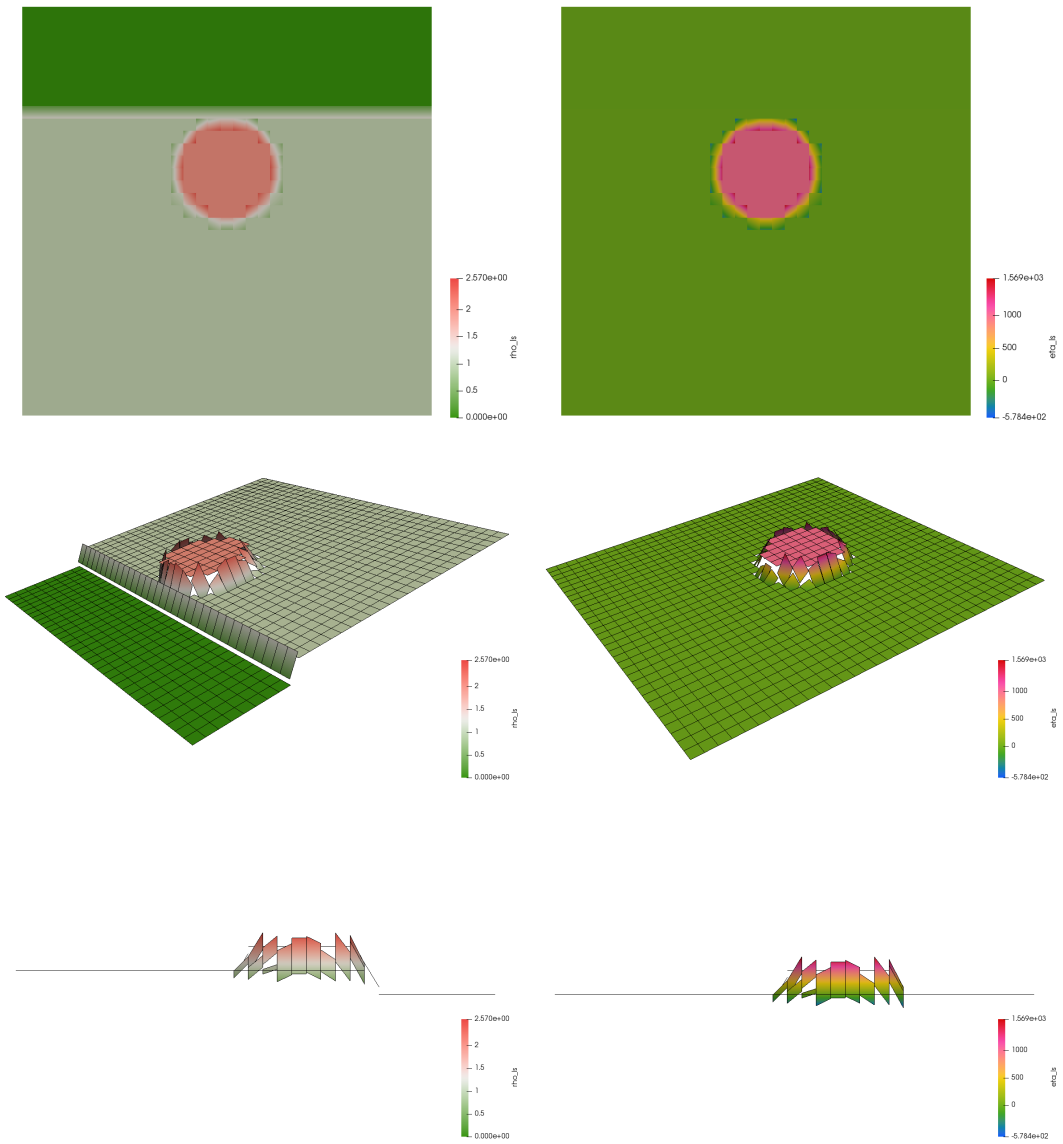
It is immediately apparent that the least-square case yields smooth velocity profiles, even at low resolutions. Also it is clear that nodal approaches tend to smear out the transition zone but do not yield a visible improvement in terms of velocity.



Total mass and root mean square velocity v_{rms} for all 7 projections.

A word about the need for a limiter

The following figures are obtained with a standard linear least square algorithm.



Describe here limiter...

Stone 42

The no flow test ($Q_1 \times P_0$, $Q_2 \times Q_1$, $Q_3 \times Q_2$, $Q_4 \times Q_3$)

```
2D
box
Q1P0
Q2Q1
Q3Q2
sparse_matrix
analytical_benchmark
stokes_equation
paraview_output
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_42`

The idea behind this stone comes from the following excerpt of Fortin & Fortin (1985) [111]:

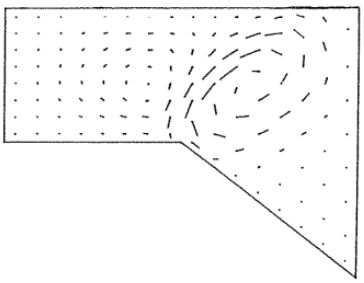


Figure 5

On the boundary Γ of the domain Ω of Figure 5, we impose homogeneous boundary conditions $\mathbf{u} = 0$ and we let $\mathbf{f} = (0, g)$ (non-zero external forces). Here again, the analytic solution of the Navier–Stokes equations is known, that is

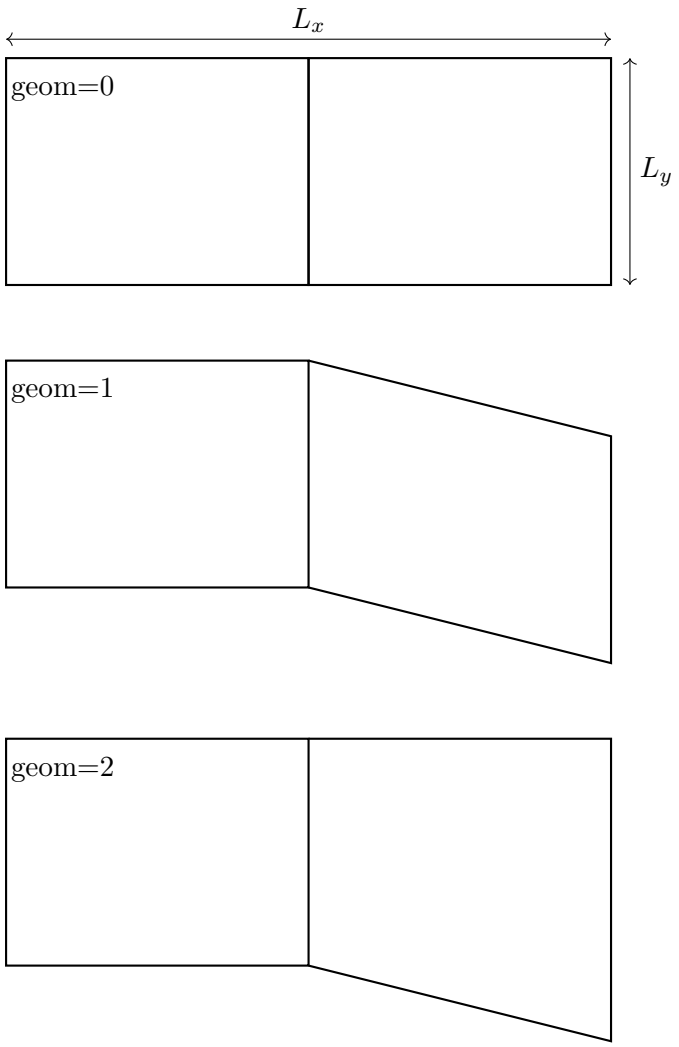
$$\mathbf{u} = \mathbf{0}, \quad p = gy + c. \tag{27}$$

Using an 8×16 mesh with the Q_1–P_0 and R_2–P_0 elements, we get $O(h)$ vortices illustrated in Figure 5. The problem here is not due to a failure in satisfying the B.B. condition but to the fact that we are using only piecewice constant pressure. Consequently, the linear analytic pressure cannot be reproduced exactly and we have a striking example of the dependency of the approximation of the pressure. It is clear that the cure to this is to use piecewise linear (at least) pressure. Indeed, the Q_1^+–P_1 element passes the no-flow test remarkably well, yielding the exact solution for both velocity and pressure (without chequerboarding!).

I then set out to reproduce this and to compare the results with continuous-pressure elements. On the boundary of the domain Ω I impose $\vec{\mathbf{v}} = \vec{0}$ and the viscosity and density are set to 1. The gravity vector points downwards with $|\vec{g}| = 1$. The analytical solution is $\vec{\mathbf{v}}(x, y) = \vec{0}$ and $p(x, y) = ay + b$, where the constants a and b are determined by the geometry and the buoyancy force.

Based on the figure above I set $L_x = 16$ and $L_y = 6$ and the angle is set to 45° (while it is about 15° on the figures below). The chosen number of elements is 16×8 as in [111]. Rather arbitrarily the pressure is set to zero on the top right corner (This is not really important since the pressure is defined up to a constant in this case).

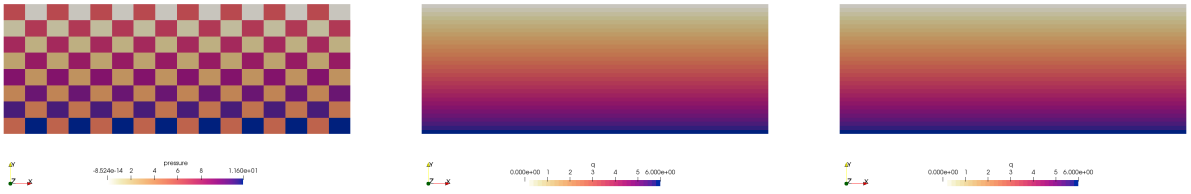
I consider the following three different domain geometries:

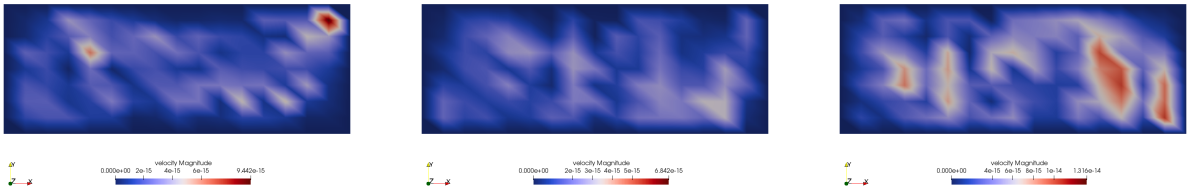


and three different elements: $\boldsymbol{Q}_1 \times P_0$, $\boldsymbol{Q}_2 \times Q_1$, and $\boldsymbol{Q}_3 \times Q_2$.

geom=0

I find that the checkerboard mode is present for the unstable $\boldsymbol{Q}_1 \times P_0$ element. However, for all three elements the velocity is zero, down to machine precision. For the two continuous-pressure elements the pressure at the top is 0 and 6 at the bottom as expected.

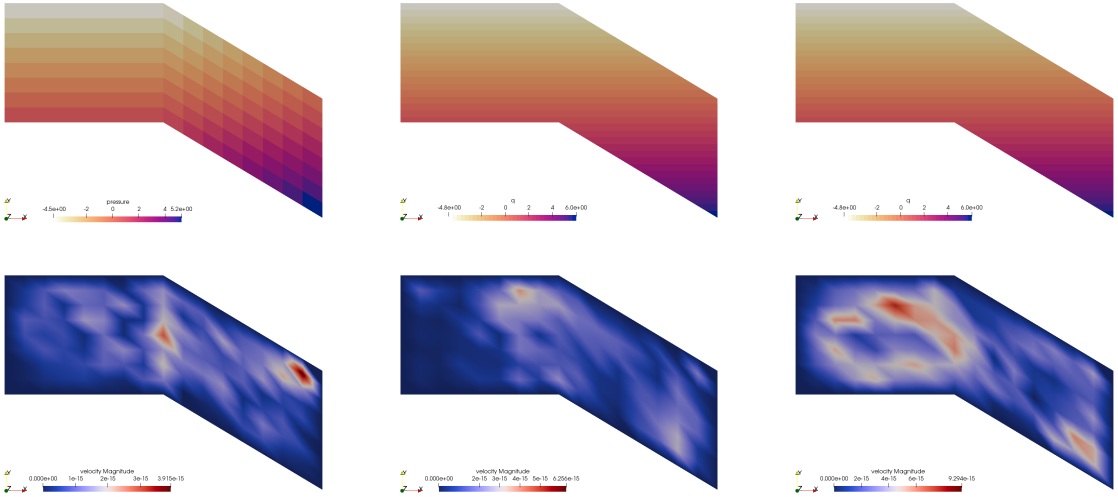




From left to right: $\boldsymbol{Q}_1 \times P_0$, $\boldsymbol{Q}_2 \times Q_1$, and $\boldsymbol{Q}_3 \times Q_2$.

geom=1

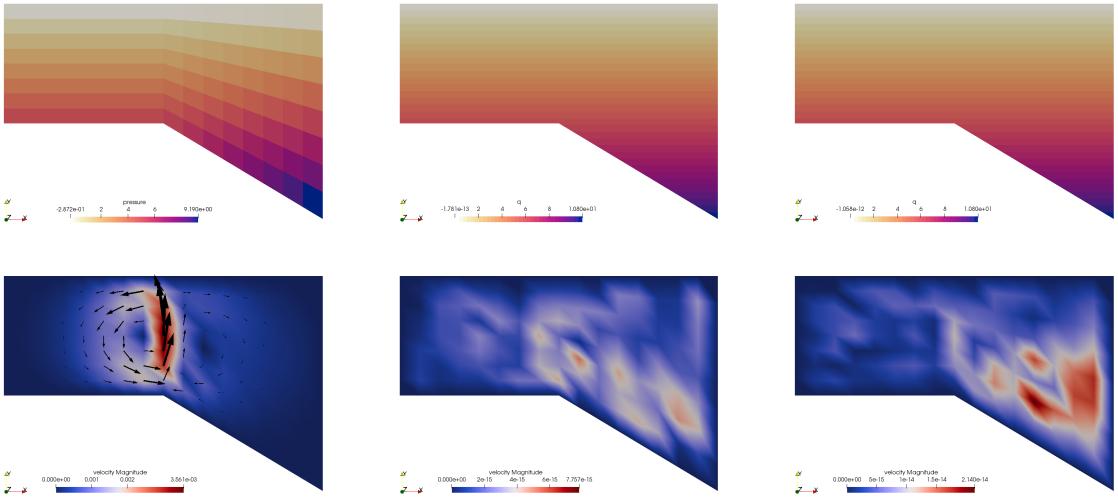
We find that the checkerboard mode is now absent and velocities are still zero for all three elements. The pressure is hydrostatic and only depends on the y-coordinate.



From left to right: $\boldsymbol{Q}_1 \times P_0$, $\boldsymbol{Q}_2 \times Q_1$, and $\boldsymbol{Q}_3 \times Q_2$.

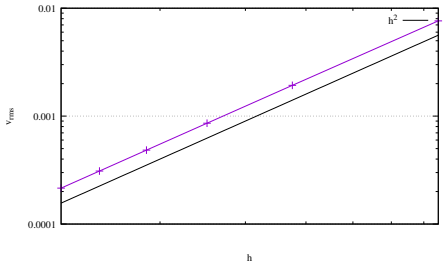
geom=2

As in Fortin & Fortin (1985) we find that the velocity field now showcases 2 "convection cells" for the $\boldsymbol{Q}_1 \times P_0$ element (although they have different shapes than the ones in their publication) while the velocity remains zero for the other two elements.



From left to right: $\boldsymbol{Q}_1 \times P_0$, $\boldsymbol{Q}_2 \times Q_1$, and $\boldsymbol{Q}_3 \times Q_2$.

On the following figure the root mean square velocity obtained with the $\boldsymbol{Q}_1 \times P_0$ element is shown as a function of the element size h. We find that it decreases quadratically with h.



Stone 43

Time-dependent advection problems (Q_1, Q_2)



Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_43`

The following experiments are "pure advection" experiments (i.e. there is no diffusion at all). As such the Peclet number is infinite and the γ value of the SUPG algorithm as shown in Section 9.6 tends to 1 and the τ parameter is simply $h/2|\vec{\mathrm{v}}|p$ (where p is the order of the shape function polynomials).

Note that although the code can deal with linear elements, all results hereafter are obtained with quadratic elements (unless specified otherwise).

A very important remark: In step-9 of deal.II (which is concerned with solving the steady-state advection equation) we read: " The mathematical theory states that we must not pose any boundary condition on the outflow part of the boundary." Upon my asking for more context to Wolfgang Bangerth (the author of step-9), his answer was: "I think the statement can be shown in the following way: If you think of the advection equation as transporting information along characteristics, then the interior can only be affected by characteristics that go from the boundary into the interior. That's exactly the ones at inflow boundary conditions. " Note that ASPECT automatically checks that $\vec{\mathrm{v}} \cdot \vec{n} < 0$ at each node where temperature is prescribed and only prescribe it there [1] In other words, one cannot prescribe the temperature on a boundary where the fluid is leaving the domain, only where it is entering.

This means that in the experiments presented hereafter this needs to be checked and corrected if needed!

[1] `https://github.com/geodynamics/aspect/blob/main/source/simulator/core.cc`, Lines 689-726

Experiment 1 - rotating cone

This benchmark can be found Donea and Huerta [94] and it is also carried out in Beuchert and Podladchikov [19] (2010). A product-cosine hill is advected in a prescribed velocity field. The initial temperature is:

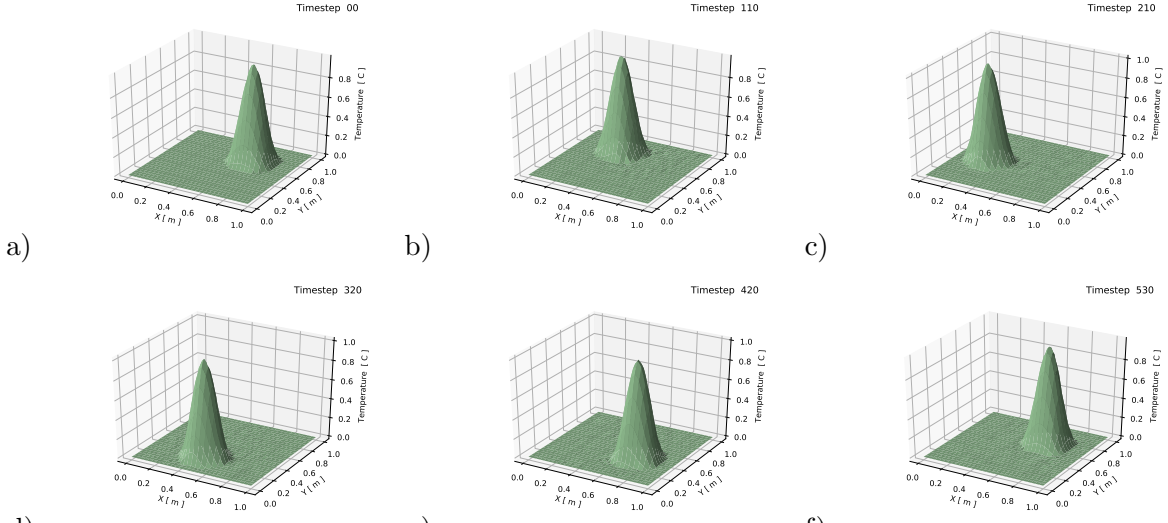
$$T_0(x,y) = \begin{cases} \frac{1}{4}\left(1+\cos\pi\frac{x-x_c}{\sigma}\right)\left(1+\cos\pi\frac{y-y_c}{\sigma}\right) & \text{if } (x-x_c)^2+(y-y_c)^2 \leq \sigma^2 \\ 0 & \text{otherwise} \end{cases} \tag{43.1}$$

The boundary conditions are $T(x,y) = 0$ on all four sides of the unit square domain, but only where the velocity field is such that the node in question accounts for an influx, not an outflux (see remark above). In what follows we set $x_c = y_c = 2/3$ and $\sigma = 0.2$. The velocity field is analytically prescribed: $\vec{\mathbf{v}} = (-(y - L_y/2), +(x - L_x/2))$. Resolution is set to 30×30 quadratic Q_2 elements.

In what follows we test the time integration scheme by setting $\alpha_T = 1$ (fully implicit formulation), $\alpha_T = 0$ (fully explicit formulation) and $\alpha_T = 1/2$ (Crank-Nicolson). In the book Donea & Huerta set the timestep is set to $\delta t = 2\pi/200$ which corresponds to a CFL number of approximately 0.666. If we want to be able to run this experiment at higher resolution we need to adapt the timestep to the mesh size (CFL criterion). We therefore set the CFL number to 0.5 and compute δt accordingly (see Section 15.9). The density and heat capacity values are set to 1. We monitor the minimum and maximum value of the temperature field, as well as the total thermal energy E_T in the system during the full rotation:

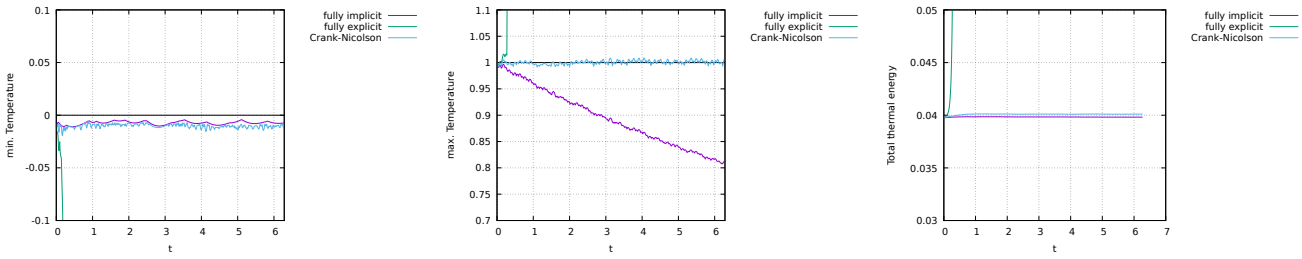
$$E_T = \int_\Omega \rho_0 C_p T dV = \int_\Omega T dV = |\Omega|\langle T\rangle \qquad \text{where} \qquad \langle T\rangle = \frac{1}{|\Omega|}\int_\Omega T dV$$

The time evolution of the temperature with the Crank-Nicolson algorithm is shown hereunder:



a,b,c,d,e,f) Temperature field throughout the 2π rotation.

Turning now to the statistics, we plot $\min / \max(T)$ and E_T as a function of time:

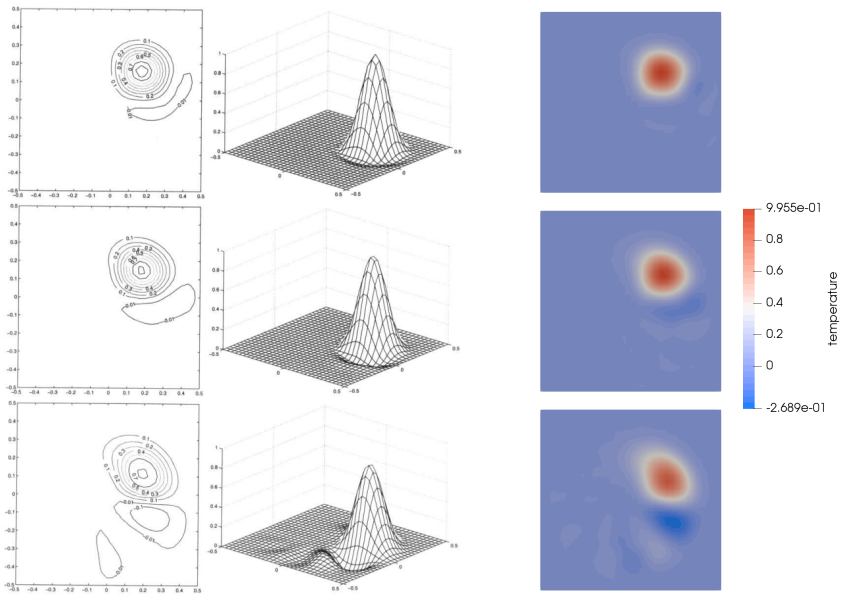


Time evolution of the min and max temperature and the total energy

The conclusions are clear: the explicit method diverges quickly and is unusable. The fully implicit and Crank-Nicolson method yield similar energy conservation but the fully-implicit showcases a clear decrease in maximum temperature.

Effect of time step value

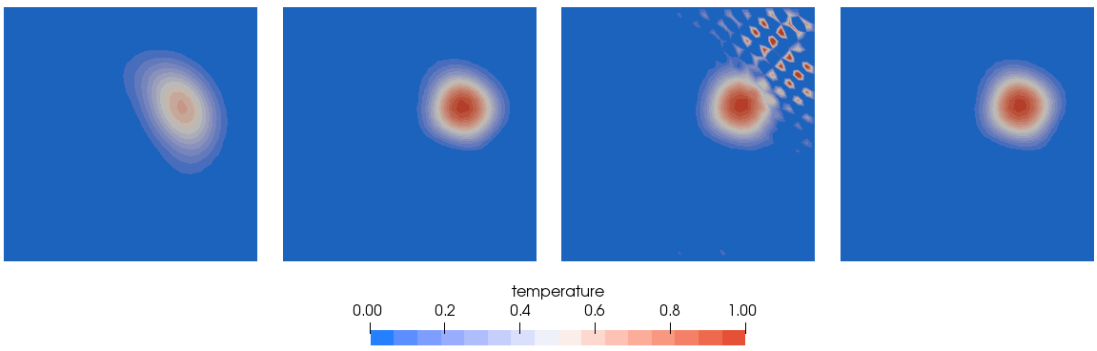
We can run the experiment (still a 2π rotation) with three different time steps ($\delta t = 2\pi/30, 2\pi/60, 2\pi/120$) and we recover very similar results to those presented in Donea and Huerta [94]:



From top to bottom: $\delta t = 2\pi/120, 2\pi/60, 2\pi/30$ with Crank-Nicolson. Left panel is taken from donea & Huerta [94]. Results obtained with linear elements.

On the use of BDF methods

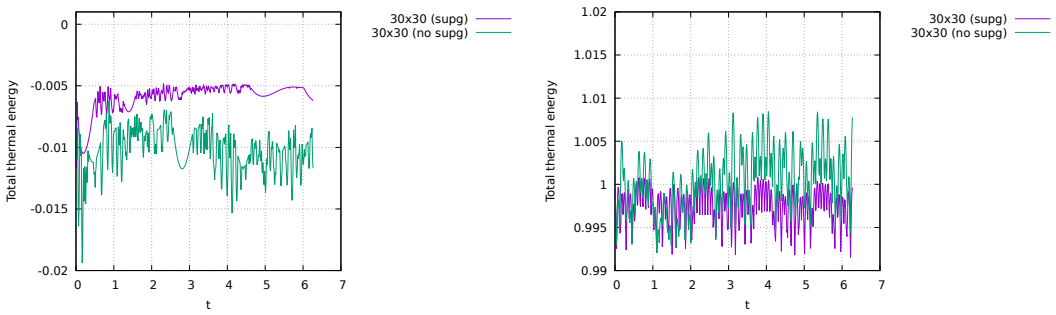
I have also implemented BDF1,2,3,4,5 (see Section 9.3). BDF2 outperforms BDF1 and is comparable to Crank-Nicolson. For reasons unknown to me, the BDF3 diverges after 100 time steps. So do BDF4 and BDF5 after even less timesteps. In the following picture the temperature is shown for BDF1,2,3 and Crank-Nicolson after a full rotation.



Left to right: BDF1 (i.e. implicit euler), BDF2, BDF3, Crank-Nicolson

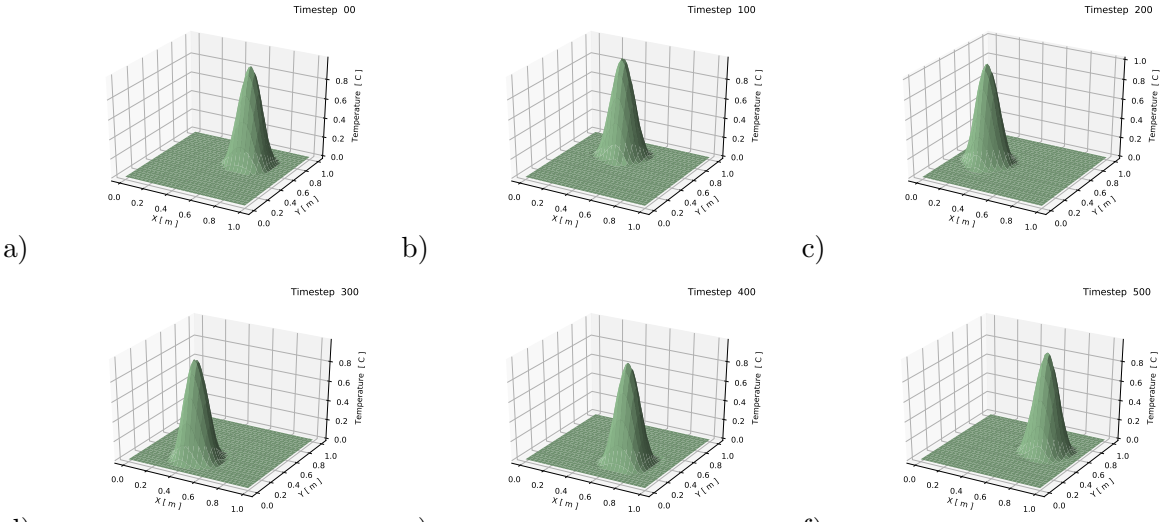
On the effect of SUPG

I now turn to another aspect of this problem: what is the effect of the SUPG stabilisation scheme on the solution? In what follows Crank-Nicolson is used.



Time evolution of min/max temperature with and without SUPG.

We find that SUPG yields a solution which min/max values are closer to the analytical ones.



a) b) c)

d) e) f)

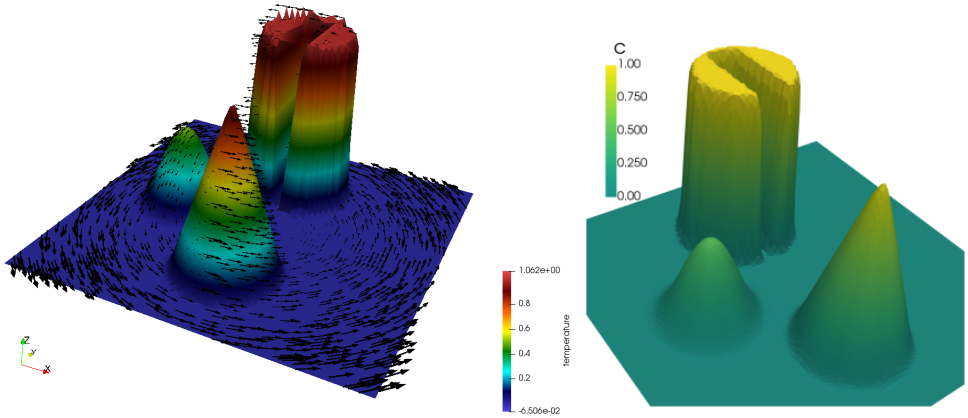
a,b,c,d,e,f) Temperature field throughout the 2π rotation.

Experiment 2 - Three objects rotation

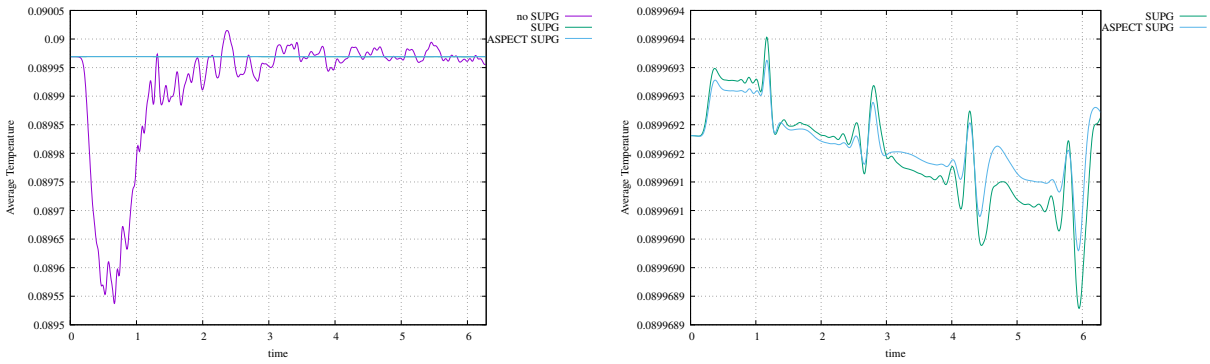
This setup is inspired by the one in the ASPECT manual[2]. The cone of the previous experiment is now replaced by three 'objects': a Zalesak disk [360], a sharp cone and a truncated cosine hill:

```
for i in range(0,NV):
    xi=x[i]
    yi=y[i]
    if np.sqrt(xi**2+(yi-0.5)**2)<0.3 and (np.abs(xi)>=0.05 or yi>=0.7):
        T[i]=1
    if np.sqrt((x[i])**2+(y[i]+0.5)**2)<0.3:
        T[i]=1-np.sqrt((x[i])**2+(y[i]+0.5)**2)/0.3
    if np.sqrt((x[i]+0.5)**2+(y[i])**2)<0.3:
        T[i]=0.25*(1+np.cos(np.pi*np.sqrt((xi+0.5)**2+yi**2)/0.3))
```

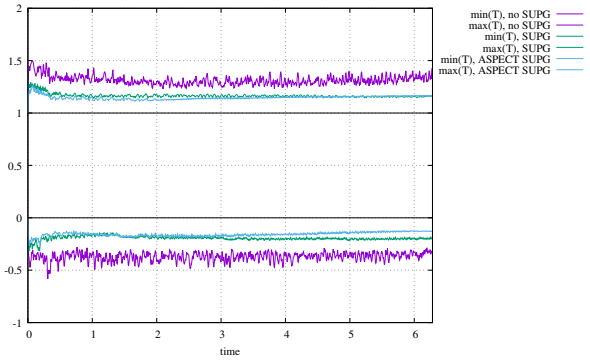
The domain is 2×2, centered on the origin. The velocity is $\vec{v} = (-y, x)$. Temperature is set to zero on all four sides (only on influx subsets). For this experiment the CFL number is set to 0.5 and the resolution is 64×64 elements.



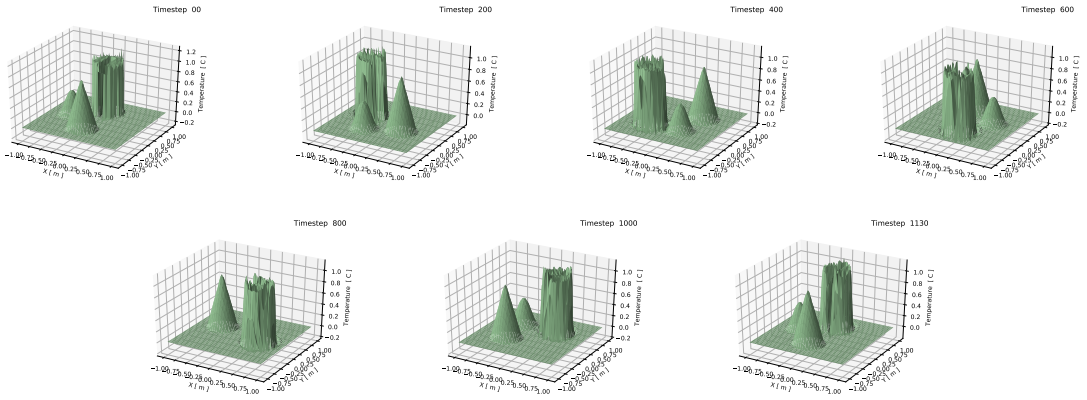
Left: ASPECT manual; Right: taken from Kohl, Mohr, Eibl, and Rüde [197] (2022).



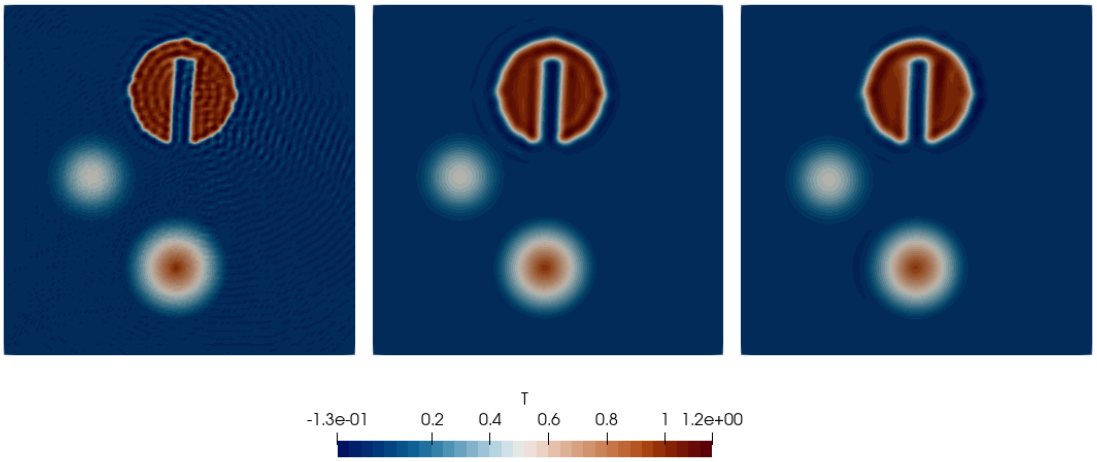
Averave temperature in the domain as a function of time. Left: all three cases. Right: only SUPG cases.



[2] https://aspect-documentation.readthedocs.io/en/latest/user/benchmarks/benchmarks/advection/doc/advection.html



Time evolution of the temperature field with Crank-Nicolson, with SUPG.

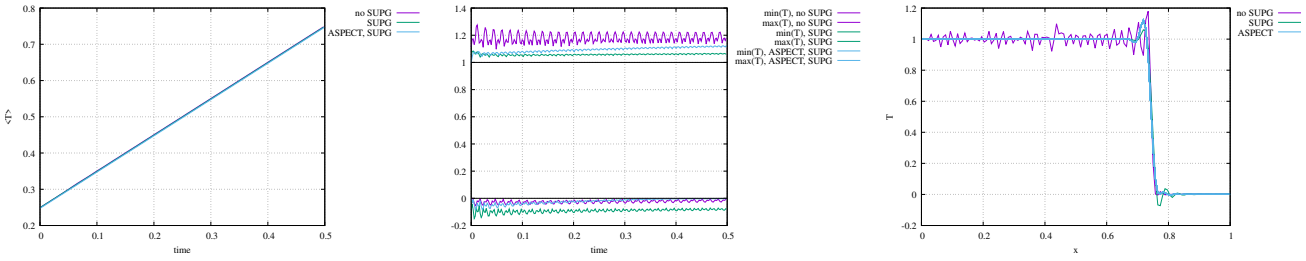


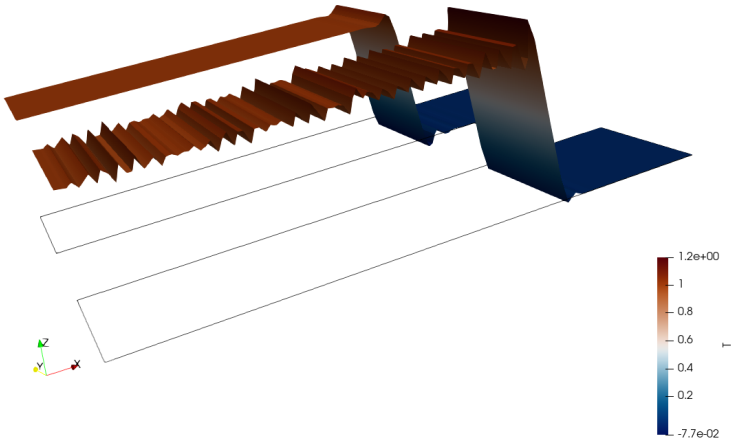
Left: no SUPG; middle: with SUPG; right: ASPECT with SUPG

Experiment 3 - 1D step advection

This experiment comes from Appendix A of Thieulot (2011) [326]. The unit segment is discretised by means of 64 elements, over which a unit velocity field is prescribed. The discontinuity is initially placed at $x = 1/4$ and after a time $t = 0.5$, it is expected to have reached the position $x = 3/4$. Temperature boundary conditions are $T = 1$ on the left. Note that the simulation is actually carried out in 2D with a 1×0.25 domain discretised by means of 64×16 elements. CFL number is set to 0.25.

$$\tau_{supg} = \frac{h}{2d|\vec{\mathrm{v}}|} = \frac{\sqrt{2}/64}{2 \cdot 2 \cdot 1} = 0.005524$$

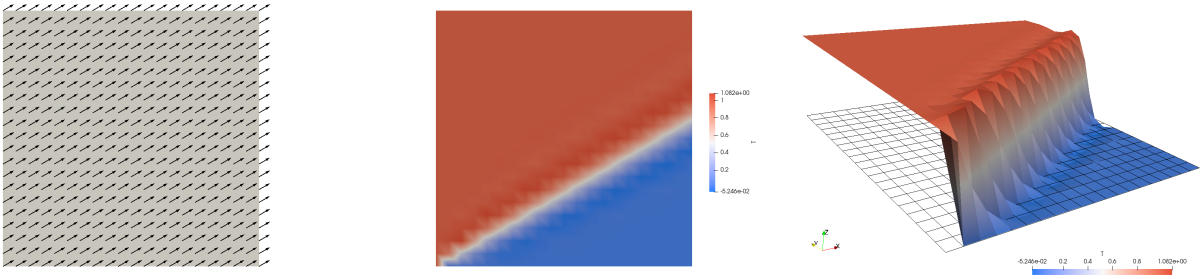


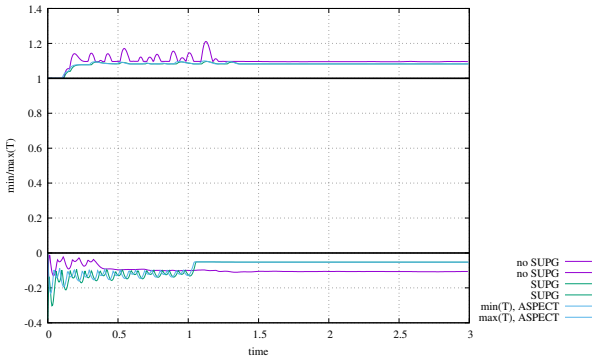


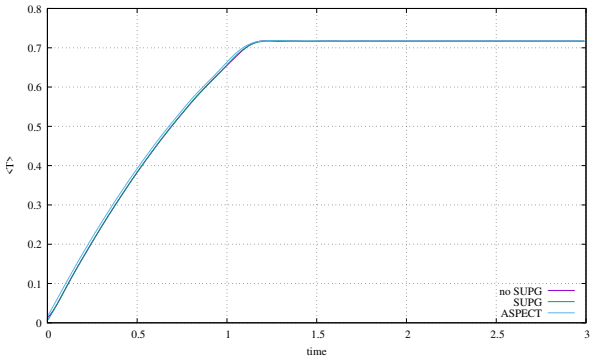
Temperature field at the end of the simulation. No SUPG in the front, with SUPG in the back.

Experiment 4 - Steady state sideways 2D advection

This experiment is somewhat similar to the one in STONE 65. The domain is a unit square, and the velocity is given by $\vec{\mathrm{v}} = (\cos\theta, \sin\theta)$ with $\theta = \pi/6$. The temperature is prescribed on the left side only, i.e. $T = 1$ for $x = 0$, $y = [0, 1]$ (corners included), and on the bottom (left corner excluded) with $T = 0$. The other two sides are left free. Initial temperature is $T = 0$. The resolution is 10×10 and the CFL number is set to $C_{\mathtt{CFL}} = 0.1$. The model is run up to $t = 3$ (steady state is then reached).

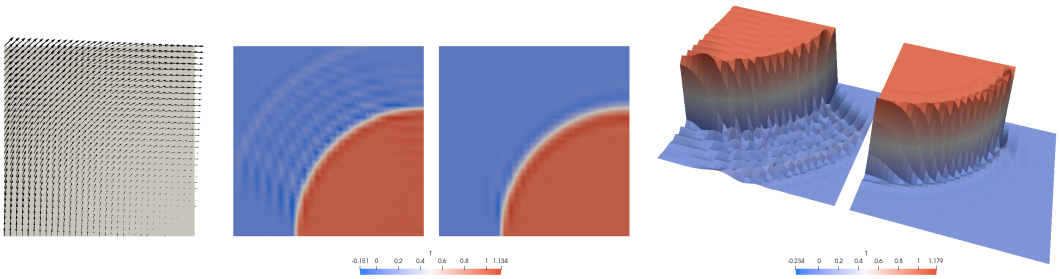




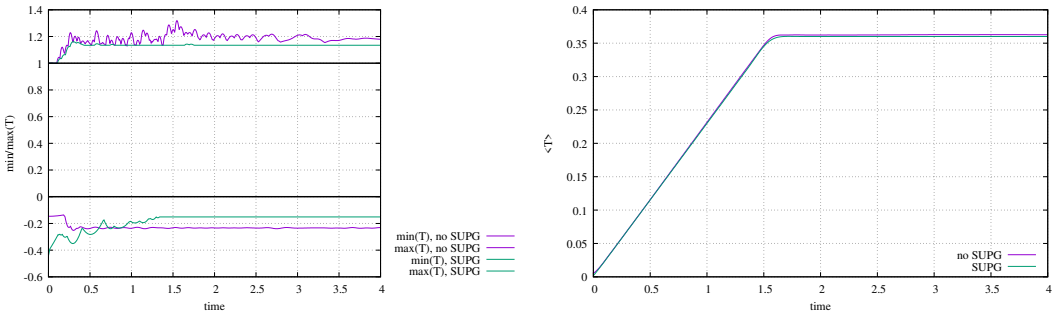


Experiment 5 - Steady state rotation

The domain is a unit square. The velocity field is given by $\vec{\mathbf{v}} = (y, 1 - x)$. The initial temperature is $T(x, y) = 0$. Influx boundary conditions are prescribed on the left and bottom boundaries so $T = 0$ is prescribed on the left while at the bottom $T = 0$ is prescribed if $x < 1/3$ and $T = 1$ otherwise. Resolution is 16×16 and the CFL number is set to $C_{CFL} = 0.25$. The model is run up to $t = 4$ (steady state is then reached).



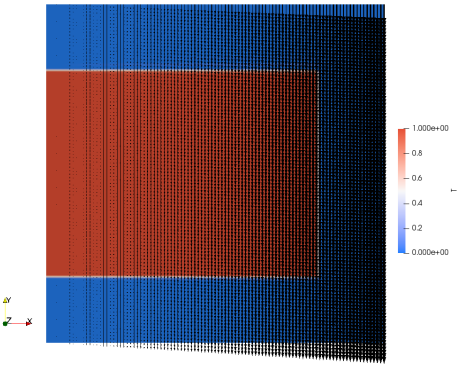
Left: no SUPG, Right: SUPG.



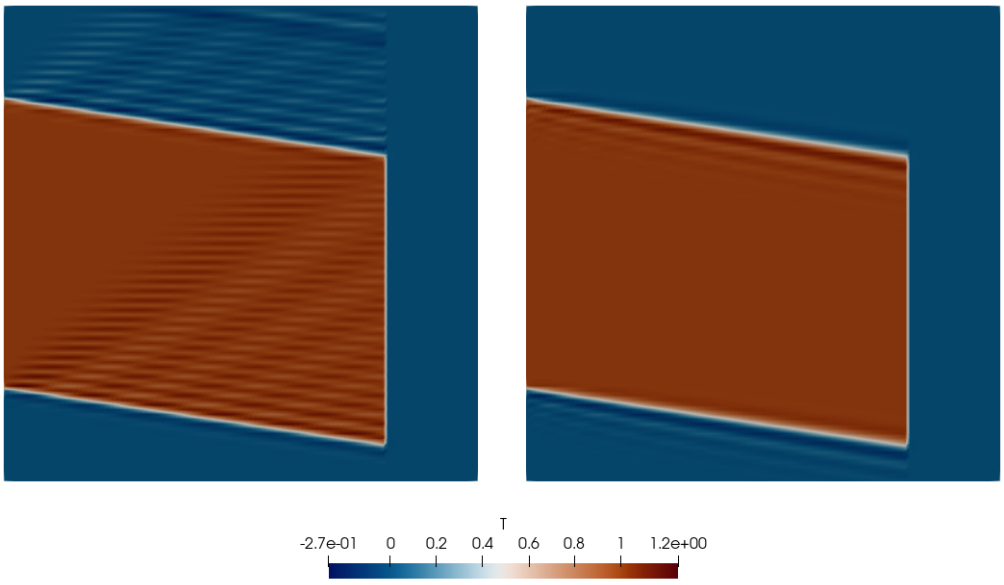
Note that it is currently not possible to run this experiment with ASPECT (?).

Experiment 6 - Bending beam

The domain is a Cartesian box of size 1000×1000km, discretised with 50×50 elements. The CFL number is set to 0.25. The top boundary is an inflow while the bottom boundary is an outflow. We then prescribe $T = 0$ at the top. On the left side $T = 1$ if prescribed if $200 < y < 800$km and 0 elsewhere. The initial temperature field is a 800×600km square attached to the left (see figure). The velocity is given by $\vec{\mathbf{v}} = (0, -x/Lx \cdot v_0)$ with $v_0 = 1$ cm yr^{-1}:

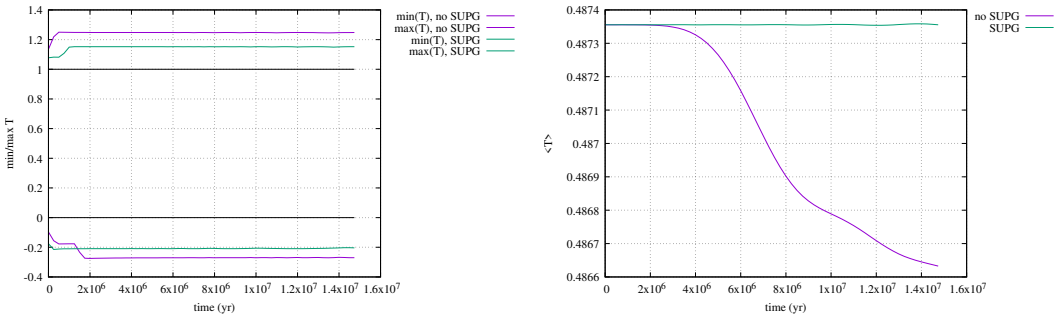


Note that in this case, because of the very small velocity and the large size of the elements, the reported τ value is about 10^{13-15}.

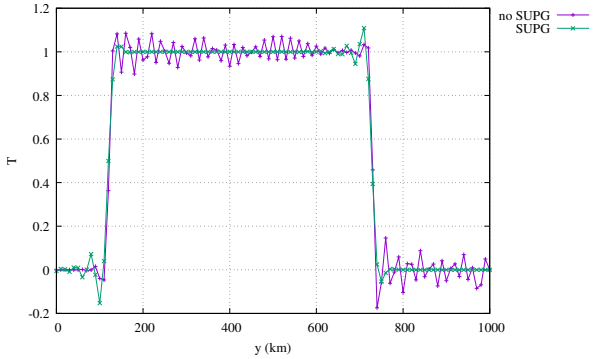


Left: no SUPG; right: with SUPG

Although SUPG removes the small wiggles, it does unfortunately not get rid of the large scale under-/overshoots.

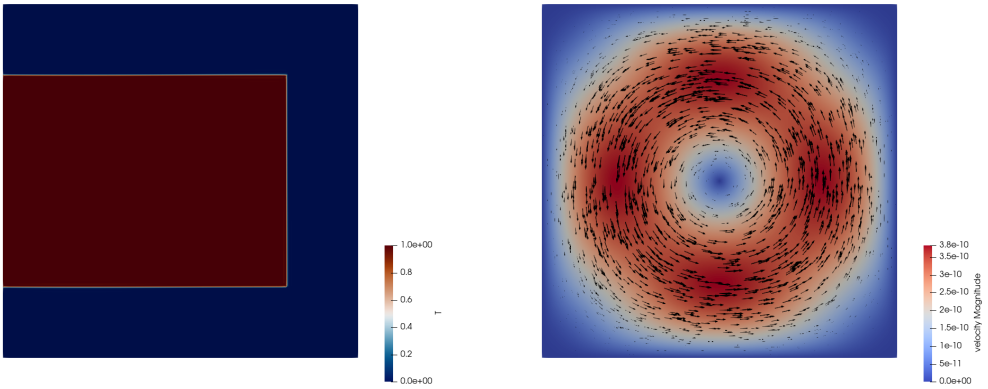


We can also plot the temperature on the vertical line given by $x = L_x/2$:

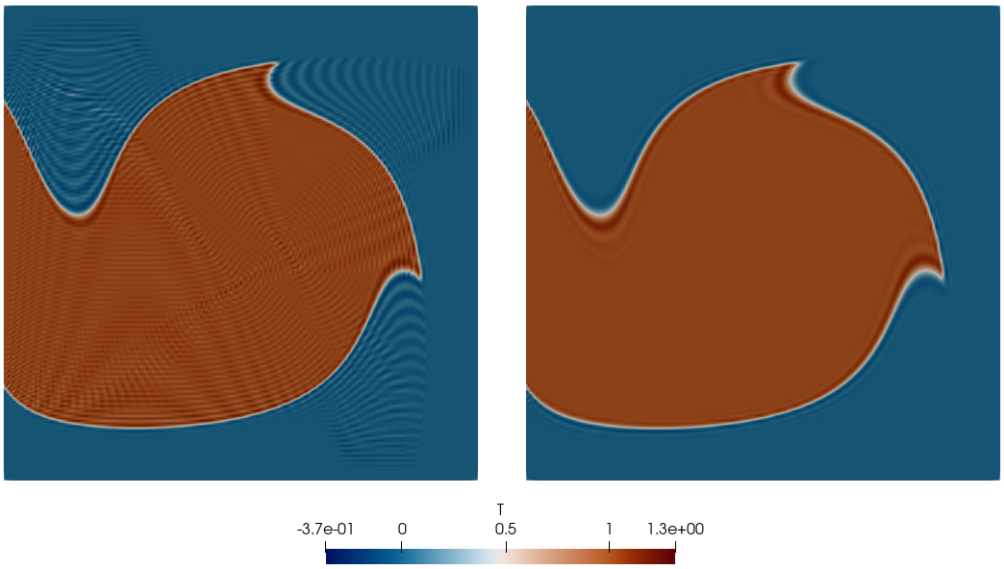


Experiment 7 - Beam in a whirl

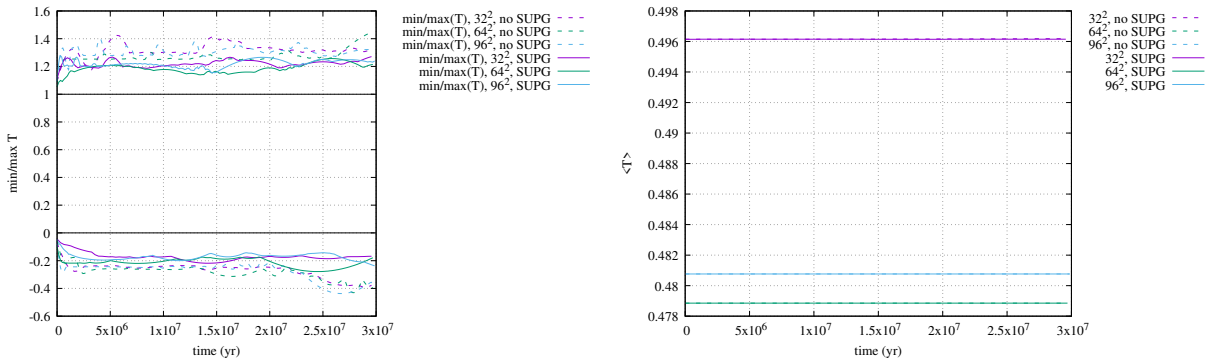
The setup is identical to experiment 6 but the velocity is now given by the equations of Section 18.2 which have been multiplied by 100 cm/year to arrive at a max velocity in the domain of approximately 1cm/year. Since the velocity is parallel/zero on all four boundaries we can prescribe the temperature on all four boundaries. The CFL number is set to 0.25.



Left: Initial temperature field in the domain. Right: velocity field.

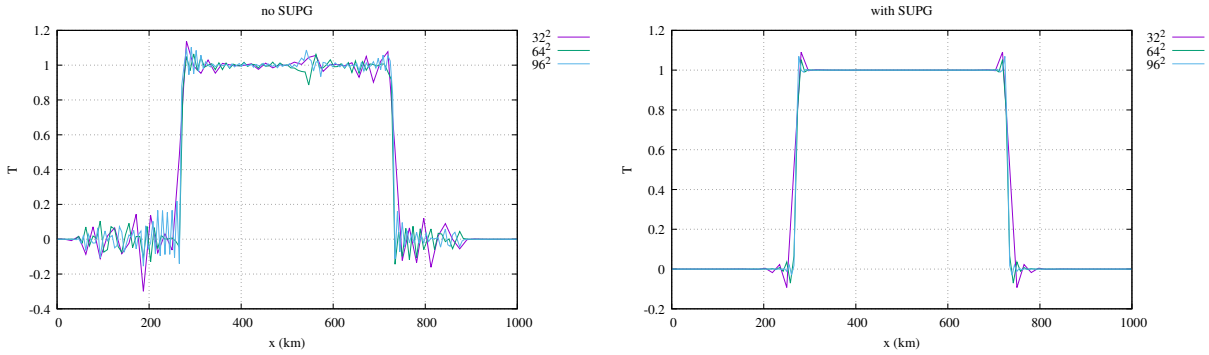


Left: no SUPG; right: with SUPG. Resolution 96^2.



Temperature statistics (min,max,avrg) as a function of time for various resolutions with and without SUPG.

We can also look at a transect of the temperature field from the upper-left corner down to the lower-right corner:



With SUPG the ripples are mostly gone, but over/undershoots near large gradients are about the same with and without supg.

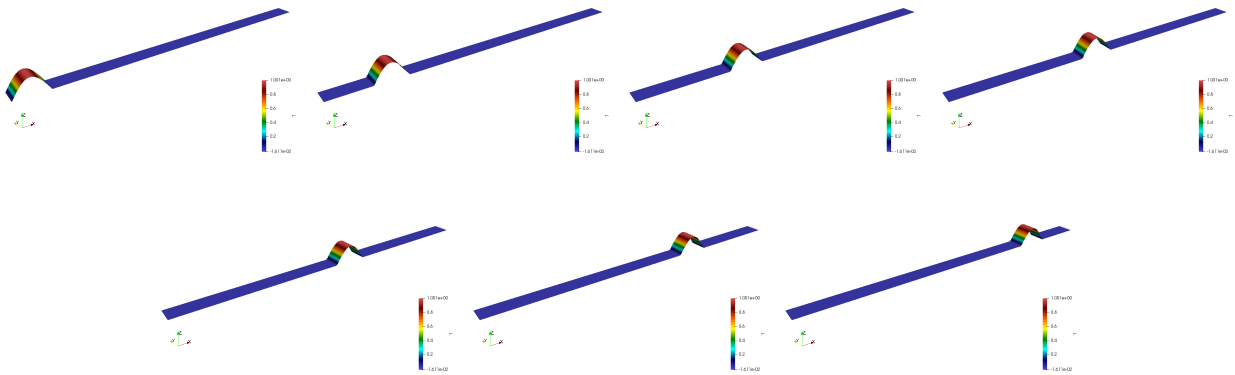
(note that the results above were obtained with slightly different b.c.)

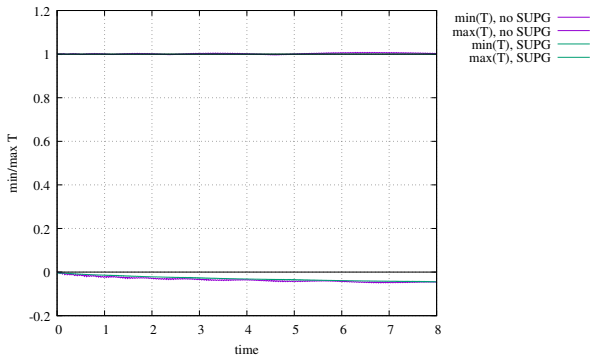
Experiment 8 - 1D Cosine hill advection

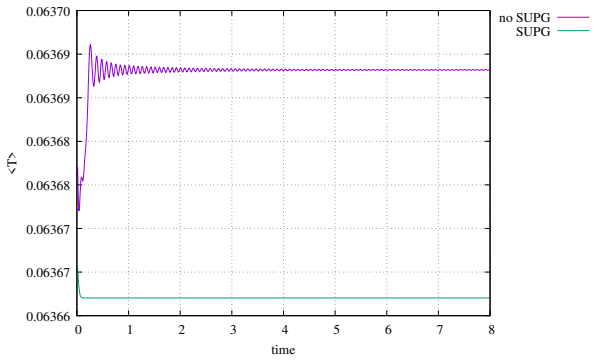
The setup originates in Li [214, ex 5.2]. The domain has dimension $L_x = 1$. The temperature is prescribed on the left to be zero (the right boundary is an outflux no no temperature can be prescribed there). The initial temperature is given by

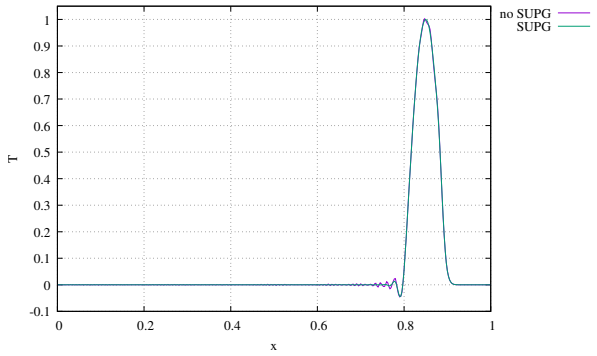
$$T(x,0) = \begin{cases} \sin(10\pi x) & \text{for } x < 0.1 \\ 0 & \text{for } x \geq 0.1 \end{cases}$$

The velocity is set to $u = 0.1$. We use a mesh of 200×4 elements and a CFL number of 0.5. We run the model to time $t = 8$.









Experiment 9 - wiggles advection (deal.II step-9)

The setup for this experiment is borrowed from step-9 of deal.II, found at `https://www.dealii.org/current/doxygen/deal.II/step_9.html`. I have only changed the notations of the main variables to be consistent with the previous experiments of this STONE .

We wish to solve the steady-state advection equation $\vec{\mathrm{v}} \cdot \vec{\nabla} T = f$ where $\vec{\mathrm{v}}(\vec{r})$ is is a vector field that describes the advection direction and speed, $f(\vec{r})$ is a source function, and $T(\vec{r})$ is the solution.

At the inflow, the above equation needs to be augmented by boundary conditions: $T = g$ on $\partial\Omega_-$ where $\partial\Omega_-$ describes the inflow portion of the boundary and is formally defined by

$$\partial\Omega_- = \{\vec{r} \in \partial\Omega : \vec{\mathrm{v}} \cdot \vec{n} < 0\}$$

and $\vec{n}$ being the outward normal to the domain. This definition is quite intuitive, since as $\vec{n}$ points outward, the scalar product with $\vec{\mathrm{v}}$ can only be negative if the transport direction $\vec{\mathrm{v}}$ points inward, i.e. at the inflow boundary. The mathematical theory states that we must not pose any boundary condition on the outflow part of the boundary.

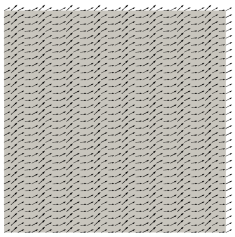
Unfortunately, the equation stated above cannot be solved in a stable way using the standard finite element method. The problem is that solutions to this equation possess insufficient regularity perpendicular to the transport direction: while they are smooth along the streamlines defined by the "wind field" $\vec{\mathrm{v}}$, they may be discontinuous perpendicular to this direction.

We set $\Omega = [-1, 1]^2$, with

$$\begin{aligned}
\vec{\mathrm{v}} &= \left(2, 1 + \frac{4}{5}\sin(8\pi x)\right) \\
f &= 1/10s^2 \qquad \text{with} \quad s = 0.1 \quad \text{for} \quad |\vec{r} - \vec{r}_0| < s \quad \text{otherwise } 0 \\
\vec{r}_0 &= (-\frac{3}{4}, -\frac{3}{4}) \\
g &= \exp(5(1 - |\vec{r}|^2))\sin(16\pi|\vec{r}|^2)
\end{aligned} \tag{43.2}$$

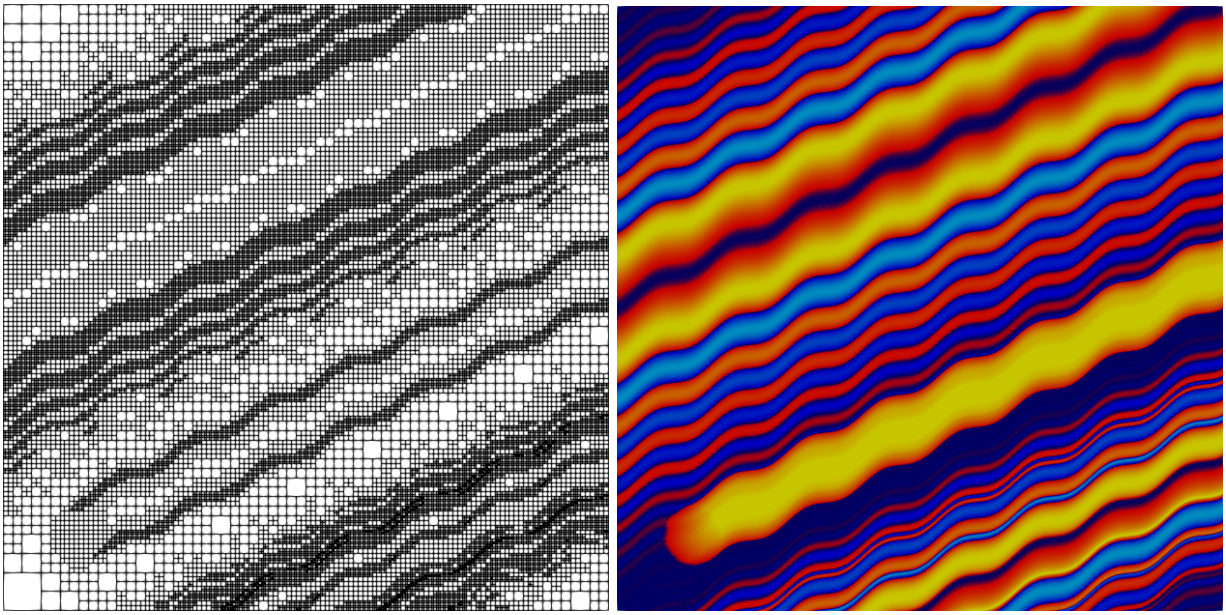
Finally a few additional comments are in order:

1. The advection field $\vec{\mathrm{v}}$ transports the solution roughly in diagonal direction from lower left to upper right, but with a wiggle structure superimposed;



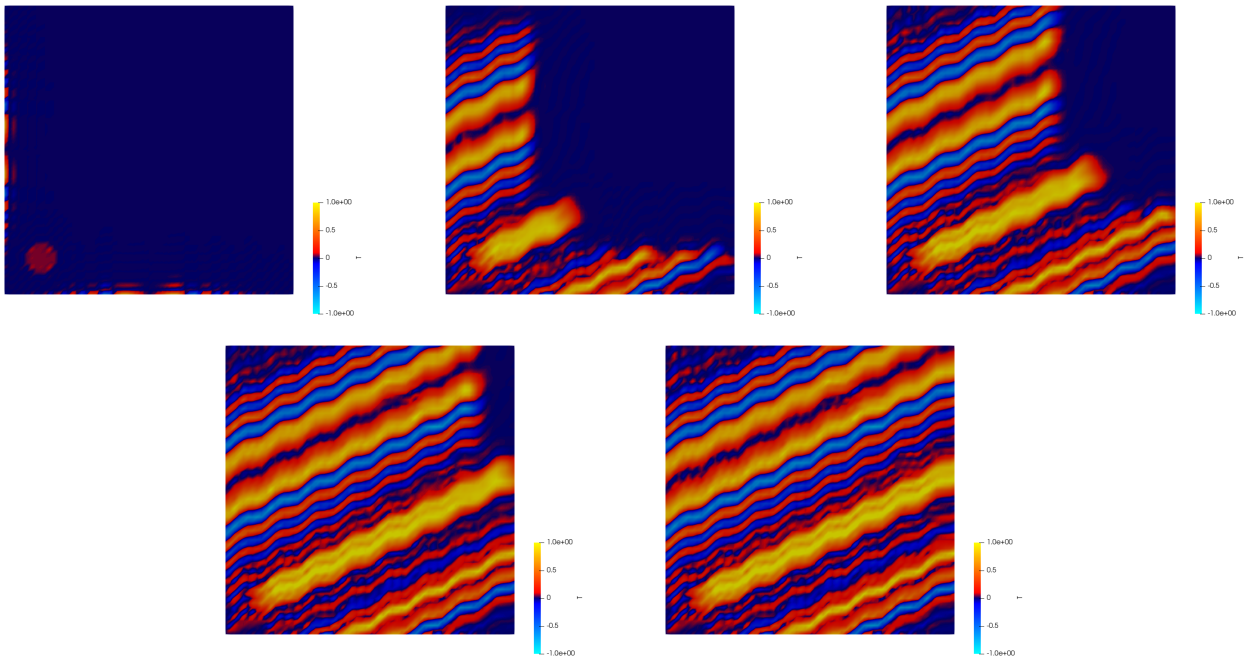
2. The right hand side adds to the field generated by the inflow boundary conditions a blob in the lower left corner, which is then transported along;

3. The inflow boundary conditions impose a weighted sinusoidal structure that is transported along with the flow field. Since $|\vec{r}| \geq 1$ on the boundary, the weighting term never gets very large.



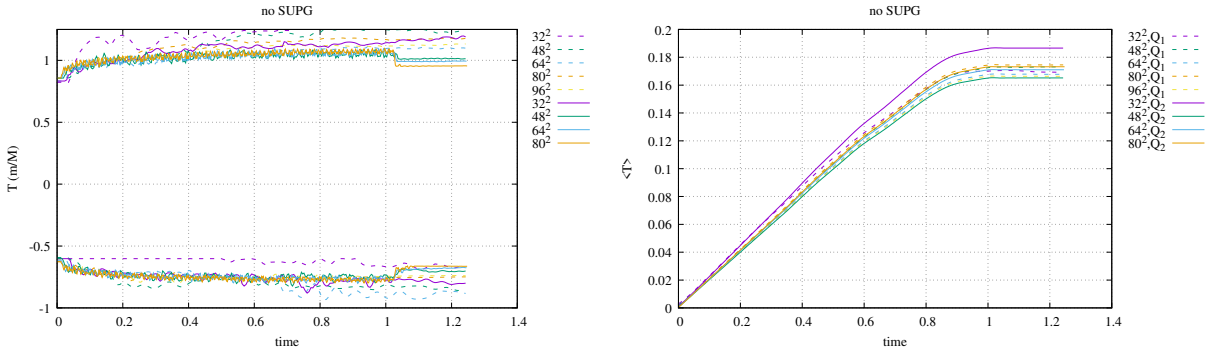
Taken from step-9 webpage. Grid and solution.

The code implements time stepping so I am running models until steady state is reached. Given the velocity field and the size of the box, I found that $t_{final} = 1.25$ is sufficient. The initial temperature field is set to zero. The source term generates a hot spot in a circle in the lower left part of the domain while the boundary conditions imposed on the left and bottom boundaries generate a temperature field that is advected inward:

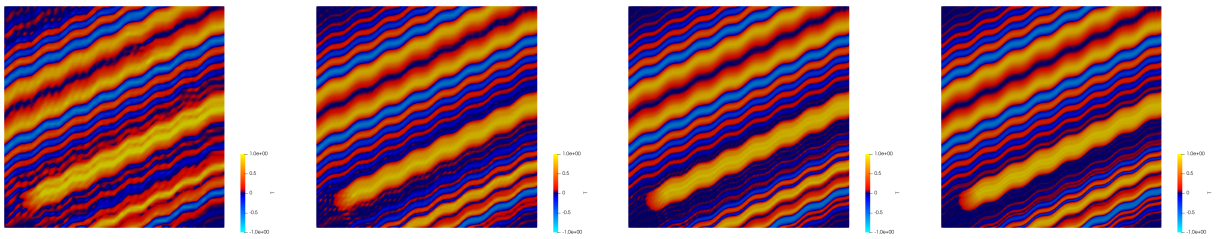


Time evolution of the temperature field using Q_2 basis functions. Results obtained with 32×32 grid for $C_{CFL} = 0.5$ without SUPG.

The following plots show the min/max values of the field and its average for three different resolutions:

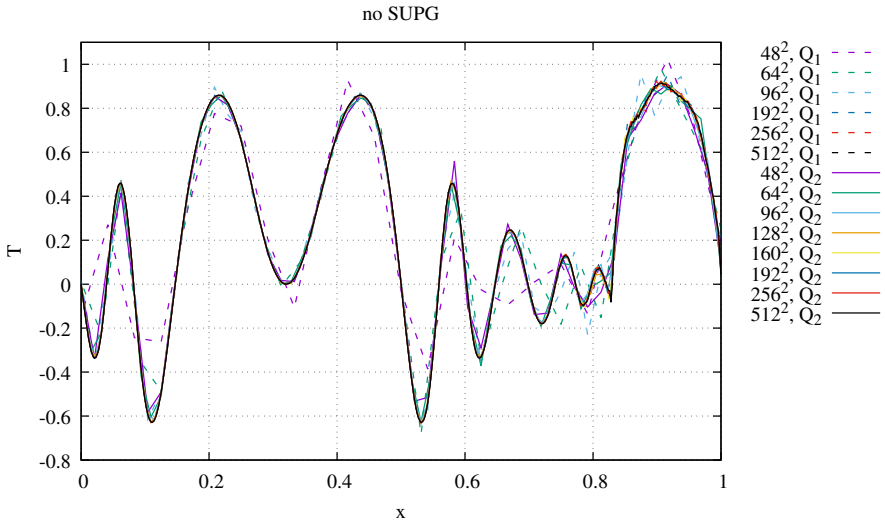


Results obtained with Q_1 and Q_2 basis functions for $C_{CFL} = 0.5$ without SUPG.

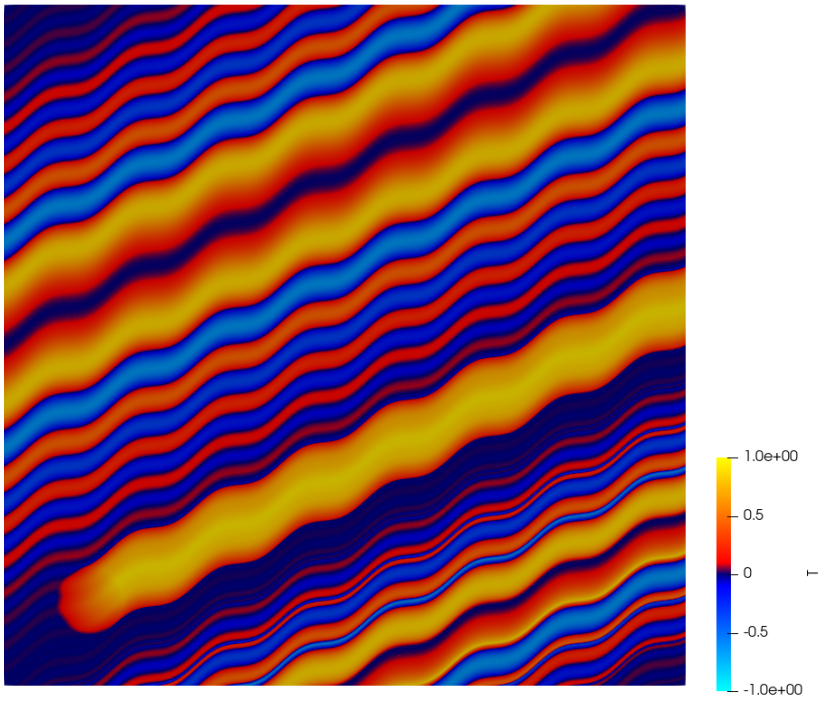


Results obtained with Q_2 basis functions for $C_{CFL} = 0.5$ without SUPG. From left to right: meshes are 32^2, 48^2, 64^2 and 80^2. Obviously this 'Cold and Hot' colorscale is objectively a very bad one but it allows me to compare (visually) with the results obtained with deal.II shown above.

One can also plot the temperature field on a line roughly perpendicular to the direction of the velocity field, i.e. from the upper-left corner to the lower-right corner:

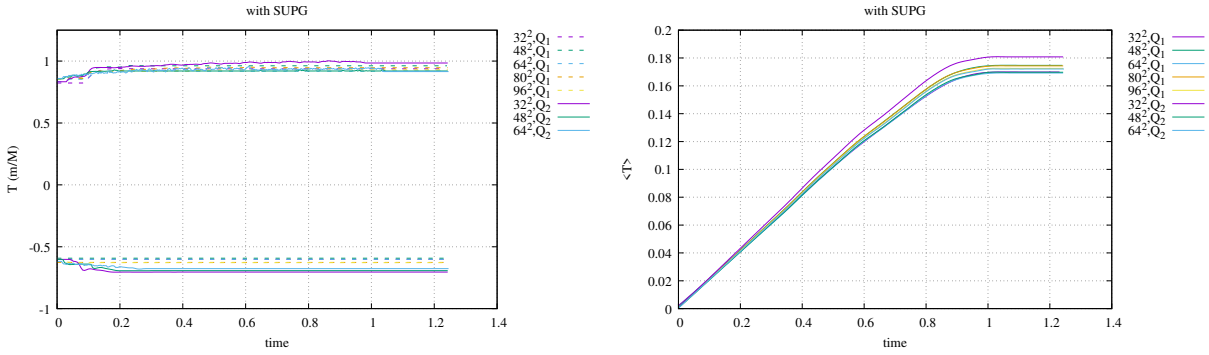


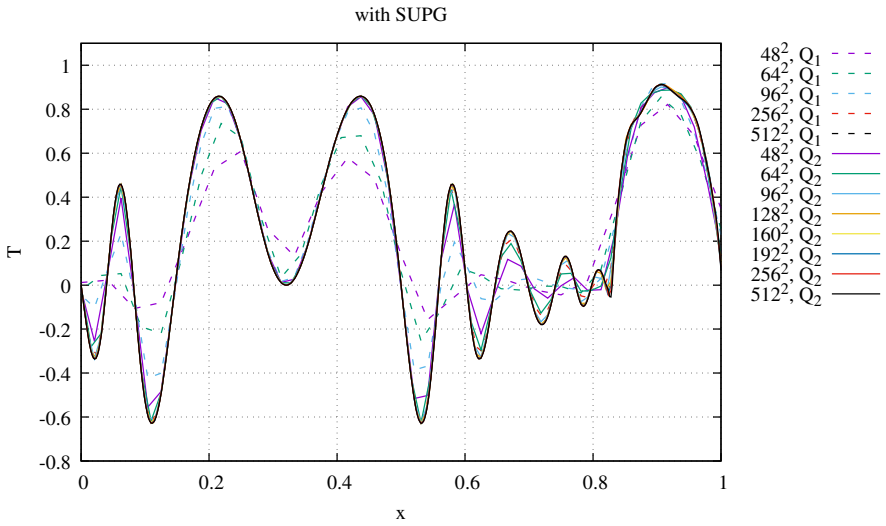
We see that in order to capture the fine features of the temperature field much higher resolutions are needed which lead me to implement a steady state mode that removes the $\partial_t T$ term and yields the desired steady state solution after only one solve. This proves to be necessary for resolutions higher than 64^2.



Steady-state temperature field with 512^2 mesh and Q_2 elements.

We can now look at the effect of SUPG on these results. We therefore set $\tau = h/2|\vec{v}|p$ where p is the order of the Q_p basis functions.





Stone 44

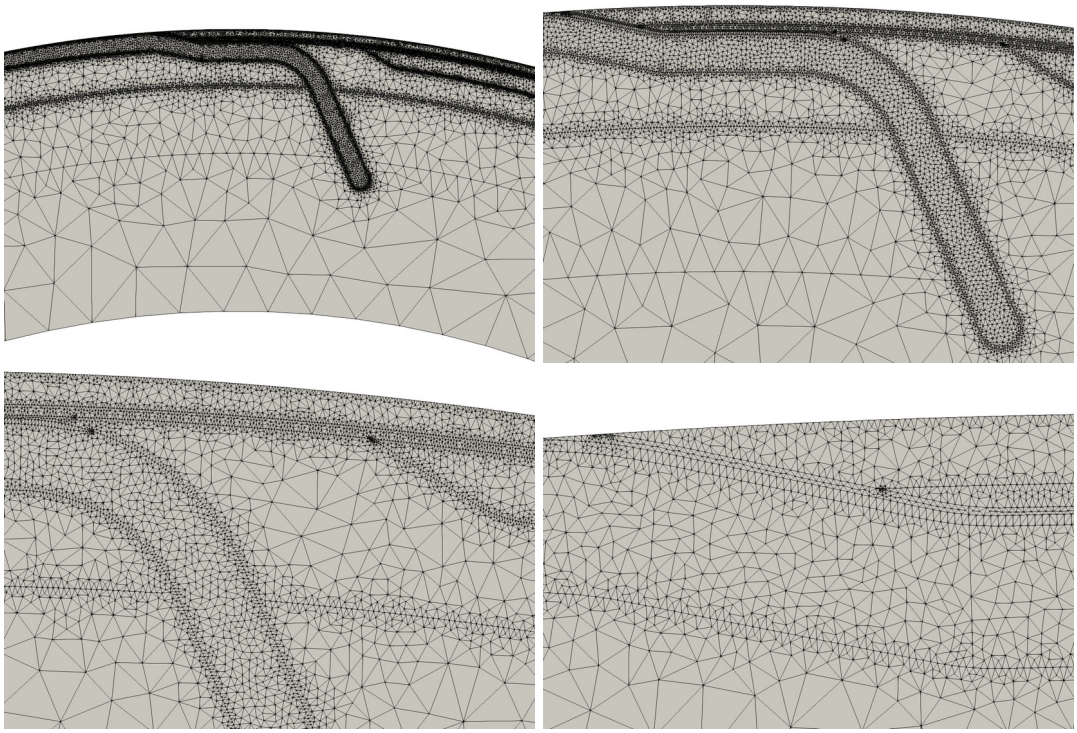
the flat slab

```
2D
chunk
P2pPm1
sparse_matrix
stokes_equation
use_mesher
viscous_rheology
buoyancy_driven
paraview_output
pressure_scaling
matplotlib
```

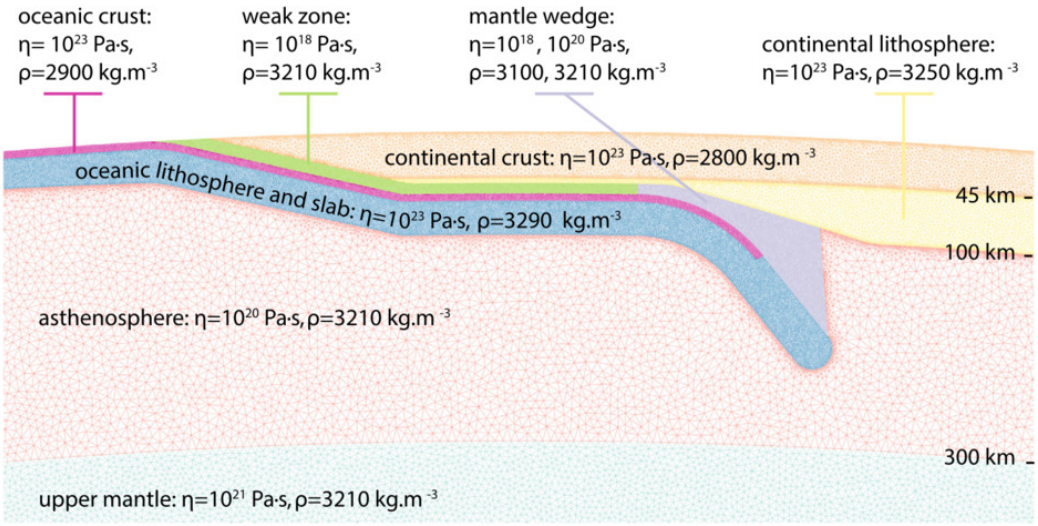
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_44`

WORK in PROGRESS. I am stuck because:

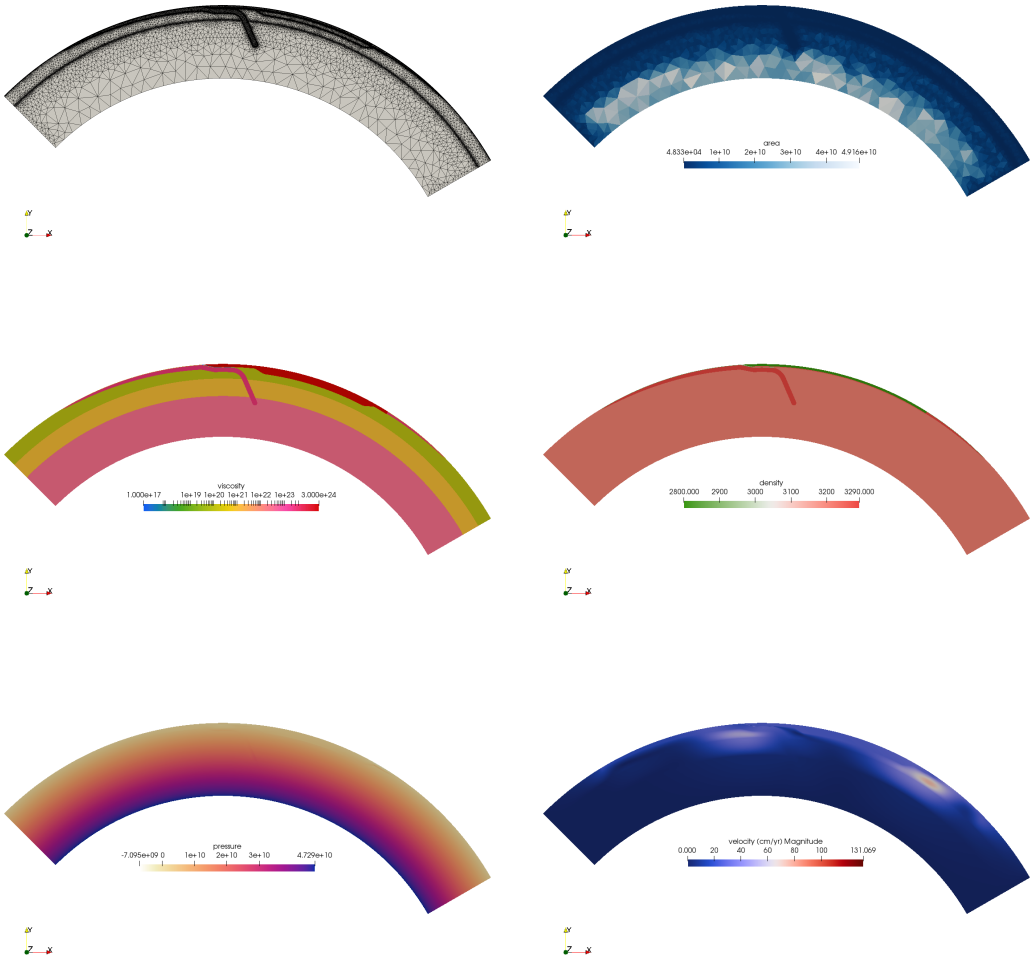
I need a list of nodes on the boundary I need a GCOORD.txt file with more decimals, bc of boundary conditions issues. I need an even lower resolution grid I need the scaling factors for rho,eta, ...



371km, 300km depth, 660km depth, 1500km.



Taken from [301]



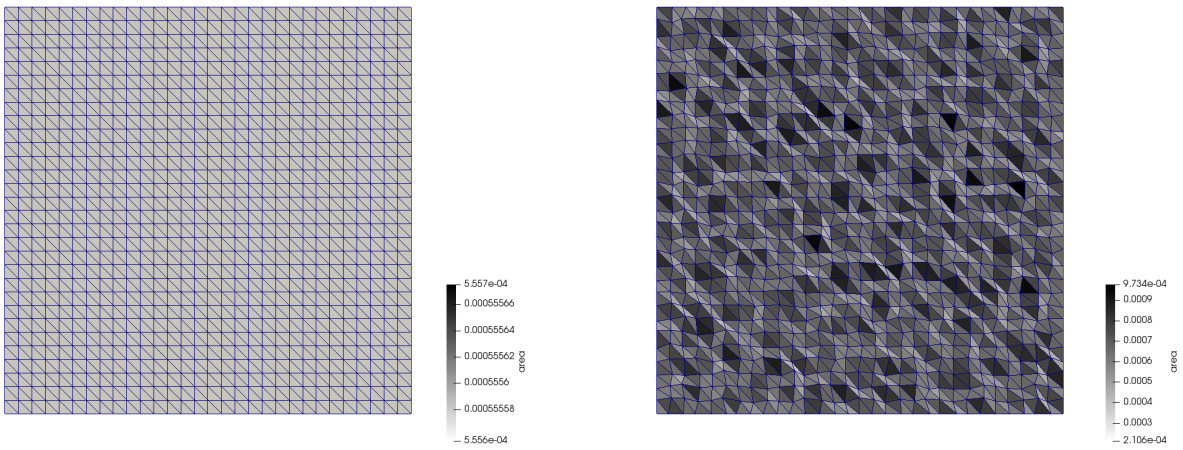
Stone 45

Rotating cone on a triangular (P_1) mesh

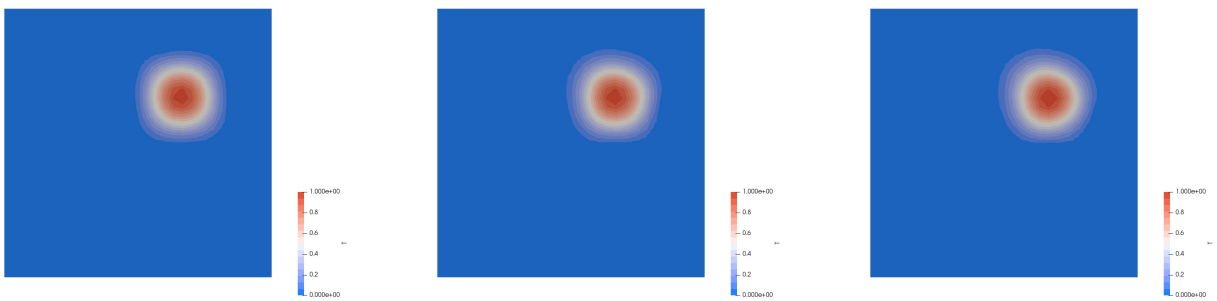
2D
box
P1
numerical_benchmark
energy_equation
time_stepping
paraview_output

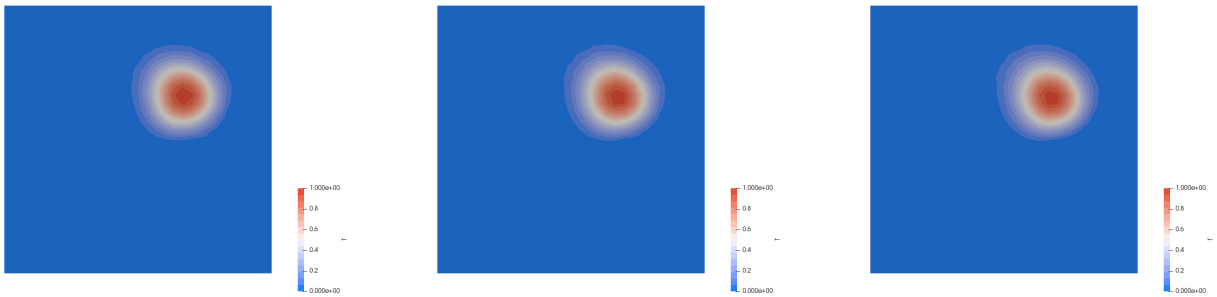
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_45`

The domain is a unit square. Only the pure advection equation is solved. The setup is identical to Experiment 1 of STONE 43. P_1 triangular elements are used. A Crank-Nicolson scheme is used for the time discretisation. The mesh is based on splitting square cells. It is either perfectly regular or has some randomness added to it:

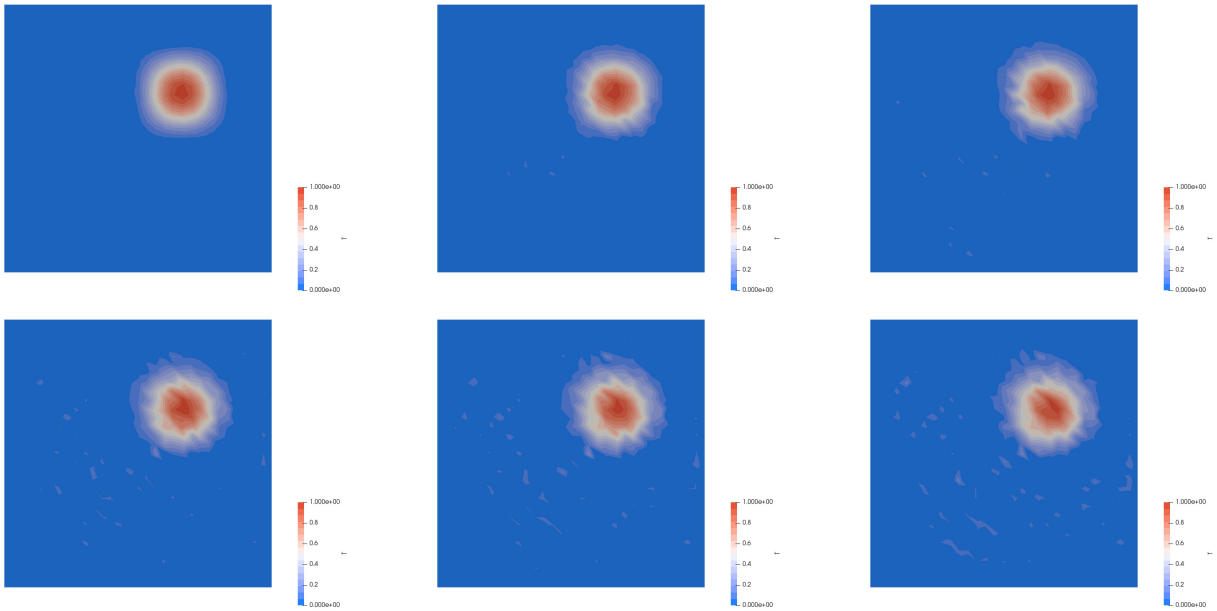


Left: regular mesh of 30x30 cells; Right: same, with random noise added

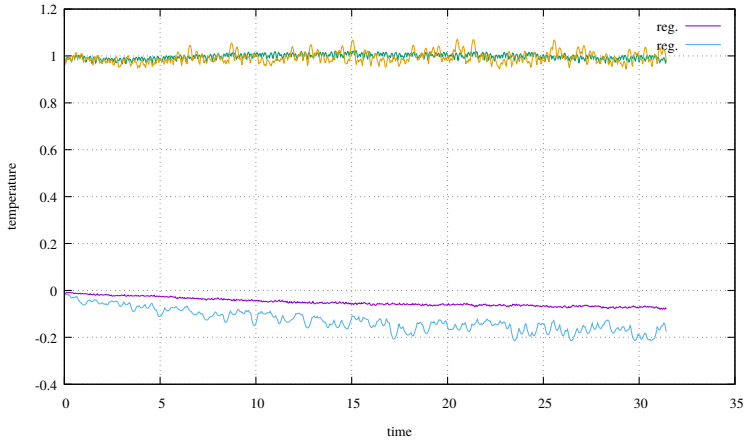




Regular mesh. Temperature field after 0, 2π, 4π, 6π, 8π, 10π rotation



Randomised mesh. Temperature field after 0, 2π, 4π, 6π, 8π, 10π rotation



Minimum/maximum temperature as a function of time for both grids.

Stone 46

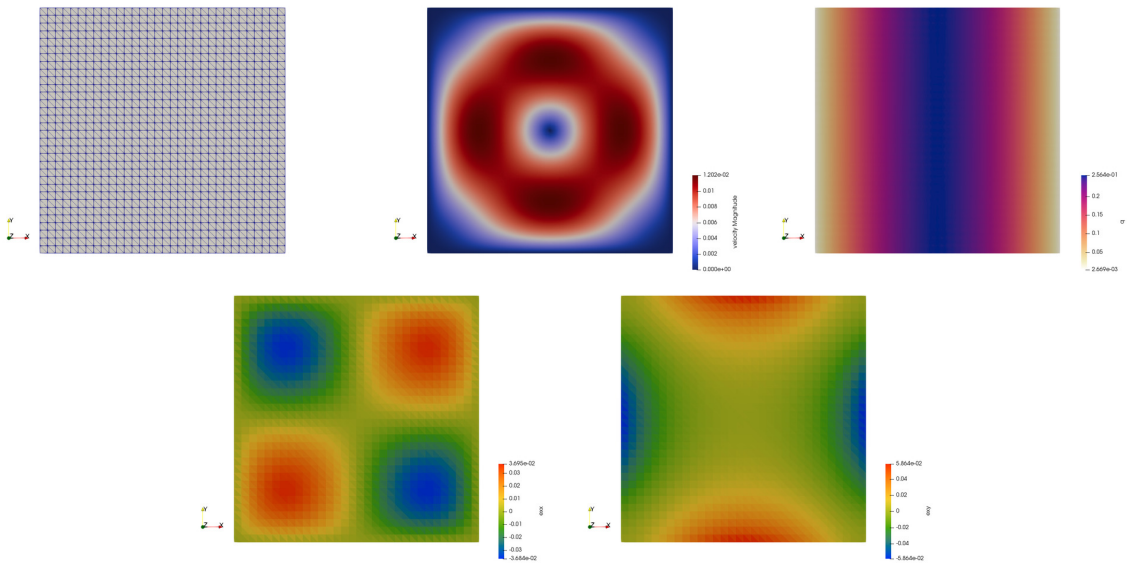
MMS1 with Crouzeix-Raviart ($P_2^+ \times P_{-1}$) elements

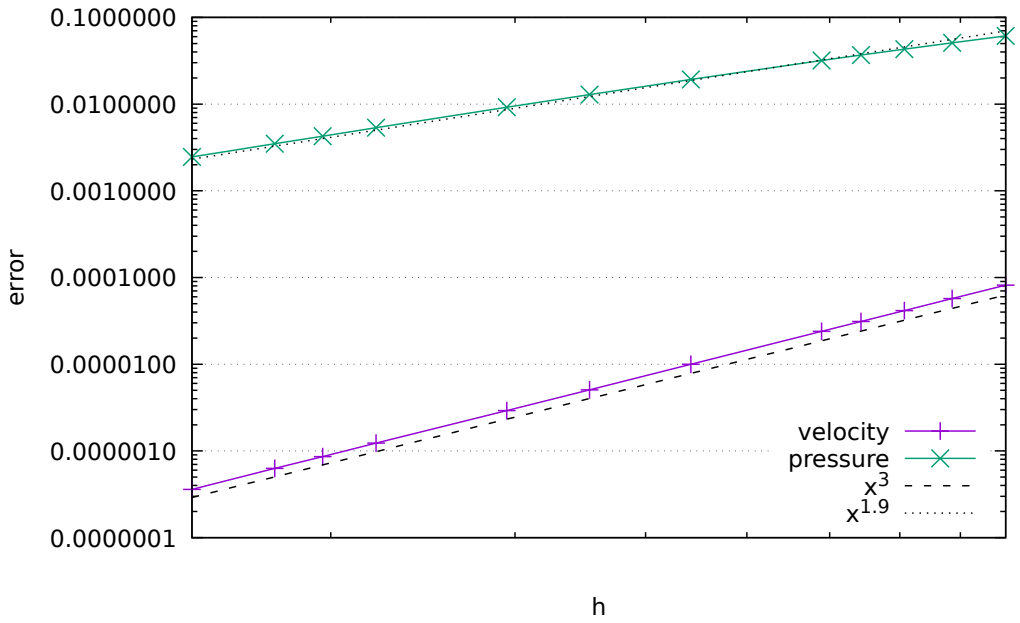
2D
box
P2pPm1
sparse_matrix
numerical_benchmark
error_convergence
stokes_equation
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_46`

This stone showcases the Crouzeix-Raviart element (see Section 10.5.9) used to solve the analytical problem "Donea & Huerta" (see Section 18.2).

Out of convenience the pressure is set to zero at location $(x,y) = (1,1)$, so that the analytical solution is now $p(x,y) = x(1-x)$.





I seem to just not quite obtain a quadratic convergence for pressure. Could it come from the p b c ?

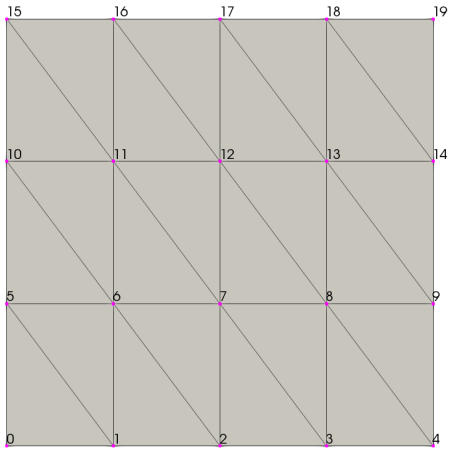
Stone 47

triangular MINI ($P_1^+ \times P_1$) element

2D
box
MINI
analytical_benchmark
error_convergence
stokes_equation
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_47`

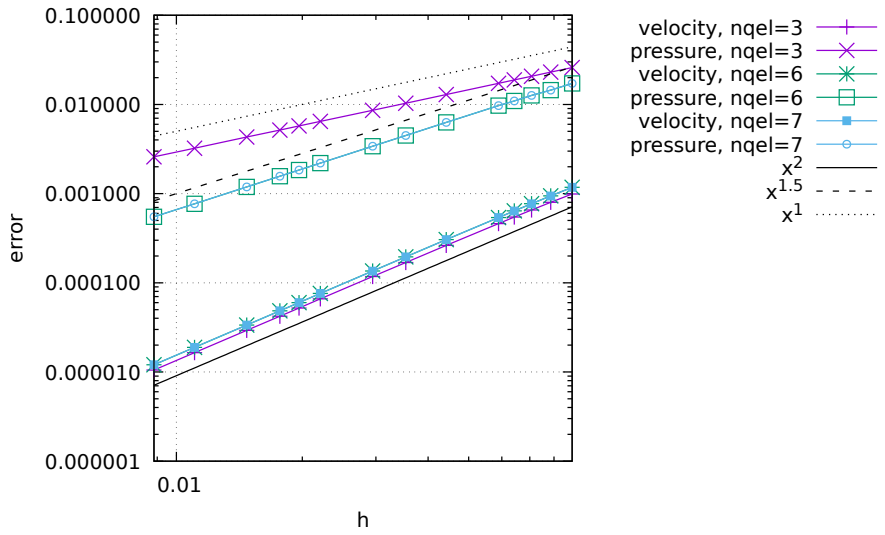
The domain is a unit square and the grid is composed of triangles but for simplicity these are obtained by splitting rectangles in two, as shown hereunder:



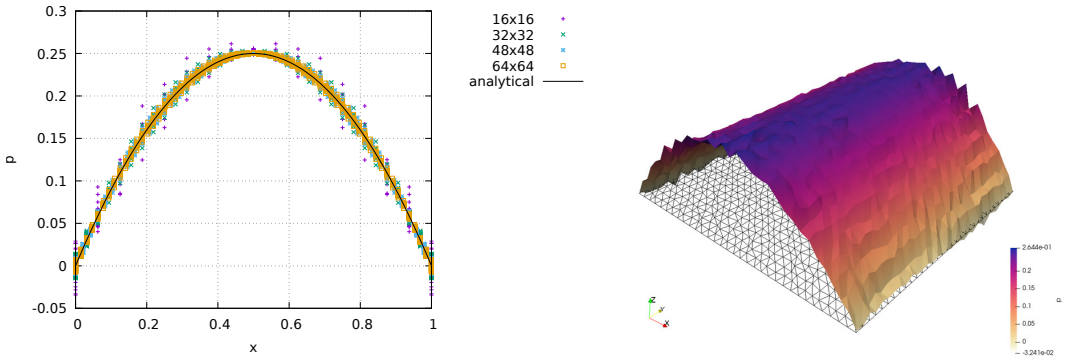
Not shown are the nodes for the bubbles in the middle of each triangle.

This stone showcases the MINI element (see Section 10.5.7) used to solve the analytical problem "Donea & Huerta" (see Section 18.2). Out of convenience the pressure is set to zero at location $(x,y)=(1,1)$, so that the analytical solution is now $p(x,y)=x(1-x)$.

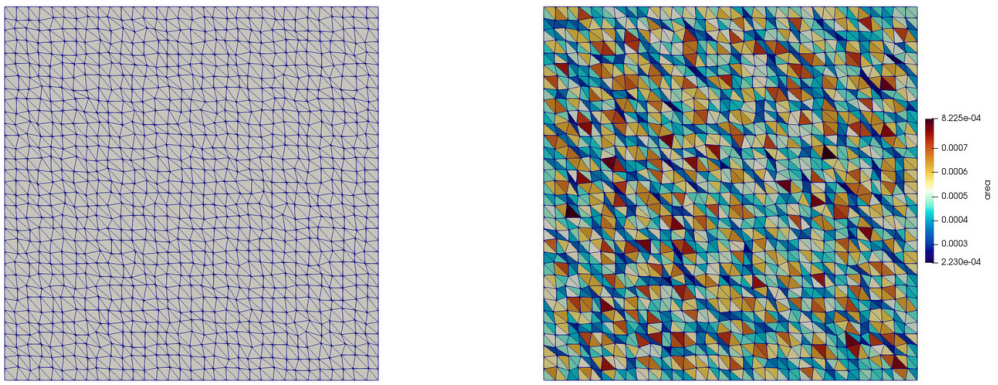
As an experiment I have run convergence tests for two cases: using `nqel=3`, `nqel=6` and `nqel=7` quadrature points. We find that the velocity and pressure errors convergence depends on this crucial parameter. For `nqel=3` the velocity and pressure errors converge quadratically and linearly respectively but for `nqel=6,7` they converge as h^2 and $h^{1.5}$ respectively:



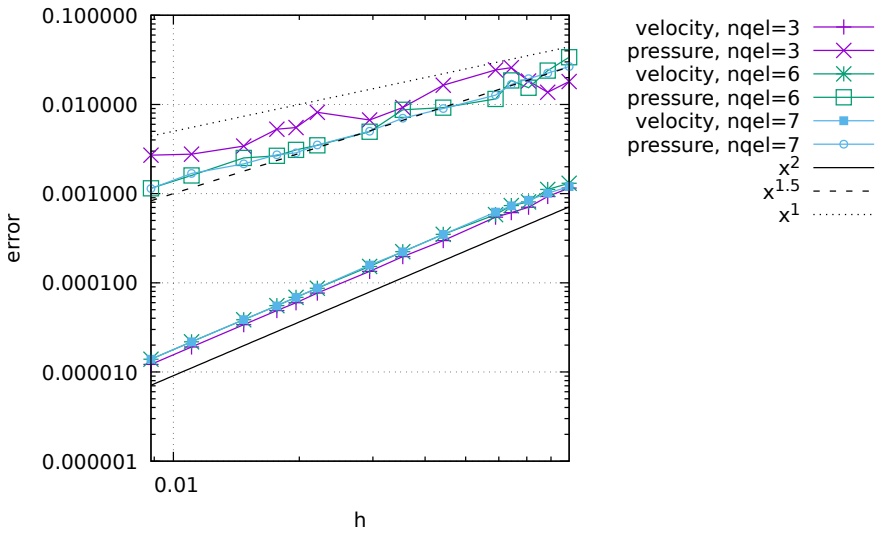
It is worth noticing that although the element is stable, and the error converges at a respectable rate, the pressure solution is not 'clean': as shown on the following figure, there is still some under/overshoot with respect to the analytical solution.



Let us now explore the case where the nodes inside the domain are randomly perturbed, i.e. a random value $(\delta_x, \delta_y) \in [-h_x/5, h_x/5] \times [-h_y/5, h_y/5]$ is added to their position (while preserving the position of the bubble as the barycenter of each triangle), as shown hereunder:



Looking again at the convergence rates of the errors, we see that the velocity errors are virtually unchanged but we observe that the pressure errors no more align on a single line and that the rates are only maintained on average.

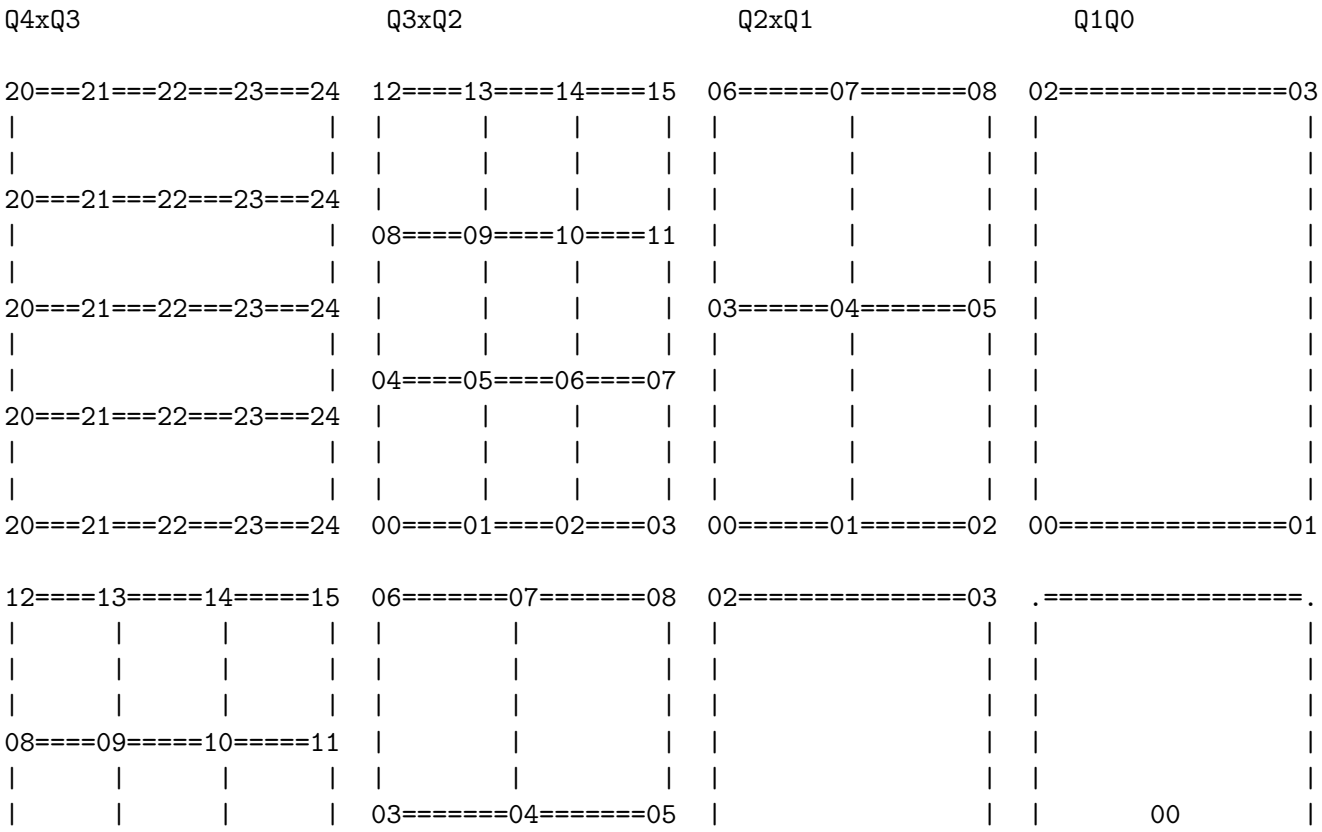

0.100000
0.010000
0.001000
0.000100
0.000010
0.000001
error
0.01
h
velocity, nqel=3
pressure, nqel=3
velocity, nqel=6
pressure, nqel=6
velocity, nqel=7
pressure, nqel=7
x2
x1.5
x1

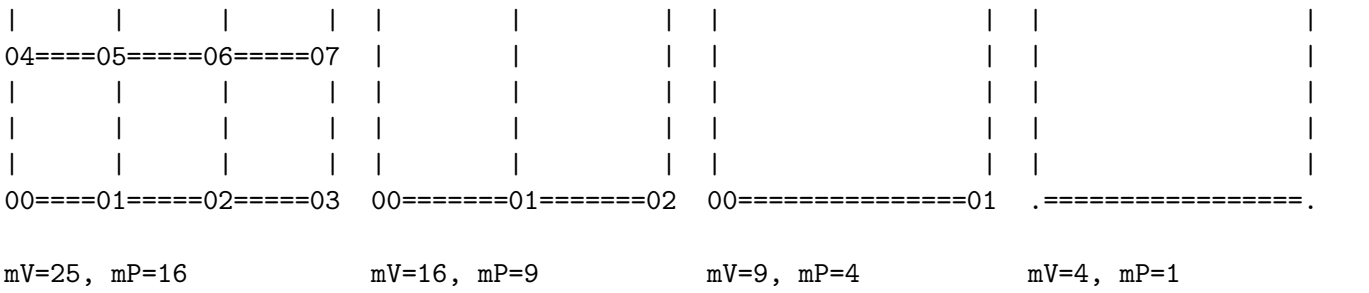
Stone 48

D&H with $Q_1 \times P_0$, $Q_2 \times Q_1$, $Q_3 \times Q_2$and $Q_4 \times Q_3$elements

```
2D
box
Q1P0
Q2Q1
Q3Q2
Q4Q3
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
paraview_output
```

In this experiment we consider 4 different finite elements. The idea behind this stone is to build a code which code(the FEM build and assembly) is common to all. The setup is the Donea & Huerta benchmark (Section 18.2), which has been modified so that the pressure is zero at the top right corner.





In the code the 'order' parameter can take values 1,2,3 and 4 which correspond to the polynomial order of the velocity approximation (Q_1, Q_2, Q_3 and Q_4).

When both nelx and nely values have been chosen, the total number of element for a regular 2D grid is simply:

```
nel=nelx*nely
```

The number of nodes in each direction is then easily computed:

```
nnx=order*nelx+1
nny=order*nely+1
```

and so is then the total number of velocity nodes:

```
NV=nnx*nny
```

The total number of pressure nodes is as follows:

```
if order==1:
   NP=nelx*nely
if order==2:
   NP=(nelx+1)*(nely+1)
if order==3:
   NP=(2*nelx+1)*(2*nely+1)
if order==4:
   NP=(3*nelx+1)*(3*nely+1)
```

Each velocity node has 2 dofs (ndofV=2) while pressure nodes have one dof (ndofP=1) so that the size of the blocks and the assembled FE matrix are given by:

```
NfemV=NV*ndofV
NfemP=NP*ndofP
Nfem=NfemV+NfemP
```

For the linear element, 2 quadrature points per dimension are enough (nqperdim=2), while 3 are necessary for the quadratic element (nqperdim=3) and 4 are used for the cubic element (nqperdim=4), and 5 for the quartic element, which can be conveniently implemented as follows:

```
nqperdim=order+1
```

The quadrature points location and weight is document in Section 7.

Because we wish to use a regular grid, the layout of the points for all three elements can be implemented easily:

```
counter=0
for j in range(0,nny):
    for i in range(0,nnx):
        xV[counter]=i*hx/order
        yV[counter]=j*hy/order
        counter+=1
```

The position of the pressure nodes follows a similar logic.

When it comes to the connectivity array, I first started by building it for each element as follows:

```
if order==1:
   counter=0
   for j in range(0,nely):
       for i in range(0,nelx):
```

```
            iconV[0,counter]=(i)*1+0+(j)*1*nnx+nnx*0
            iconV[1,counter]=(i)*1+1+(j)*1*nnx+nnx*0
            iconV[2,counter]=(i)*1+0+(j)*1*nnx+nnx*1
            iconV[3,counter]=(i)*1+1+(j)*1*nnx+nnx*1
            counter += 1

if order==2:
   counter = 0
   for j in range(0,nely):
       for i in range(0,nelx):
            iconV[0,counter]=(i)*2+0+(j)*2*nnx+nnx*0
            iconV[1,counter]=(i)*2+1+(j)*2*nnx+nnx*0
            iconV[2,counter]=(i)*2+2+(j)*2*nnx+nnx*0
            iconV[3,counter]=(i)*2+0+(j)*2*nnx+nnx*1
            iconV[4,counter]=(i)*2+1+(j)*2*nnx+nnx*1
            iconV[5,counter]=(i)*2+2+(j)*2*nnx+nnx*1
            iconV[6,counter]=(i)*2+0+(j)*2*nnx+nnx*2
            iconV[7,counter]=(i)*2+1+(j)*2*nnx+nnx*2
            iconV[8,counter]=(i)*2+2+(j)*2*nnx+nnx*2
            counter += 1

if order==3:
   counter = 0
   for j in range(0,nely):
       for i in range(0,nelx):
            iconV[ 0,counter]=(i)*3+0+(j)*3*nnx+0*nnx
            iconV[ 1,counter]=(i)*3+1+(j)*3*nnx+0*nnx
            iconV[ 2,counter]=(i)*3+2+(j)*3*nnx+0*nnx
            ...
            iconV[13,counter]=(i)*3+1+(j)*3*nnx+3*nnx
            iconV[14,counter]=(i)*3+2+(j)*3*nnx+3*nnx
            iconV[15,counter]=(i)*3+3+(j)*3*nnx+3*nnx
            counter += 1
```

Having done so, it becomes quickly apparent that the connectivity array can be computed for all elements as follows:

```
counter=0
for j in range(0,nely):
    for i in range(0,nelx):
        counter2=0
        for k in range(0,order+1):
            for l in range(0,order+1):
                iconV[counter2,counter]=i*order+l+j*order*nnx+nnx*k
                counter2+=1
        counter += 1
```

The same approach is taken to build the pressure connectivity array, although the $Q_1 \times P_0$ element requires special attention since the pressure is elemental and attributed to a single node inside the element.

For the other elements I started from:

```
if order==2:
   counter=0
   for j in range(0,nely):
       for i in range(0,nelx):
            iconP[0,counter]=(i)*1+0+(j)*1*(nelx+1)+(nelx+1)*0
            iconP[1,counter]=(i)*1+1+(j)*1*(nelx+1)+(nelx+1)*0
            iconP[2,counter]=(i)*1+0+(j)*1*(nelx+1)+(nelx+1)*1
            iconP[3,counter]=(i)*1+1+(j)*1*(nelx+1)+(nelx+1)*1
            counter += 1

if order==3:
   counter=0
   for j in range(0,nely):
       for i in range(0,nelx):
            iconP[0,counter]=(i)*2+0+(j)*2*(2*nelx+1)+(2*nelx+1)*0
            iconP[1,counter]=(i)*2+1+(j)*2*(2*nelx+1)+(2*nelx+1)*0
            iconP[2,counter]=(i)*2+2+(j)*2*(2*nelx+1)+(2*nelx+1)*0
            iconP[3,counter]=(i)*2+0+(j)*2*(2*nelx+1)+(2*nelx+1)*1
            iconP[4,counter]=(i)*2+1+(j)*2*(2*nelx+1)+(2*nelx+1)*1
            iconP[5,counter]=(i)*2+2+(j)*2*(2*nelx+1)+(2*nelx+1)*1
            iconP[6,counter]=(i)*2+0+(j)*2*(2*nelx+1)+(2*nelx+1)*2
            iconP[7,counter]=(i)*2+1+(j)*2*(2*nelx+1)+(2*nelx+1)*2
            iconP[8,counter]=(i)*2+2+(j)*2*(2*nelx+1)+(2*nelx+1)*2
            counter += 1
```

```
if order==4:
   etc ...
```

and quickly arrived at the following compact form:

```
if order>1:
   om1=order-1
   counter=0
   for j in range(0,nely):
       for i in range(0,nelx):
           counter2=0
           for k in range(0,order):
               for l in range(0,order):
                   iconP[counter2,counter]=i*om1+l+j*om1*(om1*nelx+1)+(om1*nelx+1)*k
                   counter2+=1
           counter += 1
```

The core of the code is rather similar if not identical to other stones (i.e. the loop over elements, the calculation of the elemental matrices, their assembly, and the solve).

What is here somewhat elegant is the projection of the pressure field onto the velocity grid nodes (mostly for plotting purposes). For each element I loop over each velocity node, and evaluate the pressure shape function at this location, compute the pressure with it and add it in the array q while keeping count of how many contributions there are in total per velocity node.

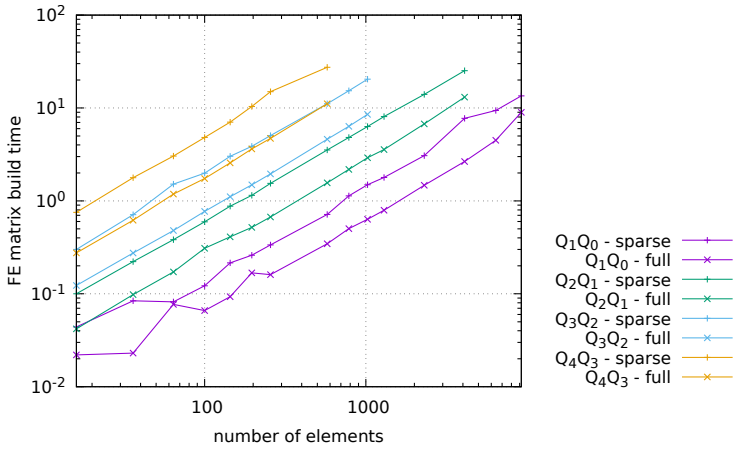
```
for iel in range(0,nel):
    for i in range(0,mV):
        NNNP[0:mP]=NNP(rVnodes[i],sVnodes[i],order)
        q[iconV[i,iel]]+=np.dot(p[iconP[0:mP,iel]],NNNP[0:mP])
        c[iconV[i,iel]]+=1.

q=q/c
```

Finally, since the vtu format does not support higher order elements, I here chose to only extract the corner values for each element, which translates as follows:

```
vtufile.write("<DataArray type='Int32' Name='connectivity' Format='ascii'> \n")
if order==1:
   for iel in range (0,nel):
       vtufile.write("%d%d%d%d\n" %(iconV[0,iel],iconV[1,iel],iconV[3,iel],iconV[2,iel]))
if order==2:
   for iel in range (0,nel):
       vtufile.write("%d%d%d%d\n" %(iconV[0,iel],iconV[2,iel],iconV[8,iel],iconV[6,iel]))
if order==3:
   for iel in range (0,nel):
       vtufile.write("%d%d%d%d\n" %(iconV[0,iel],iconV[3,iel],iconV[15,iel],iconV[12,iel]))
if order==4:
   for iel in range (0,nel):
       vtufile.write("%d%d%d%d\n" %(iconV[0,iel],iconV[4,iel],iconV[24,iel],iconV[20,iel]))
vtufile.write("</DataArray>\n")
```

The following results are obtained by running one of the four scripts *script_errorsX* where X=1,2,3,4. The gnuplot script is to be found in the *images* folder.

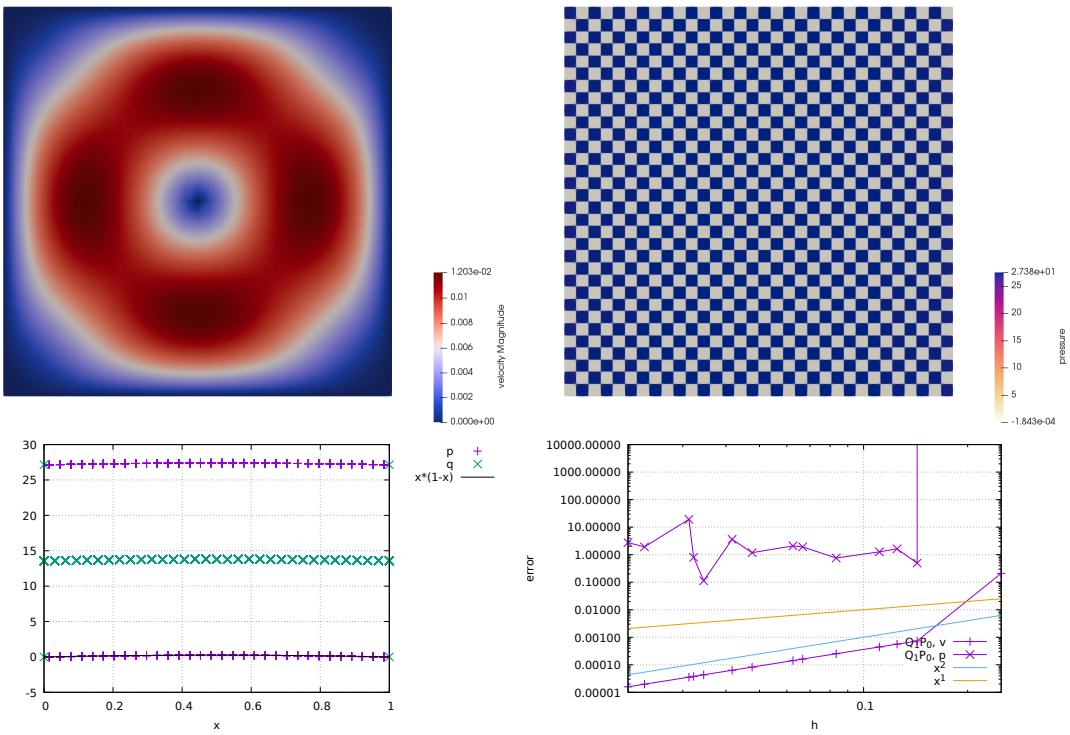
The stone implementes two ways of building the FE matrix. When the flag *sparse* is false, the $\mathbb{K}$ and $\mathbb{G}$ matrices are built as full arrays, later assembled in a larger full array, and then only converted to Compressed Sparse Row it is passed to the solver. When the flag is true, the global FE matrix is defined as a *lil_matrix* (a List of Lists) and it grows in size/memory every time a new term is added to it. As shown on the following plot, it is about twice as slow compared to the first option, but it uses only a fraction of the memory that the first one does.



Also not very surprising: the cost of building the FE matrix increases with the order of the used elements. A matrix corresponding to 100 $Q_1 \times P_0$ elements can be built in about 1s, while it will take 7s when $Q_4 \times Q_3$ elements are used.

The parameter 'mode' in the code allows to switch from the regular form of the Stokes equation to the one presented in Section **??**.

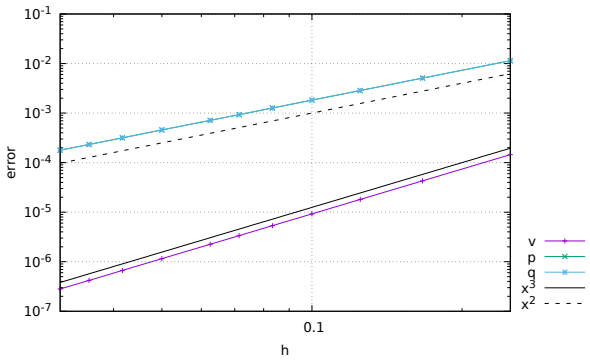
Results with $Q_1 \times P_0$ element



We see that we recover a second order convergence rate for velocity (as expected) but because of the checkerboard pattern the pressure convergence is simply random. The smoothed pressure q shows virtually no checkerboard pattern, except on the boundaries. This is a perfect example for the use of more accurate/clever smoothing procedure, see Section 15.8.

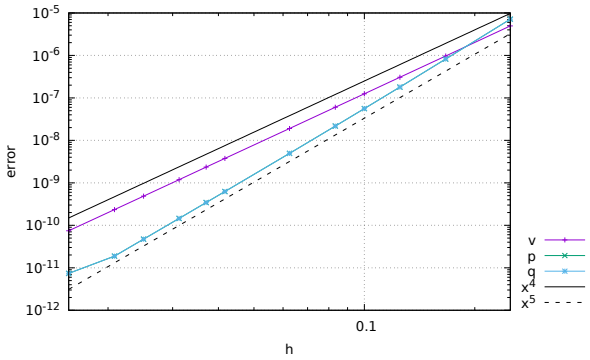
Results with $Q_2 \times Q_1$ element

We recover the cubic convergence for the velocity error and the quadratic convergence for the pressure:



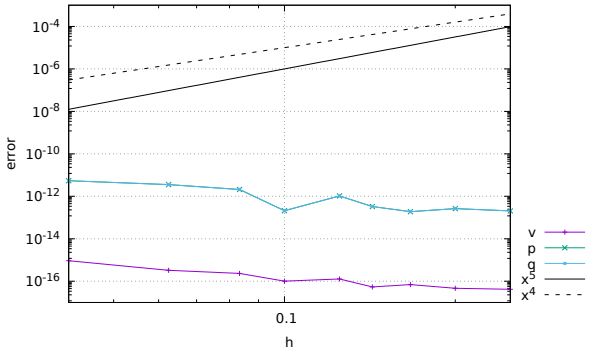
Results with $Q_3 \times Q_2$ element

The analytical solution is a second order polynomial, which means that the pressure shape functions can adequately represent the solution. We recover a fourth-order convergence for the velocity error and a superconvergent (fifth order) pressure error (but why is it 5th order ?).



Results with $Q_4 \times Q_3$ element

Rather interestingly, now both velocity and pressure analytical solutions can be represented exactly by their respective polynomial spaces, so that the errors are at machine precision.



Stone 49

Consistent Boundary Flux method on D&H benchmark with 4 elements

2D
box
Q1P0
Q2Q1
Q3Q2
Q4Q3
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_49`

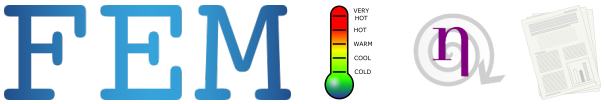
The domain is a unit cube. Free slip boundary conditions

Looking at the four different elements

Looking at the influence of the mas matrix lumping

Stone 50

Lithosphere extension ($Q_2 \times Q_1$)



python_codes/fieldstone_50/text.tex

2D
box
Q2Q1
sparse_matrix
stokes_equation
energy_equation
thermo_mechanical
nonlinear
viscous_rheology
plastic_rheology
paraview_output
matplotlib
pressure_scaling

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_50`

The setup for this stone comes from J. Naliboff.

The code is based on quadrilateral $Q_2 \times Q_1$ elements. The domain is 400 km $\times$ 100 km and the mesh is composed of 160 $\times$ 40 elements. Gravity is Earth-like. There are 4 materials in the domain:

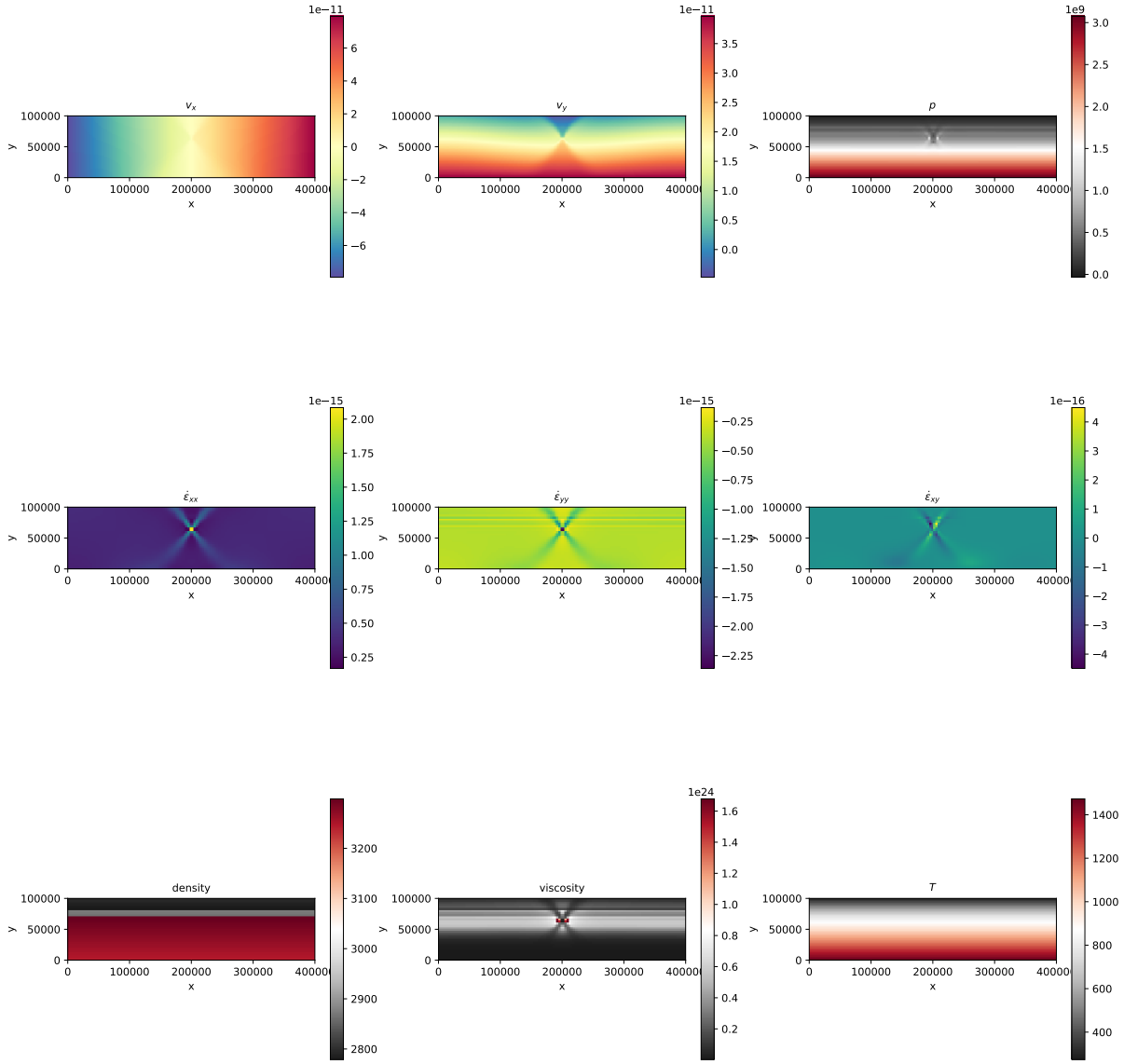
- upper crust (20 km thick layer)
- lower crust (10 km thick layer)
- mantle lithosphere (70 km thick layer)
- weak seed for $60 < y < 68$km and $198 < x < 202$km

Each element is assigned a material number ('imat'). The rheology is visco-plastic, with a dislocation creep rheology for the viscous branch and a Drucker-Prager yield criterion. Nonlinear (Picard) iterations are carried out until a relative tolerance $tol = 10^{-4}$ is reached. TODO!!! The viscosity and density are computed at each quadrature point. For visualisation purposes these are averaged per element.

Kinematical boundary conditions are precribed on the sides and bottom boundaries: $u = -0.25$ cm yr^{-1} on the left, $u = +0.25$ cm yr^{-1} on the right, and $v = +0.125$ cm yr^{-1} on the bottom.

The temperature field is piecewise linear in the y-direction: from 0 °C to 500 °C at 30 km depth, and then to 1500 °C at the bottom. Pressure p is interpolated onto the velocity node (q field). There is no time stepping and the heat transport equation is not solved.

Results are exported to vtu format (solution.vtu) but also by means of matplotlib (solution.pdf). Note that I'd be happy if anybody knows how to fix the size of the colorbars.



For simplicity I compute the following convergence indicators:

$$\xi_u = \frac{||\vec{\mathcal{U}}^k - \vec{\mathcal{U}}^{k-1}||_2}{||\vec{\mathcal{U}}^k + \vec{\mathcal{U}}^{k-1}||_2}$$

$$\xi_v = \frac{||\vec{\mathcal{V}}^k - \vec{\mathcal{V}}^{k-1}||_2}{||\vec{\mathcal{V}}^k + \vec{\mathcal{V}}^{k-1}||_2}$$

$$\xi_p = \frac{||\vec{\mathcal{P}}^k - \vec{\mathcal{P}}^{k-1}||_2}{||\vec{\mathcal{P}}^k + \vec{\mathcal{P}}^{k-1}||_2}$$

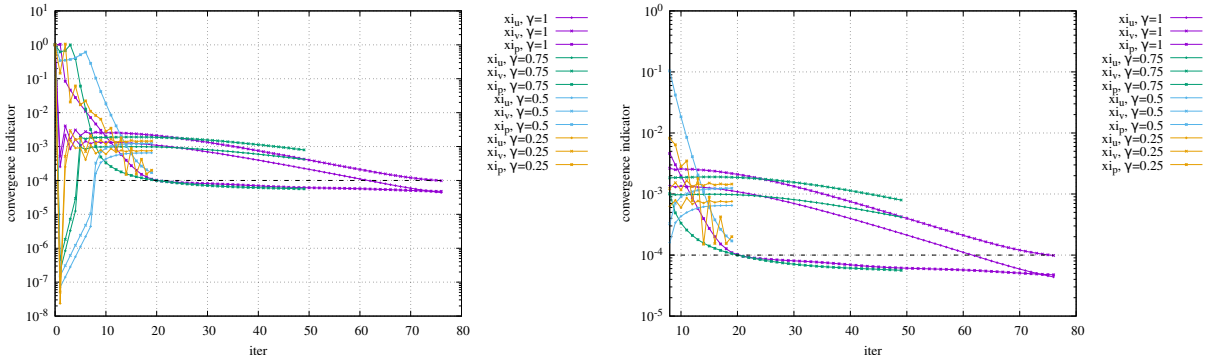
where k denotes the nonlinear loop increment and $\vec{\mathcal{U}}, \vec{\mathcal{V}}, \vec{\mathcal{P}}$ denote the solution vectors. When/if the fields converge then the denominator converges to a single non-zero value and the numerator goes to zero.

Relaxing the viscosity

As an experiment, I have implemented a simple relaxation scheme on the nonlinear viscosity. At each iteration the newly computed viscosity at a quadrature point is weighed by a factor γ and added to the previously obtained viscosity weighed by a factor $(1-\gamma)$.

```
etaq[counterq]=etaq[counterq]*gamma_eta+(1-gamma_eta)*etaq_mem[counterq]
```

If $\gamma = 1$ then no relaxation takes place. Convergence results are shown in the following figure:



$\eta_{ref} = 10^{23}$. Note that I have not run all simulations to convergence.

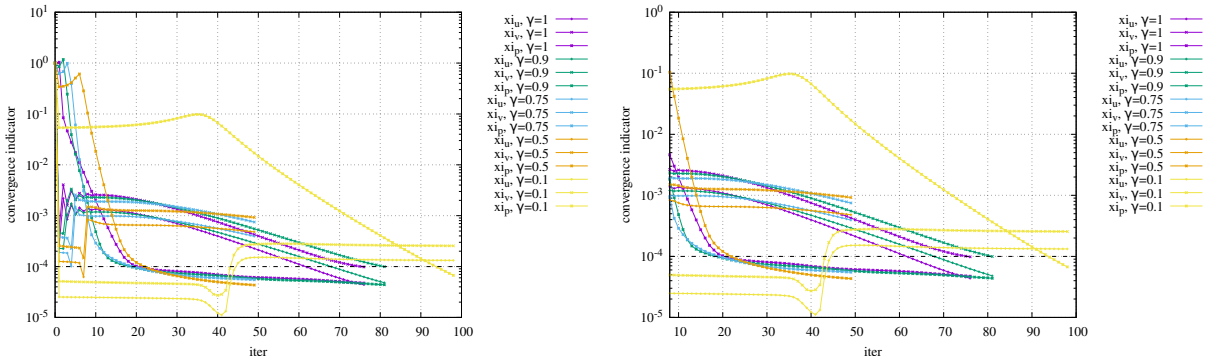
We see that 1) all curves seem to plateau first and then seem to converge very slowly; b) this relaxation scheme does not improve things much at all...

Let us explore the parameters which could alter these convergence results and let us start with the reference viscosity. Changing it to 1e21 does not alter the results.

Relaxing the velocity and pressure

The same approach is now taken for the velocity and pressure (while the viscosity is left alone).

```
u=u*gamma_uvp+u_old*(1-gamma_uvp)
v=v*gamma_uvp+v_old*(1-gamma_uvp)
p=p*gamma_uvp+p_old*(1-gamma_uvp)
```



As before we can conclude that such simple relaxation scheme does not substantially improve (if at all) the convergence.

To Do, open questions, future work?

- compute nonlinear residuals instead of ξ vallues
- extract velocity and/or pressure value at key points and monitor as a function of the nonlinear iterations loop increment

Stone 51

Convection in a triangular domain (MINI)



python_codes/fieldstone_51/text.tex

2D
MINI
numerical_benchmark
stokes_equation
energy_equation
thermo_mechanical
viscous_rheology
buoyancy_driven
vrms
nusselt_number
rayleigh_number
paraview_output
steady_state

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_51`

This fieldstone was developed in collaboration with L. van de Wiel.

The following problem is studied in John *et al.* (2017) [172]. The equations that they solve are the thermo-mechanically coupled steady state equations:

$$-\vec{\nabla}p + \Delta\vec{\mathrm{v}} + \mathsf{Ra}T\vec{e}_y = 0 \tag{51.1}$$

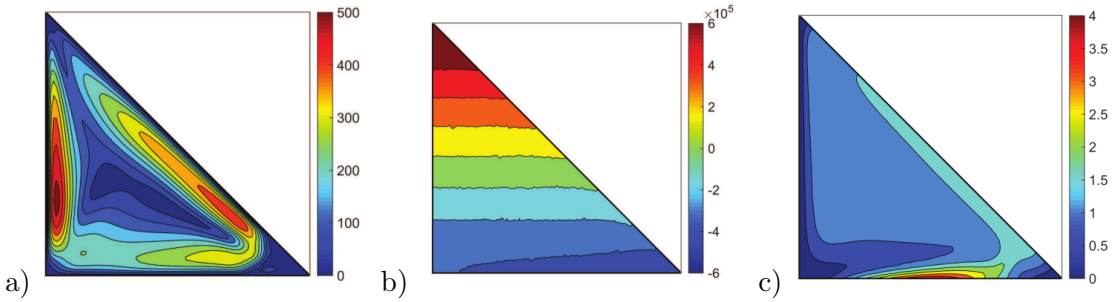
$$\vec{\nabla}\cdot\vec{\mathrm{v}} = 0 \tag{51.2}$$

$$-\Delta T + \vec{\mathrm{v}}\cdot\nabla T = 0 \tag{51.3}$$

In our case the code is based on the MINI element (a.k.a. $P_1^+ \times P_1$), see Section 10.5.7 and we set $\mathsf{Ra} = 10^6$.

The domain is chosen to be the right triangle with vertices $(0,0)$, $(1,0)$, and $(0,1)$. The boundary is considered to be solid walls (no-slip). For the temperature, a sinusoidal heat source is enforced on the bottom boundary with a Dirichlet condition ($T(x) = 2(1-\cos(2\pi x))$), the left wall is set to a constant temperature of zero, and the hypotenuse wall is perfectly insulated so that a Neumann boundary condition is appropriate.

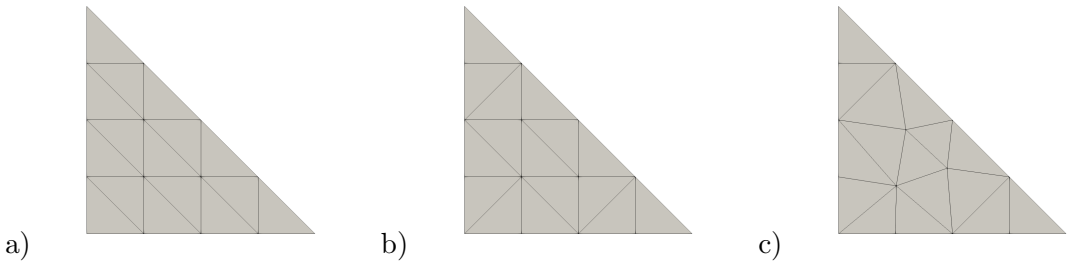
The steady state velocity pressure and temperature fields are as follows:



Taken from [172]. Steady state fields: a) velocity, b) pressure, c) temperature.

Although it is not mentioned in the original article it appears that the pressure field has been normalised so that $\langle p \rangle = \int_\Omega p dV = 0$ (because of the nature of the velocity boundary conditions the pressure solution has a nullspace).

As opposed to the mesh presented in [172] I build a regular mesh. An example of such a mesh is shown hereunder (a) for $n = 5$ (the number of nodes per side of the triangular domain). In order to generate a mesh which is more isotropic some edges between triangles can be flipped (b). Note that this mesh can also be modified in such a way that the position of nodes inside the domain is perturbed by a small random value (c). In what follows I denote by h the distance between nodes on the horizontal (or vertical) boundaries, i.e. $h = L_x/(n-1) = L_y/(n-1)$.



a) regular mesh for n=5. b) flipped edges mesh. c) randomized+flipped edges mesh.

Since I am solving for the steady-state solution I set the mass matrix in the heat transport part of the code to zero. However, since I am solving the stokes equations and the heat transport equation alternatively until convergence is reached, it is well known that this approach does not converge fast (if at all). I then implement a simple relaxation scheme (e.g. Vynnytska, Rognes, and Clark [343]). After I have solved for velocity (using the most recent temperature field in the rhs), I do:

$$\vec{\mathbf{v}}^k = \gamma \vec{\mathbf{v}}^k + (1-\gamma)\vec{\mathbf{v}}^{k-1}$$

and after having solved for temperature having used the most recent velocity field, I do the same for temperature:

$$T^k = \gamma T^k + (1-\gamma)T^{k-1}$$

where the relaxation parameter γ is between 0 and 1.

Additionally I measure:

- the Nusselt number defined by

$$Nu = \int_{y=0} \vec{\nabla} T \cdot \vec{n} dS = -\int_{x=0}^{x=1} \frac{\partial T}{\partial y} dx$$

  It is reported to be 24.535 in John, Linke, Merdon, Neilan, and Rebholz [172] (2017).

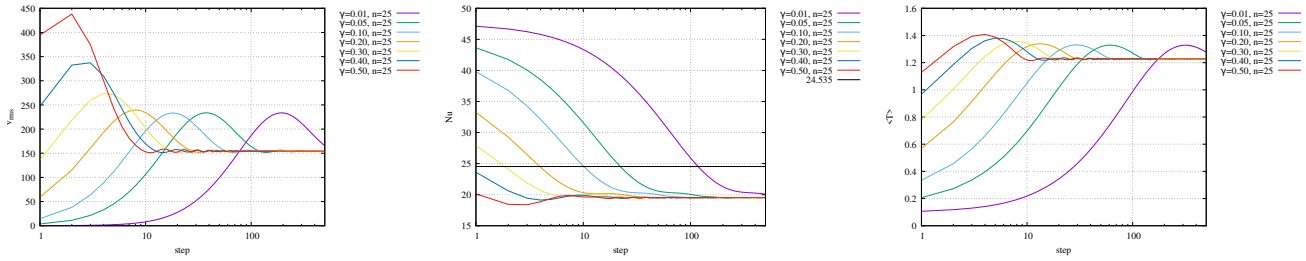
- the temperature on the hypotenuse.

- the root mean square velocity

- the mean temperature $\langle T \rangle = |\Omega|^{-1} \int_\Omega T \, dV$

On the importance of the relaxation parameter γ

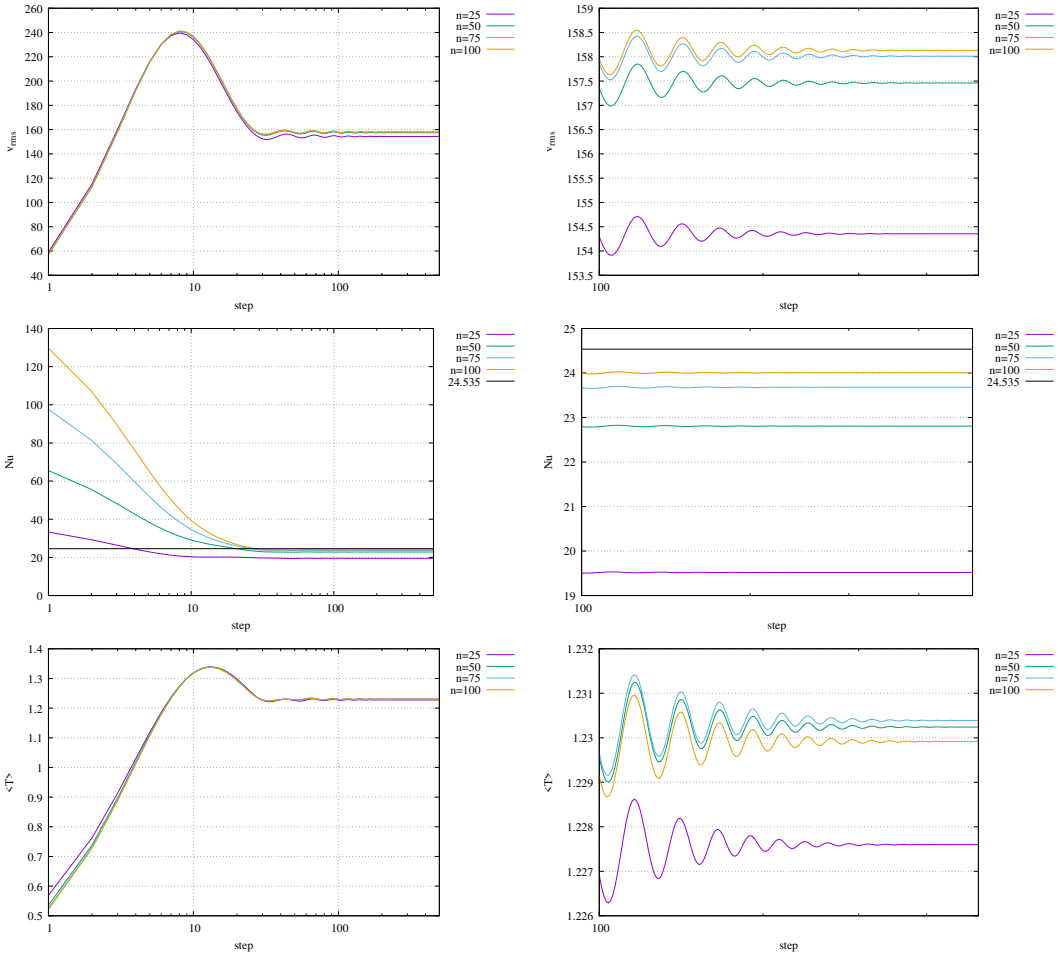
In what follows the internal node coordinate randomness is switched off. I have run the model for various values of γ for $n = 25$, and results are shown on the following figures. We see that all simulations seem to converge to the same steady state, which is very reassuring. However, it looks like too small a value of γ delays greatly the convergence and too large a value also seems detrimental. In light of this, I have chosen $\gamma = 0.2$ for all what follows.



Left to right: Root mean square velocity, Nusselt number and average temperature as a function of the iteration counter.

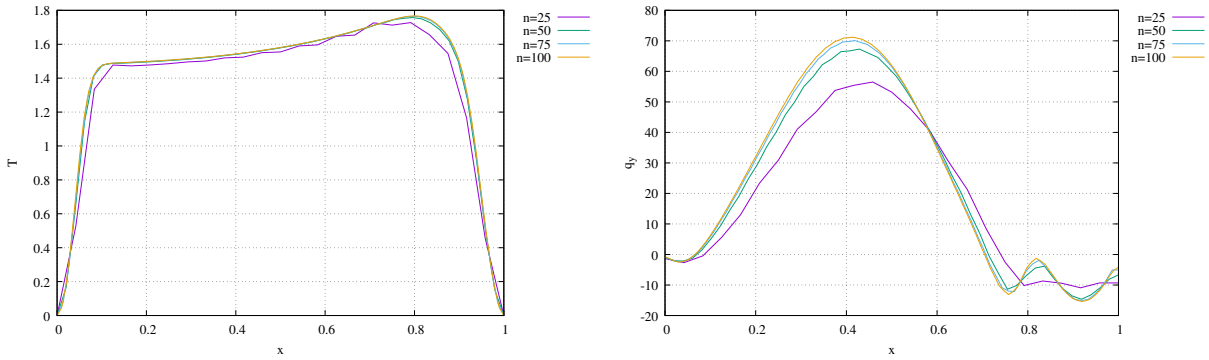
On the influence of mesh resolution

I now explore the influence of the mesh resolution on the results and run steady state calculations for $n = 25, 50, 75, 100$.

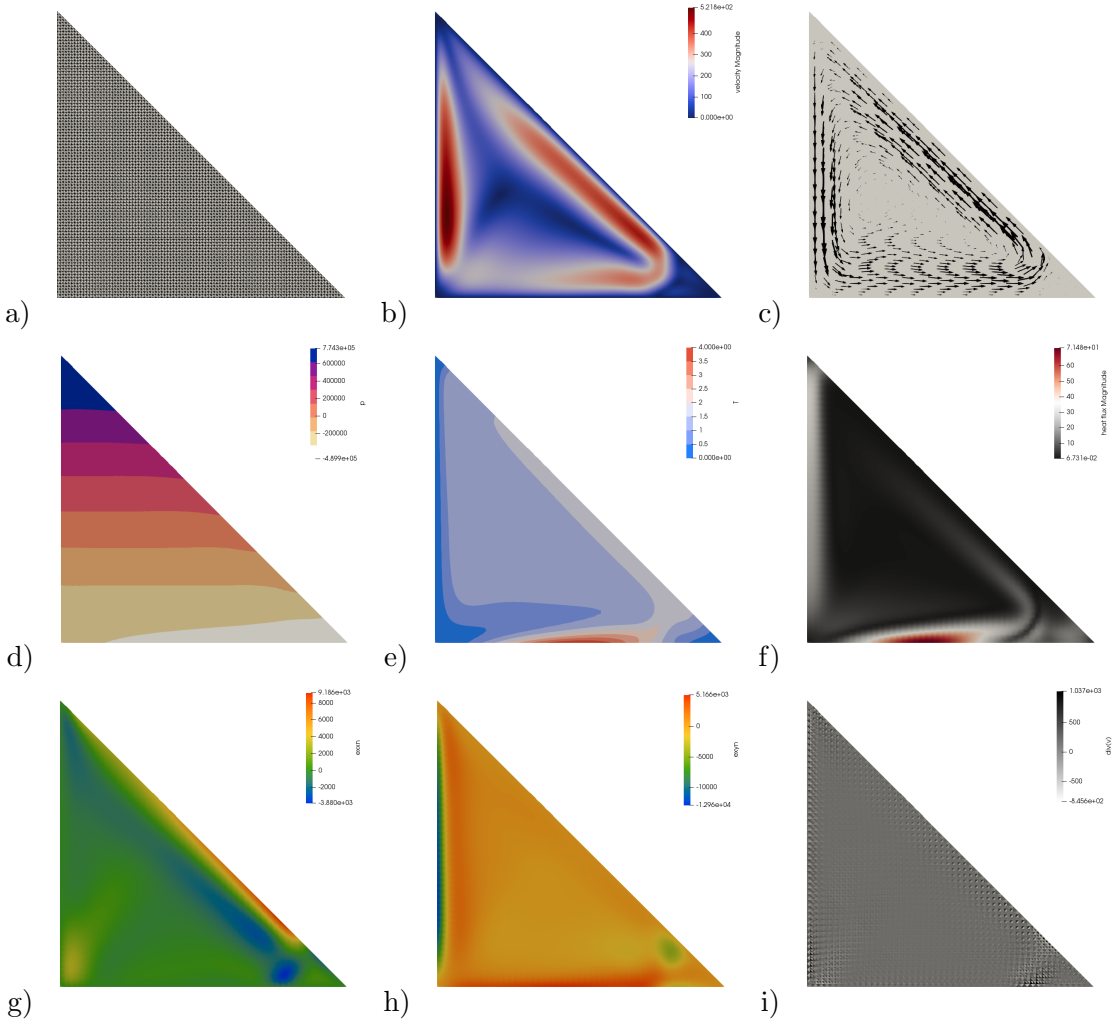


resolution (n)	v_{rms}	Nu	$\langle T \rangle$
25	154.35406	19.51906	1.22759
50	157.46000	22.80683	1.23024
75	158.01207	23.67889	1.23038
100	158.13157	24.00510	1.22991

The following two plots show the temperature along the hypotenuse (as a function of x for simplicity) at steady-state and the heat flux measured at the bottom.



Finally, here are the fields for $n = 100$ at steady state:



Steady state fields: a) grid (n=100), b,c) velocity, d) pressure, e) temperature, f) heat flux, g) $\dot{\varepsilon}_{xx}$, h) $\dot{\varepsilon}_{xy}$, i) velocity divergence measured in the middle of the element.

Stone 52

Serendipity element in 2D ($Q_2 \times Q_1$, $Q_2^{(8)} \times Q_1$, $QH8 - C1 \times Q_1$)



python_codes/fieldstone_52/text.tex

```
2D
box
Q2Q1
Q2(8)Q1
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
paraview_output
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_52`

The domain is a unit square and we solve the manufactured solution problem of Section 18.36:

$$\begin{aligned} u(x,y) &= 200x^2(1-x)^2y(1-y)(1-2y) \\ v(x,y) &= -200x(1-x)(1-2x)y^2(1-y)^2 \\ p(x,y) &= 10\left[(x-1/2)^3y^2 + (1-x)^3(y-1/2)^3\right] - 5/4 \end{aligned}$$

The 5/4 term in the pressure equation insures that the pressure is exactly zero in the upper right corner (this boundary condition is implemented to get rid of the pressure nullspace). It is also the value of the average pressure over the domain.

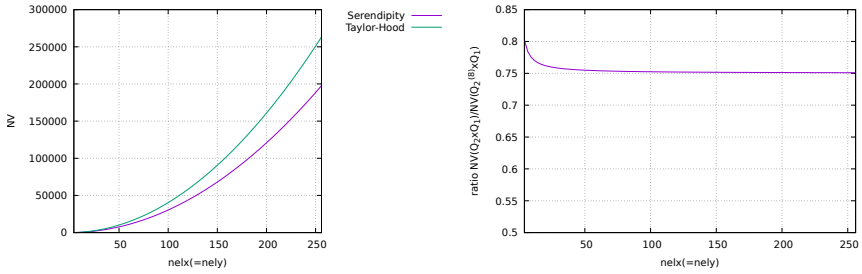
We will look at three second order elements:

- The Taylor-Hood $Q_2 \times Q_1$
- The serendipity element $Q_2^{(8)} \times Q_1$. The shape functions and their derivatives are in Section 8.3.5.
- A modified serendipity element by Zhang & Xiang (2020) [363], the $QH8 - C1 \times Q_1$.

The global numbering of nodes for both types of elements is shown in this simple 4×3 element mesh:

```
      Q_2 X Q1 (serendipity)                    Q_2 X Q_1 (regular)

15--47--16--48--17--49--18--50--19      54--55--56--57--58--59--60--61--62
 |       |       |       |       |       |   :   |   :   |   :   |   :   |
42      43      44      45      46      45..46..47..48..49..50..51..52..53
 |       |       |       |       |       |   :   |   :   |   :   |   :   |
10--38--11--39--12--40--13--41--14      36--37--38--39--40--41--42--43--44
 |       |       |       |       |       |   :   |   :   |   :   |   :   |
33      34      35      36      37      27..28..29..30..31..32..33..34..35
 |       |       |       |       |       |   :   |   :   |   :   |   :   |
05--29--06--30--07--31--08--32--09      18--19--20--21--22--23--24--25--26
 |       |       |       |       |       |   :   |   :   |   :   |   :   |
24      25      26      27      28      09..10..11..12..13..14..15..16..17
 |       |       |       |       |       |   :   |   :   |   :   |   :   |
00--20--01--21--02--22--03--23--04      00--01--02--03--04--05--06--07--08

iel= 0:                                  iel= 0
node  0 : 0 at pos. 0.0 0.0              node  0 : 0 at pos. 0.0 0.0
node  1 : 1 at pos. 1.0 0.0              node  1 : 2 at pos. 1.0 0.0
node  2 : 6 at pos. 1.0 1.0              node  2 : 20 at pos. 1.0 1.0
node  3 : 5 at pos. 0.0 1.0              node  3 : 18 at pos. 0.0 1.0
node  4 : 20 at pos. 0.5 0.0             node  4 : 1 at pos. 0.5 0.0
node  5 : 25 at pos. 1.0 0.5             node  5 : 11 at pos. 1.0 0.5
node  6 : 29 at pos. 0.5 1.0             node  6 : 19 at pos. 0.5 1.0
node  7 : 24 at pos. 0.0 0.5             node  7 : 9 at pos. 0.0 0.5
                                         node  8 : 10 at pos. 0.5 0.5
```

We see that the serendipity element-based mesh counts only 51 nodes, as opposed to 63 for its counterpart. Setting $nelx = nely$, we can look at the number of velocity nodes for each as a function of $nelx$, as shown hereunder:

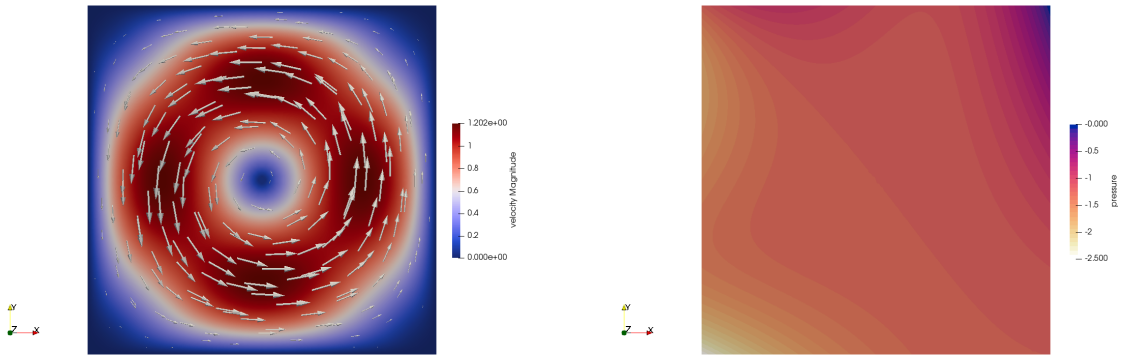


Left: NV for both elements as a function of nelx. Right: ratio of the two.

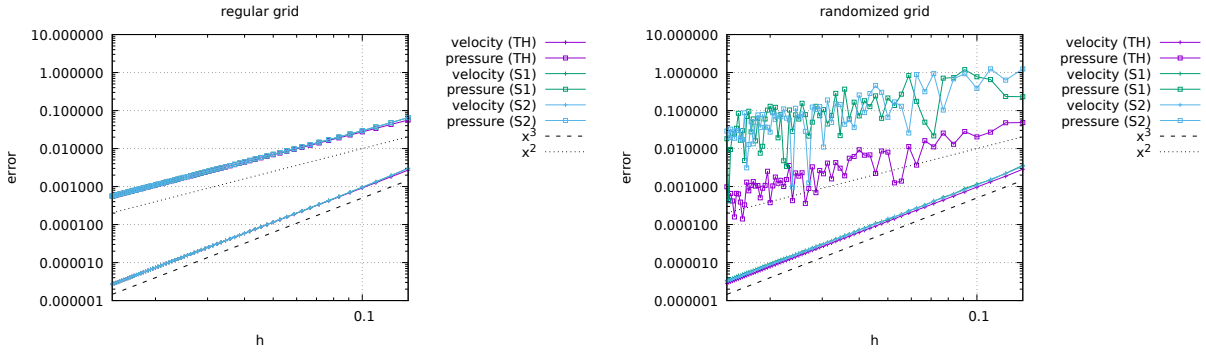
Looking at the ratio between both, we see that ultimately at high resolution, a mesh composed of serendipity elements will count 25% less nodes than a mesh with Taylor-Hood elements. Since there is no free lunch, what is the price paid in terms of accuracy when using the cheaper serendipity?

Although the vtk format does not support the Q_2 element in 2D or 3D, it surprisingly does support the serendipity element in 2D (type=23) and 3D (type=25).

Two types of grids are generated. The first one is a mesh composed of square (or rectangular) elements. The second one has random noise added to the first one: the position of each corner node (with the exception of those on the domain boundaries) is perturbed by a randomly computed (positive or negative) distance equal to $0.1h$ at the maximum. Note that the nodes on the edges between corner nodes are relocated to the middle of the edge so as to generate straight-side quadrilaterals.



We then compute the velocity and pressure error as a function of resolution h:



'TH' is Taylor-Hood, 'S1' is serendipity $Q_2^{(8)} \times Q_1$, 'S2' is $QH8C1 \times Q_1$ element.

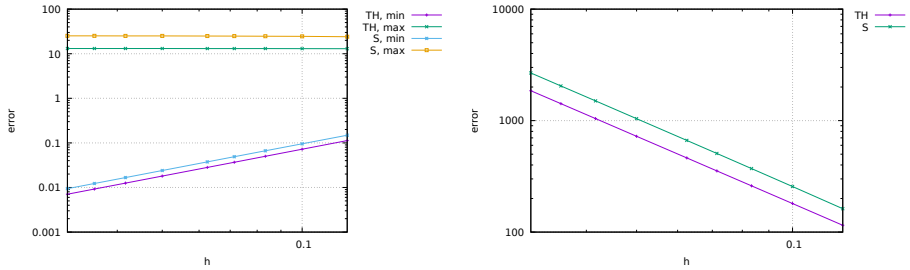
Note that the $QH8C1 \times Q_1$ element is identical to the regular $Q_2^{(8)} \times Q_1$ element in the case of regular meshes since in this case $m_x = m_y = 0$, so that its results are the same as those of the regular serendipity element.

For regular grids the serendipity element yields the same errors and error convergence rates as its Taylor-Hood counterpart. Since it is cheaper in terms of dofs, one could think that it should be preferred. However, most modern codes use an iterative solver approach to solve the discretised Stokes problem, and often the $\mathbb{K}$ matrix (which is SPD) is 'solved' with a conjugate gradient solver. The convergence of this type of solver depends on the condition number of the matrix itself, i.e. the ratio of the largest and smallest eigenvalues. Note that this is rather trivial with Python:

```
print('condition number:', nel, linalg.cond(K_mat))
```

However, since I was also curious about the values of the eigenvalues, I implemented it as follows:

```
eigvals, eigvecs = linalg.eig(K_mat)
print('eigenvalues:',nel,eigvals.min(),eigvals.max())
```



Left: min and max eigenvalues for both types of elements as a function of h; Right: condition number

As it turns out, the condition number is twice as high for the serendipity element, which means that the CG would have to iterate more to arrive at the solution, thereby offsetting the benefit of less dofs.

Looking back at the error rates for the randomised grids we see that the error velocity is virtually unchanged. Even though the pressure error still decreases quadratically with the mesh size, it does now do so in a much more erratic way. Also the Taylor-Hood element outperforms the serendipity elements by about an order of magnitude.

Question: isoparametric mapping vs Q1 mapping?

Stone 53

The sinking block benchmark ($Q_2 \times Q_1$, $Q_2^{(8)} \times Q_1$)

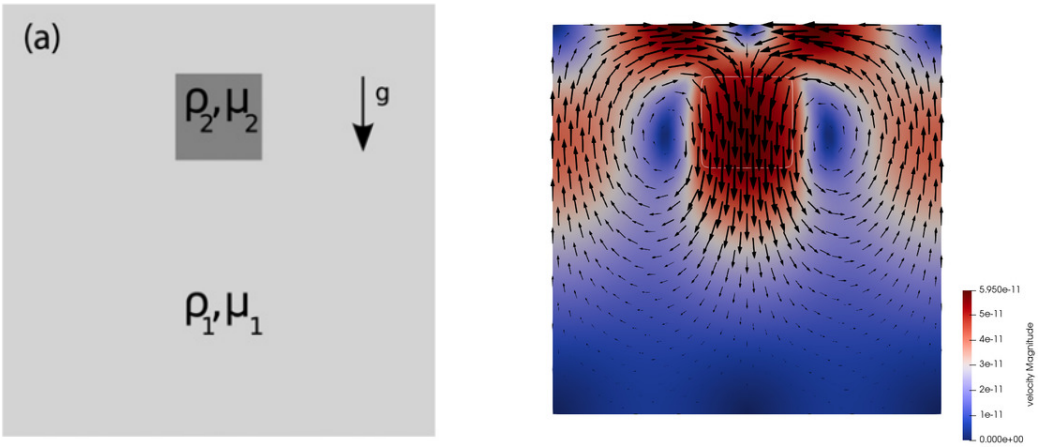


python_codes/fieldstone_53/text.tex

```
2D
box
Q2Q1
Q2(8)Q1
penalty
sparse_matrix
numerical_benchmark
error_convergence
stokes_equation
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_scaling
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_53`

This simple benchmark provides challenging numerical experiments dealing with large viscosity variations within the simulation domain. It appears in Gerya [128] (2010) and consists of a bulk of fluid 1 (ρ_1, η_1) in which a block of fluid 2 (ρ_2, η_2) falls under its own weight. The domain is a square of size $L_x = L_y = 512$ km and the block is initially centred at point ($x = 256$ km, $y = 384$ km) with size $L_b \times L_b = 128$ km $\times$ 128 km:

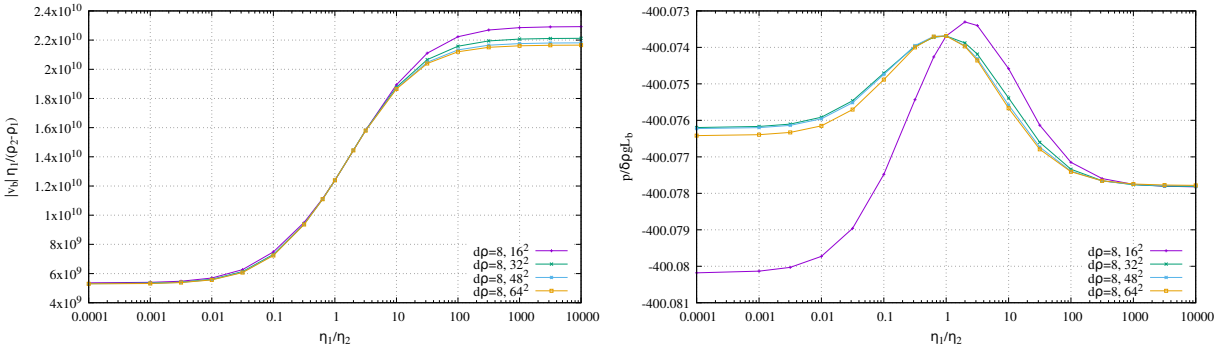


Left: setup. Right: velocity field for $\rho_2 = 3208$, $\eta_1 = 10^{21}$ and $\eta_2 = 10^{22}$.

The simulation is carried out on 32×32, 48×48 and 64×64 grids. Free slip boundary conditions are imposed on all sides of the domain. In all experiments the density of the surrounding fluid is $\rho_1 = 3200\ \mathrm{kg\,m^{-3}}$. The velocity v_b of the falling block is measured in its centre (note that due to symmetry the horizontal component should be zero).

As explained in Thieulot [326] (2011), following physical intuition, one expects the velocity v_b of the block to (a) decrease when the viscosity of the surrounding medium η_1 increases; (b) increase with the density contrast $\rho_2 - \rho_1$. The quantity $v_b \eta_1/(\rho_2 - \rho_1)$ is therefore monitored and shown hereunder as a function of the viscosity ratio. We also then plot $p^\star = p/(\delta\rho g L_b)$ measured in the block center as function of η_1/η_2.

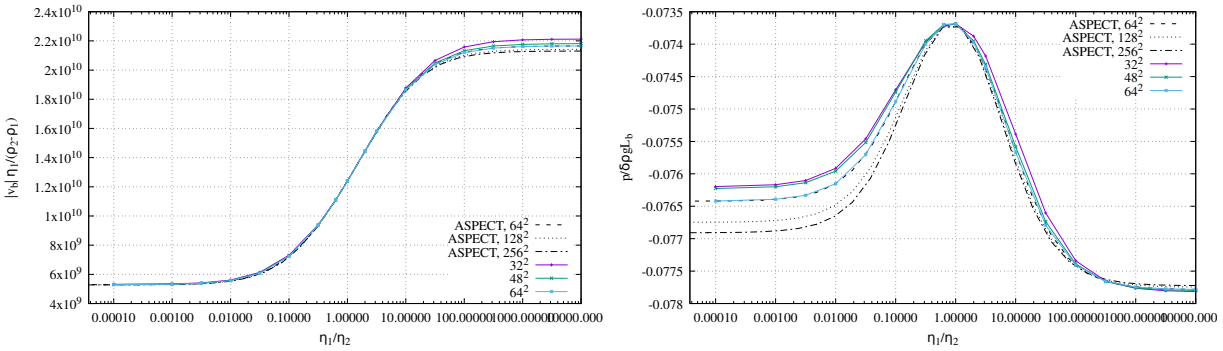
full density



Series of experiments have been conducted with $\rho_2 = 3208, 3216$, $\log_{10}(\eta_1) = 21$ and $\log_{10}(\eta_2) = 17, 18, \ldots 23, 23.5, 24, 25$, all with 4 mesh resolutions.

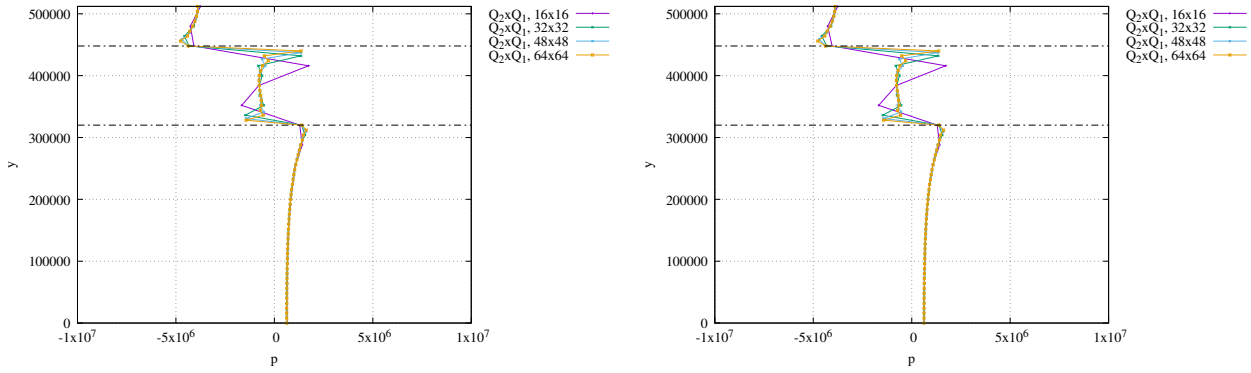
All experimental points line up on a single curve which further indicates that the code can deal with gravity driven simulations in the presence of large viscosity contrasts. These results have been succesfully compared with those obtained with ASPECT with the same setup.

reduced density Following the approach in Thieulot and Bangerth [328] (2022), we set $\rho_1 = 0$ and $\rho_2 = 8$ so that we are only solving for the dydnamic pressure, the lithostatic pressure having been removed.



Series of experiments have been conducted with $\rho_2 = 8$, $\log_{10}(\eta_1) = 21$ and $\log_{10}(\eta_2) = 17, 17.5, 18, ..22.5, 23, 23.5, 24$, all with multiple mesh resolutions.

We see that resolution 64×64 is close but not quite *the* solution of this experiment.
The pressure along the line $x = L_x/2$ was measured for 4 resolutions and both elements:



Left: $Q_2 \times Q_1$ element; Right: $Q_2^{(8)} \times Q_1$ element.

Stone 54

Arbitrary Lagrangian Eulerian formulations

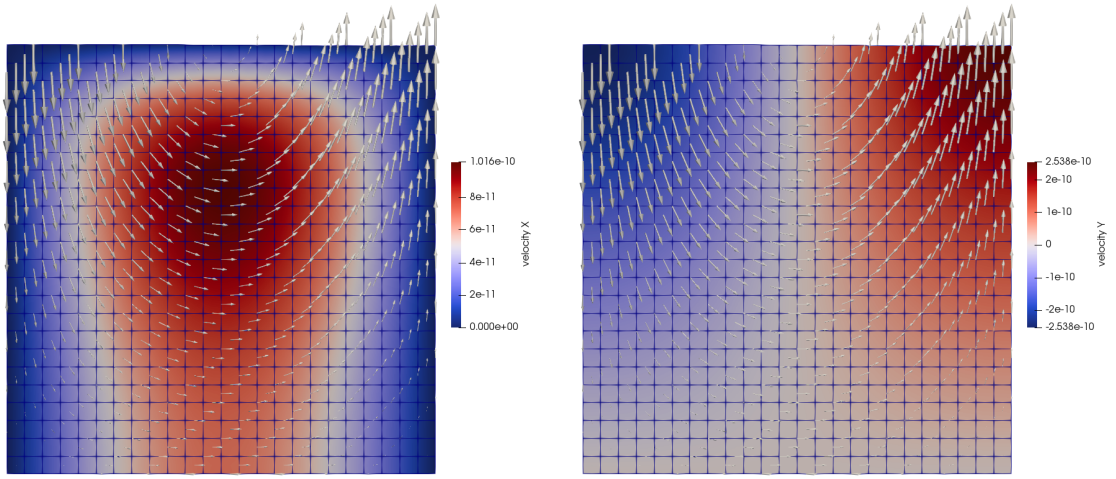
```
2D
box
Q2Q1
sparse_matrix
analytical_benchmark
numerical_benchmark
stokes_equation
time_stepping
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_scaling
free_surface
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_54`

This stone implements three different free surface/mesh deformation algorithms. The first one has all the nodes move with the computed velocity (Lagrangian method) and is coined 'method 1'. The second one ('method 2') only has the top row of nodes moving with the computed velocity, while all the nodes underneath are static (this is obviously not a viable method for large deformations). The third one ('method 3') is the method used in ASPECT and described in Rose, Buffet, and Heister [269] (2017). Its implementation is described in Section 15.1.

Experiment 1 - relaxation of topography

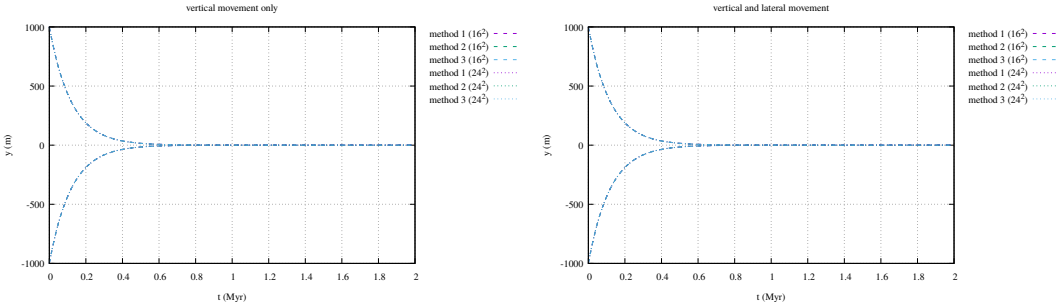
The domain is a 2D Cartesian box of size 512×512km, with free slip on left, bottom and right sides, free surface at the top. Mantle material characterised by $\rho_m = 3200$ and $\eta_m = 10^{22}$. Gravity is vertical and Earth like. The surface is perturbed at startup by $\delta y = A\cos(\pi x/L_x)$ with A=1km. 200 time steps with $\delta t = 10$kyr are carried out. The root mean square velocity, the total volume of the domain, the min/max elevation values of the surface are recorded over time.



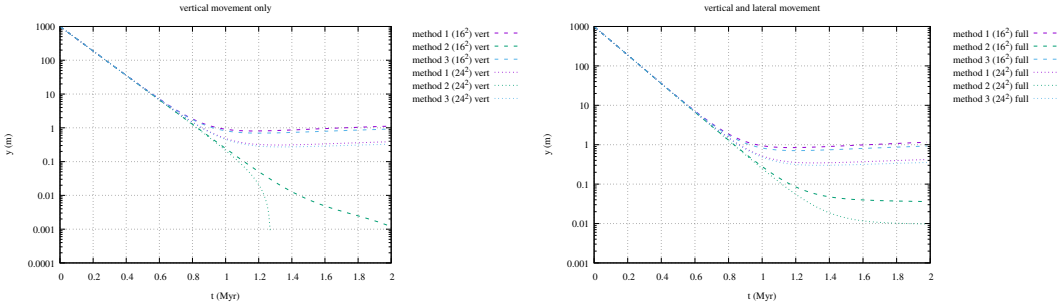
velocity field at $t = 0$ with resolution 24x24

The results hereunder are obtained for all three methods at two different resolutions (16x16 and 24x24 elements). The plots on the left column are obtained with the movement of the top nodes being constrained in the vertical direction, while the plots on the right column are obtained with nodes being allowed to move in both x and y directions (note that for method 3 the normals are not -yet- computed with the method of Eq. 49 in [269] but instead by a simple geometric rule).

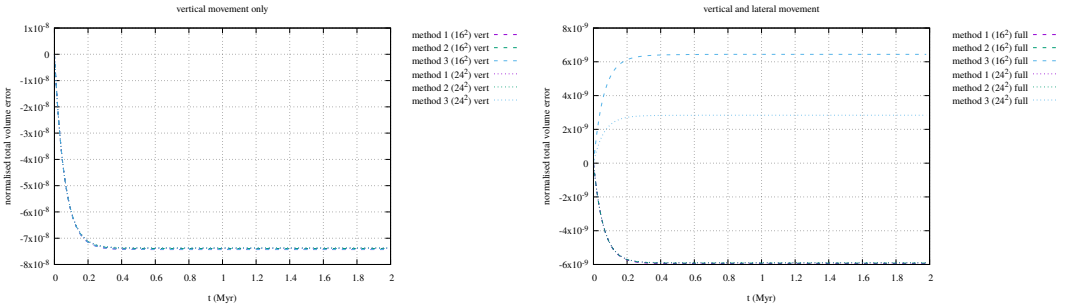
a)

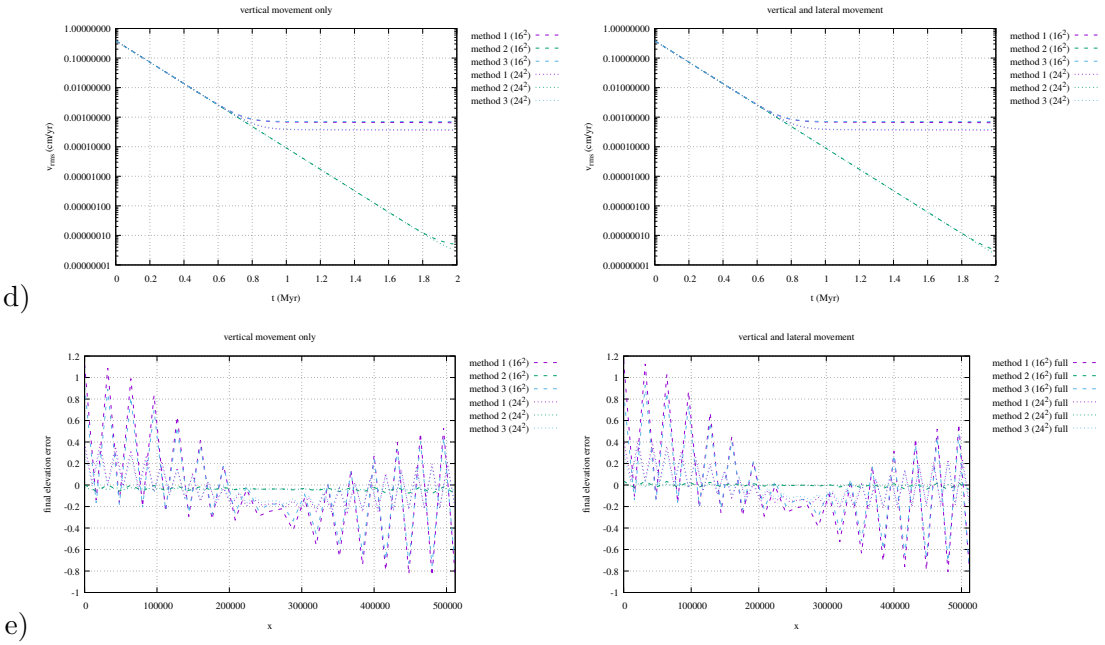


b)



c)

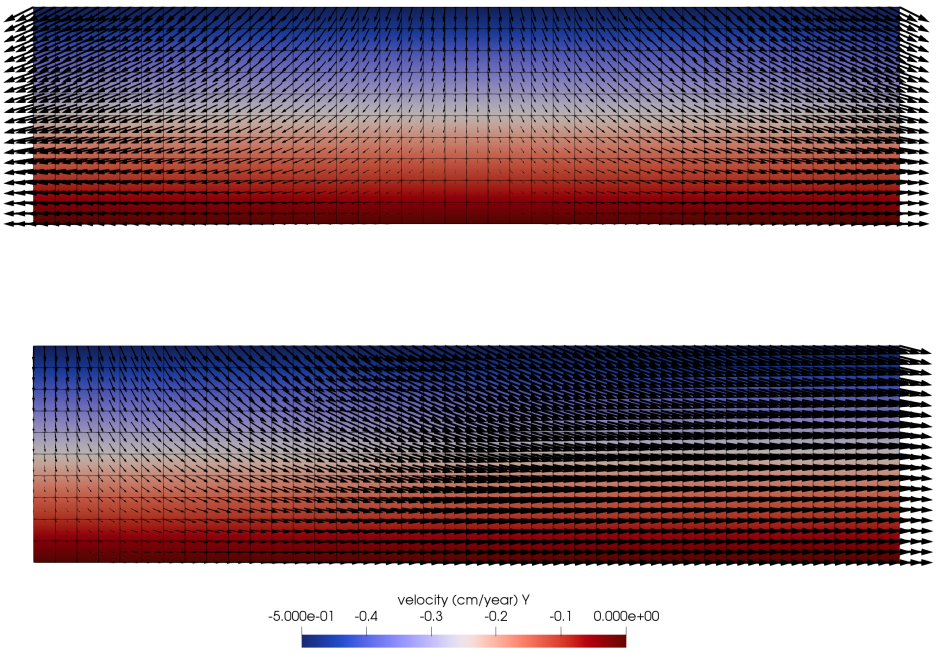




a) min/max of free surface topo as a function of time; b) free surface topo maximum (in log scale) as a function of time; c) measured volume of the domain with numerical quadrature normalised by the expected volume $L_x L_y$; d) root mean square velocity as a function of time; e) final elevation at the 200th time step.

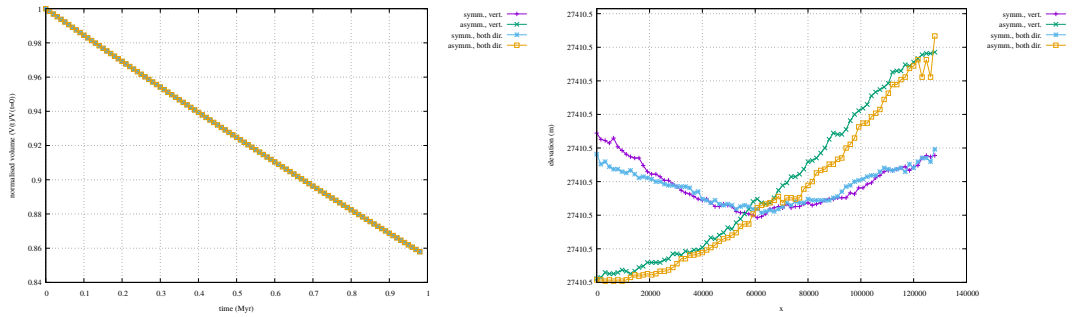
Experiment 2,3 - extension

We now consider a rectangular domain (crust sized) of 128×32km, discretised by means of 40×10 elements. The fluid is identical to the one above and so is gravity. Extensional boundary conditions are applied. In the first case, a 1cm/yr horizontal velocity is applied on both sides, while in the second case a 2cm/yr velocity is applied on the right while 0 is prescribed to the left. Free slip conditions are otherwise prescribed on the sides and bottom. 100 time steps are carried out with $\delta t = 10^4$yr. Only method 3 is used here.

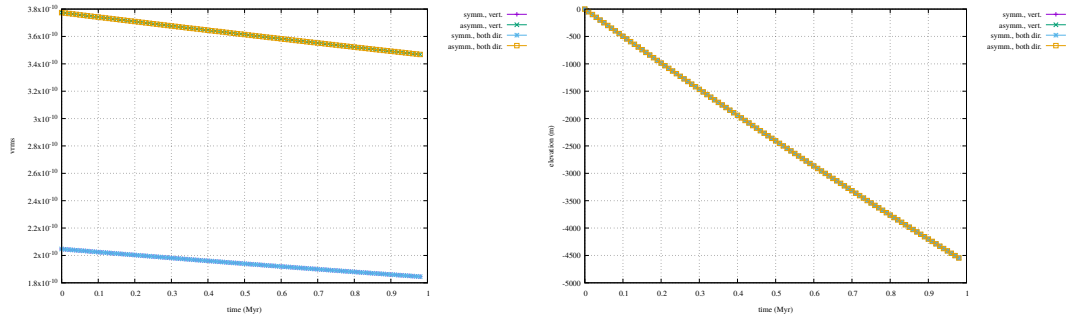


Velocity field for both experiments at $t = 0$.

Despite the asymmetry in the boundary conditions, we expect the same evolution of the domain geometry. This is indeed what we recover with surprising accuracy. 'vert' stands for only vertical movement allowed (i.e. $n_x = 0$, $n_y = 1$) while 'both dir' stands for the use of dynamically computed normal vectors (based on geometrical consideration).



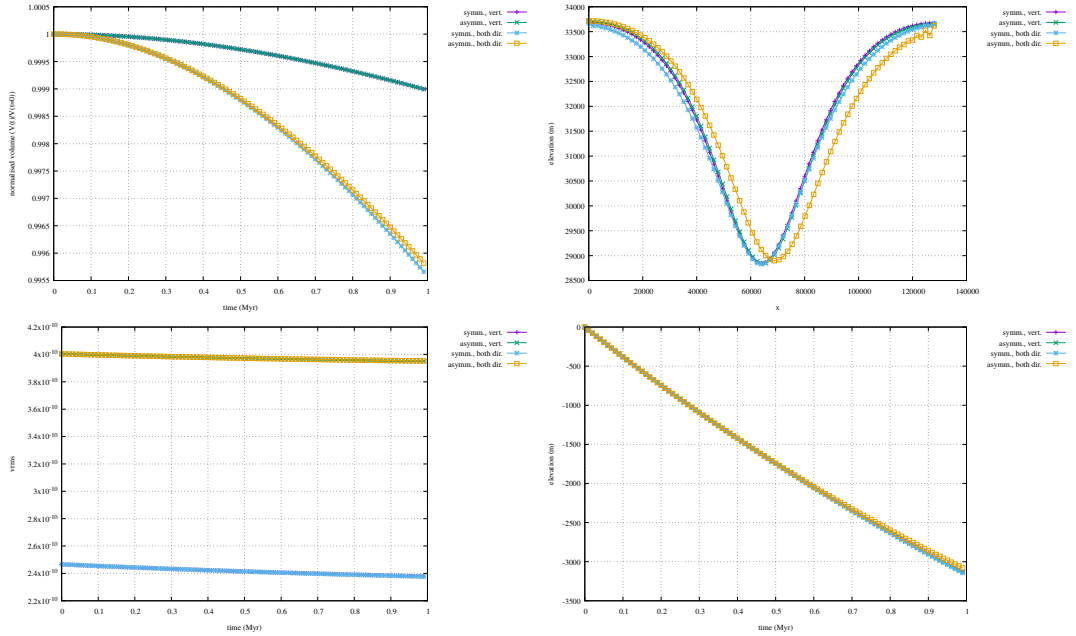
Left: time evolution of the normalised volume for both boundary condition types; Right: surface at the end of the simulation.



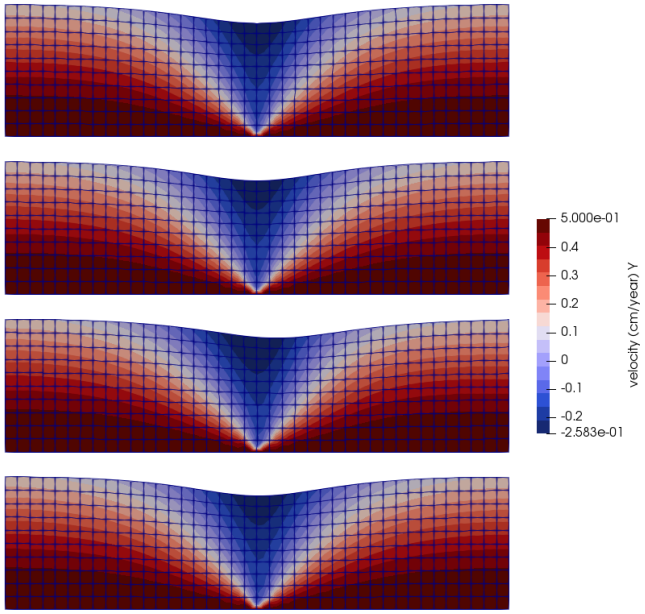
Left: Root mean square velocity as a function of time; Right: time evolution of the elevation (min and max virtually indistinguishable).

Experiment 4,5 - extension with bottom inflow

This is the same setup as above, but we now impose an influx boundary condition at the bottom: $v = 0.5cm$ (this balances the outflux exactly) and the horizontal component at the bottom is set to the left value (-1 or 0 cm/yr) for $x \leq L_x/2$ and to the right value +1 or +2 cm/yr for $x \geq L_x/2$.

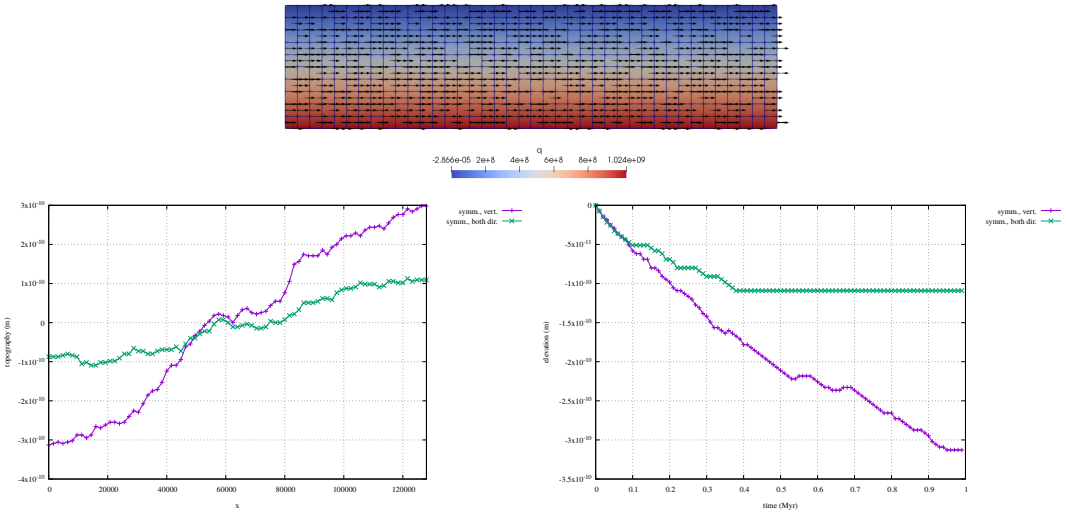


We can conclude that the 'vertical movement only' conserves volume/mass better and the free surface remains symmetrical unless the full normal is used which is simply explained by having $\vec{v} \cdot \vec{n}$ as a boundary condition: in the asymmetric extension case, the velocity is always to the right while the normal has an x component which is negative and positive, therefore introducing an asymmetry in the surface boundary conditions.



Top to bottom: symmetric extension, full normal vector; symmetric extension, vertical normal vector; asymmetric extension, full normal vector; asymmetric extension, vertical normal vector

Experiment 6 - pure advection test



Results after 100 time steps = 1Myr

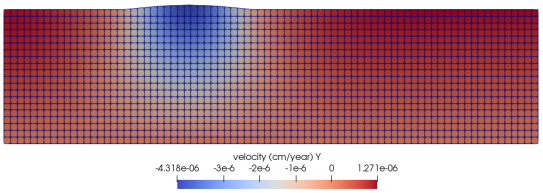
Experiment 7 - pure advection test of a cosine bump

The initial topography bump is given by

$$y(x) = A\left[1 + \cos\left(\frac{x - x_0}{w}\pi\right)\right]$$

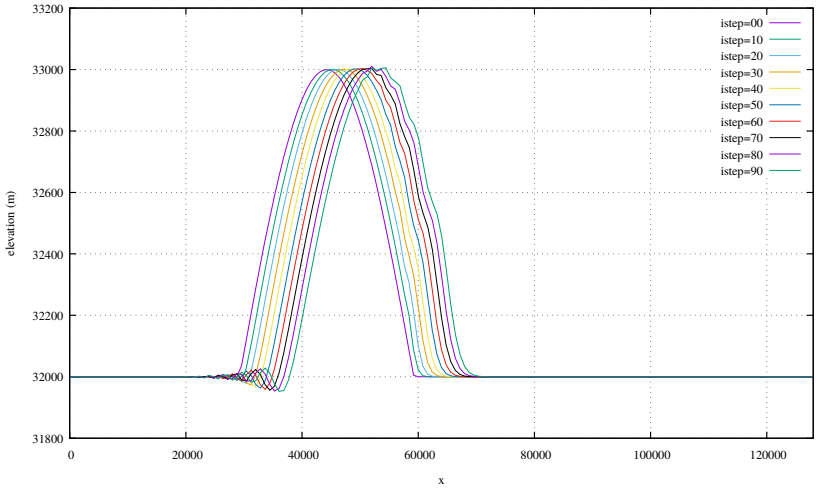
with $A = 1000$m, $x_0 = 0.345678L_x$ and w =15km. This is a somewhat ideal case since the transition from flat to bump is very smooth.

The viscosity is set to 10^{26}Pa·s so that the velocity of the viscous relaxation of the topography is negligible with regards to the advection velocity (+1cm/yr on left and right boundaries). I also choose a resolution of 80x20 elements.

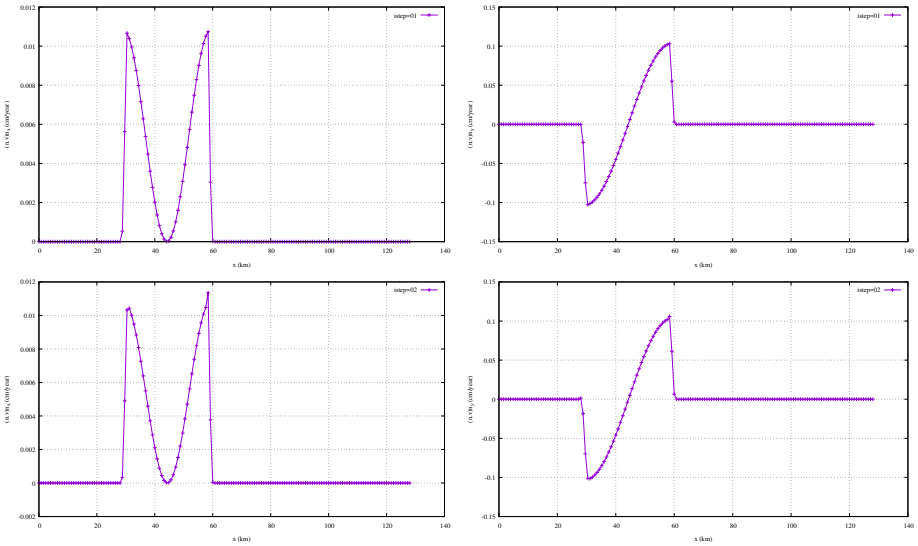


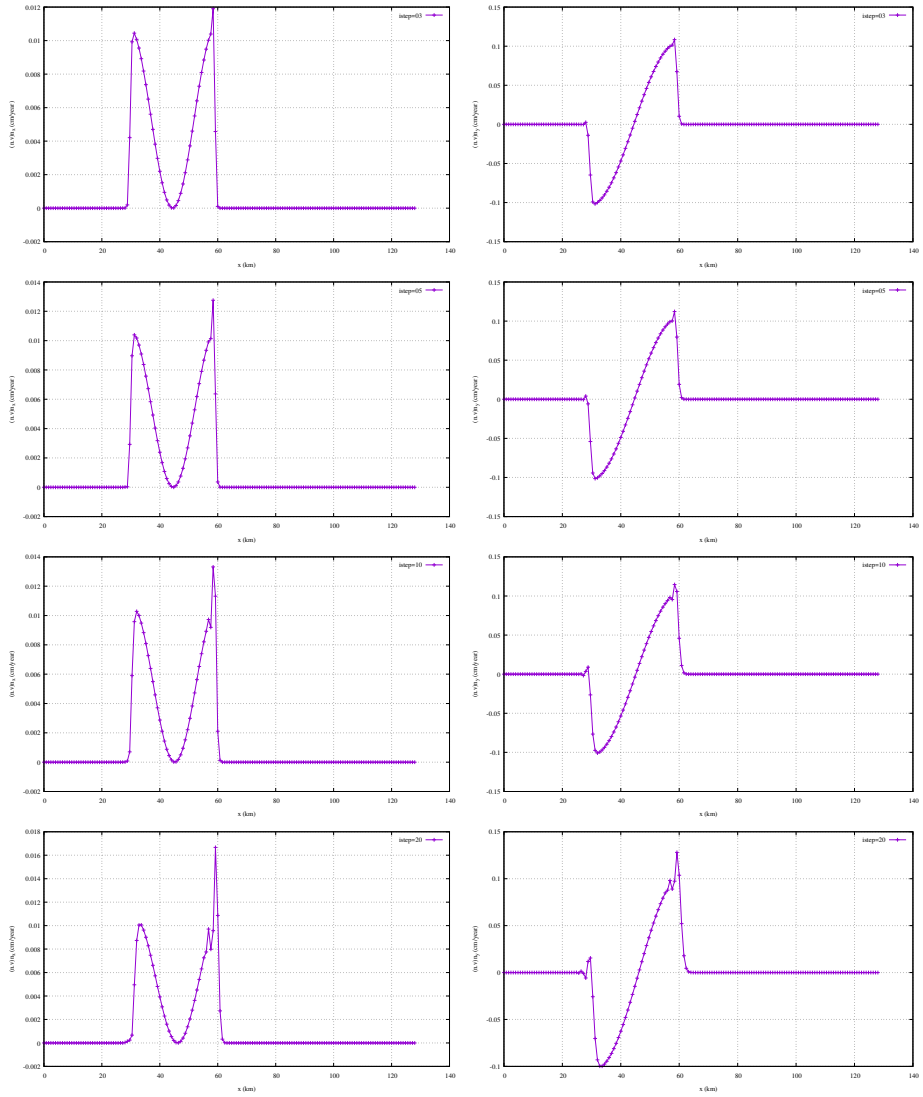
If only the vertical normal is used then *nothing* moves since the velocity is always perpendicular to the normal.

What follows is obtained when the full normal is used. After 20 timesteps the topography has been advected and we already observe some visible asymmetry (and oscillations) on the surface:



On the following plots the left column shows $(\vec{\mathbf{v}} \cdot \vec{n})n_x$ as a function of the x-coordinate and the right column shows $(\vec{\mathbf{v}} \cdot \vec{n})n_y$, both for timesteps 1,2,3,5,10,20. Both quantities form the boundary conditions for the mesh deformation. Since I have chosen x_0 such that it does not fall on a node, the initial topography is *not* symmetric and therefore the normal vectors at the nodes on the left and right of the peak are not identical. From the second timestep (the first one for which the free surface algo uses a non zero Stokes velocity as boundary condition) we see that both components $(\vec{\mathbf{v}} \cdot \vec{n})n_x$ and $(\vec{\mathbf{v}} \cdot \vec{n})n_y$ are asymmetric!

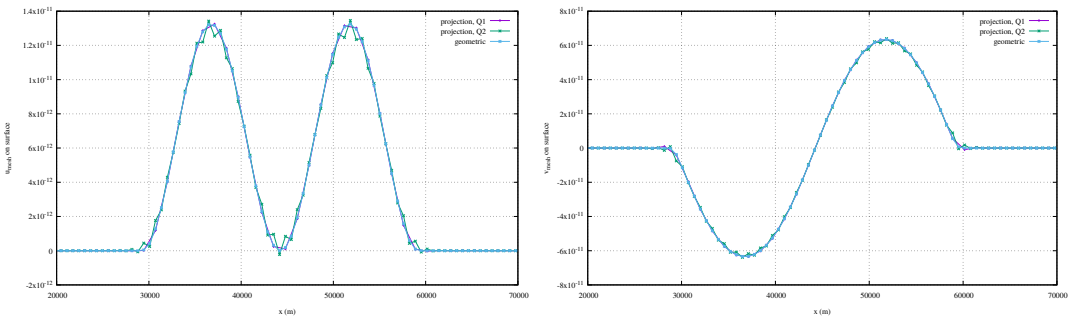


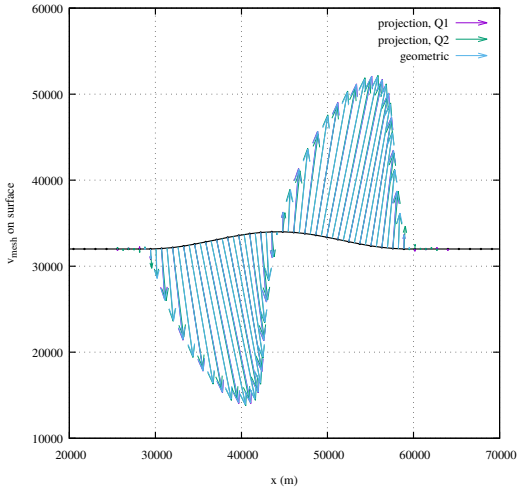


The inescapable conclusion is that the algorithm (as it is now implemented with the normal vector) is incapable of advecting a bump.

In what follows, the Stokes system is solved once, and the obtained velocity is used to compute the mesh velocity boundary condition for the Laplace system. The resolution is set to 100x25. I have implemented the L_2 projection approach of [269] for both Q1 and Q2 elements. For such a smooth topography the Q1 and geometrical approach (i.e. using geometrically computed normal vectors) are very similar although Q1 produces tiny undershoots. Q2 however generates oscillations which makes the computed velocity not suited as a boundary condition to move surface nodes.

From left to right: horizontal component, vertical component, and vector form of the computed mesh velocity bc.

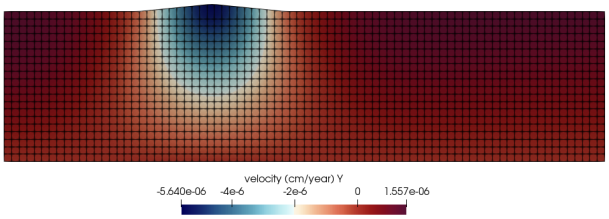


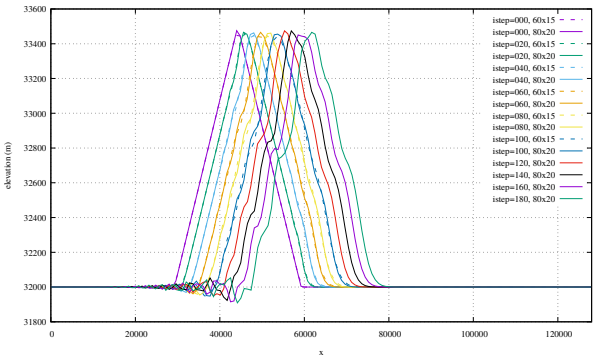


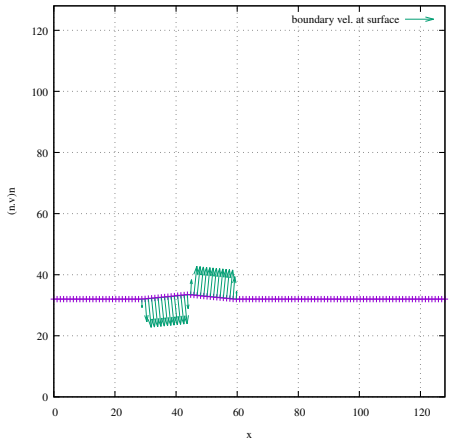
I haven't used any of the projections to carry out timestepping.

Experiment 8 - pure advection test of a pyramidal bump

This is a nearly identical experiment as the previous one, but now the bump is composed of two straight lines, of slopes ± 0.1. Pyramid centered at $x = 0.345678 L_x$, of half width 15km. $\rho = 3000$, $\eta = 10^{26}$. Resolution 60x15. +1cm/yr prescribed on left and right. dt=10kyr.



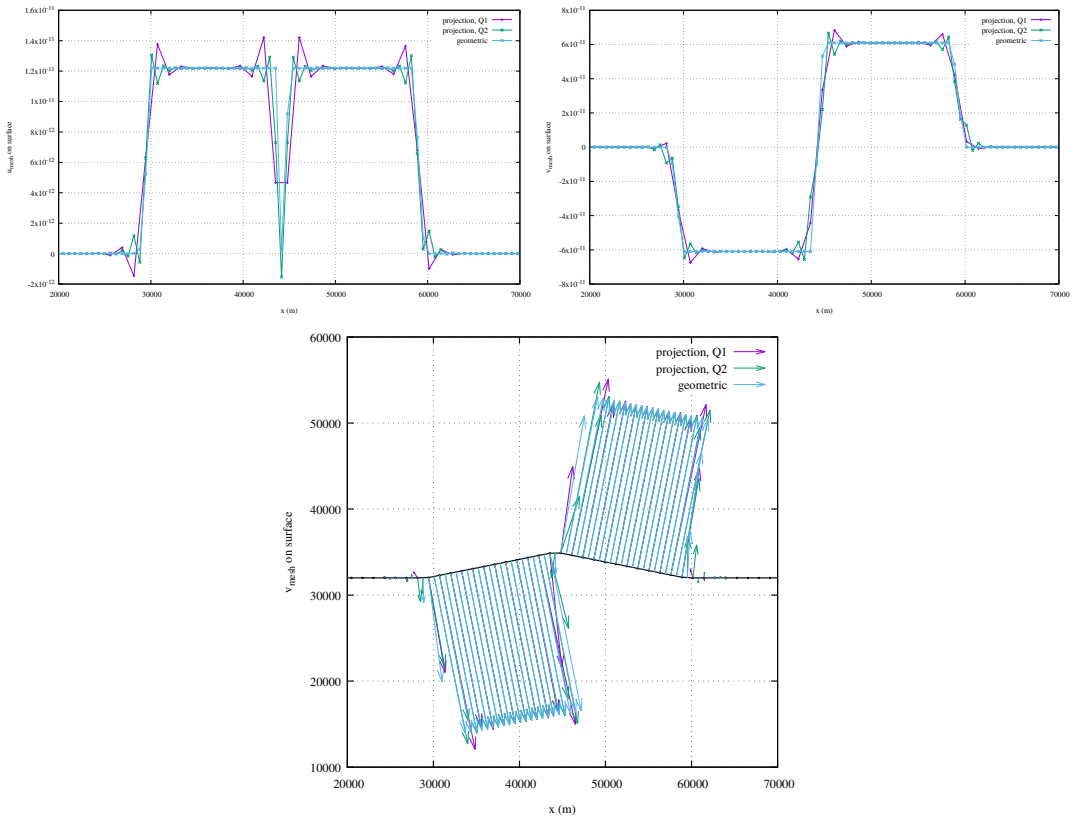




Results are not crazy bad, but we should be able to do better ... the pyramid becomes more and more deformed after only 20km of advection (but surprisingly retains its height).

In what follows, the Stokes system is solved once, and the obtained velocity is used to compute the mesh velocity boundary condition for the Laplace system. The resolution is set to 100x25. I have implemented the L_2 projection approach of [269] for both Q1 and Q2 elements. For such a broken line topography the geometrical approach (i.e. using geometrically computed normal vectors) is better. Q1 and Q2 both produce under/overshoots/oscillations with no clear winner.

From left to right: horizontal component, vertical component, and vector form of the computed mesh velocity bc.



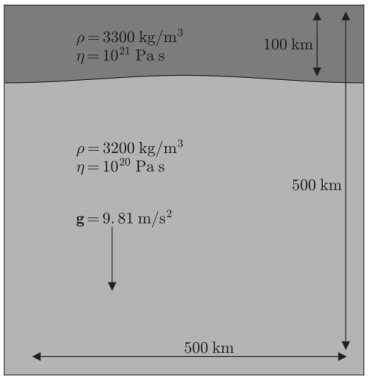
I haven't used any of the projections to carry out timestepping.

Experiment 9 - Rayleigh-Taylor instability

The setup originates in Kaus et al (2010) [180]. It is a Rayleigh-Taylor instability of a dense, more viscous layer ($\rho = 3300 kg/m^3$, $\eta = 10^{21} Pa \cdot s$), sinking through a less dense fluid ($\rho = 3200 kg/m^3$, $\eta = 10^{20} Pa \cdot s$). Side boundaries are free slip, the lower boundary is no-slip and the upper boundary is a free surface. The box is 500 x 500 km in size, and gravitational acceleration was 9.81m/s^2 . The initial perturbation was sinusoidal with initial amplitude A of 5 km (i.e. 1% of the box size) and centered at $y_0 = 400$km:

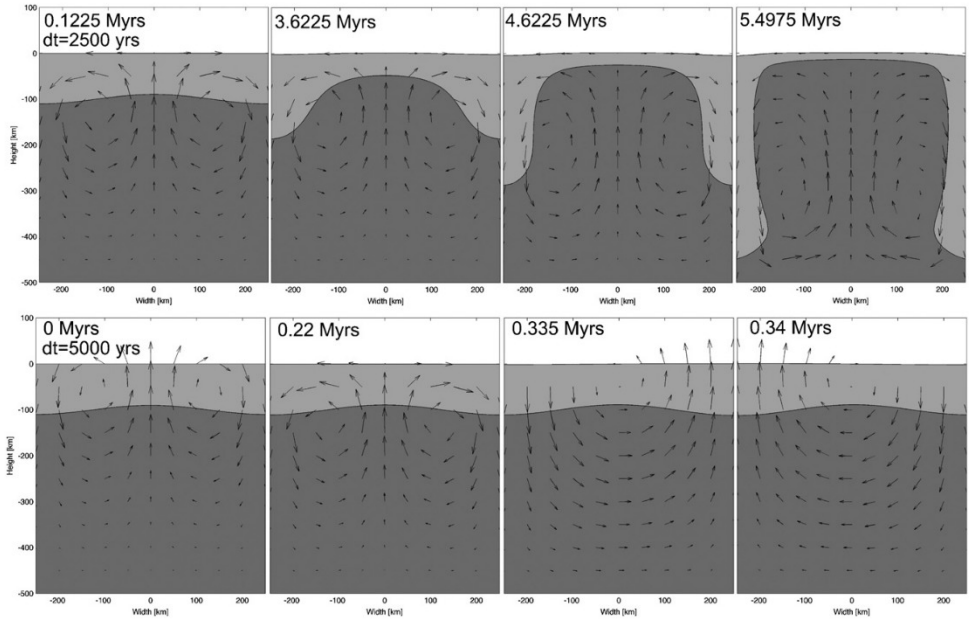
$$y(x) = y_0 - A \cos(2\pi x / L_x)$$

The position $y(t)$ of the free surface point situated at $x = L_x$ is monitored and plotted ...



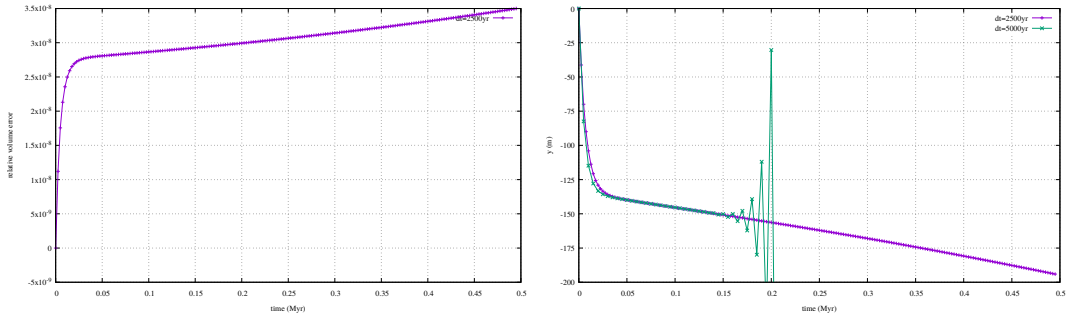
Taken from Rose et al (2017) [269].

Because I do not have compositional fields or markers implemented in this code I have to align the mesh with the initial sinusoidal perturbation and run the model in Lagrangian mode. Also, as shown in [180], this experiment is prone to drunken sailor instabilities and I have therefore fixed dt=2500yr with a 25x25 resolution. As a consequence, the model will run until elements become too distorted.

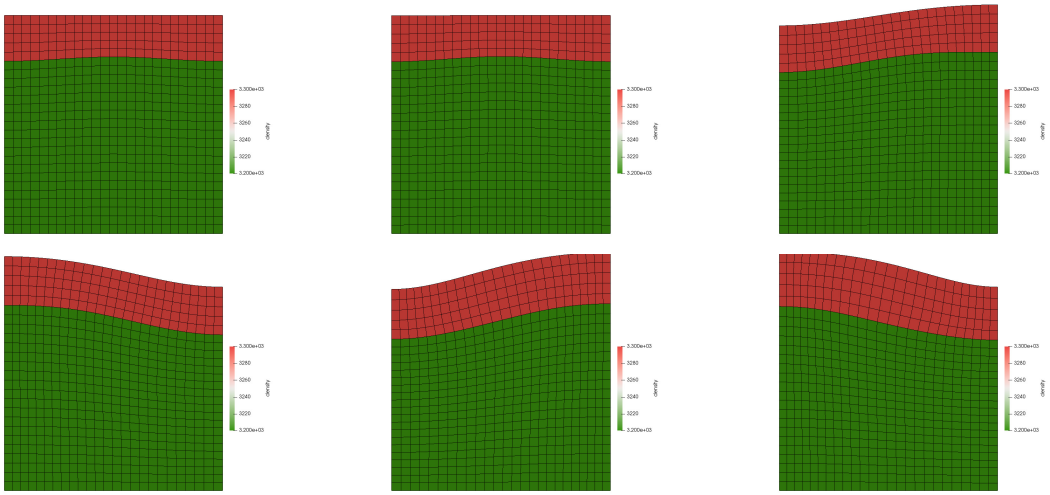


Taken from Kaus et al (2010) [180].

Kaus et al show that the system is stable for $\delta t = 2500$yr and quickly unstable for $\delta t = 5000$yr. This is also what we observe:



Results obtained with method=1. 25x25 resolution.



mesh at time steps 0,40,50,51,52,53 for $\delta t = 5000$yr.

Stone 55

Subduction as a thin-sheet problem



2D
box
P2pPm1
sparse_matrix
numerical_benchmark
stokes_equation
time_stepping
use_mesher
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_scaling

`https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_55`

This stone was developed in collaboration with Erik van der Wiel and Neil Ribe.

Parameters for the setup are defined in *parameters.py*. This file is used in *generate_nodes.py* which produces the *subd.node* file which contains the coordinates of all key points on the boundary of the domain and along the material interfaces. This file is then further processed by the triangle program[1] as follows:

```
./triangle -q -a200000000 -o2 subd.node
```

The '-q' option adds vertices to the mesh to ensure that all angles are between 20 and 140 degrees. The '-a' makes sure that no triangle has an area larger than the supplied number. The '-o2' generates a mesh composed of second order triangles (six nodes per element, rather than three) and the three extra nodes of an element fall at the midpoints of the three edges. This generates two files: 'subd.1.ele' which contains the connectivity of all generated triangles and 'subd.1.node' which contains the coordinates of all nodal points. These two files are then read in *fieldstone.py* and stored in the xV, yV and iconV arrays.

Gravity is vertical and Earth-like. Free-slip boundary conditions are imposed on the top while the other boundaries are free (in/outflow determined freely based on the internal dynamics). Crouzeix-Raviart elements are used, see Section 10.5.9. The density of the mantle is set to zero while the subducting plate has a density $\delta\rho$.

[1] `https://www.cs.cmu.edu/~quake/triangle.html`

In order to remove the horizontal null space (when the side boundaries are fully open) the average horizontal velocity of the domain is set to zero, which we will revisit a bit later. Neil wrote: "The BEM solves the nullspace problem naturally because the Green functions all decay to zero at infinity. That means that the velocity solutions are all relative to a state of rest at infinity."

As an exercise, we can ask ourselves where the center of mass of the slab is located. If the mass distribution is continuous with the density $\rho(\vec{r})$ within a solid Q, then the integral of the weighted position coordinates of the points in this volume relative to the center of mass $\vec{R}$ over the volume V is zero, that is

$$\iiint_Q \rho(\vec{r})(\vec{r} - \vec{R})dV = 0$$

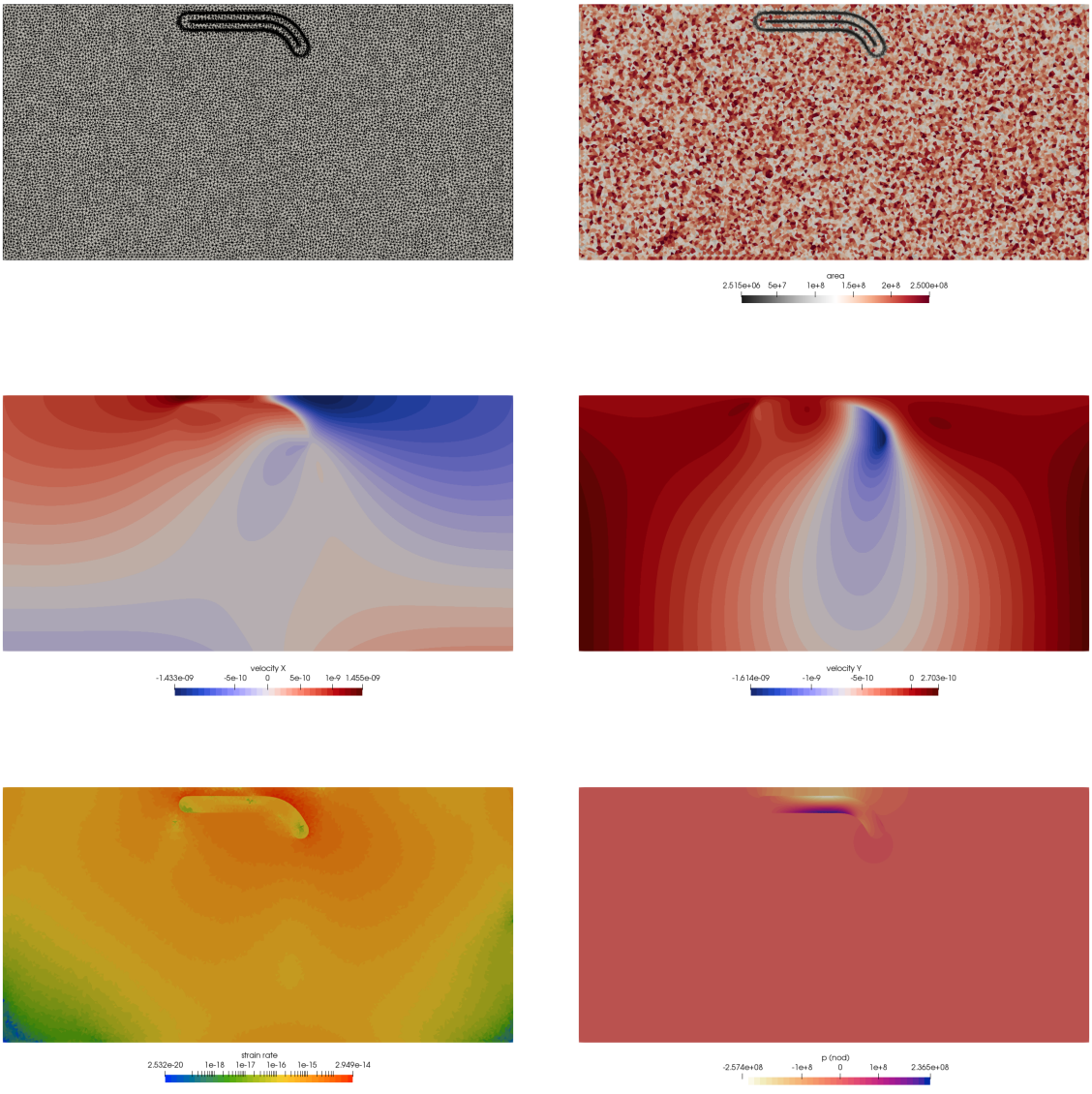
Since the density in the slab is constant this simplifies to

$$\iiint_Q (\vec{r} - \vec{R})dV = 0$$

or,

$$\vec{R} = \frac{\iiint_Q \vec{r}dV}{\iiint_Q dV}$$

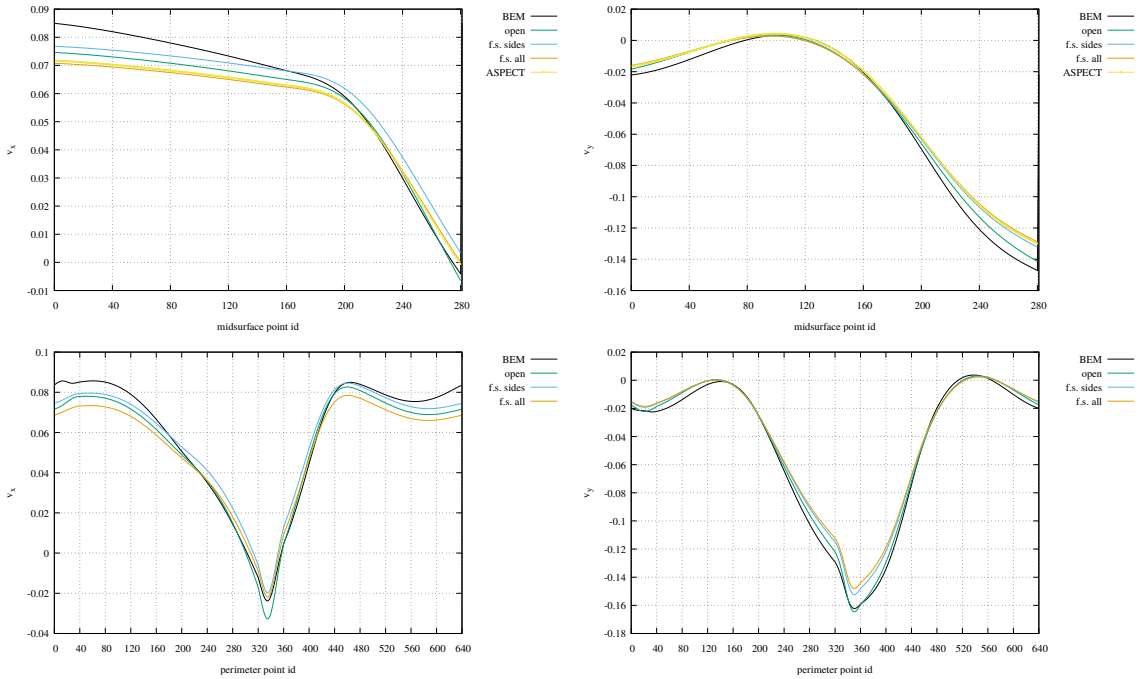
We find $\vec{R} = (940.7808, 873.6642)km$ for a 2000x1000km domain.





Mesh composed of 31,765 triangles. Other parameters: $\theta_0 = 60°$, $\eta_1 = 10^{21}$, $\gamma = 100$, L_x=3000km, L_y=1500km, $\delta\rho = 100$, L =400km, h =100km, d =50km. Note that the bottom boundary is open.

We have also run this model with ASPECT for the case where free slip boundary conditions are prescribed on all sides. Velocities on the midsurface are reported hereunder alongside those obtained with fieldstone and the BEM method.

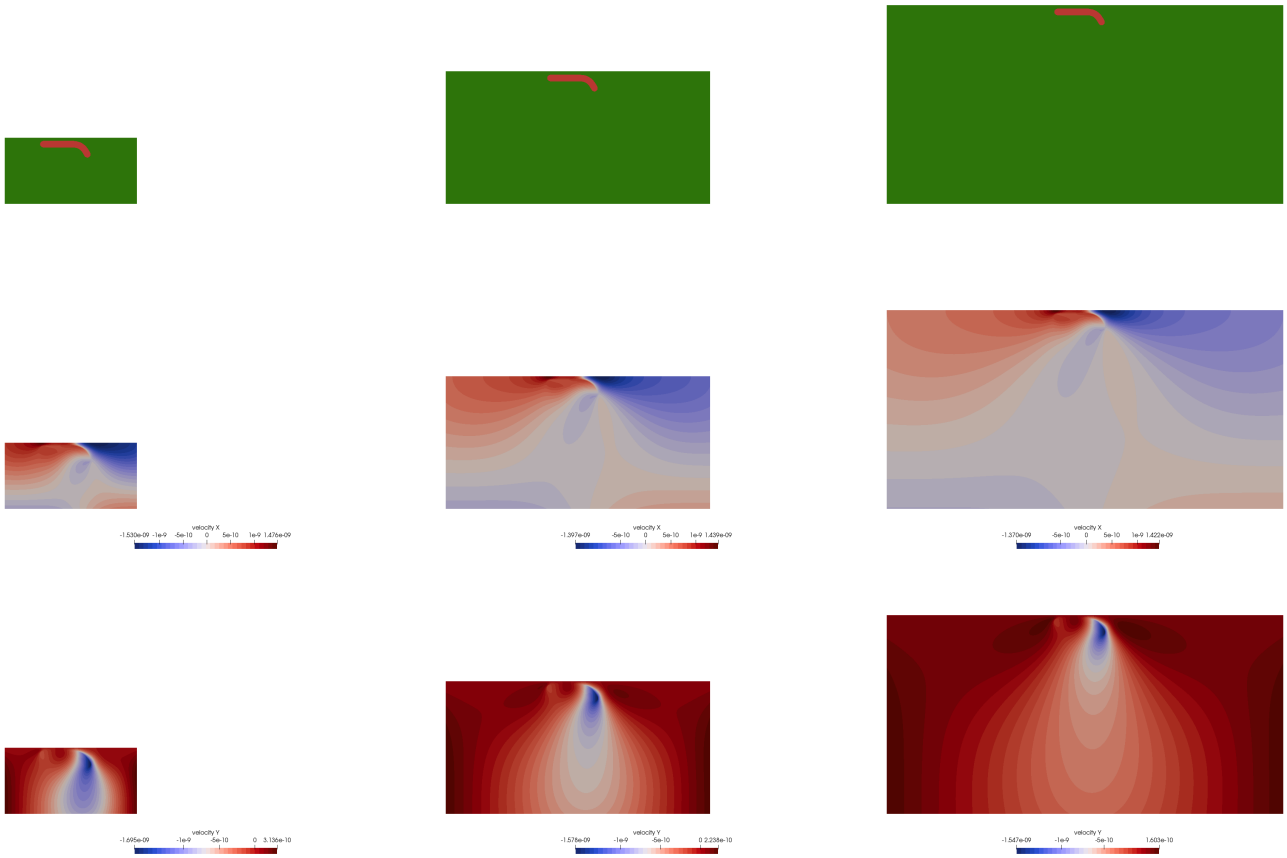


Top row: midsurface velocity measurements. Bottom row: slab/plate perimeter velocity measurements. 'open' means no b.c. on sides and bottom; 'f.s. sides' means free slip b.c. on left and right sides, open at the bottom; 'f.s. all' means free slip b.c. on sides and bottom.

Exploring the influence of the domain size

The BEM method is such that the domain is actually a semi-infinite plane. All I can do in a FEM code is prescribe no kinematic boundary conditions on the side and bottom boundaries and use a 'large enough' domain. But what is large enough? Let us investigate.

I have explored domains from size 2000x1000 up to 6000x3000 km. The slab remains at the same location below the surface and in the middle of the domain in the x-direction as shown here:

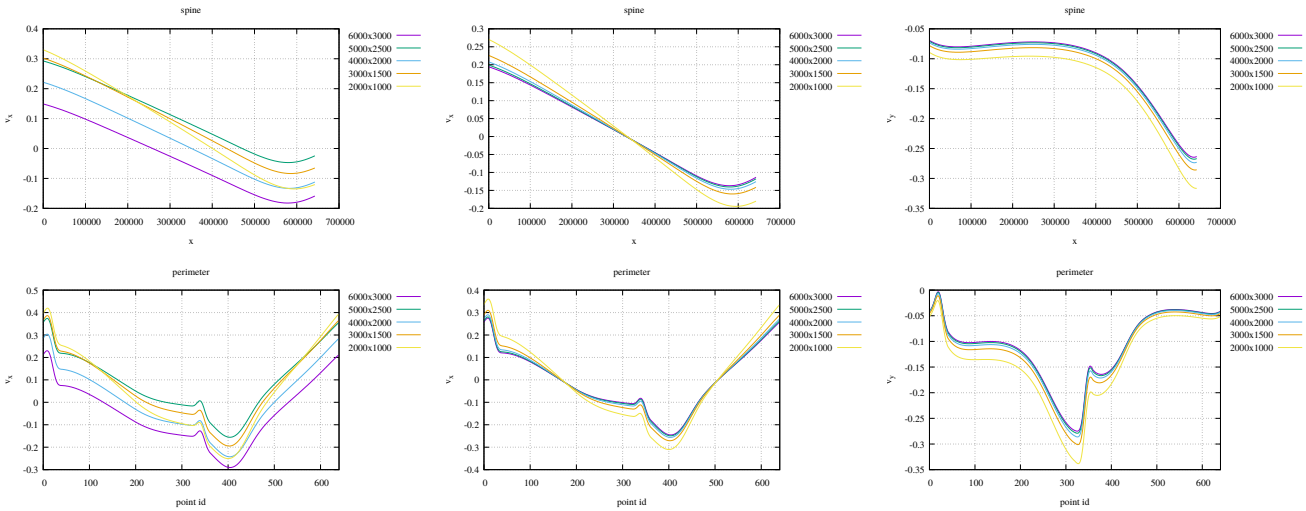


From left to right: 2000x1000, 4000x2000, 6000x3000km domains. Density field.

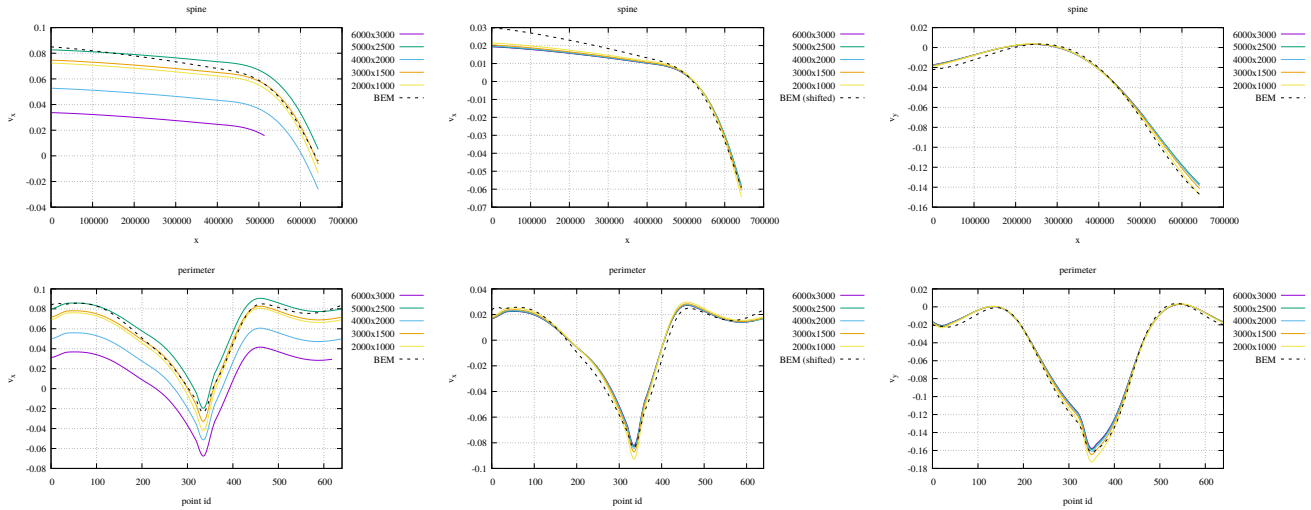
I also revisit the null space removal algorithm. Because the domain size changes using the total average horizontal velocity makes little sense. Instead I compute the average horizontal velocity of the slab only.

We define also the reference velocity as:

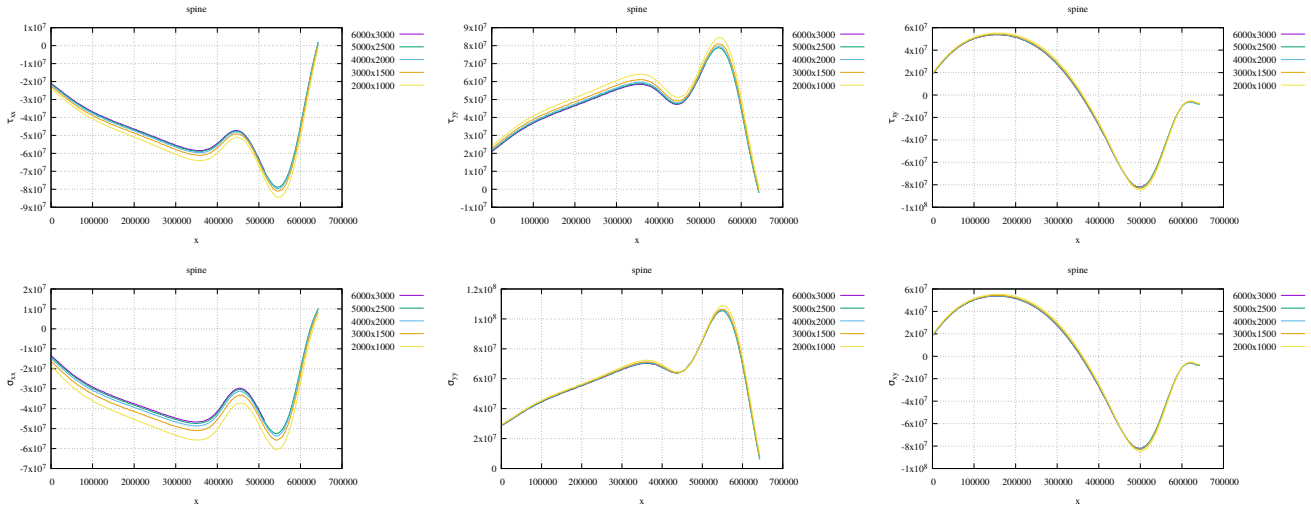
$$V_{ref} = \frac{h^2 g \delta\rho}{\eta} = \frac{100e3^2 \cdot 9.81 \cdot 100}{1e21}$$



Viscosity contrast $\gamma = 1$. From left to right: u, zero slab velocity-normalised u and v as measured on the spine (top row) and on the perimeter (bottom row). Velocities are normalised by V_{ref}



Viscosity contrast $\gamma = 100$. From left to right: u, zero slab velocity-normalised u and v as measured on the spine (top row) and on the perimeter (bottom row). Velocities are normalised by V_{ref}



Deviatoric and full stress components on the spine of the slab.

We see that results seem to converge to a domain size-independent profile, for both u and v fields as measured on the spine and on the perimeter of the slab.

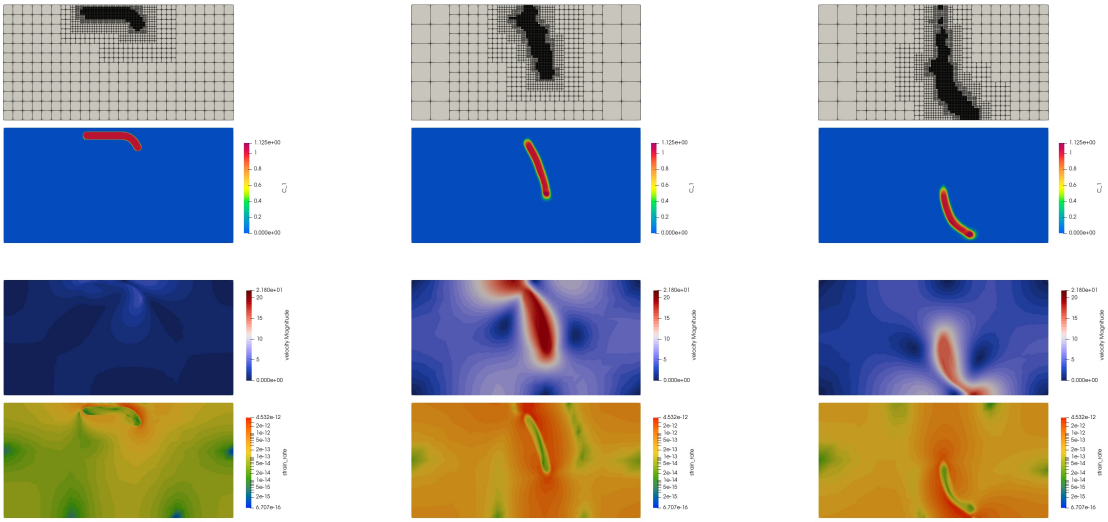
I have also quickly explored the influence of the mesh resolution for a given domain size and it appears that the resolution used for these calculations is sufficient to capture a resolution independent velocity field.

The question remains as to why BEM results still differ substantially from my high resolution large domain results.

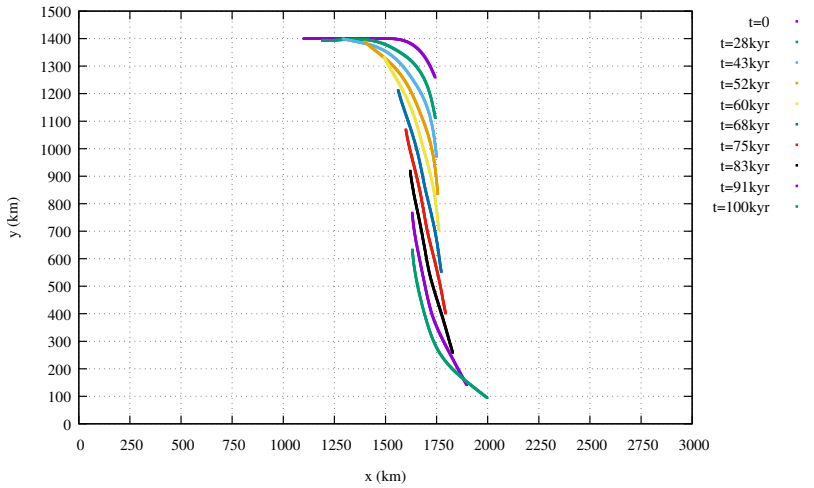
Neil wrote: "From the point of view of thin-sheet theory, the dominant stress component should be σ_{ss}, where s is the coordinate parallel to the spine. σ_{ss} should vary linearly across the plate is deformation is dominated by bending, and should be constant across the plate if it is dominated by stretching. In the flat part of the plate, $sigma_{ss} = \sigma_{xx}$. So your image of τ_{xx}, with a linear dependence across the flat part, corresponds to bending deformation. The bending here is so strong because the lubrication layer is thick: that allows the slab to bend down the whole interior of the plate while not undergoing strong bending itself. If possible, it would probably be a good idea to decrease the lubrication layer thickness from $0.5h$ to $0.3h$ or (better yet) $0.2h$. "

Time evolution with ASPECT

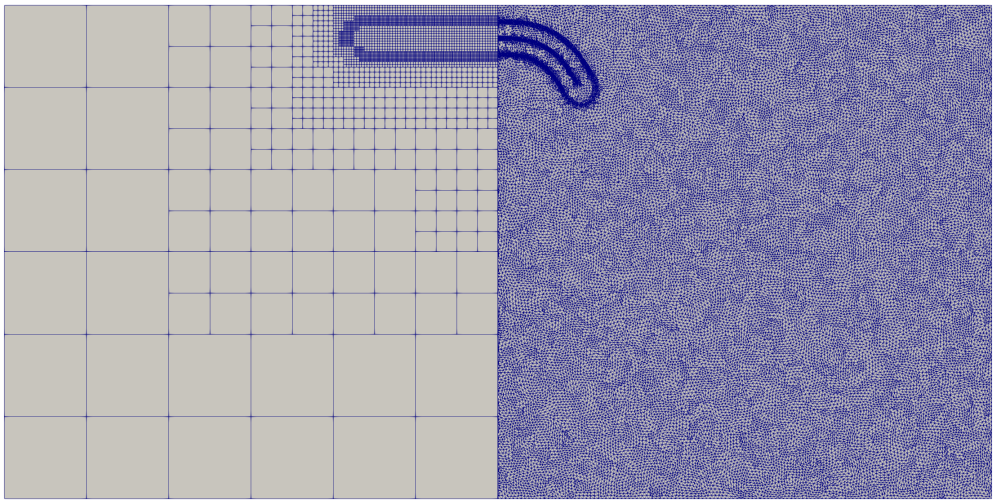
Finally, we have run this experiment over time with ASPECT (unfortunately viscosities were 100 times too small so that the times should be multiplied by 100):



Left: t=0, middle t=64kyr, right: t=100kyr.



Time evolution of the midline of the slab.



Literature: [113]

Stone 56

The Diamond-square algorithm



python_codes/fieldstone_56/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_56`

This stone was developed in collaboration with A. Hendrickx.

Purpose: provide an implementation of the diamond-square algorithm.

According to Wikipedia[1]: "The diamond-square algorithm is a method for generating heightmaps for computer graphics. It is a slightly better algorithm than the three-dimensional implementation of the midpoint displacement algorithm, which produces two-dimensional landscapes. It is also known as the random midpoint displacement fractal, the cloud fractal or the plasma fractal, because of the plasma effect produced when applied.

The diamond-square algorithm starts with a two-dimensional grid, then randomly generates terrain height from four seed values arranged in a grid of points so that the entire plane is covered in squares."

The idea was first introduced by Fournier, Fussell, and Carpenter [114] at SIGGRAPH[2] in 1982.

The function[3] `generate_fractal_map` is taken from the MTE code[4]. That function was 100% written by Agnes.

This specific implementation of the diamond-square algorithm also adds some diffusion to the resulting surface by means of the `gaussian_filter` function from the `scipy.ndimage` module. It is controlled by the `sigma` parameter[5].

This algorithm only works on a square surface of size $L \times L$ discretised by means of a $N_x \times N_y$ grid where $N_x = N_y = 2^{size} + 1$. For example if $size = 8$ then the grid counts 257×257 nodes. The `roughness` parameter determines the amplitude of the generated topography.

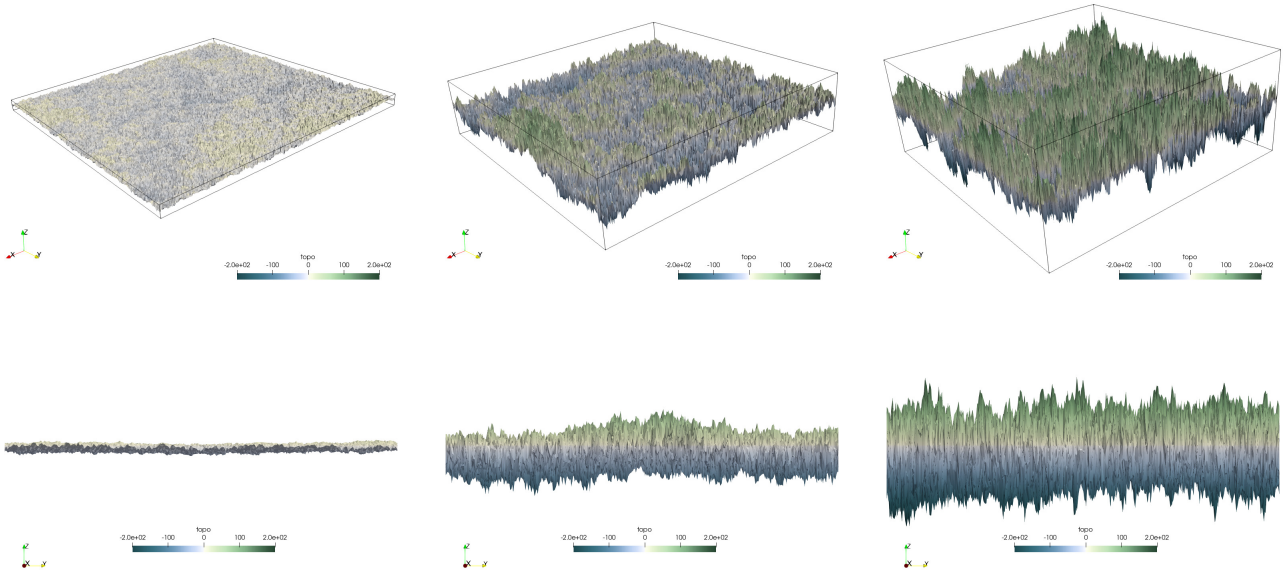
[1] `https://en.wikipedia.org/wiki/Diamond-square_algorithm`

[2] SIGGRAPH (Special Interest Group on Computer Graphics and Interactive Techniques) is an annual conference centered around computer graphics organized by ACM, starting in 1974.

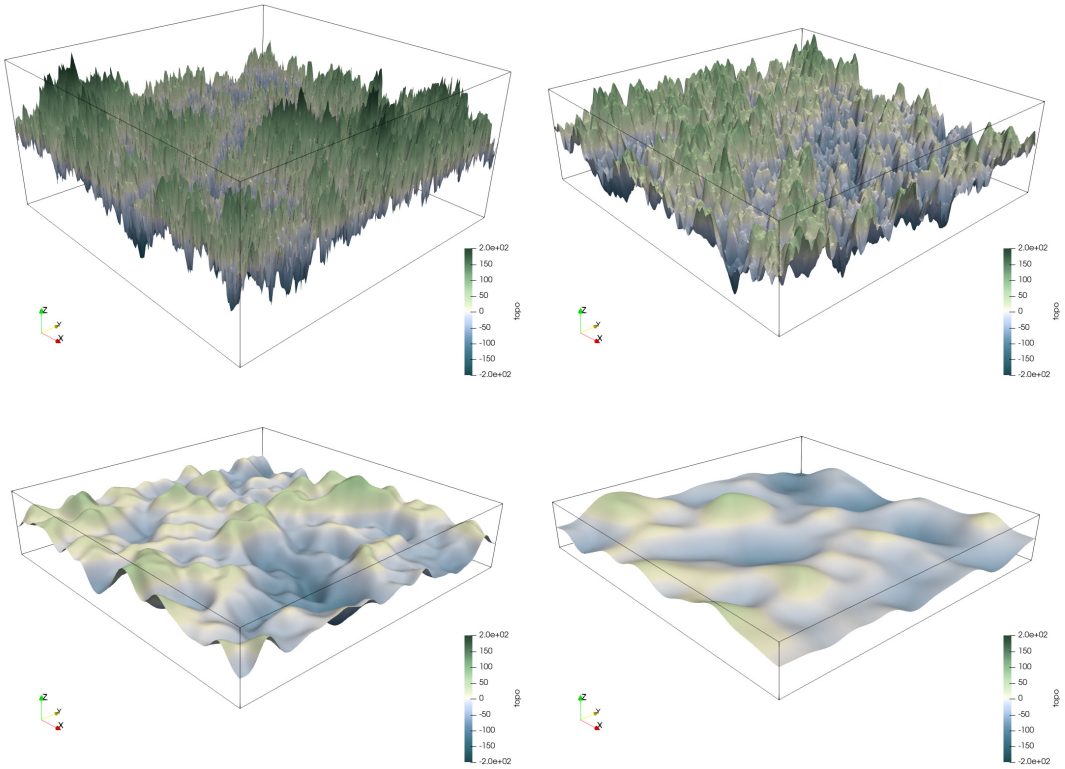
[3] `https://github.com/AgnesHendrickx/MTE/blob/main/art_DEM.py`

[4] `https://mte.readthedocs.io/en/latest/`

[5] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.gaussian_filter.html`



Resulting topography for $L = 1000$m, grid 257x257, $\sigma = 0$ and for roughness=10,50,100 from left to right.



Resulting topography for $L = 1000$m, grid 257x257, roughness=100 and $\sigma = 0, 1, 5, 10$.

Stone 57

DG-FEM: 1D steady state diffusion

```
1D
energy_equation
discontinuous_galerkin
analytical_benchmark
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_57`

This stone was developed in collaboration with Jort Jansen.

Boundary conditions are $T(x=0)=0$ and $T(x=1)=1$. All material coefficients are set to 1 so that we are solving $d^2T/dx^2=0$ over the domain $[0,1]$. The analytical solution is of course $T(x)=x$ and $q(x)=1$ (Note that because of the conventions used in the book we used q is here positive!).

The code

We follow the discretisation presented in Section 13.2 and use the successive substitution to arrive at the solution.

There are `nelx` linear elements and therefore `nnx` nodes. We start by generating an array containing the coordinates of the nodes:

```
x=np.linspace(0,Lx,nnx)
```

We then need to declare arrays for temperature and heat flux, both on the left and on the right of each node:

```
T_min=np.zeros(nnx,dtype=np.float64)
T_plus=np.zeros(nnx,dtype=np.float64)
q_min=np.zeros(nnx,dtype=np.float64)
q_plus=np.zeros(nnx,dtype=np.float64)
```

and we will need the corresponding arrays keeping track of their previous values:

```
T_min_old=np.zeros(nnx,dtype=np.float64)
T_plus_old=np.zeros(nnx,dtype=np.float64)
q_min_old=np.zeros(nnx,dtype=np.float64)
q_plus_old=np.zeros(nnx,dtype=np.float64)
```

The boundary condition at $x=0$ is imposed by setting

```
T_min[0]=T_left
```

while the boundary condition at $x=L_x$ is imposed by setting

```
T_plus[nnx-1]=T_right
```

Because the element size h_x is constant we can precompute the three required matrices:

```
K=np.array([[hx/3,hx/6,-C  ,-0.5],
            [hx/6,hx/3,0.5,-C   ],
            [C   ,-0.5,E   ,0   ],
            [0.5 ,C   ,0   ,E   ]])
```

which corresponds to the matrix of Eq. (13.28),

```
K_left=np.array([[hx/3,hx/6,-0.5,-0.5],
                 [hx/6,hx/3, 0.5,-C   ],
                 [0.5 ,-0.5, E   ,0   ],
                 [0.5 ,C   , 0   ,E   ]])
```

which corresponds to the matric of of Eq. (**??**),

```
K_right=np.array([[hx/3,hx/6,-C  ,-0.5],
                  [hx/6,hx/3,0.5,0.5 ],
                  [C   ,-0.5,E   ,0   ],
                  [0.5 ,-0.5,0   ,E   ]])
```

which corresponds to the matrix of Eq. (**??**).

We then start iterating

```
for it in range(0,niter):
```

and update the flux at the boundaries:

```
q_min[0]       =q_plus[0]    -E*(T_min[0]      -T_plus[0])       # left boundary
q_plus[nnx-1]=q_min[nnx-1]-E*(T_min[nnx-1]-T_plus[nnx-1]) # right boundary
```

We then need to loop over all elements, of each we find the corresponding value for k (`k`) and $k+1$ (`kp1`). If the loop index points at the first element (`iel==0`) then the matrix $\boldsymbol{K}_{el}$ is $\boldsymbol{K}_{left}$, if the loop index points at the last element (`iel==nel-1`) then the matrix $\boldsymbol{K}_{el}$ is $\boldsymbol{K}_{right}$, otherwise $\boldsymbol{K}_{el}$ is $\boldsymbol{K}$ from Eq.(13.28).

In all three cases the rhs vector `rhs_el` is filled according to Eqs. (**??**), (**??**) and (13.28) respectively.

When the elemental matrix and rhs are established for the element under consideration then the systen is solved and q_k^+, q_{k+1}^-, T_k^+ and T_{k+1}^- are updated.

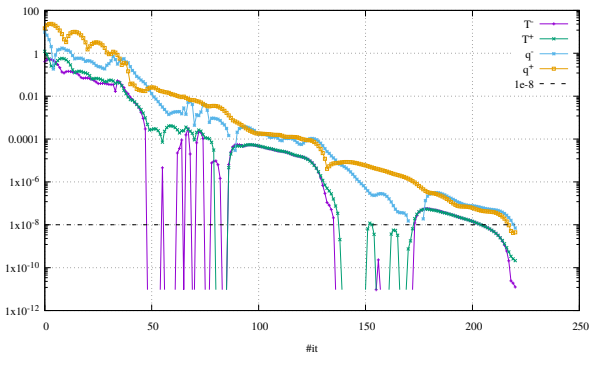
The iteration loop continues until convergence has been reached, i.e. when the difference between the new T, q at all points and the previous values is smaller than the chosen tolerance.

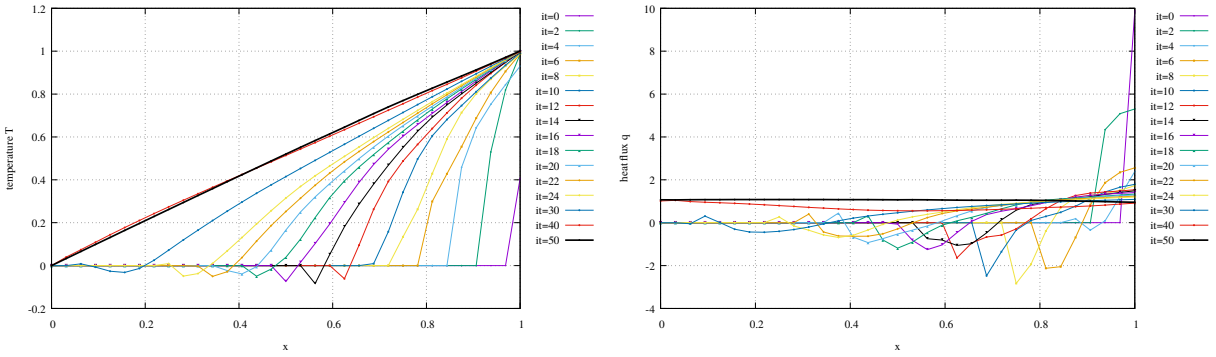
Finally, still inside the iteration loop we must transfer the newly obtained values to the 'old' arrays:

```
T_min_old[:] =T_min[:]
T_plus_old[:]=T_plus[:]
q_min_old[:] =q_min[:]
q_plus_old[:]=q_plus[:]
```

Results

Let us start by fixing $\mathcal{E}=4$ and $\mathcal{C}=0$ and setting the absolute convergence tolerance to 10^{-8}. The code converges after 220 iterations and we indeed recover the expected temperature and heat flux fields:





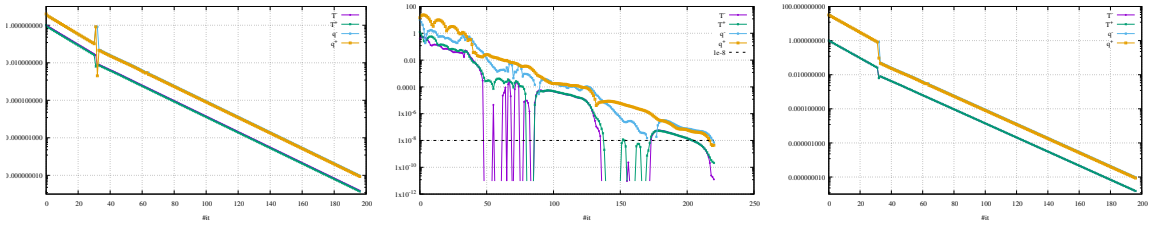
Effect of resolution Keeping the same parameters we see that the number of required iterations to convergence increases linearly with the number of elements.

nelx	# iterations
08	60
16	116
32	220
64	436
128	857

Effect of $\mathcal{E}$ & $\mathcal{C}$ values We now keep nelx=32 and explore the effect of these parameters on the required number of iterations:

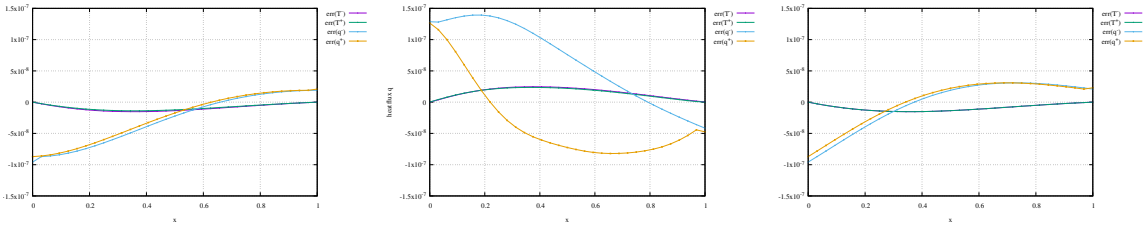
$\mathcal{E}$	$\mathcal{C}$	# iterations
1	0	772
2	0	414
3	-0.5	**104**
3	0	284
3	0.5	129
4	-1/2	196
4	0	220
4	+1/2	196
5	-1/2	256
5	0	195
5	+1/2	256
6	0	246
10	0	486

$\mathcal{E} = 4$ & $\mathcal{C} = -1/2$ $\mathcal{E} = 4$ & $\mathcal{C} = 0$ $\mathcal{E} = 4$ & $\mathcal{C} = +1/2$



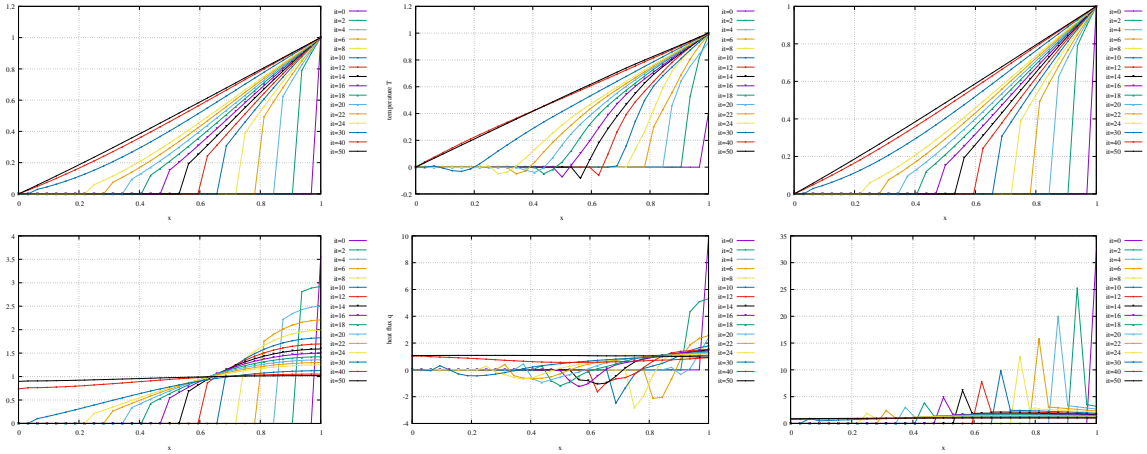
Using $\mathcal{C} \neq 0$ yields a very linear convergence.

$\mathcal{E} = 4$ & $\mathcal{C} = -1/2$ $\mathcal{E} = 4$ & $\mathcal{C} = 0$ $\mathcal{E} = 4$ & $\mathcal{C} = +1/2$



Error between computed and analytical solution as a function of x

$\mathcal{E} = 4$ & $\mathcal{C} = -1/2$ | $\mathcal{E} = 4$ & $\mathcal{C} = 0$ | $\mathcal{E} = 4$ & $\mathcal{C} = +1/2$



The path to a converged solution is quite different for the three values of $\mathcal{C}$ tested.

Stone 58

Elastic disk under compression



python_codes/fieldstone_58/text.tex

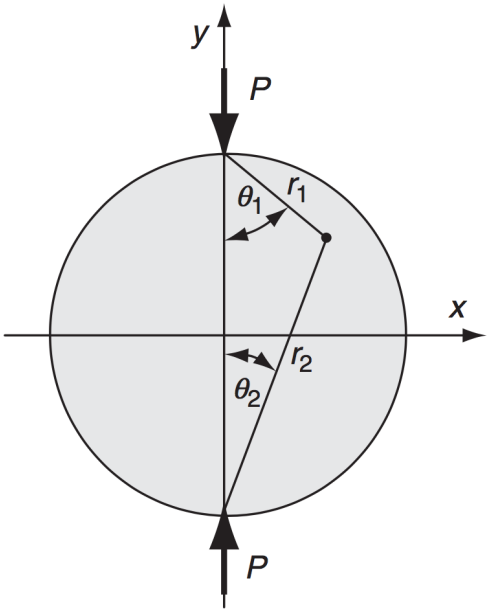
2D
disk
P1
sparse_matrix
compressible
analytical_benchmark
stokes_equation
elastic_rheology
vrms
residual
paraview_output
displacement_code

P F Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_58

This stone was developed in collaboration with L. van de Wiel and T. Shinohara.

Experiment 1

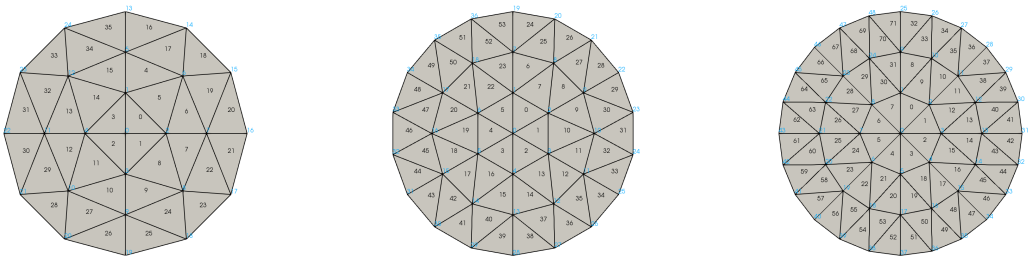
This benchmark is well document in Sadd [272]. Let us investigate the solution to the plane problem shown hereunder of a circular disk loaded by equal but opposite concentrated forces along a given diameter. This particular problem is of special interest since this geometry is used in standard testing of bituminous and other brittle materials such as concrete, asphalt, rock, and ceramics. Normally referred to as the Brazilian or indirect tension test, the sample and loading geometry create a tension zone along the loaded diameter, thus allowing determination of the tensile strength of the specimen material. Standard direct tension testing on such brittle materials has led to difficulty in establishing a failure region in the sample's central interior away from the gripping locations.



Setup of the experiment.

The mesh is regular and made of concentric layers of triangles. The original algorithm which was used in ELEFANT was written by L. van de Wiel and is present in the `mesher_fortran` folder. Given the nature of the boundary conditions we wish to apply we make sure that two faces are present at the top and bottom locations of the disc, which is why the mesh is rotated 90°after it is generated.

The mesh is composed of `nsection` sections (must be an even number) as shown hereunder (for `nLayers`=3).



Mesh composed of 4 sections (left), 6 sections (middle) and 8 sections (right).

The pressure P prescribed at $y = \pm R$ is actually given by $P\delta(\boldsymbol{r}_\pm)$. From [272] (example 8.10, p209), the stress solution is given by:

$$\sigma_{xx}(x,y) = -\frac{2P}{\pi}\left[\frac{(R-y)x^2}{r_1^4} + \frac{(R+y)x^2}{r_2^4} - \frac{1}{D}\right] \tag{58.1}$$

$$\sigma_{yy}(x,y) = -\frac{2P}{\pi}\left[\frac{(R-y)^3}{r_1^4} + \frac{(R+y)^3}{r_2^4} - \frac{1}{D}\right] \tag{58.2}$$

$$\sigma_{xy}(x,y) = \frac{2P}{\pi}\left[\frac{(R-y)^2x}{r_1^4} - \frac{(R+y)^2x}{r_2^4}\right] \tag{58.3}$$

where

$$r_1 = \sqrt{x^2 + (R-y)^2} \qquad r_2 = \sqrt{x^2 + (R+y)^2}$$

The pressure is given by:

$$\begin{aligned} p(x,y) &= -\frac{1}{2}(\sigma_{xx}+\sigma_{yy}) \\ &= \frac{P}{\pi}\left[\frac{(R-y)x^2}{r_1^4}+\frac{(R+y)x^2}{r_2^4}-\frac{1}{D}\right]+\frac{P}{\pi}\left[\frac{(R-y)^3}{r_1^4}+\frac{(R+y)^3}{r_2^4}-\frac{1}{D}\right] \quad (58.4)\\ &= \frac{P}{\pi}\left[\frac{(R-y)x^2+(R-y)^3}{r_1^4}+\frac{(R+y)x^2+(R+y)^3}{r_2^4}-\frac{2}{D}\right] \quad (58.5)\end{aligned}$$

On the x-axis ($y=0$) these results simplify to give

$$\begin{aligned} \sigma_{xx}(x,0) &= \frac{2P}{\pi D}\left(\frac{D^2-4x^2}{D^2+4x^2}\right)^2 \quad (58.6)\\ \sigma_{yy}(x,0) &= -\frac{2P}{\pi D}\left(\frac{4D^4}{(D^2+4x^2)^2}-1\right) \quad (58.7)\\ \sigma_{xy}(x,0) &= 0 \quad (58.8)\\ p(x,0) &= \frac{P}{\pi}\left[\frac{(R-y)x^2+(R-y)^3}{r_1^4}+\frac{(R+y)x^2+(R+y)^3}{r_2^4}-\frac{2}{D}\right] \quad (58.9)\\ &= \frac{P}{\pi}\left[\frac{Rx^2+R^3}{(x^2+R^2)^2}+\frac{Rx^2+R^3}{(x^2+R^2)^2}-\frac{2}{D}\right] \quad (58.10)\\ &= \frac{2P}{\pi}\left[\frac{R(x^2+R^2)}{(x^2+R^2)^2}-\frac{1}{D}\right] \quad (58.11)\\ &= \frac{2P}{\pi}\left[\frac{R}{x^2+R^2}-\frac{1}{D}\right] \quad (58.12)\\ & \quad (58.13)\end{aligned}$$

On the y-axis ($x=0$) the stresses are

$$\begin{aligned} \sigma_{xx}(0,y) &= \frac{2P}{\pi D} \quad (58.14)\\ \sigma_{yy}(0,y) &= -\frac{2P}{\pi}\left(\frac{2}{D-2y}+\frac{2}{D+2y}-\frac{1}{D}\right) \quad (58.15)\\ &= -\frac{2P}{\pi}\left(\frac{1}{R-y}+\frac{1}{R+y}-\frac{1}{D}\right) \quad (58.16)\\ \sigma_{xy}(0,y) &= 0 \quad (58.17)\\ p(0,y) &= \frac{P}{\pi}\left[\frac{(R-y)^3}{(R-y)^4}+\frac{(R+y)^3}{(R+y)^4}-\frac{2}{D}\right] \quad (58.18)\\ &= \frac{P}{\pi}\left[\frac{1}{R-y}+\frac{1}{R+y}-\frac{2}{D}\right] \quad (58.19)\end{aligned}$$

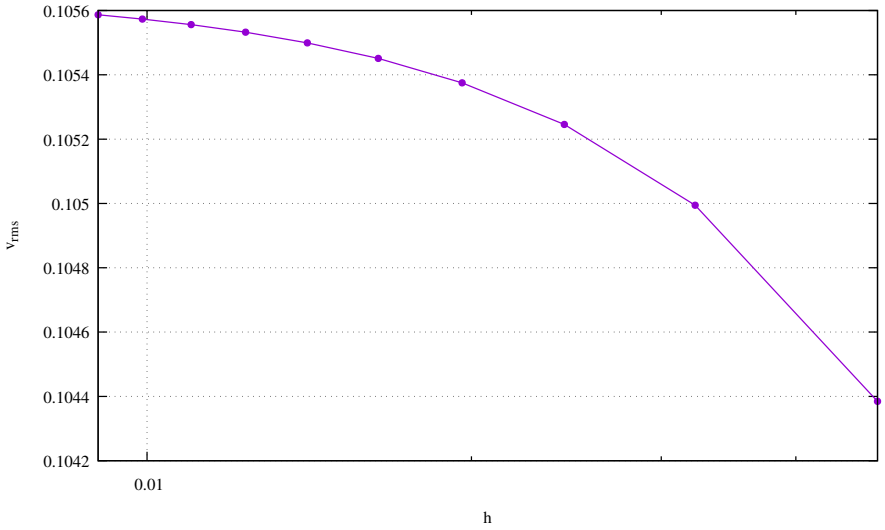
In the code the pressure is retrieved after the displacements are computed. In 2D, we have (using Eq. 3.123 for the stress tensor):

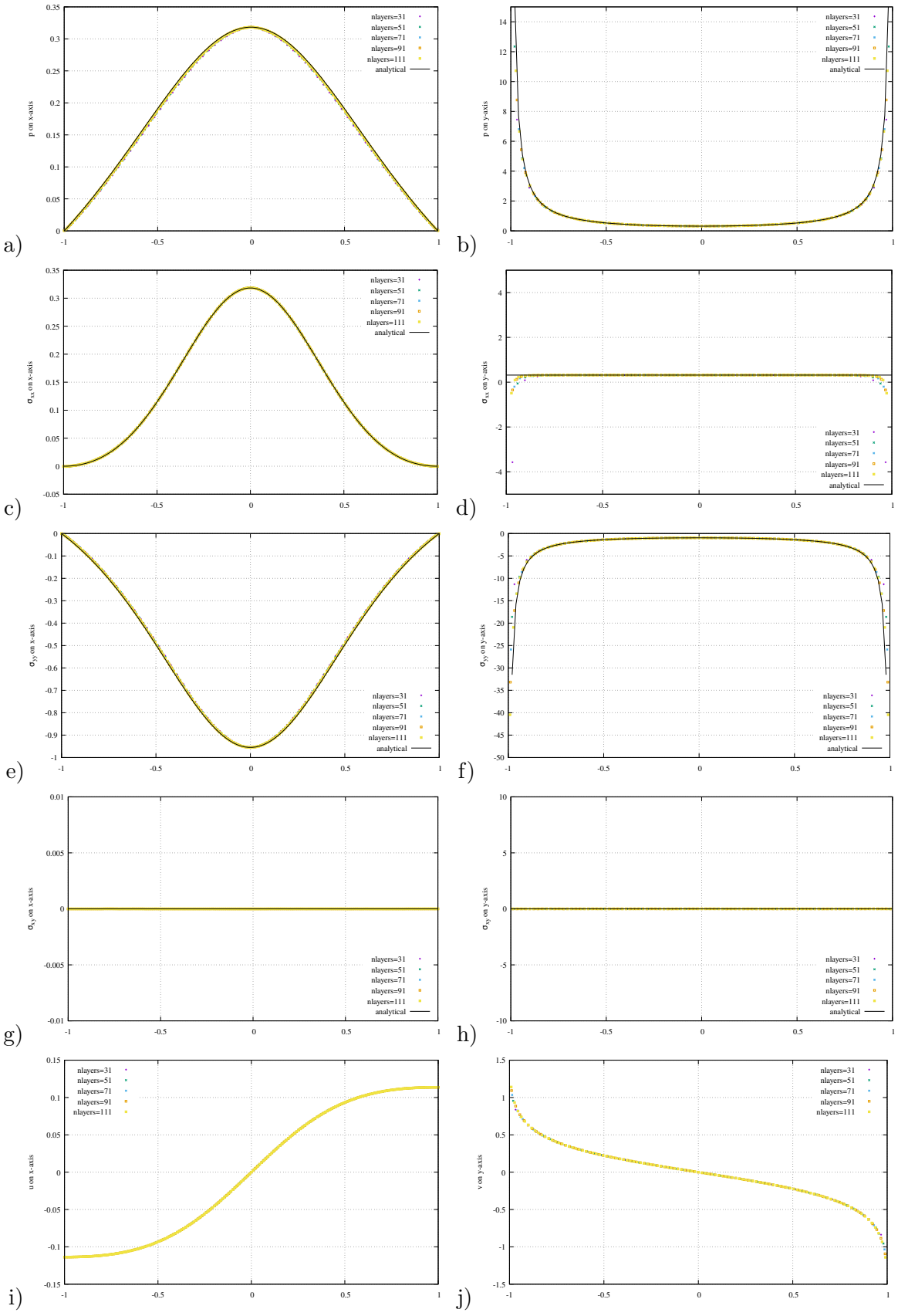
$$\begin{aligned} p &= -\frac{1}{2}(\sigma_{xx}+\sigma_{yy}) \\ &= -\frac{1}{2}\left[\lambda(\varepsilon_{xx}+\varepsilon_{yy})+2\mu\varepsilon_{xx}+\lambda(\varepsilon_{xx}+\varepsilon_{yy})+2\mu\varepsilon_{yy}\right] \quad (58.20)\\ &= -\frac{1}{2}\left[2\lambda(\varepsilon_{xx}+\varepsilon_{yy})+2\mu(\varepsilon_{xx}+\varepsilon_{yy}\right] \\ &= -(\lambda+\mu)(\varepsilon_{xx}+\varepsilon_{yy}) \quad (58.21)\end{aligned}$$

The radius is set to $R=1$, and the disc is centered on the origin. The disc is discretised by means of *nlayers* concentric layers of P_1 triangles. I set $\mu=1$ and $\nu=0.25$.

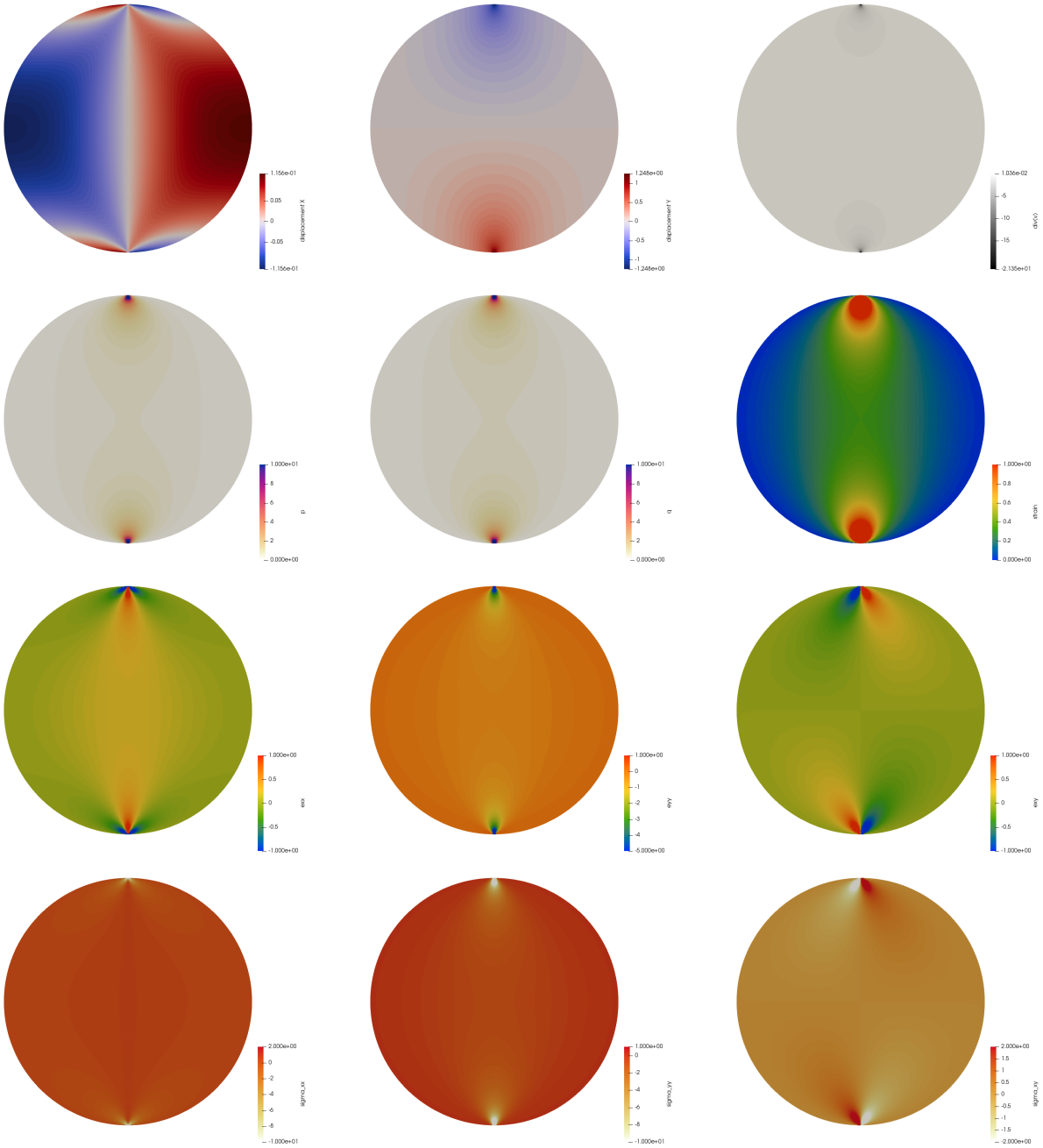
The boundary conditions are as follows: on the two vertical edges at $y = R$ and $y = -R$ the pressure is applied. Furthermore because of the symmetry of the problem, and in order to remove the expected nullspaces in the displacement field, we fix $u = 0$ on the vertical axis and $v = 0$ on the horizontal axis. In the coming plots the number of layers (which needs to be odd) is varied, from 31 to 111. Note that the displacement is not analytically known but the computed displacement nicely converge to a single smooth curve.

nlayers	NV	nel	Nfem	v_{rms}
21	1387	2646	2774	1.043847e-01
31	2977	5766	5954	1.049944e-01
41	5167	10086	10334	1.052457e-01
51	7957	15606	15914	1.053750e-01
61	11347	22326	22694	1.054509e-01
71	15337	30246	30674	1.054994e-01
81	19927	39366	39854	1.055324e-01
91	25117	49686	50234	1.055560e-01
101	30907	61206	61814	1.055735e-01
111	37297	73926	74594	1.055868e-01

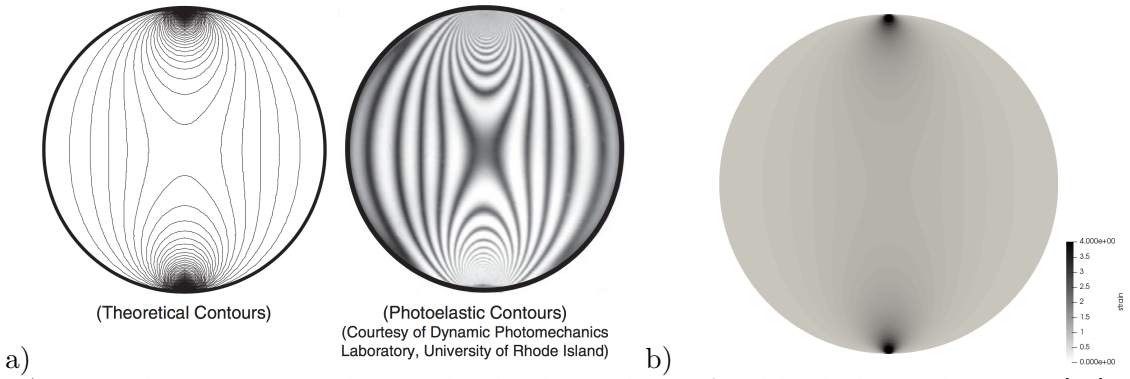




a,b) pressure along the x-axis; c,d) σ_{xx} along the $x-$ and $y-$axis; e,f) σ_{yy} along the $x-$ and $y-$axis; g,h) σ_{xy} along the $x-$ and $y-$axis; i) horizontal displacement along the $x-$axis; j) vertical displacement along the $y-$axis



Results for nLayers=11



a) Maximum shear stress contours and corresponding photoelastic isochromatic for a disk under diametrical compression [272]; b) computed second invariant of the strain tensor.

We can also look at the principal stresses. The principal direction angle θ_p defines the principal directions where the only stresses are normal stresses, and is given by the relationship:

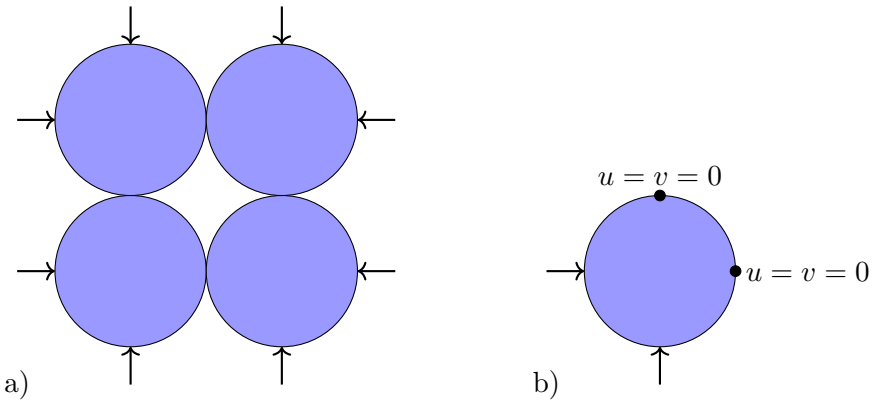
$$\tan(2\theta_p) = \frac{2\sigma_{xy}}{\sigma_{xx} - \sigma_{yy}}$$

The principal stresses are found from the original stresses via

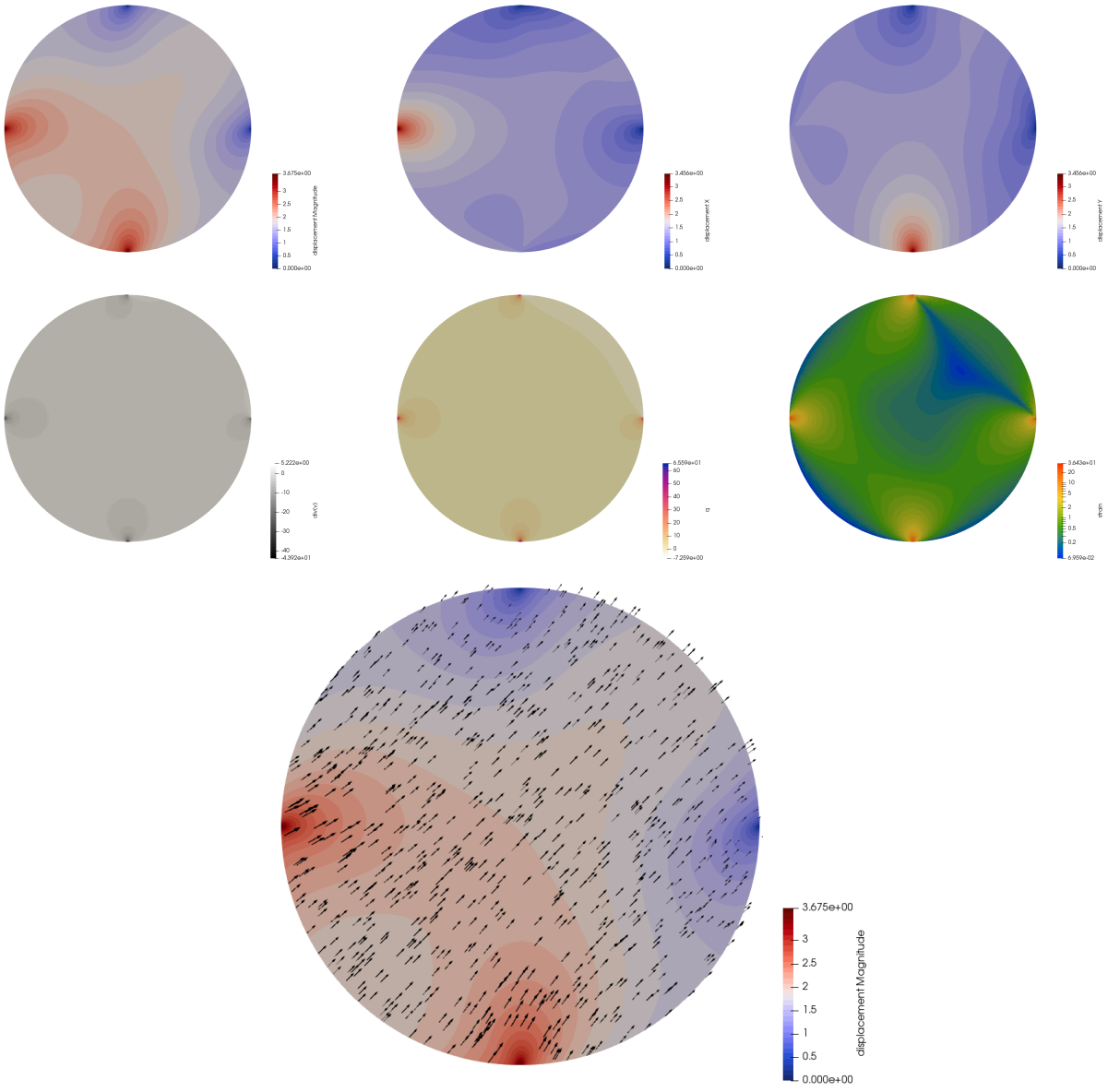
$$\sigma_{1,2} = \frac{\sigma_{xx} + \sigma_{yy}}{2} \pm \sqrt{\left(\frac{\sigma_{xx} - \sigma_{yy}}{2}\right)^2 + \sigma_{xy}^2}$$

Experiment 2

This experiment was designed in collaboration with T. Shinohara . Unlike the previous one it does not have an analytical solution. Ideally we wished to look at a cluster of grains as shown hereunder. However, because of the symmetries of the problem we can here again only model one grain, with (idealised) boundary conditions:



a) assembly of four circular grains and the pressure boundary conditions; b) simplified setup of a single grain.



Relevant Literature:

- Compaction creep of sands due to time-dependent grain failure: Effects of chemical environment, applied stress, and grain size. Brzesowsky *et al.* (2014) [41]
- Failure behavior of single sand grains: Theory versus experiment. Brzesowsky *et al.* (2011) [42]
- Time-independent compaction behavior of quartz sands. Brzesowsky *et al.* (2014) [43]
- Determination of the tensile strength of rock by a compression test of an irregular test piece. Hiramatsu *et al.* (1966) [154]
- Micromechanics of sand grain failure and sand compaction. Brzesowsky (1995) [44]
- Contact fatigue in silica sand—Observations and modeling. Wang & Michalowski (2015) [345]
- Tensile stress concentration and compressive failure in cemented granular material. Wong & Wu (1995)[353]
- Micromechanics of pressure-induced grain crushing in porous rocks. Zhang *et al.* (1990) [365]

Stone 59

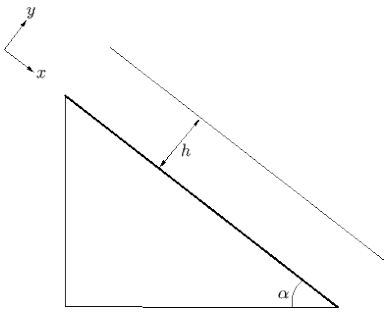
Ice Flow Down an Inclined Plane



python_codes/fieldstone_59/text.tex

2D
box
Q2Q1
sparse_matrix
numerical_benchmark
stokes_equation
nonlinear
viscous_rheology
plastic_rheology
paraview_output
pressure_scaling
schur_complement

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_59`



REDO with tikz

Linear viscous fluid We start from the Stokes equation for isoviscous fluids:

$$\eta \Delta \vec{\mathrm{v}} - \vec{\nabla} p + \rho \vec{g} = \vec{0}$$

Assuming that the fluid is incompressible and that the incline is infinite, then $\vec{\mathrm{v}} = (u(y), 0)$. The x-component of the equation then writes:

$$\eta \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) - \frac{\partial p}{\partial x} + \rho g_x = 0$$

$$\Rightarrow \qquad \eta \frac{\partial^2 u}{\partial y^2} + \rho g \sin\alpha = 0$$

since $\partial_x \to 0$. This 2nd order ODE can be integrated twice. The two integration constants are determined by setting $u(y=0)=0$ (no slip) and the shear stress to be zero at the (free) surface. The velocity profile is given by:

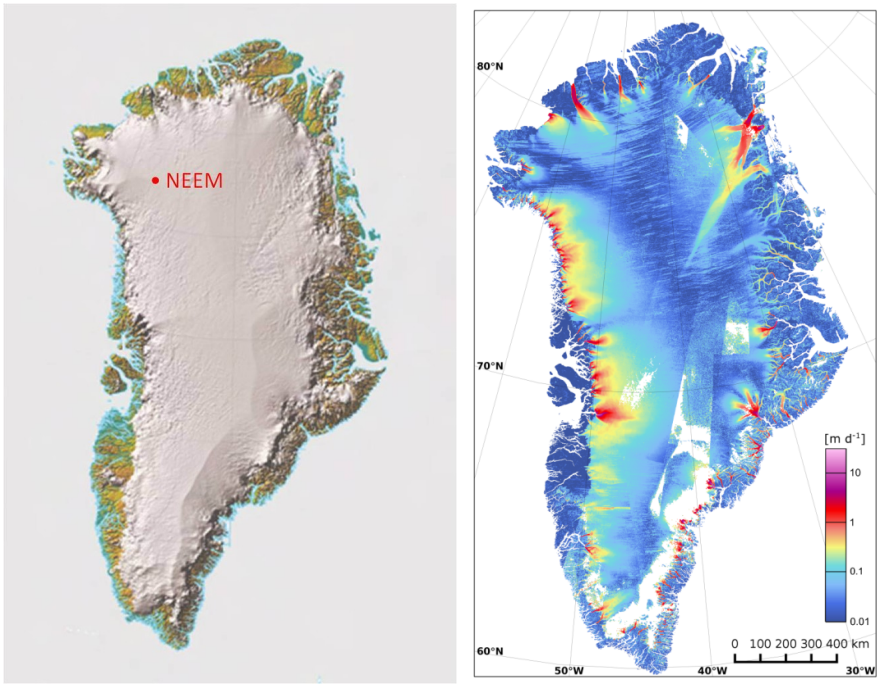
$$u(y) = \frac{\rho g \sin\alpha}{2\eta}(2h - y)y$$

We can now compute the components of the strain rate tensor:

$$\dot{\varepsilon}_{xx} = 0 \qquad \dot{\varepsilon}_{yy} = 0 \qquad \dot{\varepsilon}_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) = \frac{1}{2}\frac{\partial u}{\partial y} = \frac{\rho g \sin\alpha}{2\eta}(h - y)$$

We now make the assumption that this problem is a very simplified ice sheet flow problem. The angle α is typically small and for $\alpha \sim 0.5 - 1^\circ$, then $\sin\alpha \sim 0.01$. We also have $\rho \sim 1000$, $g \sim 10$, $h \sim 2500$ and we postulate the ice viscosity to be $\eta \sim 10^m$,. The velocity is maximum at the surface and is given by

$$u(y=h) = \frac{\rho g \sin\alpha}{2\eta} h^2 \sim \frac{1000 \cdot 10 \cdot 0.01}{2 \cdot 10^m} 2500^2 \sim 3 \cdot 10^{8-m} \tag{59.1}$$



Ice velocity map (magnitude, in logarithmic scale) of the Greenland Ice Sheet derived from SAR data of the Sentinel-1A satellite, acquired in Interferometric Wide Swath Mode (IW) between January and March 2015. Taken from Nagler, Rott, Hetzenecker, Wuite, and Potin [240] (2015).

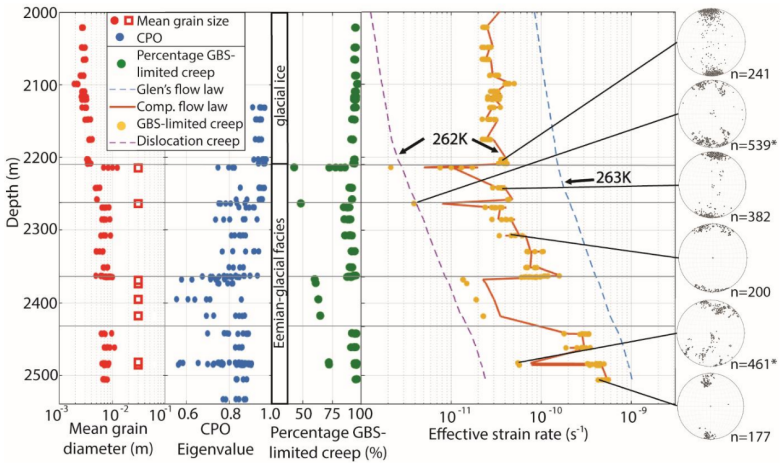
As shown on the figures above the typical ice sheet velocity values around the NEEM ice core is $\sim 0.03\ \mathrm{m\,day^{-1}} \simeq 3 \cdot 10^{-7}\ \mathrm{m\,s^{-1}}$. Then, in order for the top of the ice sheet to flow at this speed it requires $m = 15$.

Having obtained the (effective linear) viscosity of the ice, i.e. $\eta \simeq 10^{15}$ Pa s, we can compute the strain rate at the bottom and in the middle:

$$\dot{\varepsilon}_{xy}(y=0) = \frac{\rho g \sin\alpha}{2\eta} h \sim \frac{1000 \cdot 10 \cdot 0.01}{2 \cdot 10^{15}} 2500 \sim 10^{-10}\ \mathrm{s^{-1}}$$

$$\dot{\varepsilon}_{xy}(y=h/2) = \frac{\rho g \sin\alpha}{2\eta} \frac{h}{2} \sim \frac{1000 \cdot 10 \cdot 0.01}{2 \cdot 10^{15}} \frac{2500}{2} \sim 5 \cdot 10^{-11}\ \mathrm{s^{-1}}$$

which are reasonable values as confirmed by the following figure (see yellow line):



Taken from Kuiper, Bresser, Drury, Eichler, Pennock, and Weikusat [201] (2019).

The conclusion from this exercise is that *if* ice could be described by a linear viscous fluid flowing on an infinite incline, a viscosity of $\eta \simeq 10^{15}$Pa.s would yield observations that match recorded velocities and strain rates.

Nonlinear power-law viscosity However, the picture is way more complex, as it was observed very early on by Glen in 1955 [132] and many others later (see Section 5.22). It was then found that the strain rate and the stress are linked in a nonlinear way:

$$\dot{\varepsilon} = A\tau^n,$$

i.e. the ice behaves as a power-law fluid (see Section 5.3), with $n = 3$ and $A \simeq 2.4 \cdot 10^{-24}$ at 0°(REF?).

More recently Kuiper, Weikusat, Bresser, Jansen, Pennock, and Drury [202] (2019) and Kuiper, Bresser, Drury, Eichler, Pennock, and Weikusat [201] (2019) derived a composite flow law to model deformation in the NEEM deep ice core. They start from the flow law proposed by Goldsby and Kohlstedt [136] (2001):

$$\dot{\varepsilon}_T = \dot{\varepsilon}_{disl} + \left(\frac{1}{\dot{\varepsilon}_{basal}} + \frac{1}{\dot{\varepsilon}_{GBS}} \right)^{-1} + \dot{\varepsilon}_{diff}$$

where $\dot{\varepsilon}_T$ is the total strain rate, composed of strain rates for basal slip accommodated by non-basal slip or dislocation creep, $\dot{\varepsilon}_{disl}$, grain boundary sliding (GBS) accommodated by basal slip, $\dot{\varepsilon}_{basal}$, and basal slip accommodated by GBS, $\dot{\varepsilon}_{GBS}$, and diffusion creep, $\dot{\varepsilon}_{diff}$. Each of these creep mechanisms can be described by a power law relation of the form:

$$\dot{\varepsilon} = A\tau^n d^{-p} \exp\left(-\frac{Q + pV}{RT} \right)$$

where A is a material parameter, τ is the differential stress (MPa), n is the stress exponent, d is the grain size diameter (m), p is the grain size exponent, Q is the activation energy for the creep mechanism at stake (J mol^{-1}), p is the hydrostatic pressure (MPa), V the activation volume (m^3 mol^{-1}), R is the gas constant (J K^{-1} mol^{-1}), and T the absolute temperature (K). The effect of pV is assumed to be very small [100] and is ignored for the remainder of this work.

As explained in Kuiper, Weikusat, Bresser, Jansen, Pennock, and Drury [202] (2019) the composite flow law can actually be simplified to:

$$\dot{\varepsilon}_T = \dot{\varepsilon}_{disl} + \dot{\varepsilon}_{GBS}$$

The material parameters for these deformation mechanisms are available in [201]:

Creep regime	A	n	p	Q (J mol^{-1})
Glen's flow law ($T < 263$K)	$3.61 \cdot 10^5$ MPa$^{-3.0}$ s^{-1}	3.0	0	60
Glen's flow law ($T > 263$K)	$1.73 \cdot 10^{21}$ MPa$^{-3.0}$ s^{-1}	3.0	0	139
Dislocation creep ($T < 262$K)	$5.0 \cdot 10^5$ MPa$^{-4.0}$ s^{-1}	4.0	0	64
Dislocation creep ($T > 262$K)	$6.96 \cdot 10^{23}$ MPa$^{-4.0}$ s^{-1}	4.0	0	155
GBS-limited creep ($T < 262$K)	$1.1 \cdot 10^2$ MPa$^{-1.8}$ m$^{1.4}$ s^{-1}	1.8	1.4	70
GBS-limited creep ($T > 262$K)	$8.5 \cdot 10^{37}$ MPa$^{-1.8}$ m$^{1.4}$ s^{-1}	1.8	1.4	250

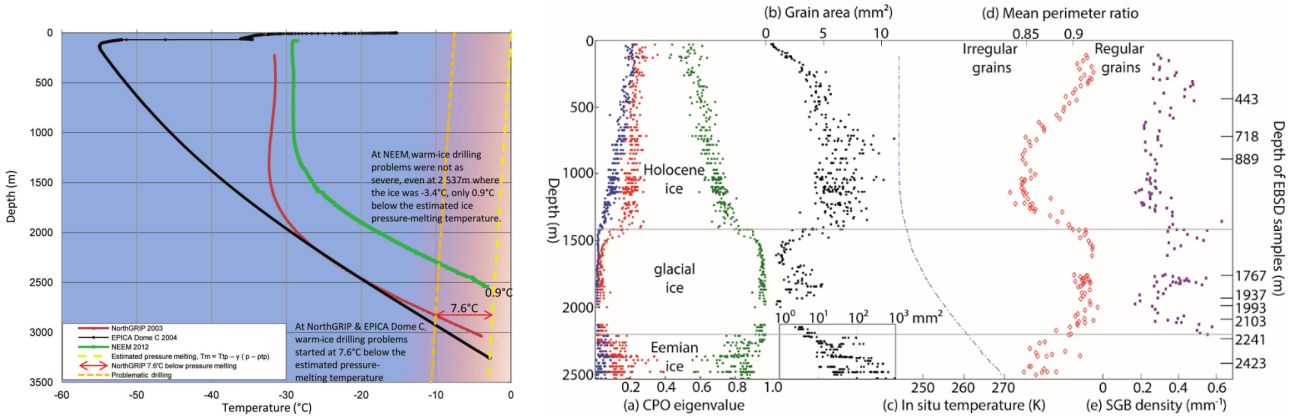
The associated effective viscosity is then given by

$$\eta = \frac{1}{2} A^{-\frac{1}{n}} \dot{\varepsilon}^{\frac{1}{n}-1} d^{\frac{p}{n}} \exp\left(\frac{Q+pV}{nRT}\right)$$

Note that the A parameter is given in MPa and not Pa so that when computing the associated effective viscosities these have to be multiplied by 10^6. Because the strain rates are added together we know that the two deformation mechanisms are in series and their effective viscosity is then the harmonic average of both viscosities:

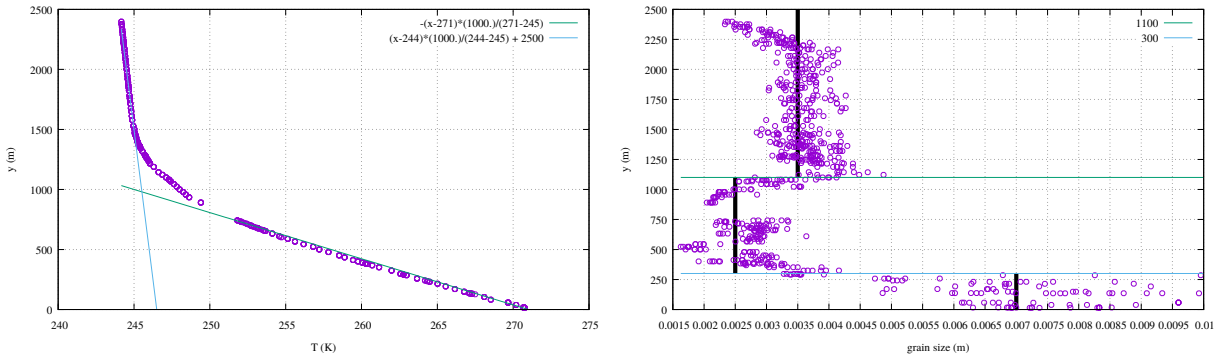
$$\eta_{eff} = \left(\frac{1}{\eta_{disl}} + \frac{1}{\eta_{GBS}}\right)^{-1}$$

Numerical setup The geometry is a 2D cartesian domain of size $L_x \times L_y$. Instead of tilting it by an angle α I choose to instead 'tilt' the gravity vector so that it becomes $\vec{g} = (\rho g \sin\alpha, -\rho g \cos\alpha)$. The boundary conditions are as follows: no slip at the bottom, free surface at the top, free slip on the left (we assume that this corresponds to a ridge - zero horizontal velocity) and we need to be careful about the right boundary which is where the ice exits the domain: if we leave the boundary open ice will rush out, generate high strain rates and the nonlinear rheologies will respond by generating low viscosities which in turn will promote a fast exit. Instead we opt to prescribe the analytical solution of Eq. (**??**) with $\eta = 10^{15}$. We neglect any kind of phase change and precipitation and only carry out a single time step. The domain is discretised with nel=nelx×nely Q_2Q_1 elements (see Section 10.4.2). We set $L_x = 125$km and $L_y = 2500$m, $\rho = 917$kg·m^{-3}, $\alpha = 0.1°$.

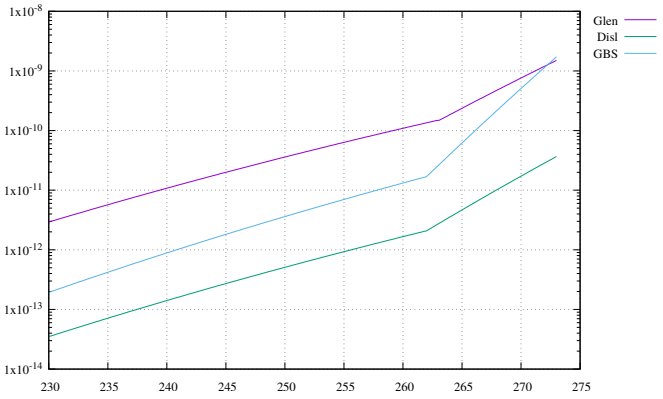


Left: Taken from Sheldon, Steffensen, Hansen, Popp, and Johnsen [292] (2014); Right: Taken from Kuiper [203] (2019).

I have obtained the raw data (curtesy from E.N. Kuiper) and it is plotted hereunder:



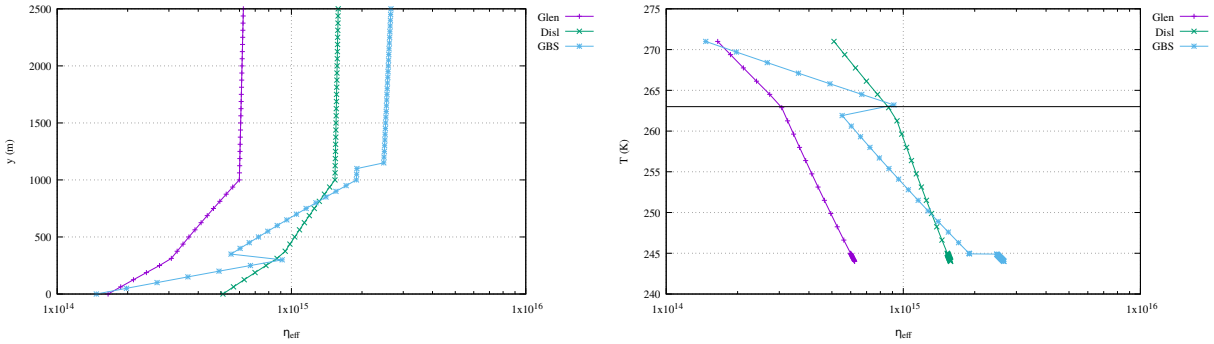
In order to verify that the values reported in the table above are the ones used in [201] I have reproduced the figure 3 of that paper (code and gnuplot script in *./data* folder):



Log strain rate versus temperature for Glen's flow law and the two mechanisms (dislocation creep and GBS-limited creep) that form the end members of the modified composite flow law, using the flow law parameters from the table above. A stress of 0.07MPa and a mean grain diameter of 5 mm were used to calculate the strain rate.

In our models, the temperature profile is simplified as shown on the plots above. Concerning the grain size distribution the ice sheet has been divided in three layers and and an average grain size is used for each.

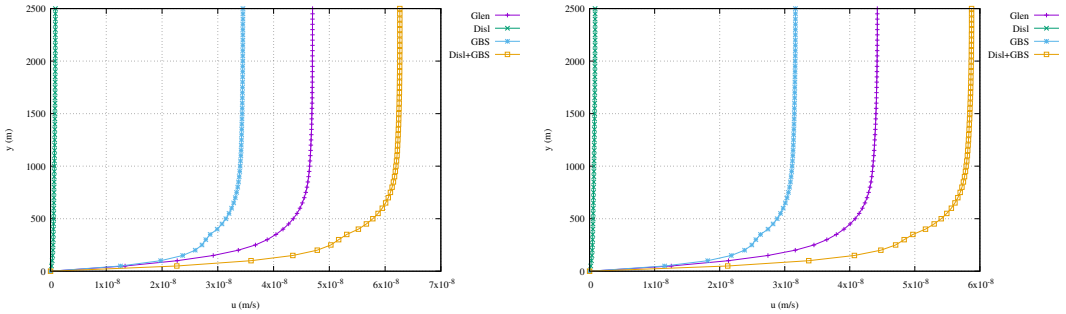
We have computed the effective viscosity for all four rheologies with the implemented temperature and grain size profiles for $\dot{\varepsilon}_e = 10^{-10}\mathrm{s}^{-1}$:

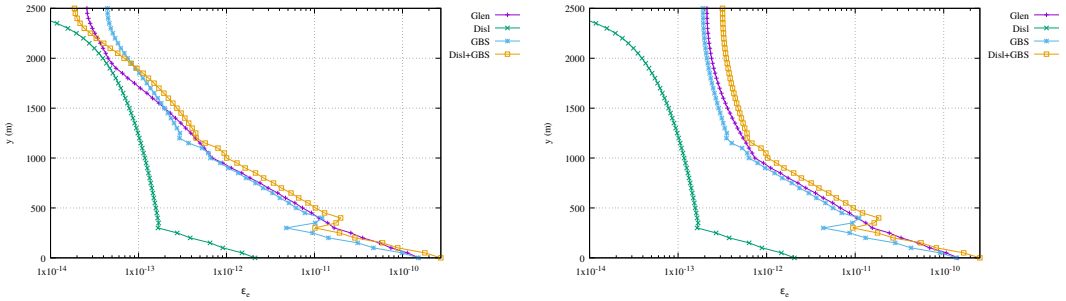


Note that smaller strain rates shift the curves to the right (higher viscosities).

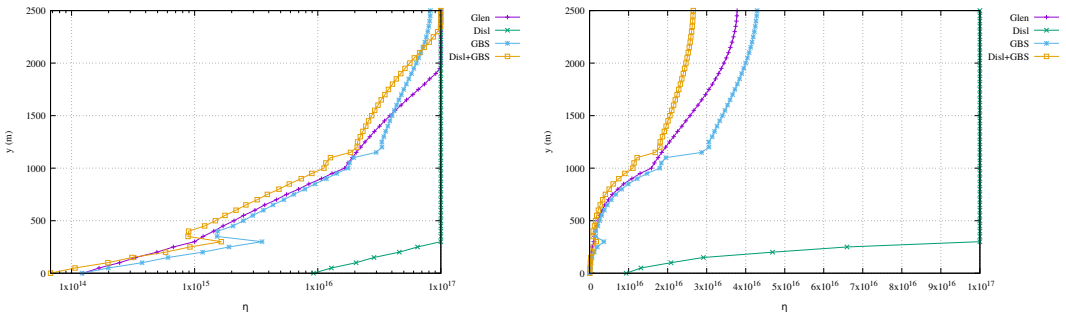
Because the rheologies are non linear we need to carry out nonlinear iterations (see Section 15.2.1). In what follows we run models for Glen's flow law (rheology=1), dislocation creep (rheology=2), GBS (rheology=3) and dislocation+GBS (rheology=4). Note that in the case of rheology=4, the partitioning of the strain rates between both mechanisms is not done (YET) so that the results are not really consistent with the rest!

Core 1 ($x = L_x/4$) Core 2 ($x = L_x/2$)

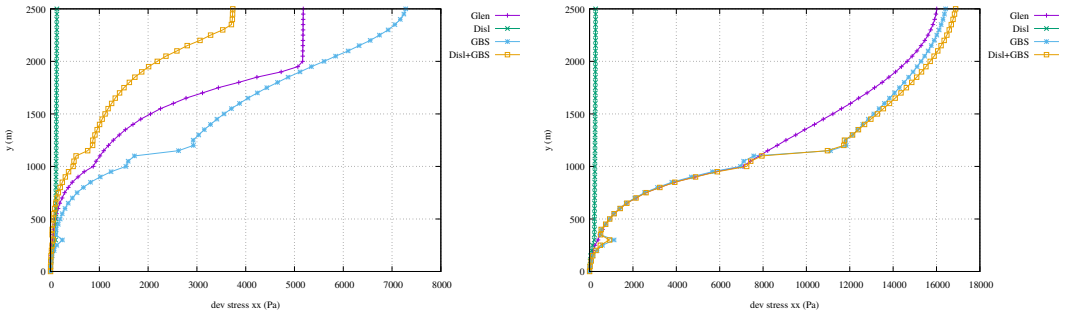



Glen
Disl
GBS
Disl+GBS
y (m)
$\dot{\epsilon}_e$

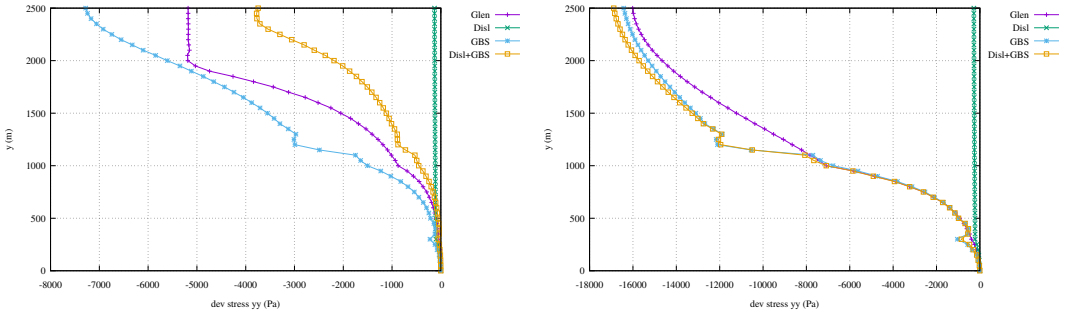
Glen
Disl
GBS
Disl+GBS
y (m)
$\dot{\epsilon}_e$


Glen
Disl
GBS
Disl+GBS
y (m)
η

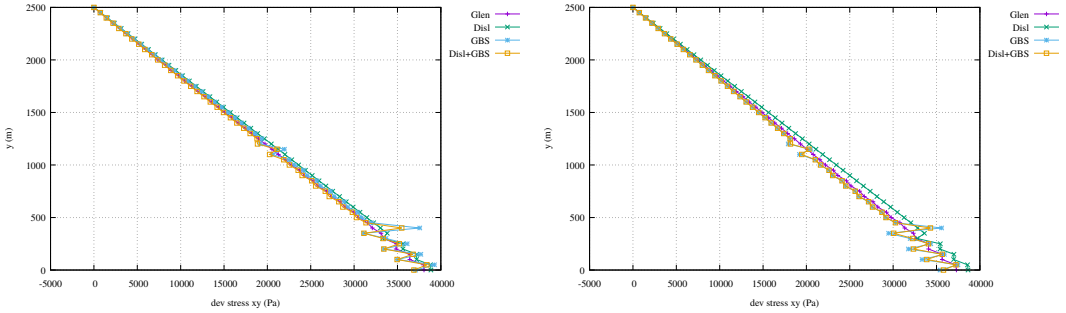
Glen
Disl
GBS
Disl+GBS
y (m)
η


Glen
Disl
GBS
Disl+GBS
y (m)
dev stress xx (Pa)

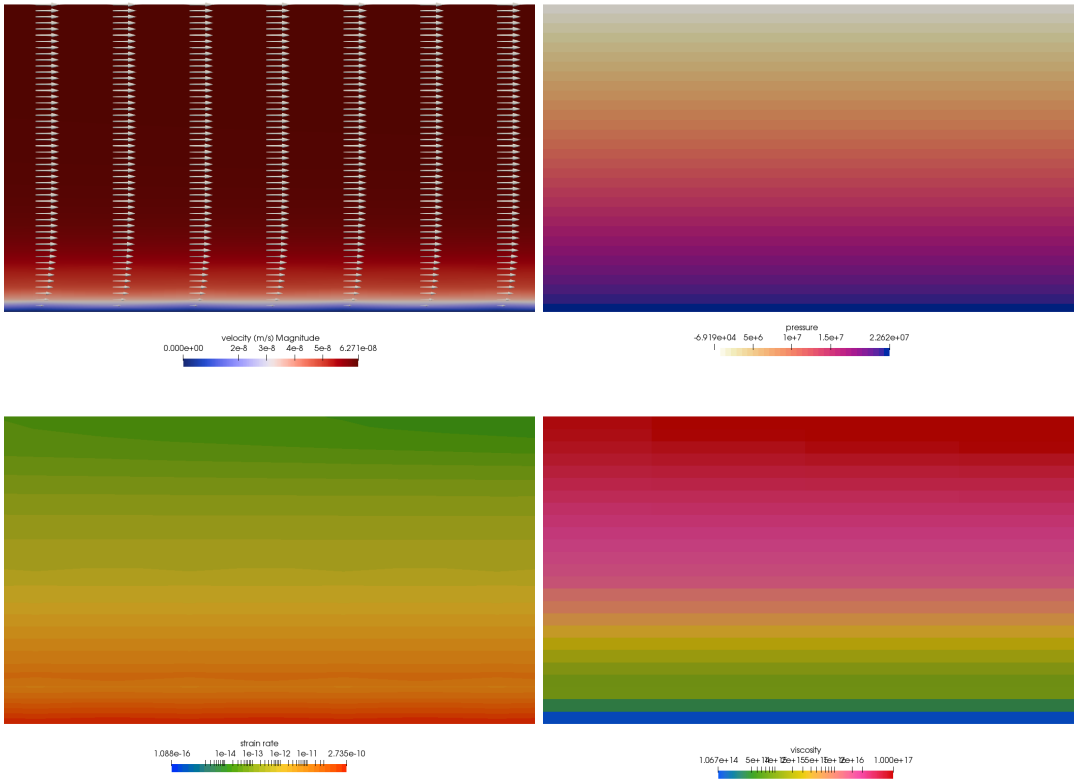
Glen
Disl
GBS
Disl+GBS
y (m)
dev stress xx (Pa)


Glen
Disl
GBS
Disl+GBS
y (m)
dev stress yy (Pa)

Glen
Disl
GBS
Disl+GBS
y (m)
dev stress yy (Pa)


Glen
Disl
GBS
Disl+GBS
y (m)
dev stress xy (Pa)

Glen
Disl
GBS
Disl+GBS
y (m)
dev stress xy (Pa)



Velocity, pressure, strain rate and effective viscosity in the middle of the ice sheet for rheology=4

Check: ELMER/ice `http://elmerice.elmerfem.org/capabilities`

Improvements: better density profile? more realistic geometry ? partitioning of strain rates. Better temperature profile. Better grain size description. Influence of tilt angle?

Relevant LiteratureBudd and Jacka [45] (1989), Zwinger, Greve, Gagliardini, Shiraiwa, and Lyly [375] (2007), Zhang, Ju, Gunzburger, Ringler, and Price [364] (2011), Leng, Ju, Xie, Cui, and Gunzburger [213] (2014), Isaac, Stadler, and Ghattas [165] (2015), Yang and Shi [356] (2015), Goelzer, Robinson, Seroussi, and Van De Wal [134] (2017), Helanow and Ahlkrona [153] (2018).

Stone 60

DG-FEM: 1D advection

```
1D
energy_equation
discontinuous_galerkin
```

This stone implements the 1D discontinuous Galerkin method to solve the simple advection equation:

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} = 0$$

The code

The mesh counts `nelx` linear elements and therefore `nnx` nodes.

The timestep is determined by means of the CFL condition:

```
dt=C*hx/u
```

The mesh is very simply built:

```
x=np.linspace(0,Lx,nnx)
```

We need to declare four arrays: the nodal temperature both on the left and on the right of each node, and their memory of the previous time step.

```
T_minus=np.zeros(nnx,dtype=np.float64)
T_minus_old=np.zeros(nnx,dtype=np.float64)
T_plus=np.zeros(nnx,dtype=np.float64)
T_plus_old=np.zeros(nnx,dtype=np.float64)
```

Initial temperatures are then prescribed on both `T_plus` and *T_minus* arrays, and then copied to the 'old' arrays.

We then enter the time stepping loop:

```
for istep in range(0,nstep):
```

At each timestep the boundary conditions are reapplied[1]:

```
T_minus[0]=T_left
T_plus[0]=T_left
T_minus[nnx-1]=T_right
T_plus[nnx-1]=T_right
```

We then loop over all elements:

```
for iel in range(0,nel):
```

and compute the corresponding k and $k+1$ values:

```
k=iel
kp1=iel+1
```

[1] It is a bit weird that we prescribe a temperature b.c. on a node with an outflow...?

The T^+ and T^- fields are then updated following Eq. (13.63):

```
T_plus[k]    =T_plus_old[k]    +C*(-3*T_plus_old[k]-T_minus_old[k+1]+4*T_minus[k])
T_minus[k+1]=T_minus_old[k+1]+C*( 3*T_plus_old[k]-T_minus_old[k+1]-2*T_minus[k])
```

Results

Experiment 1

We consider the following advection problem taken from Li [214, ex 5.2]. It is also carried out in STONE 43.

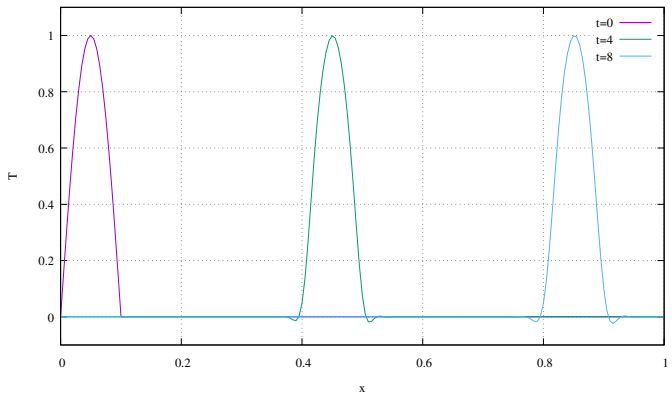
The domain has dimension $L_x = 1$. The temperature is prescribed on the left and the right boundary to be zero. The initial temperature is given by

$$T(x,0) = \begin{cases} \sin(10\pi x) & \text{for } x < 0.1 \\ 0 & \text{for } x \geq 0.1 \end{cases}$$

The velocity is set to $u = 0.1$. We use 200 elements and a time step of $\delta t = 10^{-4}$. We run the model to time $t = 8$ so we need 80,000 time stpes.

Note that the CFL-number is then very small:

$$C = \frac{\delta t \cdot u}{h} = \frac{10^{-4} \cdot 0.1}{1/200} = 0.002$$



Temperature field at three different times.

redo with standard Galerkin and compare!

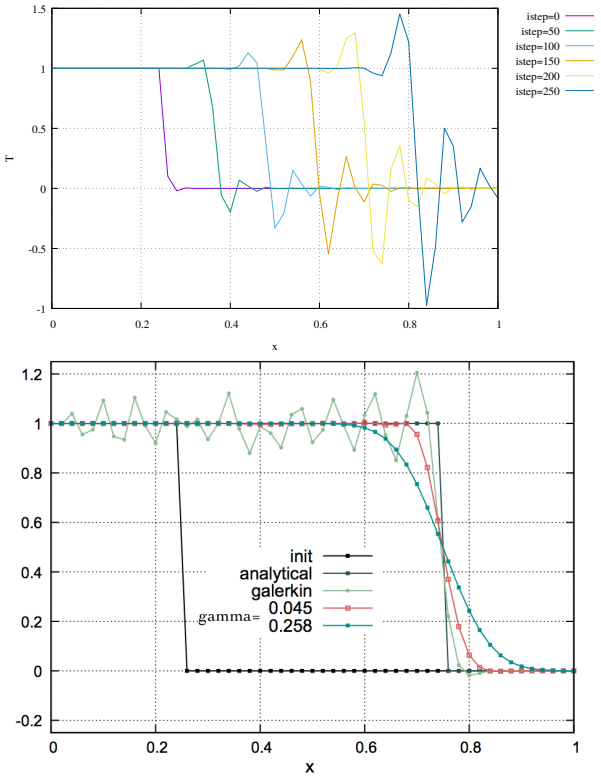
Experiment 2

This advection benchmark originates in Donea & Huerta [94] and it also to be found in Thieulot (2011) [326] (and in Section 9.6.3).

The domain has dimension $L_x = 1$. The temperature is prescribed on the left at $T = 1$ and the right boundary at $T = 0$. The initial temperature is given by

$$T(x,0) = \begin{cases} 1 & \text{for } x < 0.25 \\ 0 & \text{for } x \geq 0.1 \end{cases}$$

Velocity is set to $u = 1$, the number of elements to 50 ($h = 0.02$), the CFL number to 0.1 (so $\delta t = 0.002$), and the number of time steps to 250, so that we expect the front to be at $x = 3L_x/4$.



Left: Temperature field at different times; Right: Taken and modified from Thieulot (2011) [326]

It is rather surprising that the DG results are so much worse than the SG results (even without SUPG stabilisation).

Stone 61

Channel flow with Herschel-Bulkley rheology ($Q_2 \times Q_1$)



2D
box
Q2Q1
sparse_matrix
analytical_benchmark
stokes_equation
nonlinear
viscous_rheology
plastic_rheology
paraview_output
pressure_scaling
schur_complement

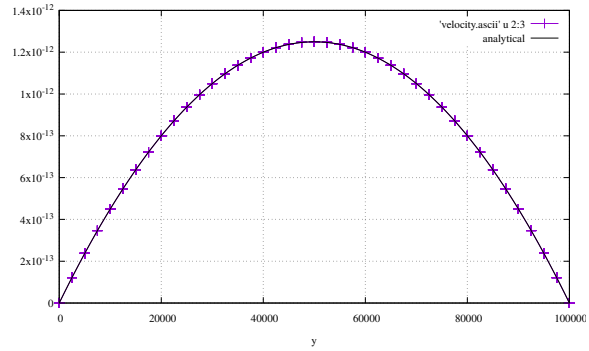
Code at `https://github.com/cedric/fieldstone/tree/master/python_codes/fieldstone_61`

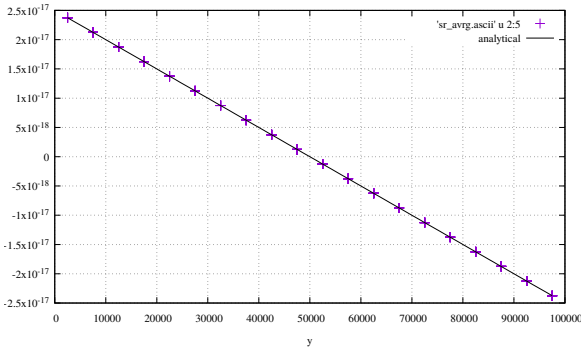
Simple Newtonian Poiseuille flow The analytical solution to this flow has been derived in Section 19.3. We set $p_{left} = 10^9$, $p_{right} = 0$, $L_x = L_y = 100$km, $\eta_0 = 10^{25}$, so

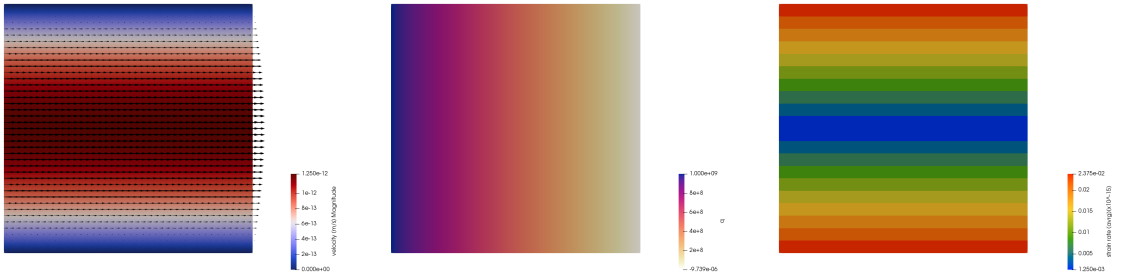
$$\Pi = \frac{p_{right} - p_{left}}{L_x} = -10^4 < 0$$

The p_{left} value is prescribed on the left (see Section 10.15) while nothing is prescribed on the right. No slip boundary conditions are prescribed on the top and bottom.

We see that we recover the analytical solution for velocity and strainrate:





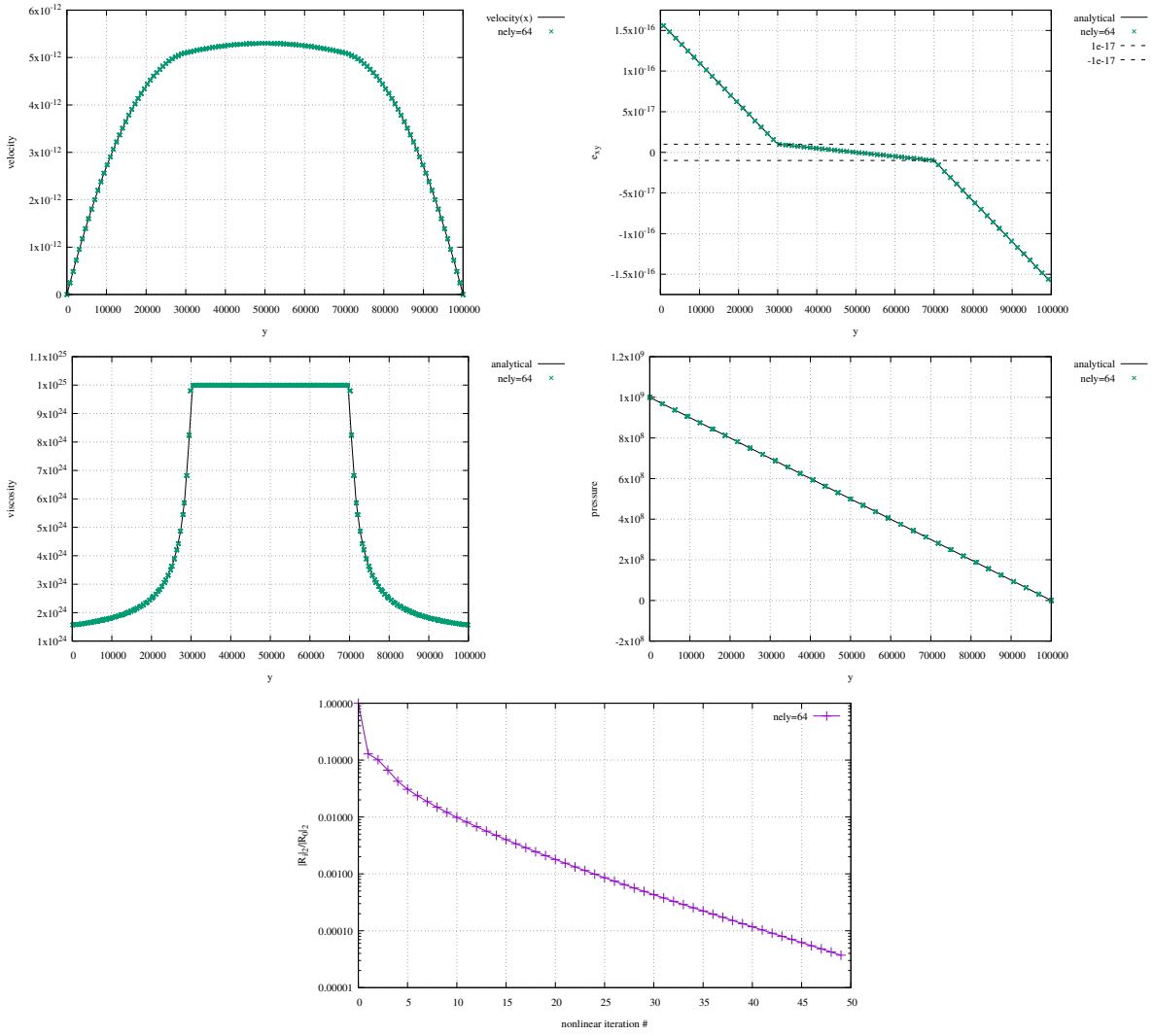


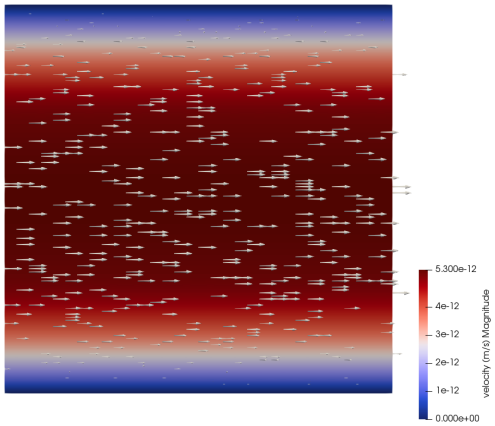
Poiseuille flow with H-B rheology - $n = 1$ The analytical solution to this flow has been derived in Section 18.37. In this case we have

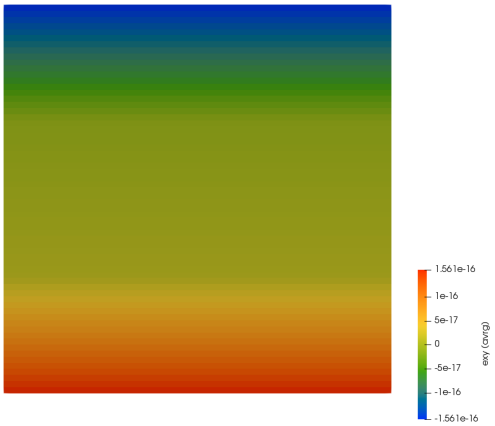
$$\eta_{HB} = \begin{cases} \eta_0 & \dot{\varepsilon}_e \leq \dot{\varepsilon}_0 \\ K + \frac{\tau_0}{\dot{\varepsilon}_e} & \dot{\varepsilon}_e \geq \dot{\varepsilon}_0 \end{cases}$$

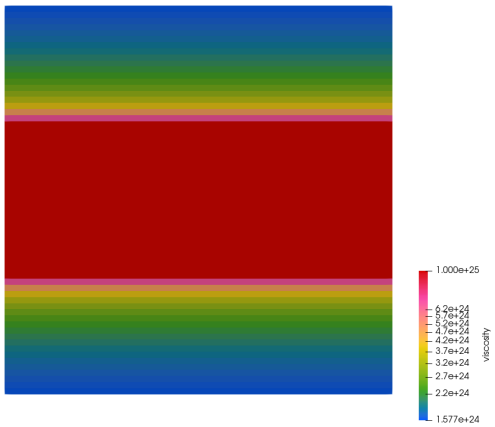
and the limiting viscosity η_0 is such that

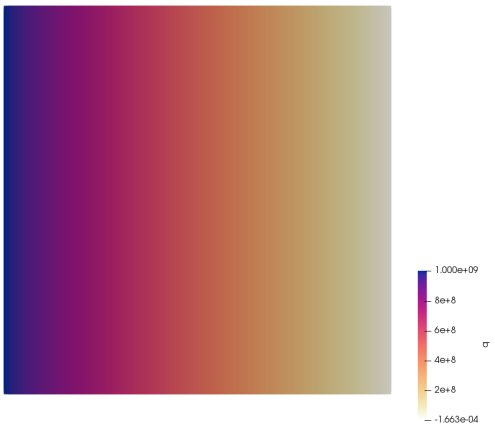
$$\eta_0 = K + \frac{\tau_0}{\dot{\varepsilon}_0}$$




5.300e-12
4e-12
3e-12
2e-12
1e-12
0.000e+00
velocity (m/s) Magnitude


1.561e-16
1e-16
5e-17
0
-5e-17
-1e-16
-1.561e-16
exy (cm/g)


1.000e+25
6.2e+24
5.7e+24
5.2e+24
4.7e+24
4.2e+24
3.7e+24
3.2e+24
2.7e+24
2.2e+24
1.577e+24
viscosity


1.000e+09
8e+8
6e+8
4e+8
2e+8
-1.663e-04
q

Stone 62

Subduction a la Quinquis Case 1 ($(P_2^+ \times P_{-1})$

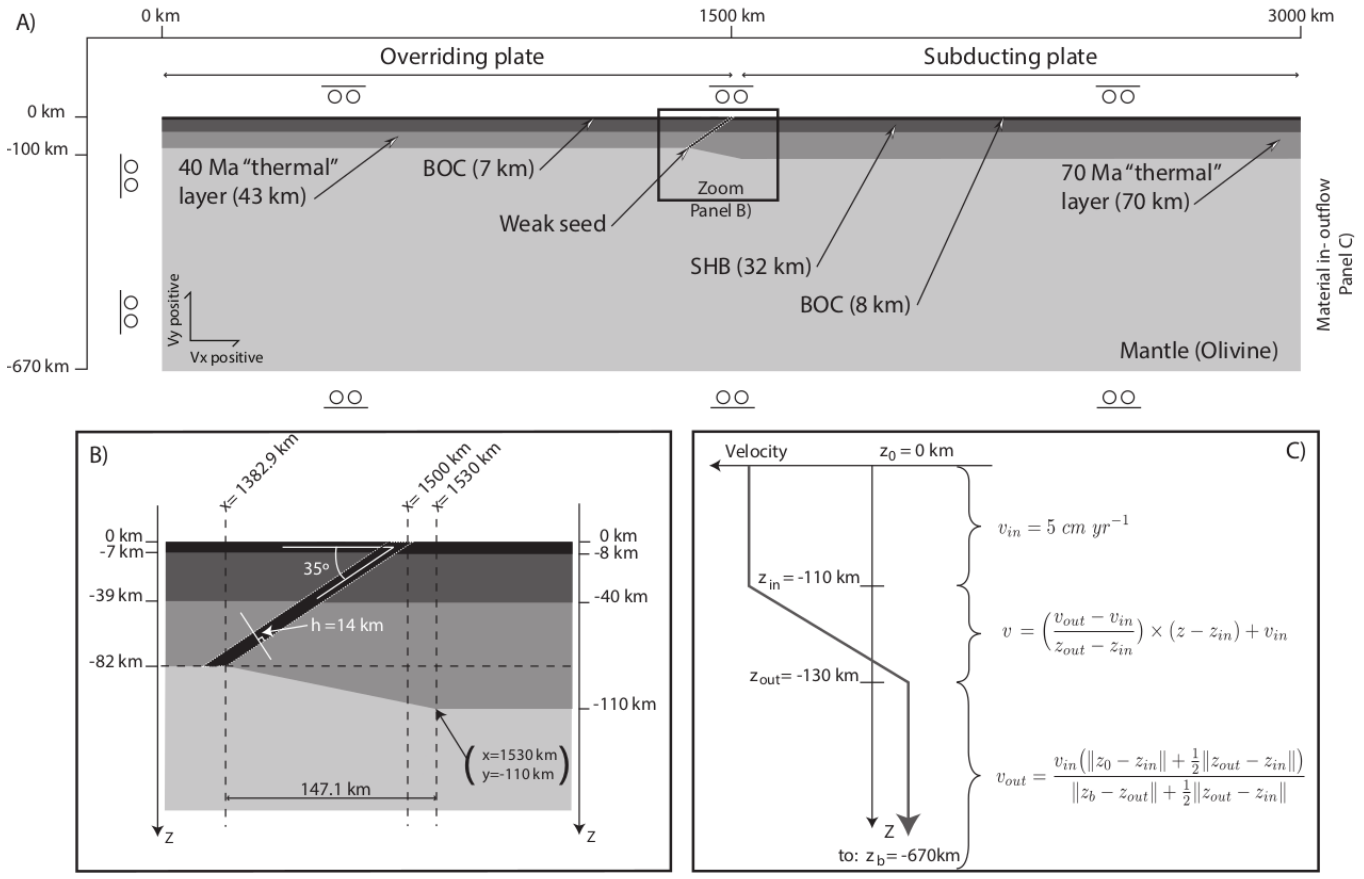


2D
box
P2pPm1
sparse_matrix
numerical_benchmark
stokes_equation
use_mesher
viscous_rheology
buoyancy_driven
vrms
pressure_scaling

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_62`

The setup is presented in the phd thesis (chapter 4) of M. Quinquis [254], In this stone we focus on the case 1 of his work and only solve the system once (single time step). The domain is 1500×670 km.

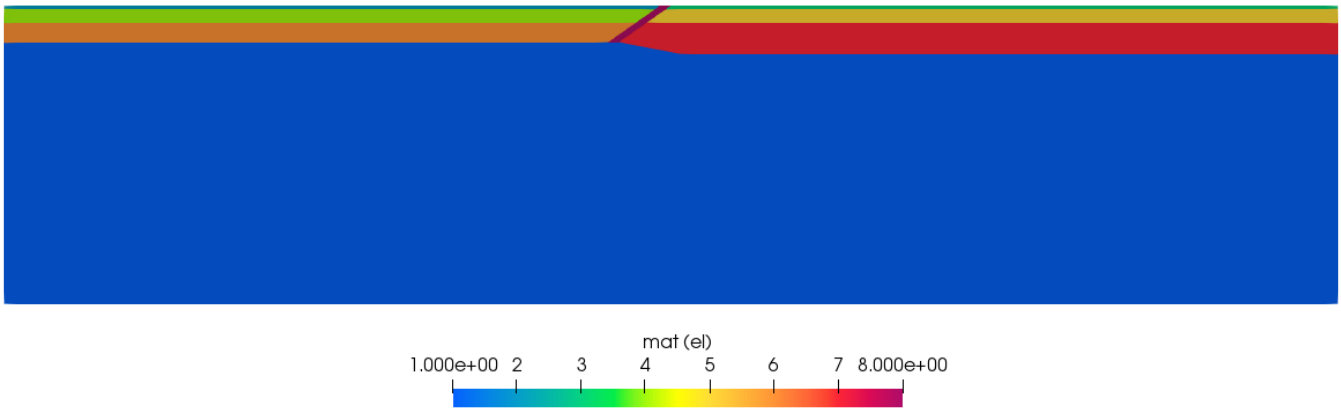
The setup and the boundary conditions are shown hereunder:



Model setup for subduction of a 70 Ma old oceanic plate under a 40 Ma old oceanic plate. A) Setup of the whole model. The top, bottom, and left boundaries are free-slip, while the right boundary condition includes material in- and outflow. S HB : Serpentinised Harzburgite, and B OC : Bulk Oceanic Composition. The 'thermal' layers are only required in the linear viscous models. B) Zoom of the setup at the trench, h is the thickness of the weak zone. C) Definition of the in- and outflow velocities on the right boundary. Taken from [254]

As opposed to the original benchmarking effort carried out by Quinquis and collaborators I build the mesh with triangles so that I can exactly represent the layers (and especially the weak zone) with an adequate resolution. The elements used are the Crouzeix-Raviart one (see Section 10.5.9).

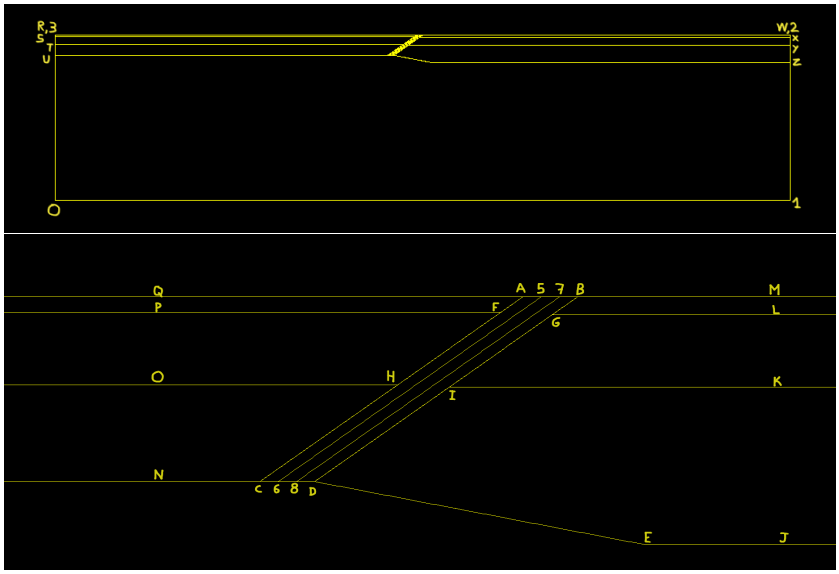
In points.py the coordinates of the key points are defined. The generate_nodes.py script generates the input file for the *triangle* program[1]. Finally the fieldstone.py script reads in the mesh, sets up the material layout, the boundary conditions and solves the FE system. Simply use the *run* script to trigger the mesh generation and the FE calculations. Note that the compiled *triangle* executable file (as compiled by you on your own machine) must be present in this folder.



Taken from [254]

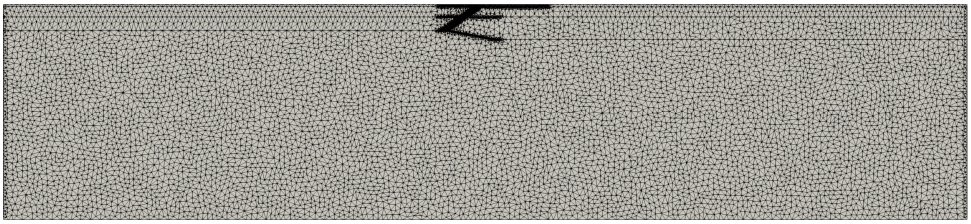
The letters referenced in the points.py file are shown hereunder:

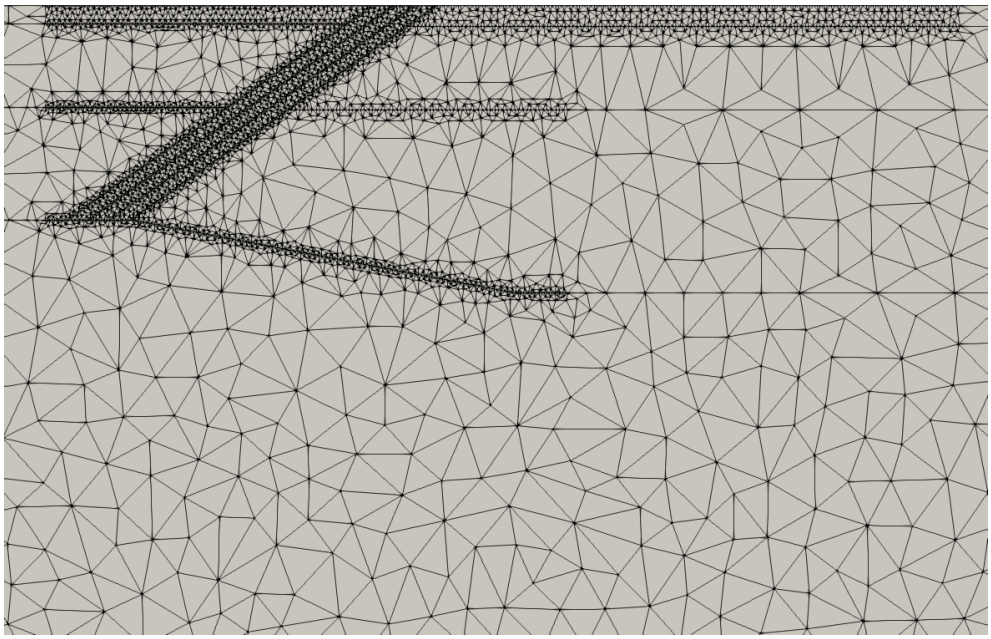
[1] `https://www.cs.cmu.edu/~quake/triangle.html`

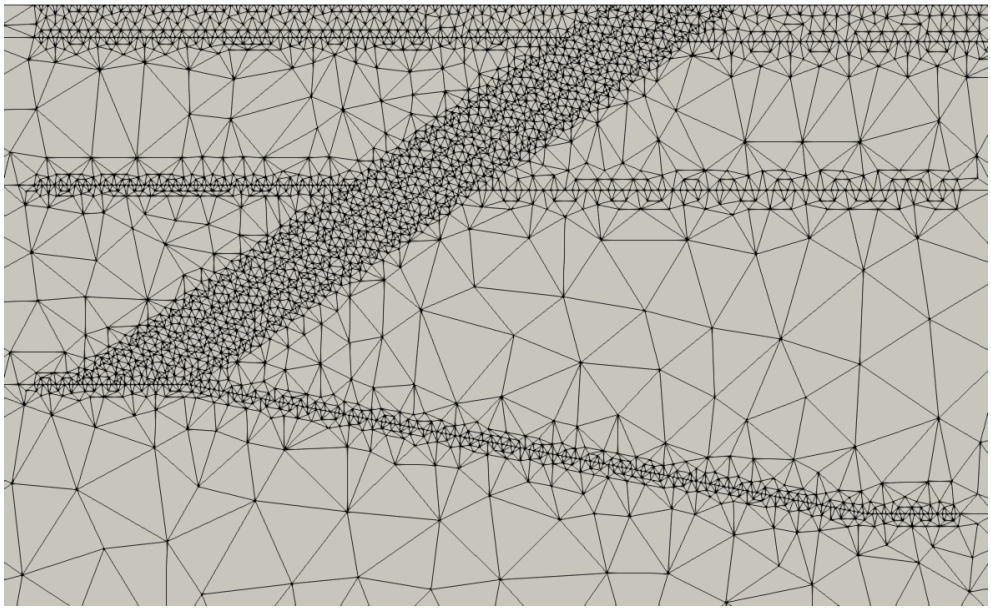


In this case 1, all materials are characterised by a constant density and a Newtonian rheology:

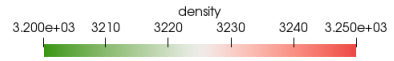
Lithology	Parameter	Unit	Value
Mantle	Density	kg m^{-3}	3200
	Viscosity	Pa s	10^{20}
Weak seed	Density	kg m^{-3}	3250
	Viscosity	Pa s	10^{20}
	Thickness	km	14
Overriding plate			
BOC	Density	kg m^{-3}	3250
	Viscosity	Pa s	10^{23}
	Thickness	km	7
SHB	Density	kg m^{-3}	3250
	Viscosity	Pa s	10^{23}
	Thickness	km	32
"Thermal layer"	Density	kg m^{-3}	3250
	Viscosity	Pa s	10^{23}
	Thickness	km	43
	Age	Ma	40
Subducting plate			
BOC	Density	kg m^{-3}	3250
	Viscosity	Pa s	10^{20}
	Thickness	km	8
SHB	Density	kg m^{-3}	3250
	Viscosity	Pa s	10^{23}
	Thickness	km	32
"Thermal layer"	Density	kg m^{-3}	3250
	Viscosity	Pa s	10^{23}
	Thickness	km	70
	Age	Ma	70



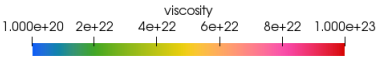


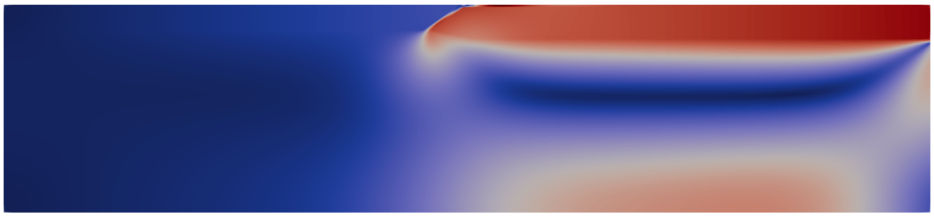


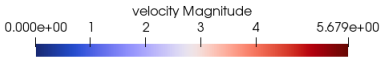


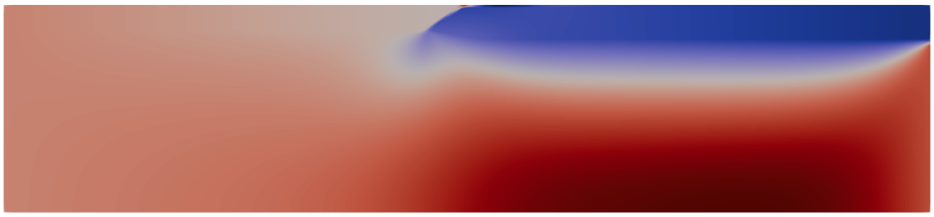

density
3.200e+03 3210 3220 3230 3240 3.250e+03

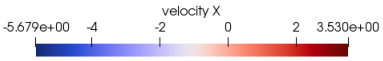


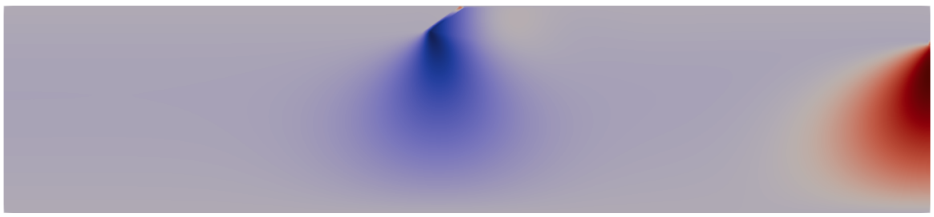

viscosity
1.000e+20 2e+22 4e+22 6e+22 8e+22 1.000e+23

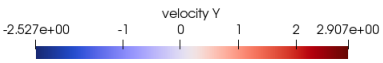


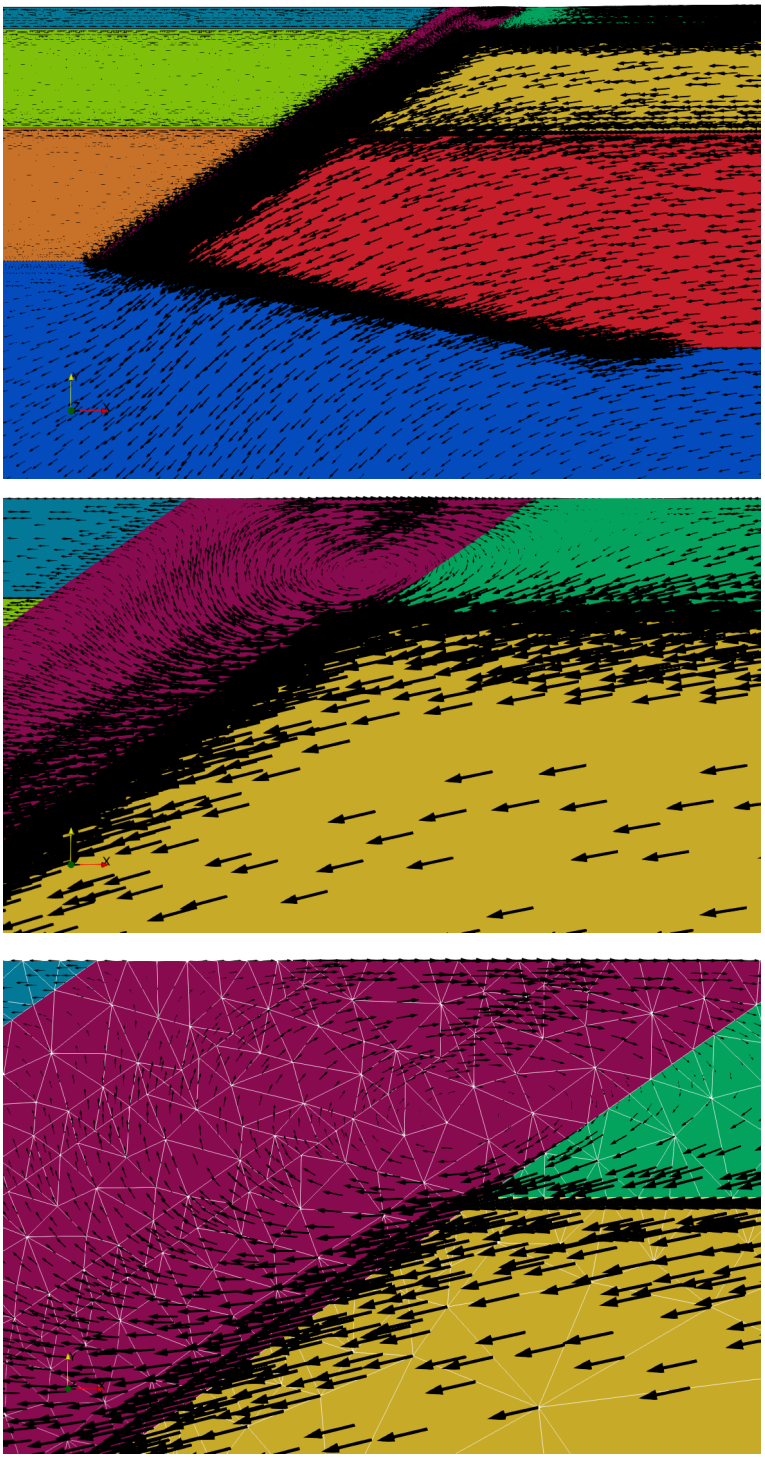

velocity Magnitude
0.000e+00 1 2 3 4 5.679e+00



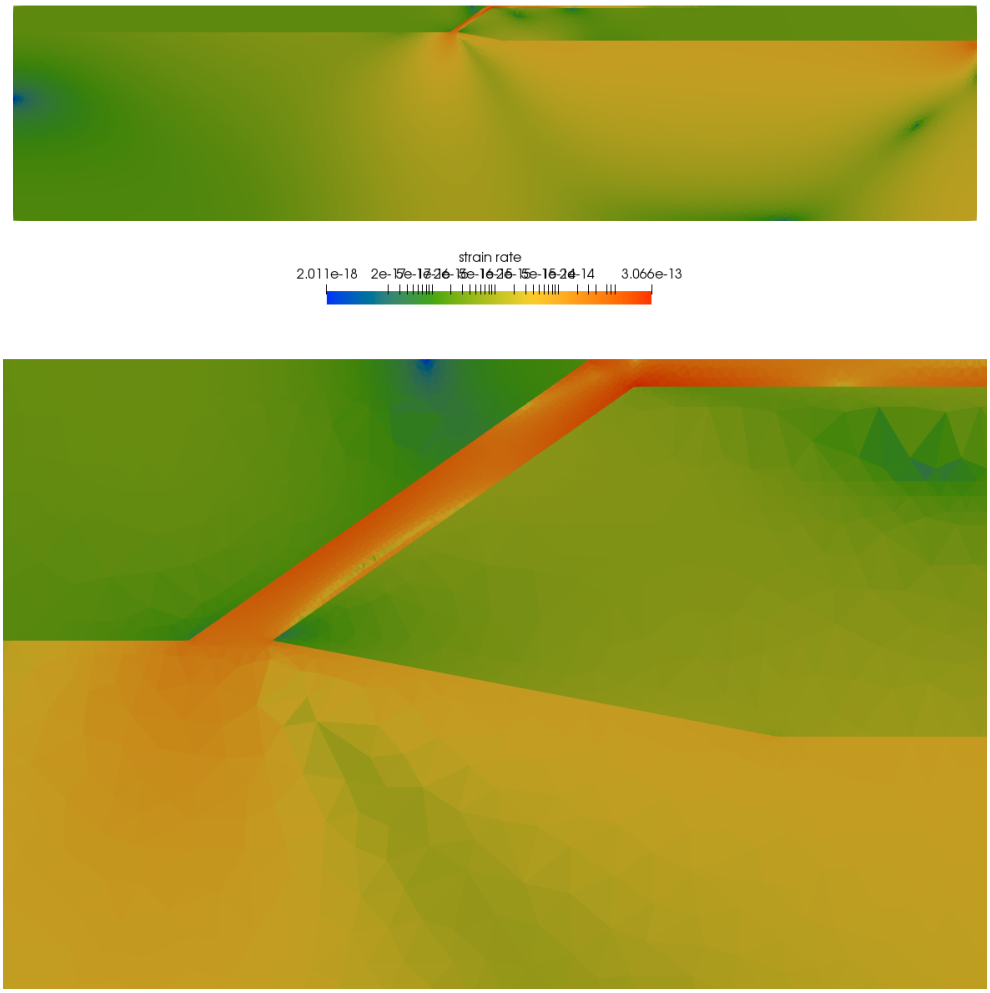

velocity X
-5.679e+00 -4 -2 0 2 3.530e+00




velocity Y
-2.527e+00 -1 0 1 2 2.907e+00



I believe that one source of our problems when we carried out resolution tests came from the fact that this small convection cell that occurs as shown above was always under-resolved. The top layer is 8km thick and the weak zone is 14km thick, and we ran simulations at max 1km resolution. However, because of the use of particle-in-cell and rectangular elements (mostly linear elements), a proper capture of this feature in the solution would have required much higher resolution. In other words, our higher resolution runs were probably not high resolution enough. Given the nature of this convection cell, it means that temperature would be advected by it, and that markers would likely gotten mixed.


strain rate
2.011e-18
3.066e-13

Stone 63

failure in cemented granular material



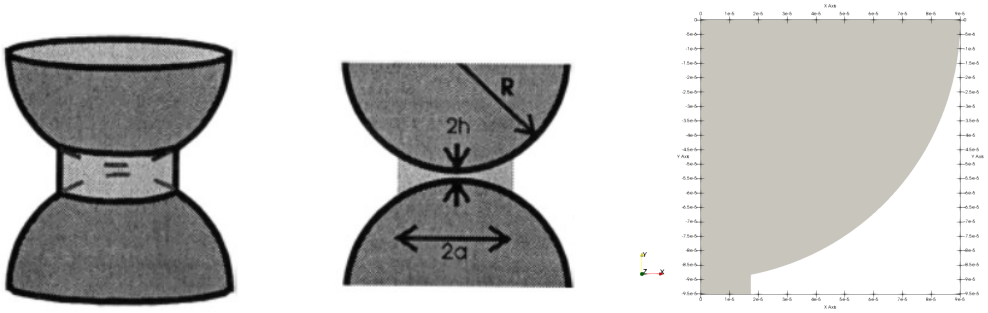
2D
P1
sparse_matrix
compressible
numerical_benchmark
stokes_equation
elastic_rheology
residual
paraview_output
displacement_code

This stone was developed in collaboration with Taka Shinohara.

This STONE is based on Wong and Wu [353] (1995) published in Geophysical Research Letters. As for every STONE aiming at reproducing results off a publication I here include de abstract of the article:

> Grain crushing and pore collapse are important micromechanical processes responsible for hydrostatic and shear-enhanced compactions in porous rocks. These processes initiate from extensile microcracks which emanate from grain contacts. Microstructural observations indicate that such extensile cracking is inhibited in the vicinity of cemented grain contacts. The finite element technique was used to simulate the tensile stress concentration and normal stiffness in a cemented aggregate. The detrital grains were assumed to be elastically identical spheres bonded by cement layers of finite thickness. The numerical simulations show that the maximum tensile stress concentration is located near the triple junction (among grain, cement and pore space), and its magnitude is significantly less than that for an uncemented system. The development of microcracking near a cemented contact is readily inhibited unless the applied stress exceeds a critical value which is at least an order of magnitude greater than that for the onset of Hertzian fracture.

In this paper the authors use the finite element method to simulate the tensile stress concentration and normal stiffness in a cemented aggregate. The grains are assumed to be elastically identical spheres bonded by cement layers of finite thickness.



Left: model of cemented granular system for axisymmetric finite element analysis (taken from [353]); Right: Domain for a_fac=0.2 and h_fac=5/90 and $R = 90\mu m$.

We define

$$a_{fac} = \frac{a}{R} \qquad \text{and} \qquad h_{fac} = \frac{h}{R}$$

which translates as follows in the code:

```
outer_radius=90e-6        #radius R (m)
a_fac = 20/90             # ratio of a to R
h_fac = 5/90              # ratio of h to R
a = outer_radius*a_fac
h = outer_radius*h_fac
```

The material properties for the grain and the cement are as follows:

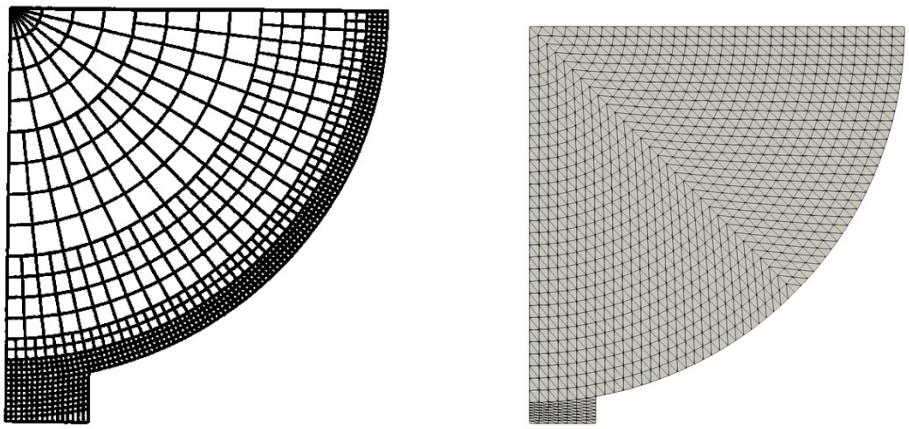
```
#Quartz mechanical properties
E1 = 95e9                          # Young's modulus (Pa)
nu1=0.08                           # poisson ratio
mu1= E1/(2*(1+nu1))                # shear modulus
lambdaa1=2.*mu1*nu1/(1.-2.*nu1)
```

and

```
#cement mechanical properties 2
E2 = 84e9                          # Young's modulus (Pa)
nu2=0.31                           # poisson ratio
mu2= E2/(2*(1+nu2))                # shear modulus
lambdaa2=2.*mu2*nu2/(1.-2.*nu2)
```

Note that since gravitational effects are neglected we set $\vec{g} = \vec{0}$ so that density actually play no role in the model.

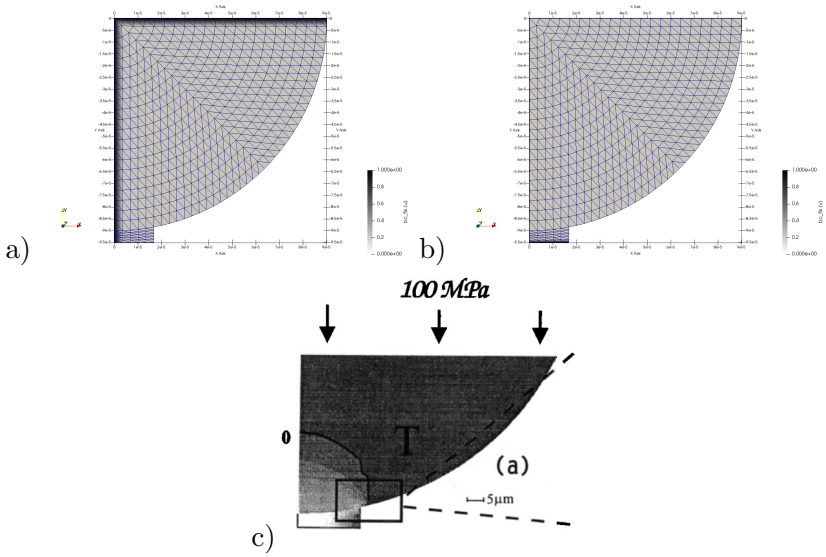
The mesh in the code is built in three phases: first, the quarter grain is meshed with triangles. Second, the cement block is meshed and shaped so as to follow the shape of the grain. Finally both meshes are merged together:



Left: quadtree-based mesh used in [353]; Right: low resolution fieldstone mesh.

Boundary conditions are as follows: $u = 0$ is prescribed on the left and top boundary, while $v = 0$ is prescribed on the bottom boundary, as shown below. Additionally, traction b.c. are prescribed at the top[1] (these are described in Section 10.15) as indicated hereunder:

[1] Note that in the axisymmetric case these then apply onto a disc and should be multiplied by 2π



a,b) Horizontal and vertical displacement boundary condition indicator. c) Imposed traction at the top surface (Taken from [353]).

Linear triangular elements are used and a 3-point numerical quadrature is used. Average elemental quantities are computed with 7 quadrature points. These same quantities are also projected onto the nodes for visualisation purposes. Principal stresses are computed as explained in Section 4.2.

The code solves the FE system and retrieves the deformation components x and y, or r and z in the axisymmetric case. Average elemental strain components are then computed. In the axisymmetric case then $\varepsilon_{\theta\theta}$ is computed and otherwise set to zero in the plane strain case. The stress tensor components are then computed as follows:

$$\sigma_{xx} = \lambda\vec{\nabla}\cdot\vec{u} + 2\mu\varepsilon_{xx} \tag{63.1}$$

$$\sigma_{zz} = \lambda\vec{\nabla}\cdot\vec{u} + 2\mu\varepsilon_{zz} \tag{63.2}$$

$$\sigma_{xz} = 2\mu\varepsilon_{xz} \tag{63.3}$$

where $\vec{\nabla}\cdot\vec{u} = \frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}$ in plane strain and $\vec{\nabla}\cdot\vec{u} = \frac{\partial u}{\partial x} + \frac{u}{x} + \frac{\partial w}{\partial z}$ in the axisymmetric case[2].

Principal stresses are such that $\sigma_1 > \sigma_2$ and

$$\sigma_{1,2} = \frac{\sigma_{xx}+\sigma_{yy}}{2} \pm \sqrt{\left(\frac{\sigma_{xx}-\sigma_{yy}}{2}\right)^2 + \sigma_{xy}^2}$$

The principal direction angle θ_p defines the principal directions where the only stresses are normal stresses, and is given by the relationship:

$$\tan(2\theta_p) = \frac{2\sigma_{xy}}{\sigma_{xx}-\sigma_{yy}}$$

We find that θ_2 is always negative, while θ_1 showcases positive and negative values. We define the tensile zone as $\sigma_1 > 0$.

In general σ_1 is the maximum (most tensile) principal stress, σ_3 is the minimum (most compressive) principal stress, and σ_2 is the intermediate principal stress.

The maximum shear stress (in 2D) is given by

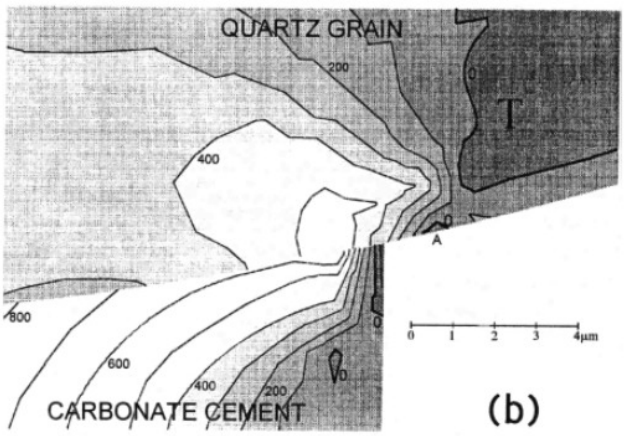
$$\tau_{max} = \frac{1}{2}(\sigma_1 - \sigma_2)$$

In the axisymmetric case the stress tensor is given by

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{rr} & 0 & \sigma_{rz} \\ 0 & \sigma_{\theta\theta} & 0 \\ \sigma_{rz} & 0 & \sigma_{zz} \end{pmatrix}$$

There are now three eigenvalues. However it is obvious that $\sigma_{\theta\theta}$ is one of them and the other two correspond to those of the 2×2 stress stensor with σ_{rr}, σ_{rz} and σ_{zz} terms. The code therefore needs not be modified to compute these.

[2]Remember that the xz plane is in fact the rz plane so that the term u_r/r becomes u/x in practice.

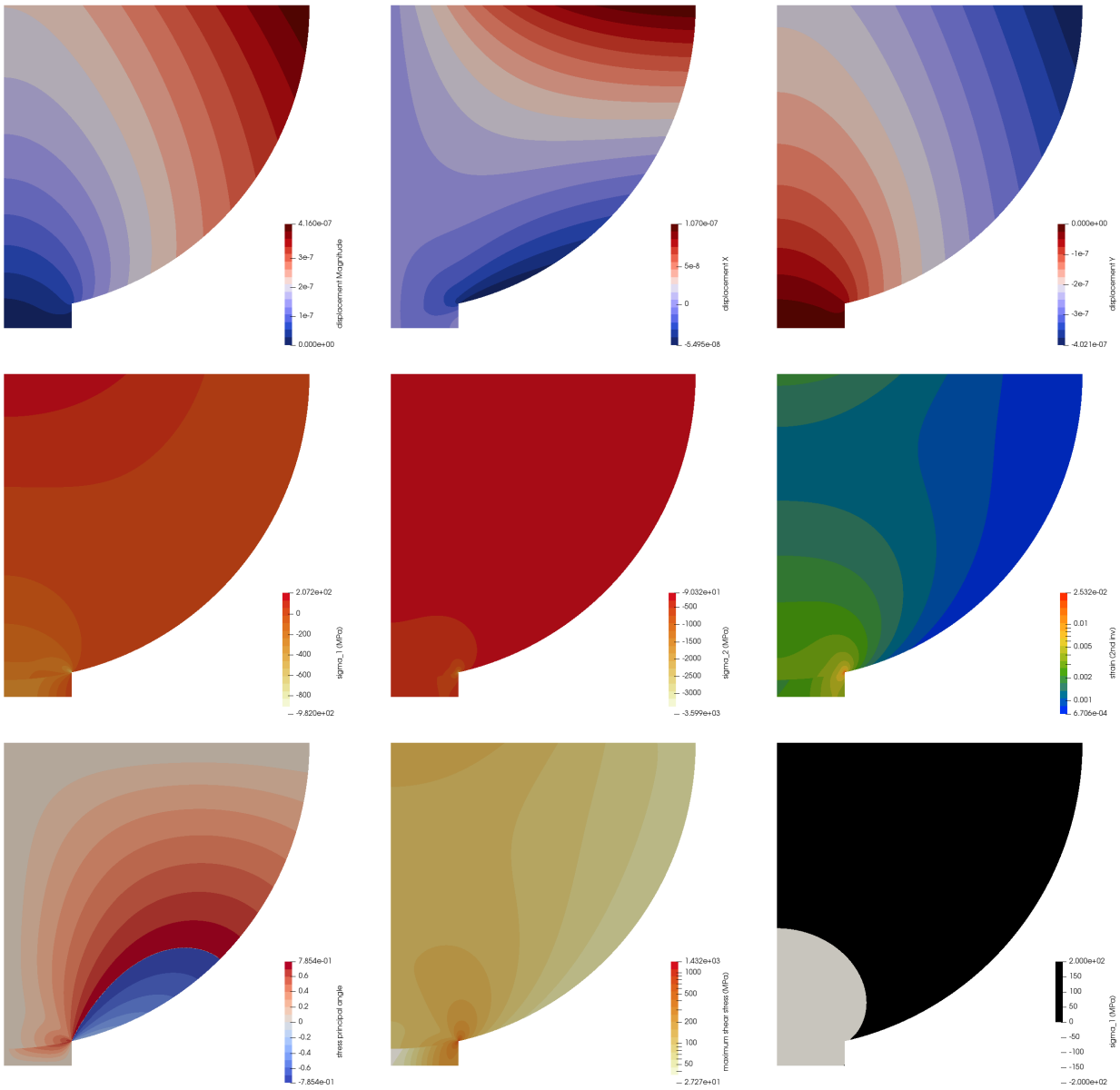


Contours for the minimum principal stress in the vicinity of the triple junction

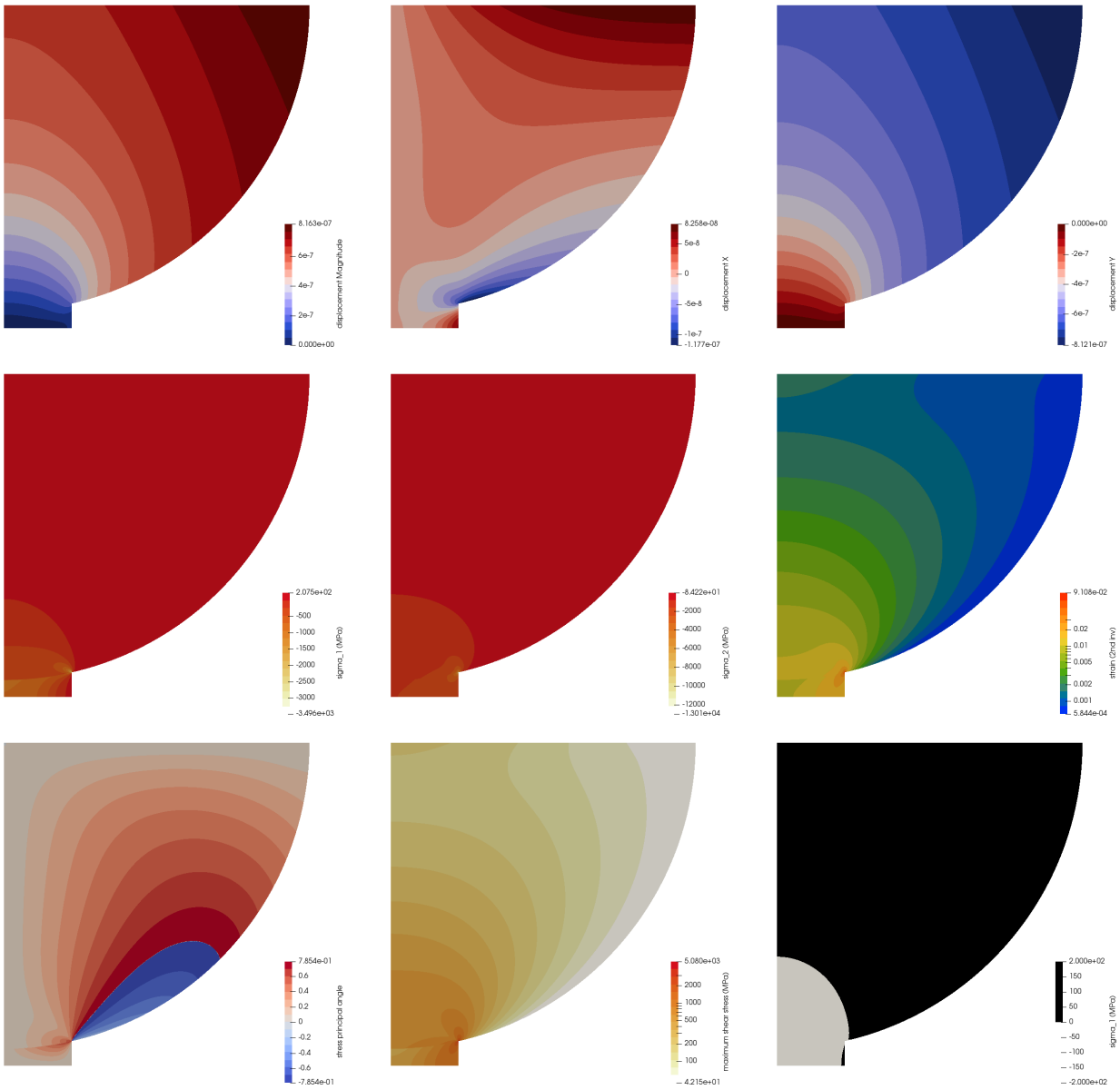
Remark. *This paper by Wong and Wu is a nice example of not-reproducible science. For one, boundary conditions are not all specified. Second, it is not specified whether the results are obtained in plane strain or axisymmetric geometry. Third, the program that was used is proprietary (ABAQUS). Fourth, no resolution tests are presented: given how small the area A is, and how comparable in size it is to the elements, this is problematic. Fifth, it is not clear how the tensile zone is defined. Sixth: the type of finite element is not specified. Seven, we are shown one field which is a derivative of the solution (displacement) but not the solution itelf.*

The following results are obtained with very high resolutions, i.e. nLayers=901, nel_h=80. This yields a mesh with about 1.6M triangles, and Nfem=1,668,488.

Results in plane strain



Results in axisymmetric geometry



Stone 64

Elasto-viscous benchmarks

```
2D
box
Q2Q1
sparse_matrix
numerical_benchmark
stokes_equation
time_stepping
elastic_rheology
viscous_rheology
buoyancy_driven
vrms
paraview_output
PIC
pressure_scaling
pressure_normalisation
free_surface
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_64`

DO NOT READ WHAT FOLLOWS. It was written for Composition based stone, but I am using markers for all benchmarks so far. Jump directly to 'stress build up in maxwell body'!

In the examples here there are only 2 compositions. Each material/composition is characterised by its $\eta_{\{1,2\}}$, $\mu_{\{1,2\}}$. With δt chosen, $Z_{\{1,2\}}$ and $\eta_{eff,\{1,2\}}$ can be computed.

C1 and C2 are initialised at startup by looping over Vnodes and assigning a value to C_1 and C_2 such that $C_1 + C_2 = 1$.

The quantity $\tau_{xx}, \tau_{yy}, \tau_{xy}, \dot{\omega}_{xy}$ (denoted by oxy in the code) are nodal and initialised at startup (they are needed for the Stokes matrix):

```
tauxx =np.zeros(NV,dtype=np.float64)
tauyy =np.zeros(NV,dtype=np.float64)
tauxy =np.zeros(NV,dtype=np.float64)
oxy   =np.zeros(NV,dtype=np.float64)
etaeff=np.zeros(NV,dtype=np.float64)
Z     =np.zeros(NV,dtype=np.float64)
rho   =np.zeros(NV,dtype=np.float64)
```

Structure of the code (the following bulletpoints are inside the time stepping loop):

- Nodal η_{eff}, Z and ρ values are computed as follows:

```
for i in range(0,NV):
    etaeff[i]=C1[i]*etaeff1+C2[i]*etaeff2
    Z[i]     =C1[i]*Z1     +C2[i]*Z2
    rho[i]   =C1[i]*rho1   +C2[i]*rho2
```

- Build Stokes matrix. Loop over elements, loop over integration points inside the element, and for each quadrature point:

$$
\begin{aligned}
Z(\vec{r}_q) &= \sum_{i=1}^{m_V} N_i^{\mathbb{V}}(\vec{r}_q) Z_i && (64.1)\\
\tau_{xx}(\vec{r}_q) &= \sum_{i=1}^{m_V} N_i^{\mathbb{V}}(\vec{r}_q) \tau_{xx,i} && (64.2)\\
\tau_{yy}(\vec{r}_q) &= \sum_{i=1}^{m_V} N_i^{\mathbb{V}}(\vec{r}_q) \tau_{yy,i} && (64.3)\\
\tau_{xy}(\vec{r}_q) &= \sum_{i=1}^{m_V} N_i^{\mathbb{V}}(\vec{r}_q) \tau_{xy,i} && (64.4)\\
\eta_{eff}(\vec{r}_q) &= \sum_{i=1}^{m_V} N_i^{\mathbb{V}}(\vec{r}_q) \eta_{eff,i} && (64.5)\\
\dot{\omega}_{xy}(\vec{r}_q) &= \sum_{i=1}^{m_V} N_i^{\mathbb{V}}(\vec{r}_q) \dot{\omega}_{xy,i} && (64.6)\\
\rho(\vec{r}_q) &= \sum_{i=1}^{m_V} N_i^{\mathbb{V}}(\vec{r}_q) \rho_i && (64.7)
\end{aligned}
$$

  The elastic memory rhs term is built following Eq. (**??**):

```
R[0]=Zq*(tauxxq+dt*oxyq*(2*tauxyq))
R[1]=Zq*(tauyyq+dt*oxyq*(-2*tauxyq))
R[2]=Zq*(tauxyq+dt*oxyq*(tauyyq-tauxxq))
f_el-=b_mat.T.dot(R)*weightq*jcob
```

- Solve system and obtain u, v, p
- Compute nodal velocity gradient $\boldsymbol{L} = \vec{\nabla}\vec{\mathrm{v}}$
- Compute derivated nodal fields

```
exx[:]=Lxx[:]
eyy[:]=Lyy[:]
exy[:]=0.5*(Lxy[:]+Lyx[:])
oxy[:]=0.5*(Lxy[:]-Lyx[:])
Jxx[:]=2*tauxx[:]*oxy[:]
Jyy[:]=-2*tauxy[:]*oxy[:]
Jxy[:]=(tauyy[:]-tauxx[:])*oxy[:]
```

- (re-)build nodal $\boldsymbol{\tau}$:

```
tauxx=2*etaeff*exx+Z*tauxx+Z*dt*Jxx
tauyy=2*etaeff*eyy+Z*tauyy+Z*dt*Jyy
tauxy=2*etaeff*exy+Z*tauxy+Z*dt*Jxy
```

- advect C_1, C_2, τ_{xx}, τ_{yy} and τ_{xy} fields
- produce vtu file

About this code This is a fairly complicated code since it features a deformable mesh (by means of a simplified ALE which only allows for the vertical movement of only the top row of mesh nodes), a particle-in-cell technique (only 1st order in space) to carry out material tracking/advection and of course the elasto-viscous rheology.

Stress build-up in viscoelastic Maxwell body - analytical solution We start from

$$\dot{\boldsymbol{\varepsilon}}_T = \dot{\boldsymbol{\varepsilon}}_e + \dot{\boldsymbol{\varepsilon}}_v = \frac{1}{2\mu}\dot{\boldsymbol{\tau}} + \frac{1}{2\eta}\boldsymbol{\tau}$$

In the case where there is no rotation then the BLABLA derivative becomes d/dt and then we have to solve

$$2\mu\dot{\boldsymbol{\varepsilon}}_T = \frac{d\boldsymbol{\tau}}{dt} + \frac{\mu}{\eta}\boldsymbol{\tau} = \frac{d\boldsymbol{\tau}}{dt} + \frac{\boldsymbol{\tau}}{t_M}$$

where $t_M = \eta/mu$ is the Maxwell time. The general solution can be arrived at by means of the Laplace transform (?!) and is given by:

$$\boldsymbol{\tau}(t) = \boldsymbol{\tau}(t_0)\exp\left(-\frac{t-t_0}{t_M}\right) + \exp\left(-\frac{t}{t_M}\right)\int_{t_0}^{t} 2\mu\dot{\boldsymbol{\varepsilon}}_T \exp\left(\frac{t'}{t_M}\right)dt'$$

If $t_0 =$ and $\boldsymbol{\tau}(t_0) = 0$ then

$$\boldsymbol{\tau}(t) = \exp\left(-\frac{t}{t_M}\right)\int_{0}^{t} 2\mu\dot{\boldsymbol{\varepsilon}}_T \exp\left(\frac{t'}{t_M}\right)dt'$$

If the strain rate and shear modulus are constant in time, then

$$\begin{aligned}
\boldsymbol{\tau}(t) &= \exp\left(-\frac{t}{t_M}\right) 2\mu\dot{\boldsymbol{\varepsilon}}_T \int_0^t \exp\left(\frac{t'}{t_M}\right)dt' \\
&= \exp\left(-\frac{t}{t_M}\right) 2\mu\dot{\boldsymbol{\varepsilon}}_T t_M \left[\exp\left(\frac{t}{t_M}\right) - 1\right] \\
&= 2\eta\dot{\boldsymbol{\varepsilon}}_T \left[1 - \exp\left(-\frac{t}{t_M}\right)\right]
\end{aligned}$$

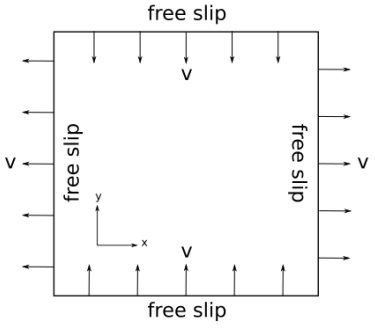
since $t_M = \eta/\mu$.

Stress build-up in viscoelastic Maxwell body under simple shear The first benchmark performed to test the viscoelastic implementation considers the stress build-up present in a viscoelastic Maxwell body. Contrary to stressed viscous materials, viscoelastic materials gradually build-up stress when sheared after which a transition to viscous deformation occurs.

An unstressed, incompressible viscoelastic Maxwell medium is subjected to a velocity field resulting in pure shear. The increase of the accumulated stress with time is given by an analytical solution:

$$\boldsymbol{\tau} = 2\eta\ \dot{\boldsymbol{\varepsilon}}\left(1 - e^{-\frac{\mu}{\eta}t}\right) \tag{64.8}$$

with t time, η the prescribed material viscosity and μ the prescribed material shear modulus. The domain size is 100×100km. The velocity prescribed at all boundaries equals $v = 1$ cm/yr in magnitude yielding a constant background strain rate of $\dot{\varepsilon} = 2\text{cm/yr}/100\text{km} \simeq 6.342 \times 10^{-15}$. The viscosity is $\eta = 10^{21}$Pa.s, the shear modulus is $\mu = 10^{10}$Pa and the gravity is set to zero. We set $\delta t = 100$yr.



Set up of the stress build-up benchmark. All domain sides have a free slip boundary condition, and pure shear velocity conditions are prescribed. Adapted from Gerya (2010) [128].

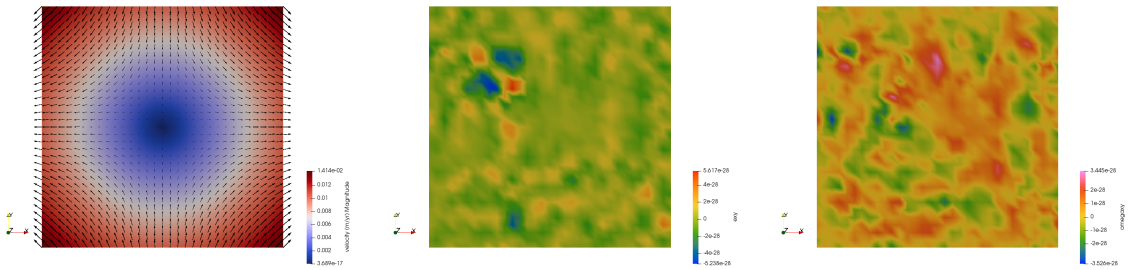
We have

$$\eta_{eff} = \frac{\eta \delta t}{\delta t + \eta/\mu} = \frac{10^{21} \cdot 3.154 \times 10^9}{3.154 \times 10^9 + 10^{21}/10^{10}} \simeq 3.0592 \times 10^{19} \text{Pa.s} \qquad \text{and} \qquad Z = \frac{\eta_{eff}}{\mu \delta t} \simeq 0.9694$$

The Maxwell time is $t_M = \frac{\eta}{\mu} = 10^{11}\text{s} \simeq 3171\text{yr}$. In the absence of elasticity (purely viscous behaviour), we have $\dot{\varepsilon}_{xx} = 6.342 \times 10^{-15}$ and $\eta = 10^{21}$ so the deviatoric stress τ_{xx} is equal to

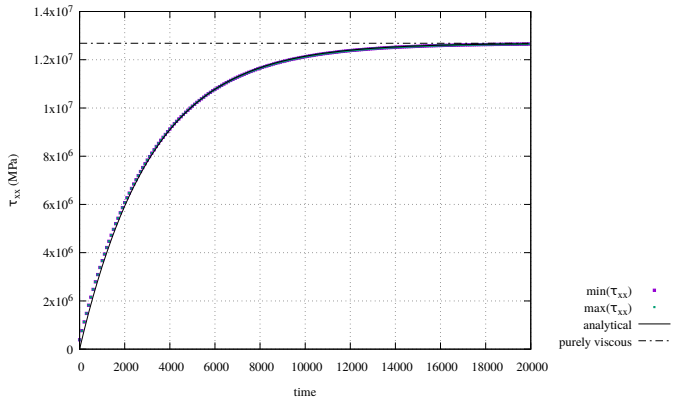
$$\tau_{xx} = 2 \cdot 10^{21} \cdot 6.342 \times 10^{-15} \simeq 12.68 \times 10^6 \text{Pa}$$

The first time that the Stokes system is solved, there is no stored stress, i.e. the elastic rhs is identically zero, so that the system is solved with a viscosity equal to η_{eff}. We can easily compute the analytical solution, and we see that $\dot{\varepsilon}_{xy} = 0$ and $\dot{\omega}_{xy} = 0$, which we recover:

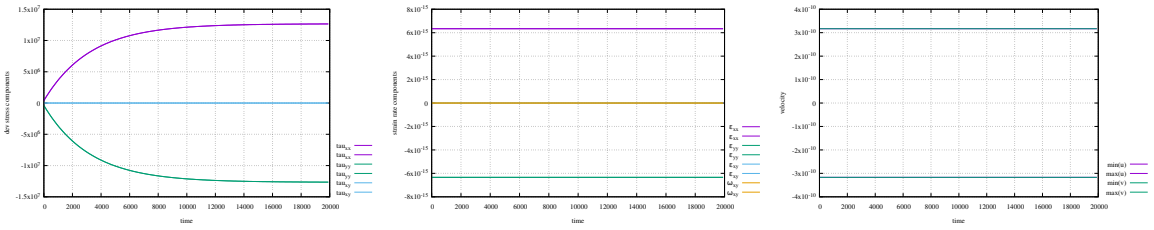


The expected stress value for τ_{xx} after the first Stokes solve is

$$\tau_{xx} = 2\eta_{eff}\dot{\varepsilon}_{xx} = 2 \cdot 3.057 \times 10^{19} \cdot 6.342 \times 10^{-15} \simeq 38.775 \times 10^4 \text{Pa}$$



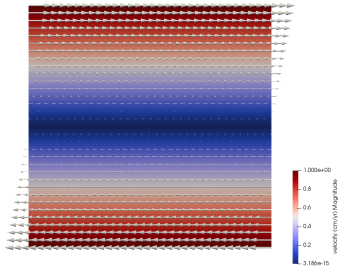
τ_{xx} as a function of time.



Remark. *Because of the outflux boundary conditions it can happen that some markers exit the domain. In order to avoid dealing with these, markers are artificially kept inside the domain (near the boundary).*

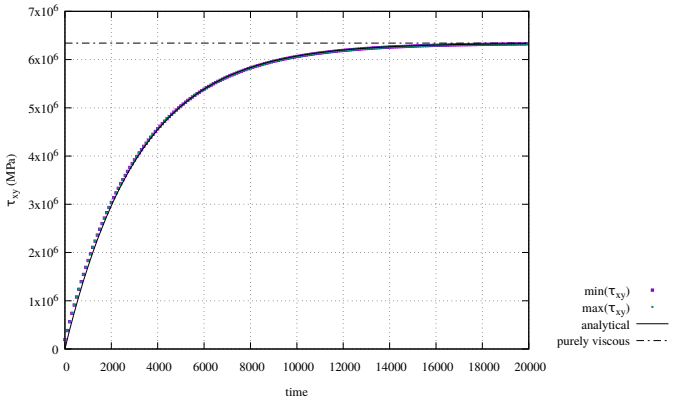
Stress build-up in viscoelastic Maxwell body under simple shear I have also created a similar problem, although this time simple shear boundary conditions are prescribed. $v = 0$ on left and right boundary, (-1,0)cm/yr at the bottom, (+1,0)cm/yr at the top.

ANALYTICAL SOLUTION!?! $\tau_{xy} \to 2\eta\dot{\varepsilon}_x y = 6.342 \times 10^6 Pa$

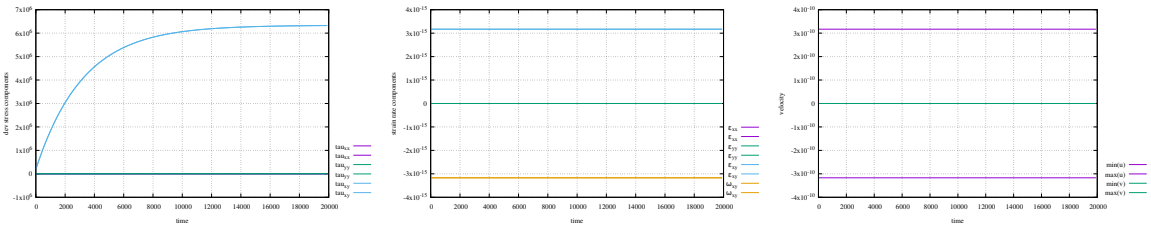


In simple shear, we have $\dot{\varepsilon}_{xx} = \dot{\varepsilon}_{yy} = 0$, and

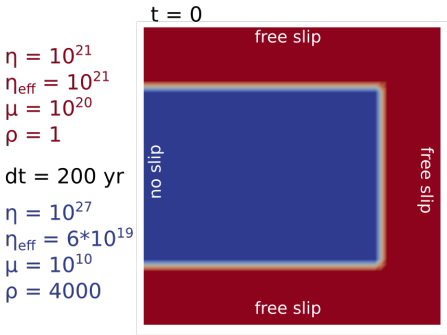
$$\dot{\varepsilon}_{xy} = \frac{1}{2}\frac{\Delta u}{L_y} = \frac{1cm/yr}{100km}$$



τ_{xy} as a function of time.



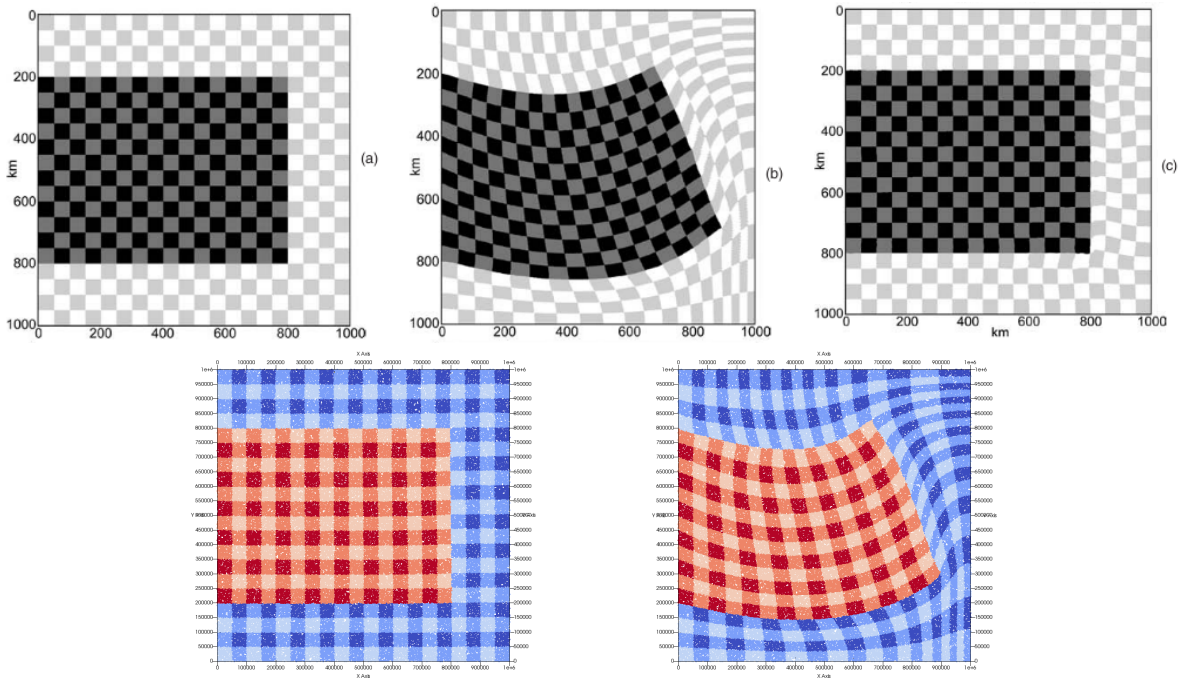
Bending of elastic slab The sinking slab benchmark consists of a beam of elastic material which is placed in a weak and viscous surrounding medium. The initially unstressed beam is attached to the left domain boundary through boundary conditions. A stress is then applied to the beam in the form of gravity. The applied gravity force results in the deformation of the beam through bending. After 20 kyr, the gravity field is turned off and the elastic properties of the beam will then force itself to its original position. The set-up of the benchmark is given in the following figure:



Set-up of the benchmark from [128]. The properties of the two materials are given on the left, together with the initial configuration of the benchmark.

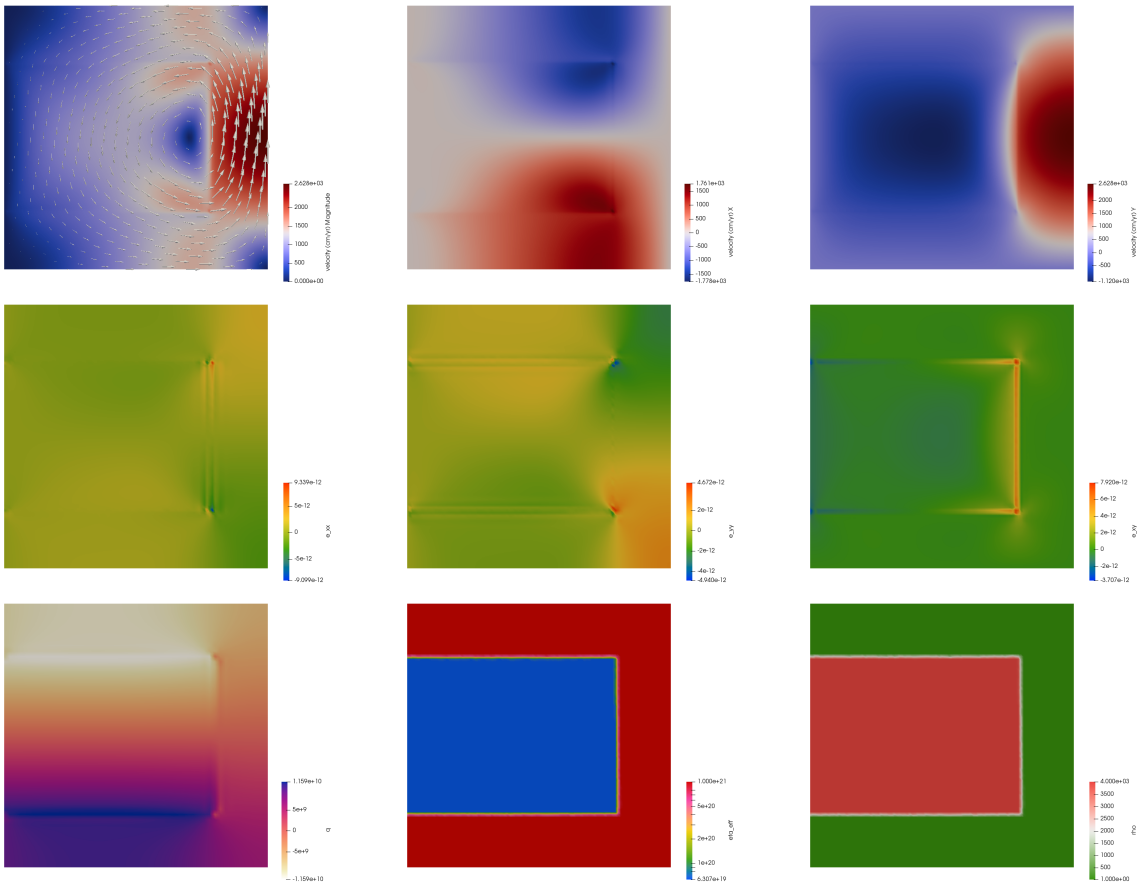
The beam is surrounded by a low-density, low-viscosity and high shear modulus medium of which the specifications are given in the following table. The boundary conditions of the domain consist of a no slip condition at the left boundary where the slab is attached and free slip boundary conditions along all other sides. The results are calculated on a grid with a resolution of 50x50 elements containing 64 randomly distributed markers at startup. The time step is set to $\delta t = 200yr$ (i.e. gravity is switched off after 100 time steps).

Material properties	*Elastic slab (fluid 1)*	*Surrounding medium (fluid 2)*
Density ρ [kg/m^3]	4000	1
Viscosity η [Pa· s]	10^{27}	10^{21}
Shear modulus μ [Pa]	10^{10}	10^{20}
Maxwell time t_M [yr]	3.17Gyr	3.17×10^{-7}yr
eff. visc. η_{eff} [Pa·s]	6.307199602192306e+19	9.999999984145105e+20
visco-elasticity factor Z [-]	0.9999999369280039	1.5854895966744522e-09

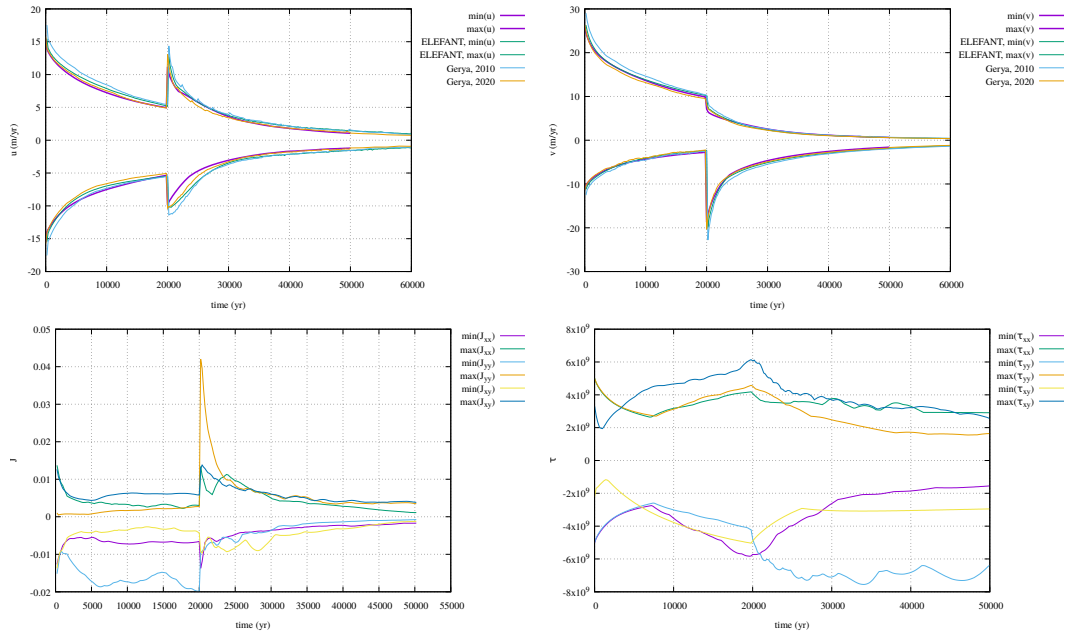


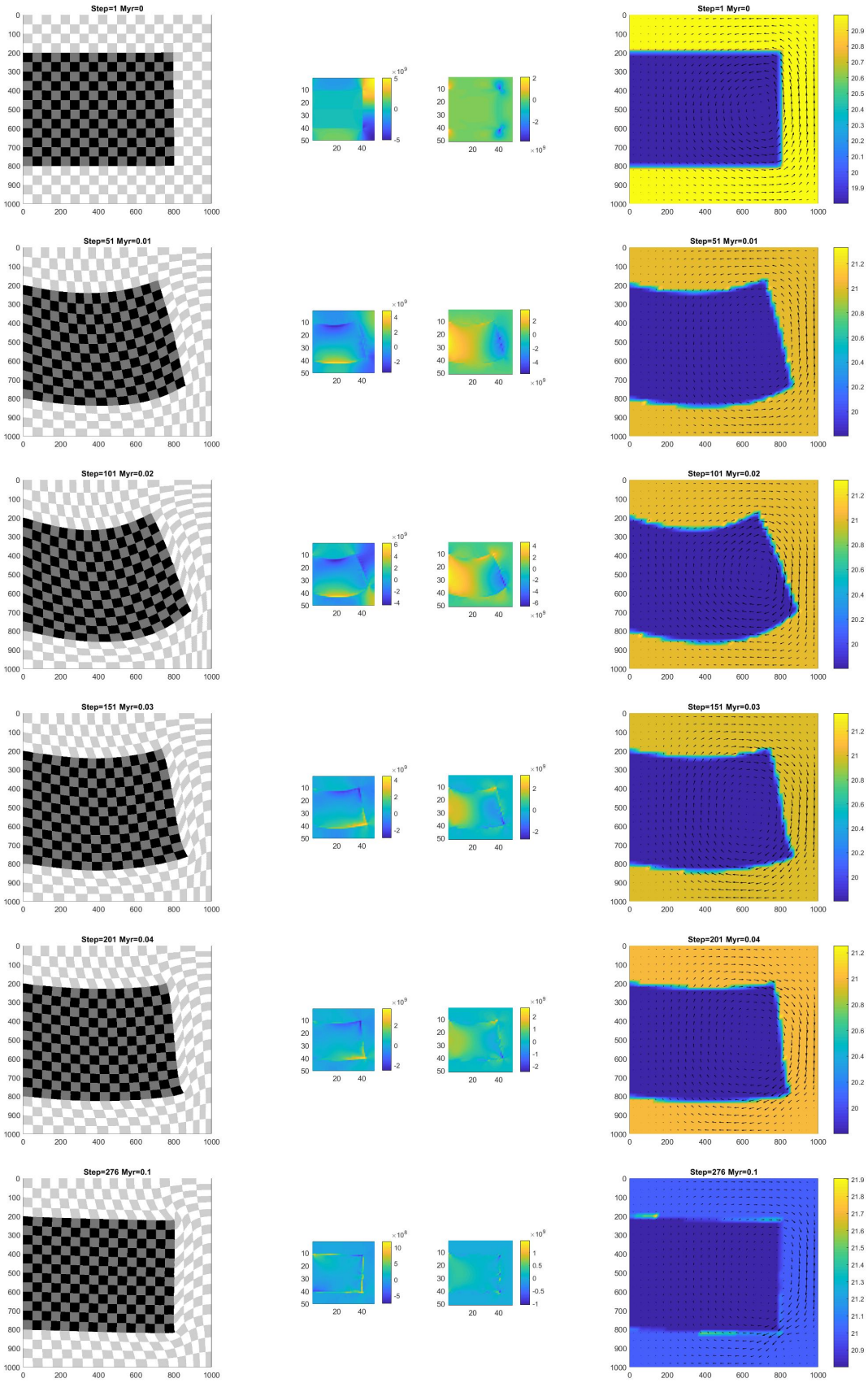
Top row taken from [128, p. 16.11]. Results of a numerical experiment for the recovery of the original shape of a visco-elastic slab (black, dark grey), embedded in a weak visco-elastic medium (light grey, white). (a) Initial configuration, (b) configuration after 20 Kyr of deformation under constant vertical gravity field ($g_x = 0$,$g_y = -10$m/s^2, (c) configuration achieved within 9980 Kyr of spontaneous deformation after switching off gravity (i.e. after $g_x = g_z = 0$ condition is applied at 20 Kyr). Numerical results are calculated at a resolution 51×51 nodes and 200×200 markers. Note the irreversible viscous deformation of the weak surrounding medium, which is visible in its perturbed checkerboard structure close to slab corners in (c).

The first time that the Stokes system is solved, there is no stored stress, i.e. the elastic rhs is identically zero, and the system is solved with a viscosity equal to $\eta_{eff,1}$ and $\eta_{eff,2}$ for the slab and mantle respectively:



Note that the Gerya data are obtained with the code from the 2010 version of the book.





Results obtained with the Matlab code provided with Gerya (2019) [129]

Stress build-up in viscoelastic Maxwell body This benchmark comes from Appendix B of Keller et al (2013) [183].

The domain is 7.5×5km. A dominantly elastic beam is fixed to, and protrudes horizontally from the left wall of the model box. Surrounding the elastic beam is a viscous, but inelastic fluid. All boundaries are free slip, except for the left wall, which is set to no slip in order to keep the bending beam fixed to the wall. The beam has a higher density than the surrounding fluid and thus will bend down elastically driven by gravity. After the beam has accumulated some elastic strain through bending down, gravity is switched off. If the stress evolution is implemented accurately, the elastic beam should now, free from the pull of gravity, move upwards again and restore its initial position.

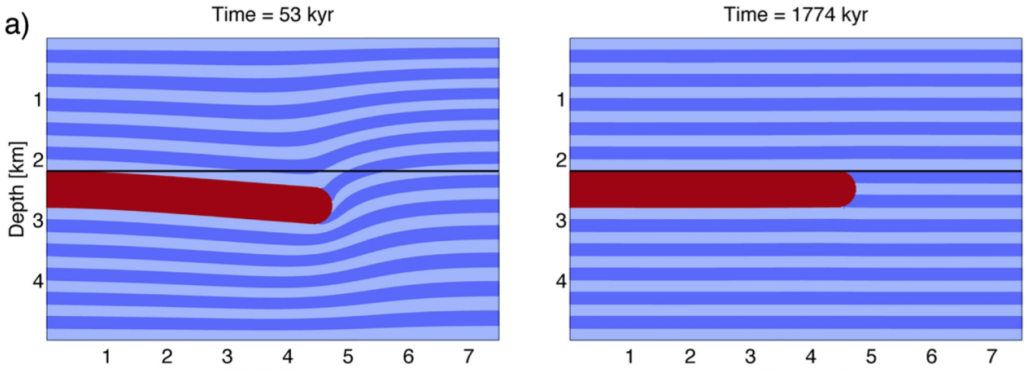
Material properties are as follows:

- beam: $\rho = 1500$, $\eta = 10^{24}$, $\mu = 10^{10}$
- fluid: $\rho = 1000$, $\eta = 10^{18}$, $\mu = 10^{11}$

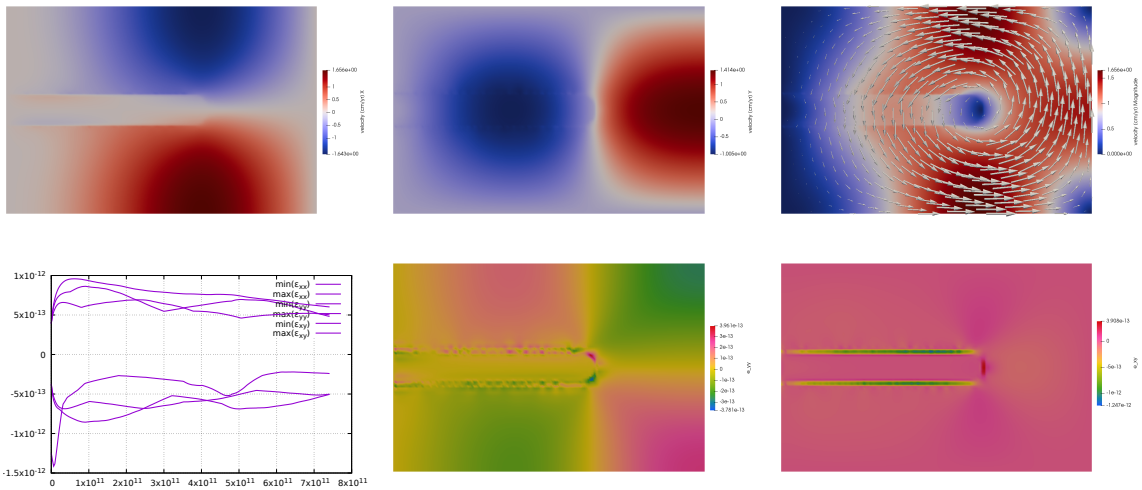
This choice of parameters leads to a Maxwell time $t_m = 0.32$ yr for the background fluid and Maxwell times of $t_m = 3.2$ Myr for the beam, meaning that the deformation in this benchmark problem, which occurs on a timescale of thousands to a million years, will lead to dominantly viscous deformation in the fluid, and dominantly elastic behaviour of the beam.

Keller et al set the numerical resolution to 300×200 elements, with 16 markers per elements for stress advection. Such a resolution is not feasible with our simple python implementation so the resolution is then set to 96×64.

The following plot comes from [183]:

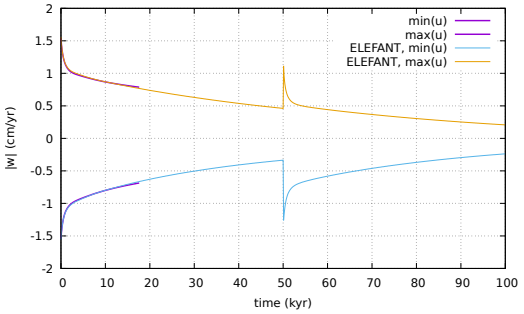


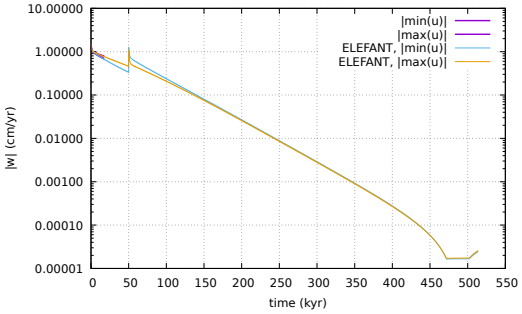
The elastic timestep is set to $\delta t_e = 100$yr and the tectonic timestep is set to the same value. This yields $\eta_{eff} = 10^{18}$ in the fluid, and $\eta_{eff} \simeq 3.15 \times 10^{19}$. After 50kyr, the gravity ($|\vec{g}| = 10$) is switched off and the model is ran for another 500kyr.

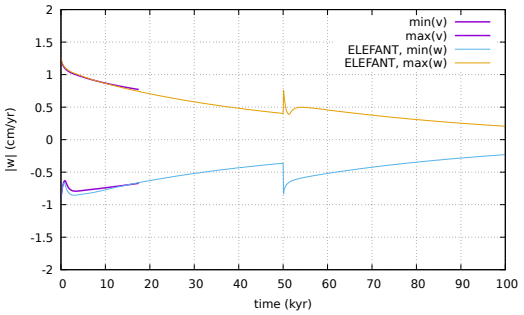


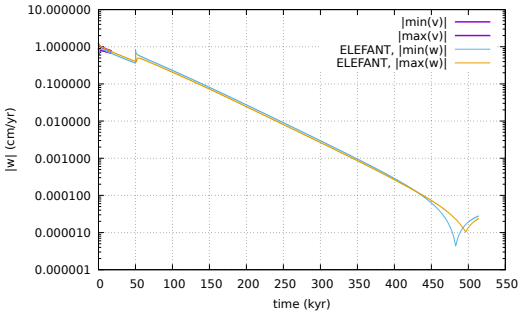


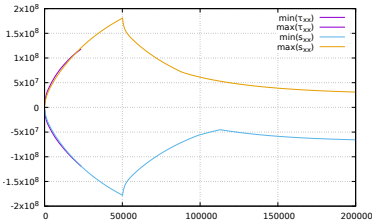
Various fields at the end of the 1st timestep: Velocity, strain rate components, Z, η_{eff}, strain rate components, pressure.

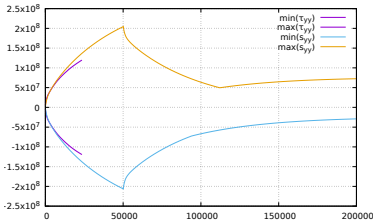


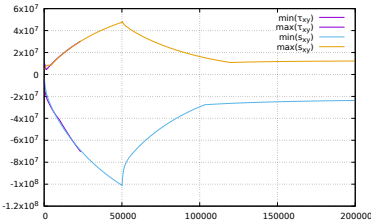




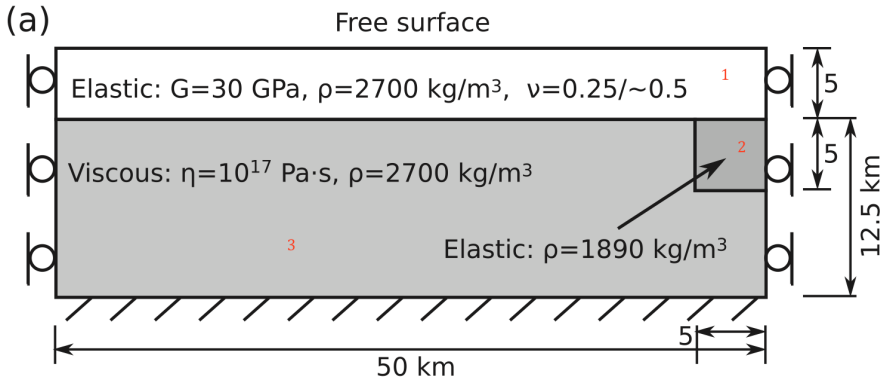








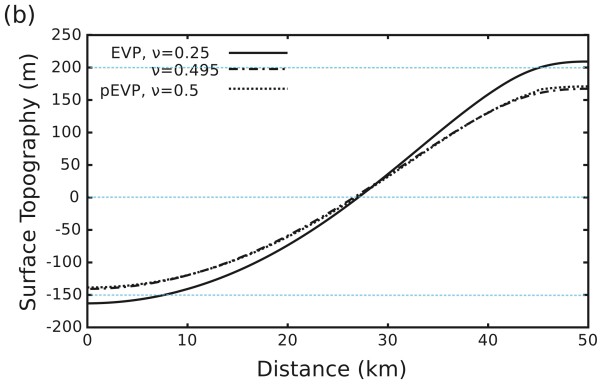
Flexure of elastic plate This benchmark is presented in Choi et al (2013) [70]. The setup is as follows:



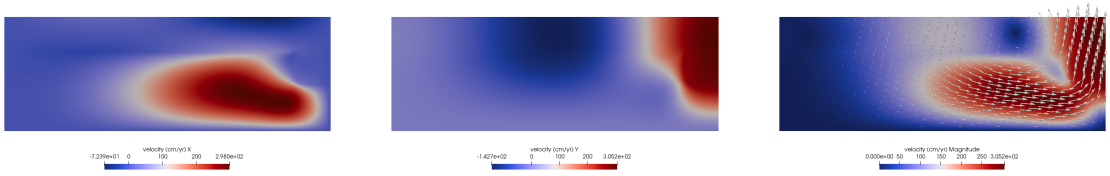
Material properties	*elastic plate (1)*	*elastic block (2)*	*viscous mantle (3)*
Density ρ [kg/m^3]	2700	1890	2700
Viscosity η [Pa· s]	10^{35}	10^{35}	10^{17}
Shear modulus μ [Pa]	$30 \cdot 10^9$	$30 \cdot 10^9$	10^{50}
Maxwell time t_M [yr]	10569930	10569930	3.1709791983764584e-41
eff. visc. η_{eff} [Pa·s]	4.73039776e+18	4.73039776e+18	1e+17
Factor Z [-]	0.9999995	0.9999995	6.3419583e-42

The value of $\eta_1 = \eta_2 = 10^{35}$ for the elastic materials was obtained through personal communication. The value of $\mu_3 = 10^{50}$ for the viscous material ensures that $\eta_{eff} = \eta_3$. Note that in the publication the authors test both compressible and incompressible formulations, but we restrict ourselves to incompressible results since our code cannot handle compressible behavior. I also use dt=5year.

The authors report a converged total relief of 306-308m:

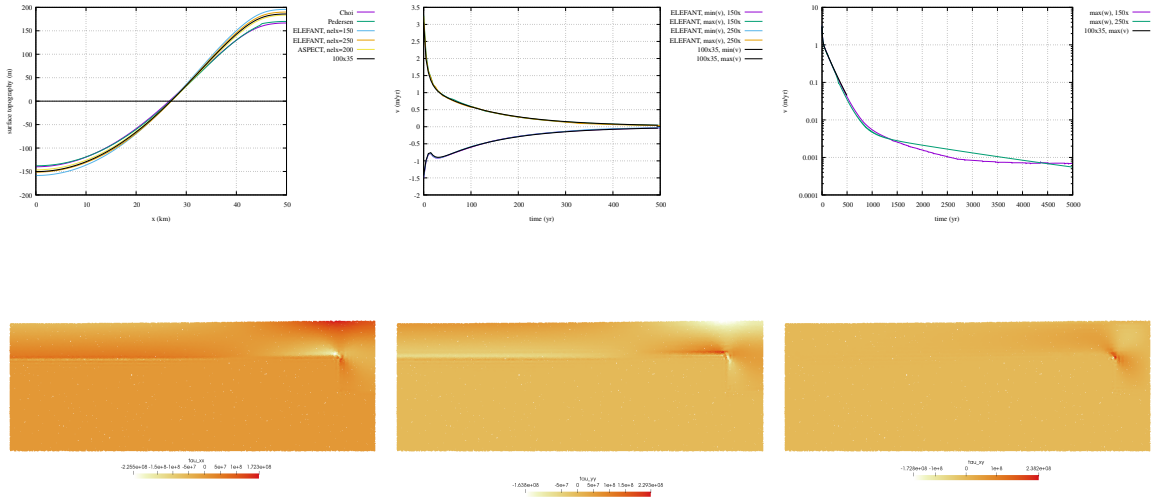


This benchmark requires either a sticky air layer (see Section 15.1) on top of the plate or a deformable mesh (ALE formulation, see Section 15.1).

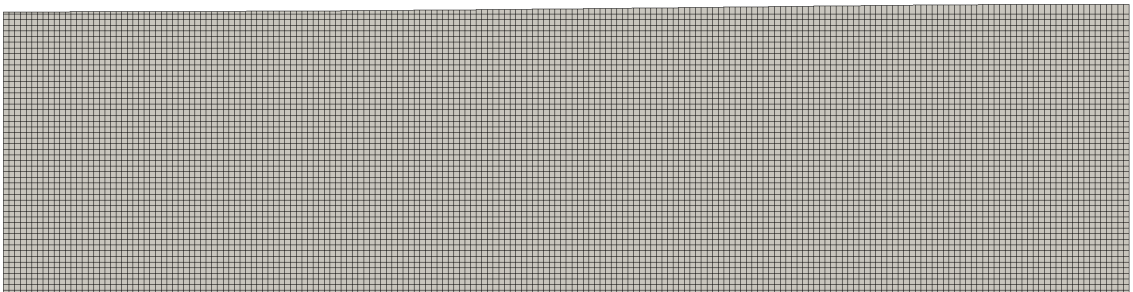


Initial velocity field

On the following figures I plot the topography and velocity statistics obtained with a resolution of 100×35 elements and 50 markers per element, next to results from Choi et al and/or results with my code Elefant [325].



accumulated deviatoric stress at time t=500yr on the markers.

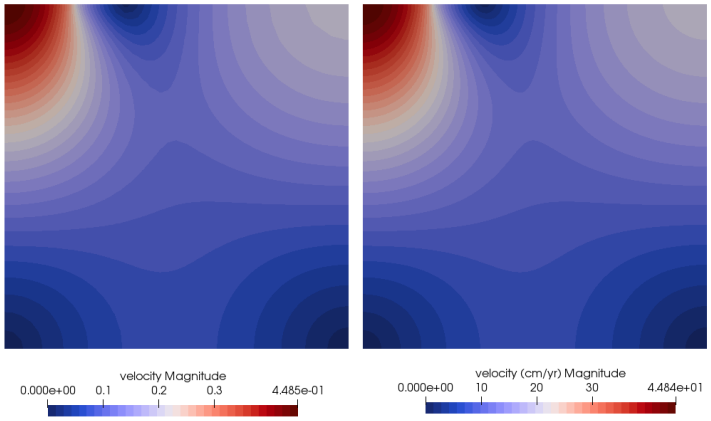


Grid at time t=500yr

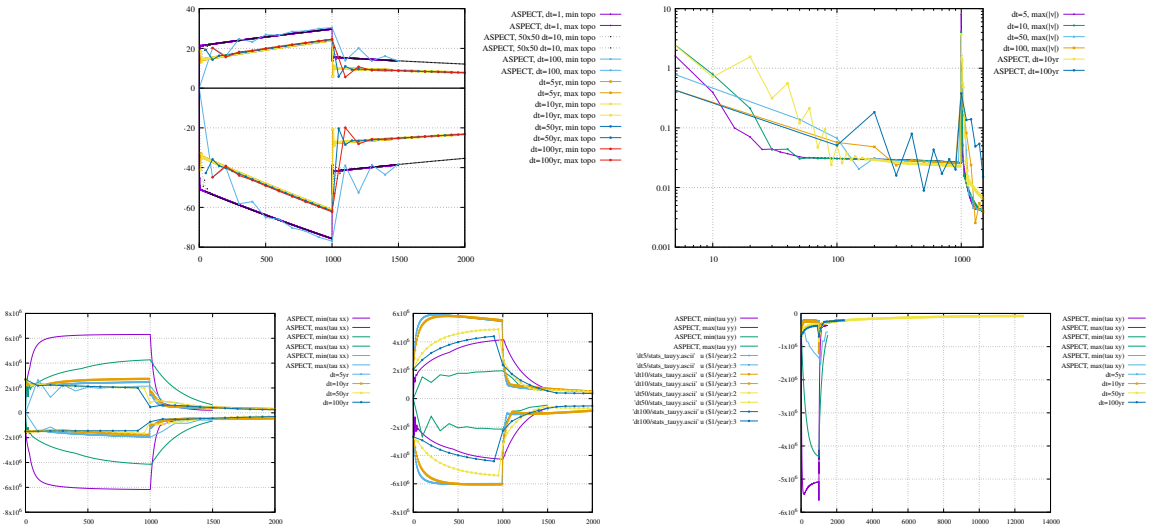
Ice load The domain is 500×500km. Vertical gravity is 9.8, density is 3300, viscosity is $3 \cdot 10^{20}$, shear modulus is 10^{10}, free slip on left, right and bottom. A normal stress is imposed on the top for $0 < x < 100$km. It corresponds to an ice sheet of density $\rho_i = 900$ of 1000m height. The timestep is set to 100yr. Resolution is set to 50×50 elements. Stress/traction b.c. are explained in Section 10.15.

Analytical solution is provided in Nakiboglu and Lambeck (1982) [241]. Note however that this is a 2D setup while the original solution is for a cylindrical load and also for a semi-infinite domain.

$$\eta_{eff} = \frac{\eta \delta t}{\delta t + \eta/\mu} = 2.85362675546547e + 19$$



Left: ASPECT; Right: stone 64



Stone 65

steady-state advection-diffusion (Q_1)

2D
box
Q1
analytical_benchmark
energy_equation
steady_state

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_65`

Let us start by remembering that the Peclet number is given by

$$\mathsf{Pe} = \frac{uh}{2\kappa} = \frac{uh\rho C_p}{2k}$$

where u is the velocity, h is the element size and κ is the diffusion coefficient.

Experiment 1

We wish to solve the advection-diffusion problem presented in Section 18.28:

$$\rho C_p u \frac{dT}{dx} - k\frac{d^2T}{dx^2} = f \qquad \text{in} \quad [0, L_x] \tag{65.1}$$

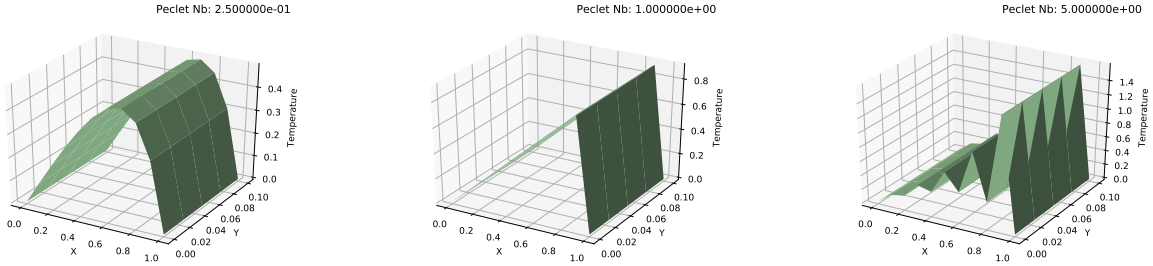
with the boundary conditions $T(x=0) = 0$ and $T(x=L_x) = 0$.

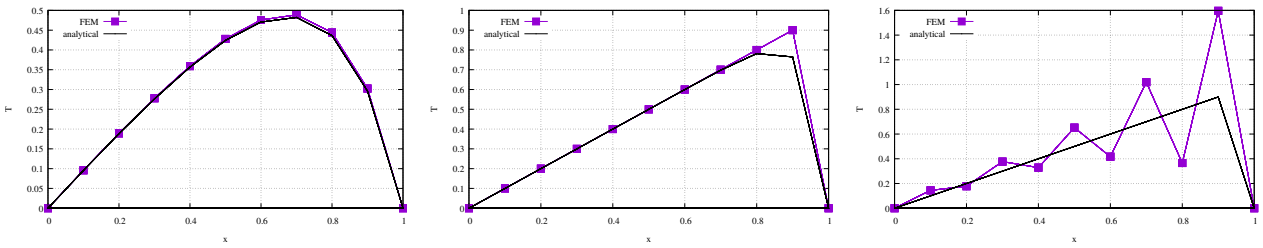
The domain is characterised by $L_x = 1$, and since L_y is irrelevant it is set to $L_x/10$. We further set $nelx = 10$, $f = 1$, $\rho = 1$, $C_p = 1$, $u = 1$. We will consider three values of the Peclet number: 0.25, 1 and 5. From these values we can compute the corresponding heat conductivity value k. Linear elements are used.

The analytical solution is given by Eq. (2.23) in Donea & Huerta [94]:

$$T(x) = \frac{f}{u_0}\left(x - \frac{1-\exp(2x\ \mathsf{Pe}/h_x)}{1-\exp(2\ \mathsf{Pe}/h_x)}\right)$$

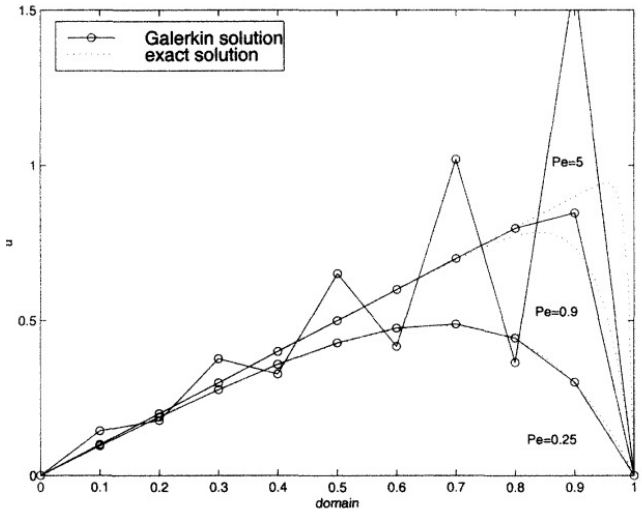
As Pe increases, a sharp gradient, sometimes called a boundary layer, develops at the right end of the domain. For high Pe values, the solution shows spurious node to node oscillations, failing to capture the highly nonlinear change. This oscillatory behavior is seen for $\mathsf{Pe} > 1$ (see Donea & Huerta [94]).





Left to right: Pe=0.25, Pe=1, Pe=5

Finally we see that we recover identical results as in Donea & Huerta [94]:



If we now use the artificial diffusion presented in Section 9.6, we have

$$\tilde{\kappa} = \beta\kappa\mathsf{Pe} = \beta\frac{k}{\rho C_p}\mathsf{Pe}$$

with

$$\beta = \coth(\mathsf{Pe}) - \frac{1}{\mathsf{Pe}}$$

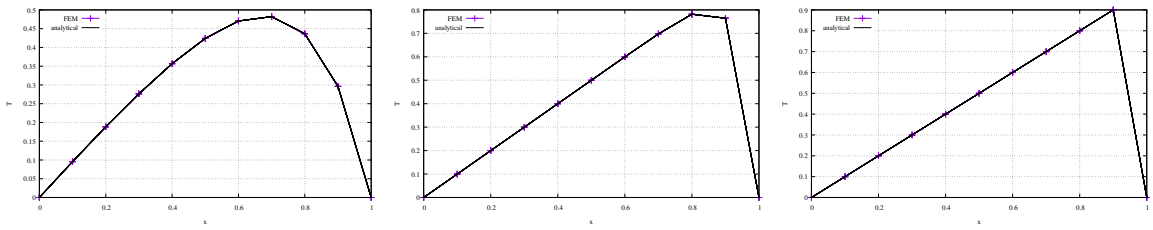
which yields:

$$\tilde{k} = \tilde{\kappa}\rho_0 C_p$$

and the code solves the following equation

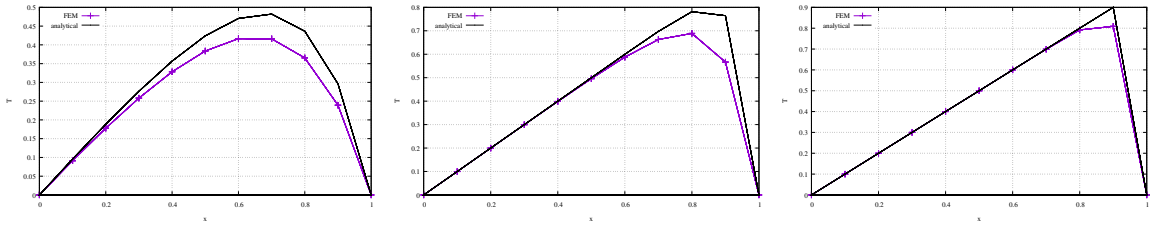
$$\rho C_p u\frac{dT}{dx} - (k+\tilde{k})\frac{d^2T}{dx^2} = f \qquad \text{in} \quad [0, L_x] \tag{65.2}$$

We observe that this approach recovers the analytical solution exactly:



Left to right: Pe=0.25, Pe=1, Pe=5

If one now uses 'full upwinding', i.e. $\beta = 1$, we see that the scheme is overly dissipative (i.e. overly diffusive):



Left to right: Pe=0.25, Pe=1, Pe=5

Pure diffusion It is also worth noting that when the velocity $u = 0$, then the ODE becomes

$$-k\frac{d^2T}{dx^2} = f \qquad \text{in} \quad [0, L_x] \tag{65.3}$$

and its solution is

$$T(x) = -\frac{k}{k}(\frac{1}{2}x^2 + ax + b)$$

Using the boundary conditions $T(x = 0) = 0$ and $T(x = L_x) = 0$, we find

$$b = 0 \qquad a = Lx/2$$

so the result is the following parabolic profile:

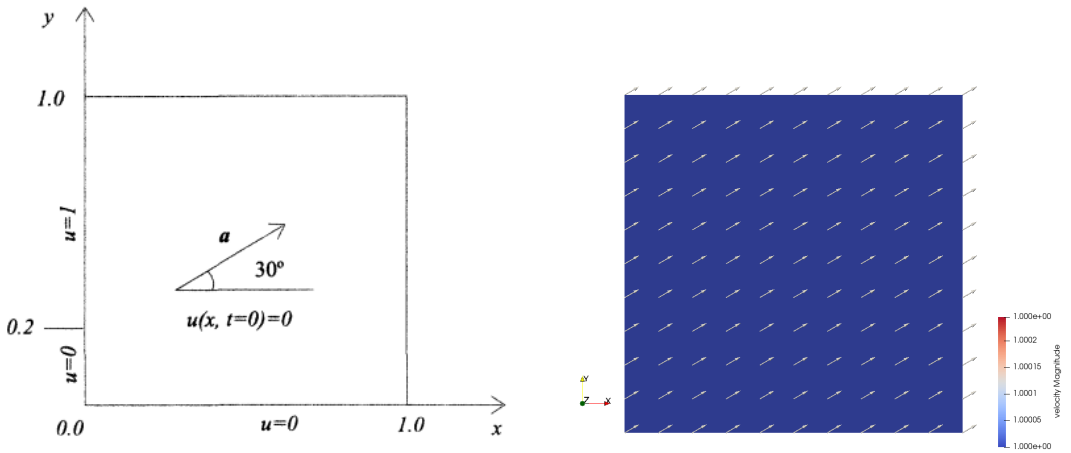
$$T(x) = \frac{f}{2k}x(L_x - x)$$

with

$$q(x) = -\frac{dT}{dx} = -\frac{f}{k}x$$

Experiment 2 - advection skew to the mesh

The domain is a unit square. Resolution is 10×10 linear elements. Initial temperature is irrelevant since we compute the steady state field.



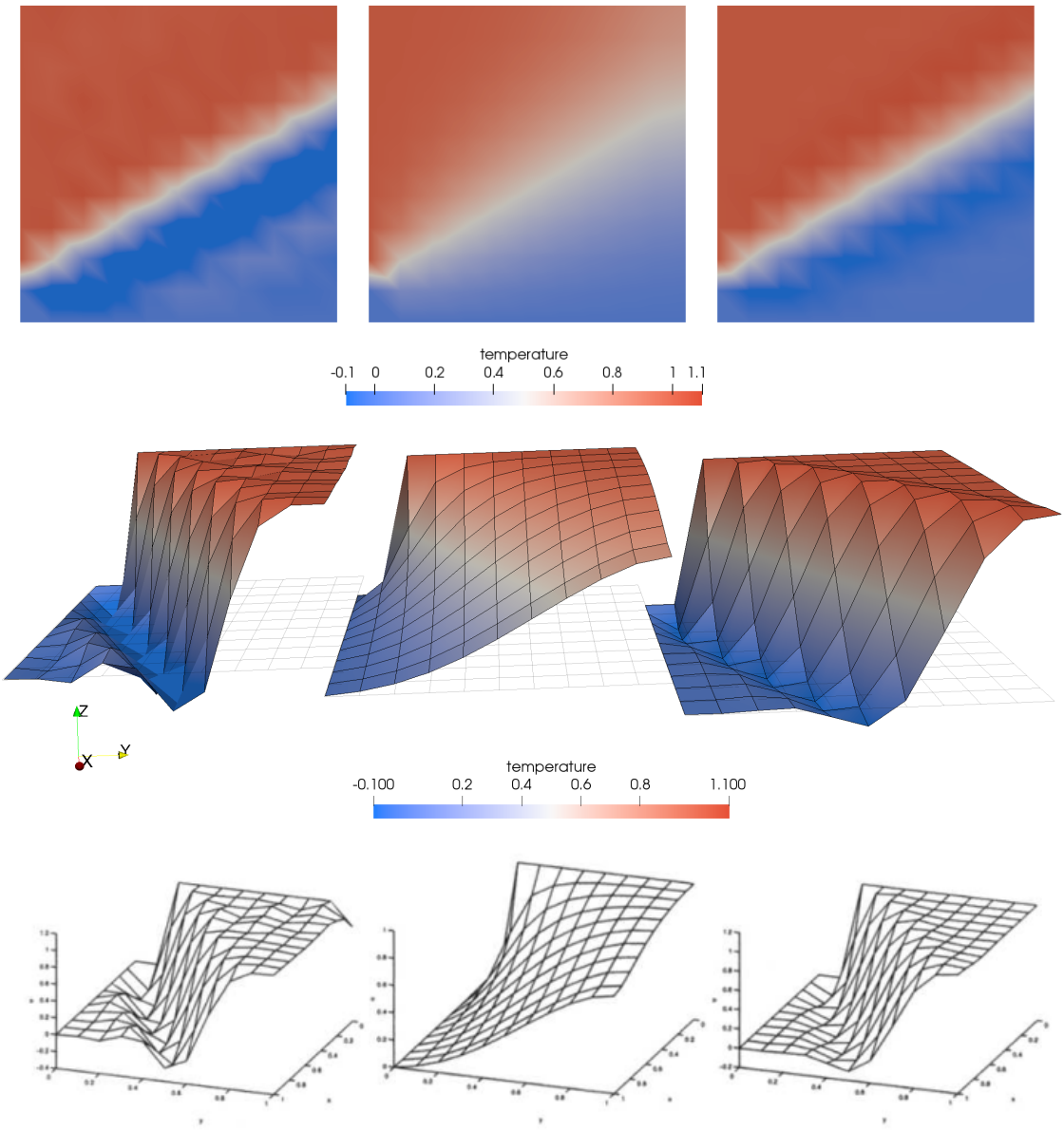
Left: Taken from Donea & Huerta [94]. In our code we replace u by the temperature T. Right: prescribed velocity field.

Remark. *On page 75 of [94] the authors state that "the diffusivity coefficient is taken to be* 10^{-4} *, corresponding to a mesh Peclet number of* 10^4*. Given that* $\mathsf{Pe} = uh/2\kappa$ *and h=0.1, this statement makes no sense. I have therefore chosen* $\mathsf{Pe} = 10^4$ *and then* $\kappa = uh/2\mathsf{Pe} = 5 \cdot 10^{-6}$*.*

The norm of the velocity vector $\vec{\mathbf{v}}$ is $|\vec{\mathbf{v}}| = 1$. Since the Peclet number is so high, we have

$$\tau = \frac{h}{2u_0}\left(\coth(\mathsf{Pe}) - \frac{1}{\mathsf{Pe}}\right) \to \frac{h}{2u_0}$$

We then conduct three experiments: one without stabilisation, one with (isotropic) artifical diffusion, and one with SUPG, from left to right on the following plots:



Left to right: Standard Galerkin method, artificial diffusion, SUPG. Bottom row taken from Donea & Huerta [94].

Experiment 3 - DG book setup

This setup comes from Li's book on Discontinuous Galerkin Methods [214], Section 5.2.3. The 2D advection-diffusion equation is solved on a unit square with the following boundary conditions:

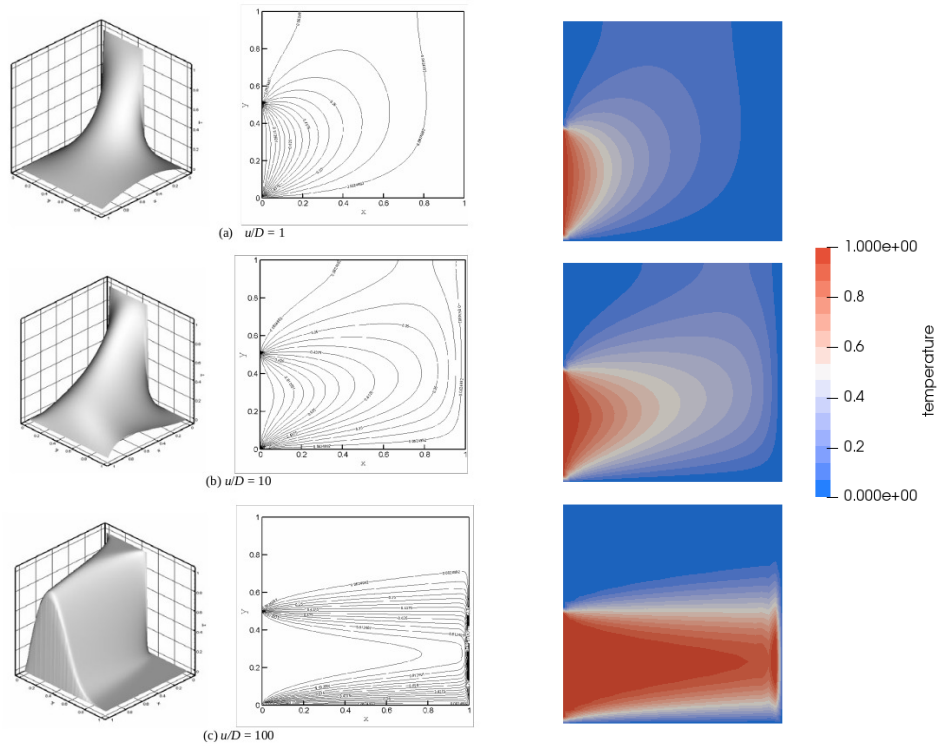
- bottom: $T = 0$
- left: $T = 0$ if $y > 0.5$, $T = 1$ otherwise
- right: $T = 0$
- top: $\partial_y T = 0$

Note that is not clear which value is actually prescribed in the lower left corner.

Coefficients are not specified but the experiment should be run at $u/\kappa = 1, 10, 100$ where D is the diffusion coefficient (in the book D is used instead of κ). We then set $\rho = C_p = 1$, $u_0 = 1$ and we define $\xi = u/\kappa$ with $\xi = 1, 10, 100$. Then

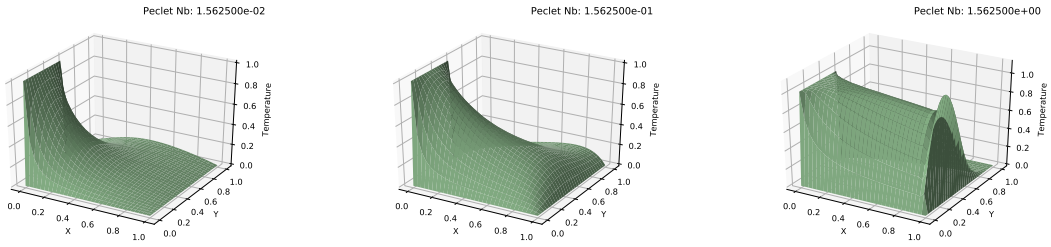
$$\mathsf{Pe} = \frac{h\xi}{2}$$

We finally set nelx=nely=32, so that the heat conductivity coefficient can be computed.



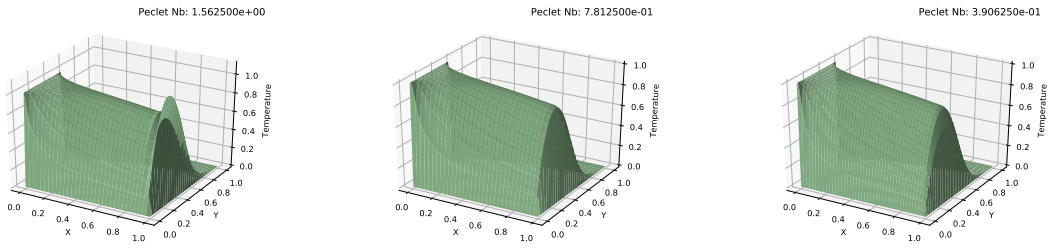
Left: Taken from Li [214]. Computed results for a 2-D convection-diffusion problem showing the effect of convection on the temperature distribution in the system. Right: results obtained without any stabilisation. From top to bottom, Pe = 0.015625, Pe = 0.15625 and Pe = 1.5625.

The code also automatically generates the following 3D plots, and we see that for $\xi = 100$ we obtain a strong overshoot near the right boundary:



No stabilisation, from left to right: $\xi = 1, 10, 100$.

If I now increase the resolution then h becomes smaller and so does the Peclet number, so that the oscillations near the right wall disappear:

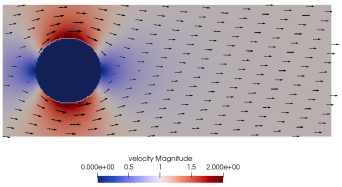


No stabilisation. $\xi = 100$. From left to right: 32x32, 64x64 and 128x128 resolution.

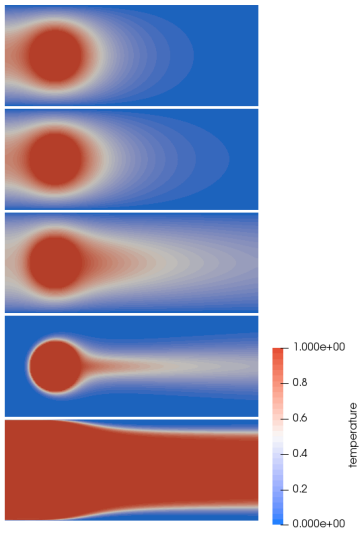
ToDo: run this experiment with SUPG.

Experiment 4 - Heat flow around a cylinder

This experiment is described in Section 19.13. We vary the Peclet number from 10^{-3} to 10^2. The prescribed velocity field is as follows:



Zero temperature is prescribed at the bottom and top boundaries. The steady state temperature fields for $\mathsf{Pe} = 0.001, 0.01, 0.1, 1, 10$ on a 250x100 grid are shown here:



Note that the velocity is not tangential to the bottom and top walls, so that we actually prescribe a tempetature on outflow regions, which is not possible!

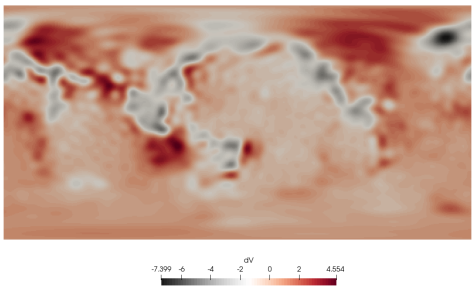
Stone 66

something about sph

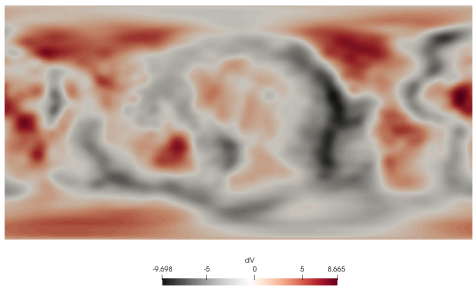
WRITE MORE about provenance of python code: `https://github.com/romaguir/sph_models`
There are several datasets available:

- *GypSum_S.sph*: Simmons et al (2010) [295]
- *S20RTS.sph*, *S40RTS.sph*: Ritsema et al (2011) [264] website[1]
- *s362ani_m_s.sph*: Kustowski et al (2008) [204]
- *SEM_WM_s.sph*: French & Romanowicz (2015) [117]
- *GAP_P4.sph*: Obayashi et al (2013) [245, 120] website[2]

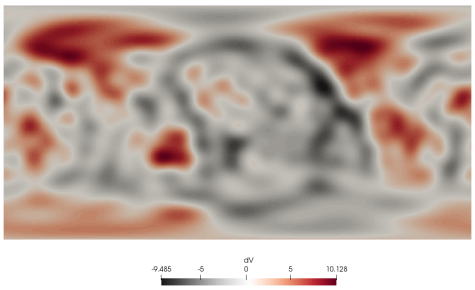
All at 100km depth:



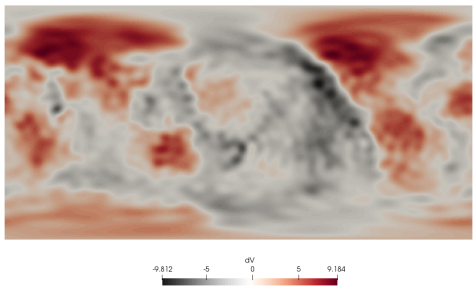
GAP_P4 ↑



GyPSum ↑

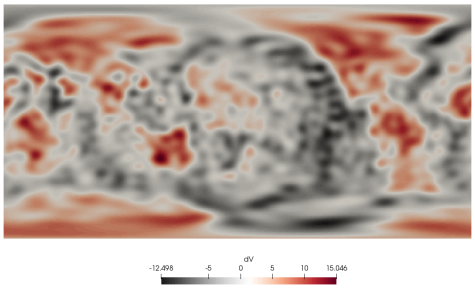


S20RTS ↑

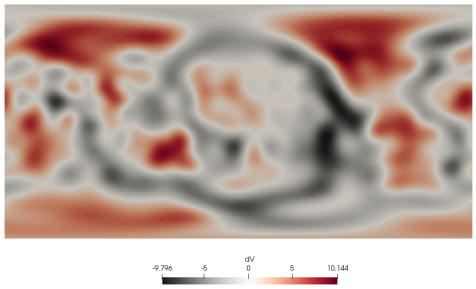


S40RTS ↑

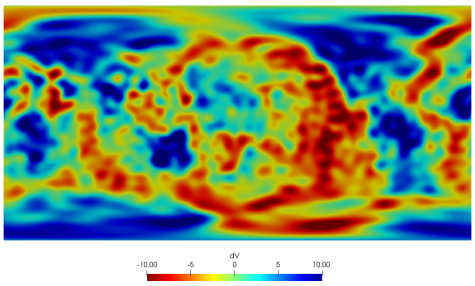
[1] `https://jritsema.earth.lsa.umich.edu/Research.html`
[2] `http://d-earth.jamstec.go.jp/GAP_P4/`

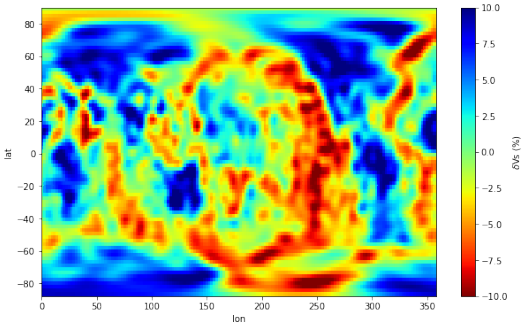


SEM_WM ↑

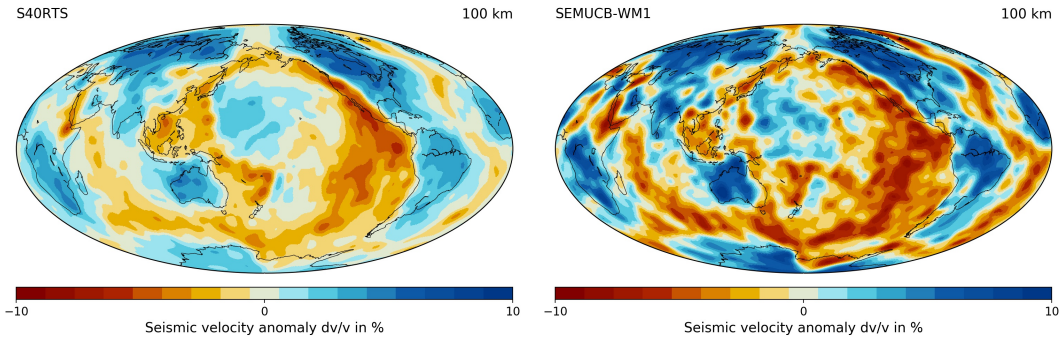


S362ANI ↑





SEM_WM. Left: fieldstone; Right: as shown in notebook.



Stone 67

Newtonian subduction setups (sinking block, subduction)



python_codes/fieldstone_67/text.tex

Codes at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_67`

Note that none of the results below should be really trusted. All setups should be rerun properly with the last version of the code, with different averagings, projections, resolutions...

This stone implements the particle in cell technique to track the multiple materials. Each particle carry a material information/identity. Since we are here considering isothermal and linear viscous setups only, this means that each particle has a density and a viscosity. The values must then be projected back onto the Eulerian mesh, either on the nodes or on the quadrature points directly. There are four types of projection, controlled by the value of the particle_projection parameter:

1. use elemental values for all quadrature points;
2. use nodal values + Q_1 shape functions to interp on q points;
3. use average nodal values to assign to all q points (elemental avrg);
4. nodal density, elemental viscosity.

Setup1: Sinking block

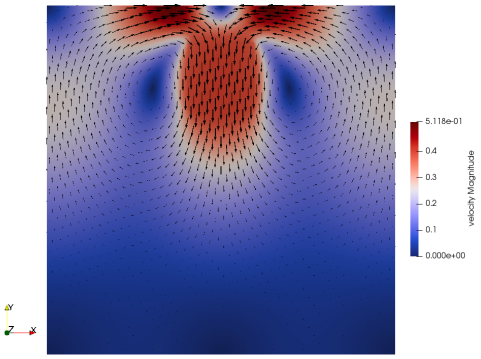
This simple benchark provides challenging numerical experiments dealing with large viscosity variations within the simulation domain. It appears in Gerya [128] (2010), Thieulot [326] (2011) and Thieulot and Bangerth [328] (2022). It consists of a bulk of fluid 1 (ρ_1, η_1) in which a block of fluid 2 (ρ_2, η_2) falls under its own weight. The domain is a square of size $L_x = L_y = 500$ km and the block is initially centered at point $(x = 250$ km$, y = 400$ km$)$ with size 100×100 km.

The simulation is carried out on a 64×64 elements grid and each element contains n^3 markers at startup. Free slip boundary conditions are imposed on all sides of the domain.

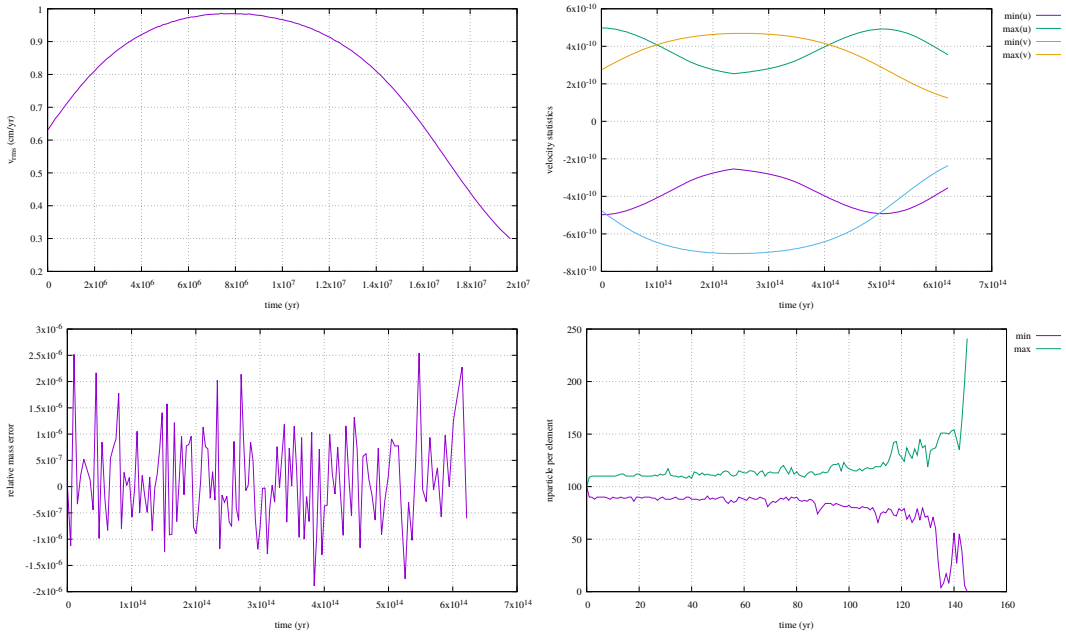
The density of the surrounding fluid is $\rho_1 = 3200$ kg m^{-3} and its viscosity $\eta_1 = 10^{20}$ Pa s. The viscosity and density of the block is varied. We define

$$\eta^\star = \eta_2/\eta_1 \qquad \rho_2 = \rho_1(1 + d\rho)$$

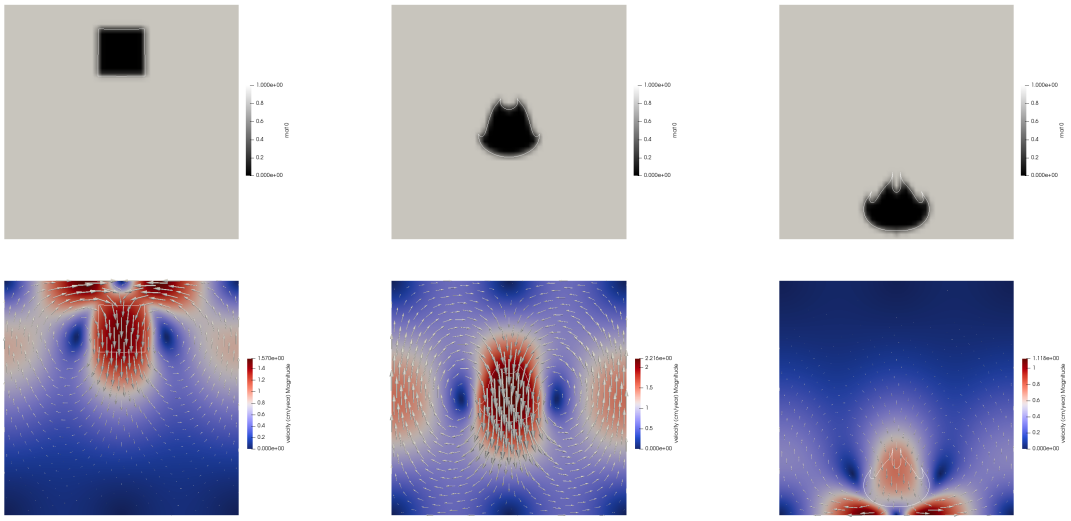
In Earth sciences, $\eta^\star$ can typically take values between 10^{-2} and 10^6, while $d\rho$ between 0.001 and 0.5 (in this context we need $d\rho > 0$ so that the cube goes down).



velocity field at $t = 0$ for $\eta_2 = 10^{23}$.



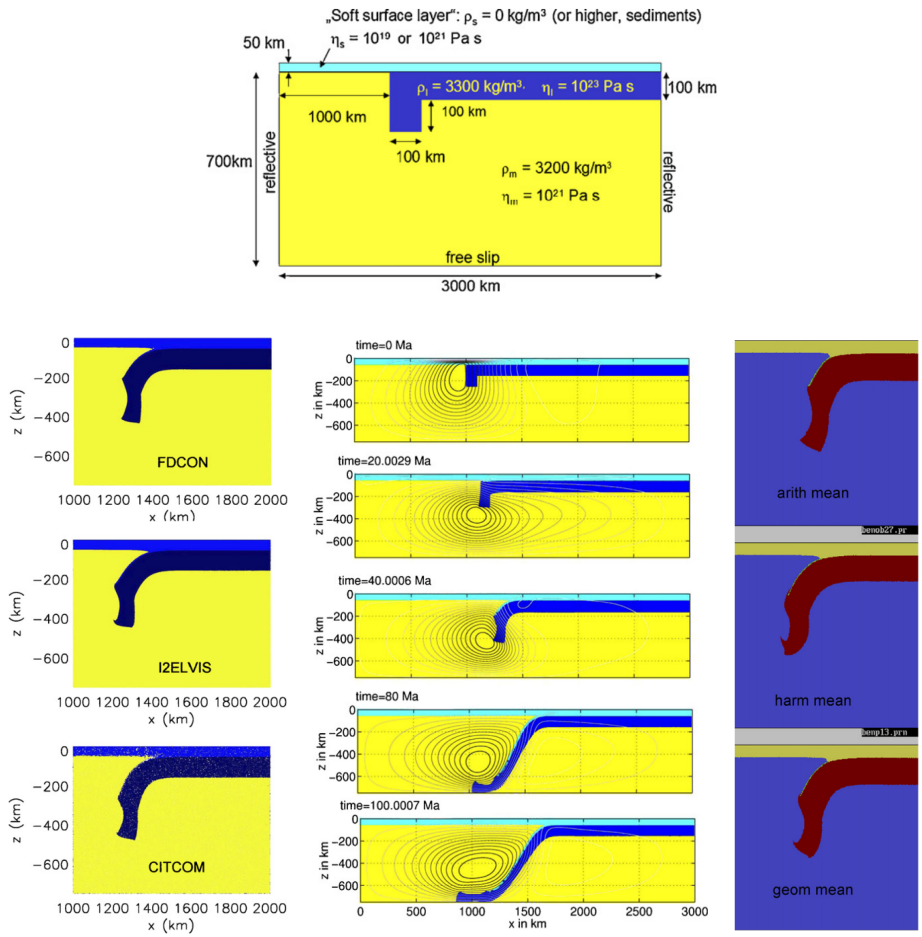
Resolution 48×48. 100 random particles per element. rk=2. $\eta* = 10$. CFL=0.25. projection=2



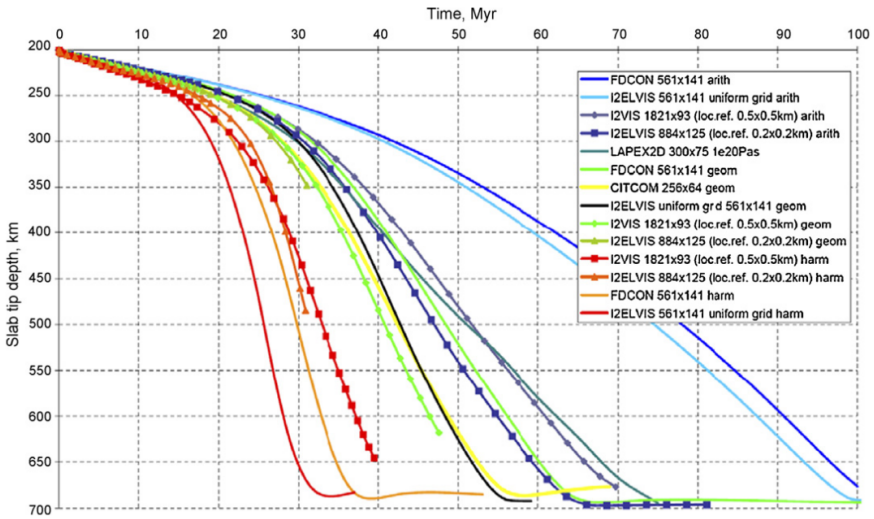
Run
- various viscosity ratios , density differences - rk1,rk2, rk3 - nmarker_per_el - reg vs random - projections

Setup2: Schmeling et al, 2008

The setup originates in Schmeling et al (2008) [282]. It consists of three linear viscous materials with a very simple layout placed in a 2D Cartesian domain with free slip boundary conditions on all sides.



Left: Typical behaviour of the model (as obtained with FDCON-4 code). Streamlines are also shown. Middle: Shapes of different case 1 models at similar stages: FDCON: 40 Myears, I2ELVIS: 34.7 Myears, CITCOM: 38.1 Myears. Viscosity averaging: geometric mean in all cases. Right: Comparison of the shapes of the slabs for different viscosity averaging methods using I2VIS. Note that the snapshots are taken at different times (59.6, 24.4, 37.8 Myears from top to bottom), so that the slab tips have reached comparable levels.



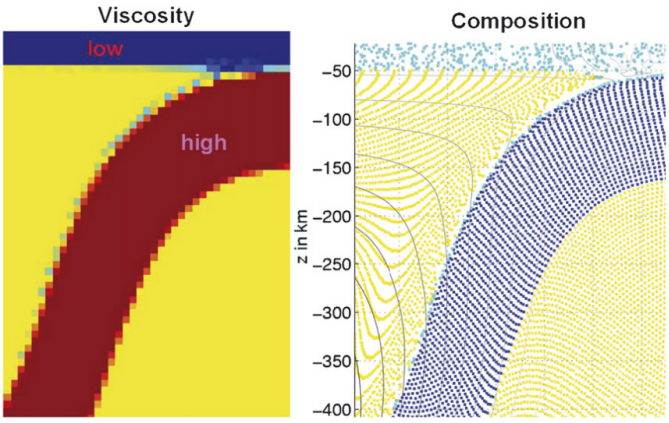
Temporal behaviour modelled by different codes with highest resolutions each. Each curve shows the position of the deepest part of the slab (slab tip) as a function of time below the initial surface of the lithosphere. See the legends for the used codes and grid resolution. Note that the codes I2VIS and I2ELVIS also use local refinement at the trench area (given in parentheses in the legend). Outside the trench area the resolution decreases to 10×46 km at model sides.

At the lower boundary the vertical resolution was 1 km. The rheological means are denoted as geom for geometric, harm for harmonic and arith for arithmetic, respectively.

The total mass of the system is given by

$$\begin{aligned} M_0 &= [3000 \times 50 \times \rho_s + (100 \times 100 + 2000 \times 100)\rho_l + (1000 \times 700 + 100 \times 500 + 1900 \times 600)\rho_m]10^6 \\ &= 210,000\rho_l + 1,890,000\rho_m \\ &= 6.741000e + 15 \text{ kg} \end{aligned} \tag{67.1}$$

Note that all published results are obtained with high resolutions (compared to what is feasible with our code). On the following figure there seems to be 5 elements in the air layer, i.e. a resolution of 10km, i.e. 300×75 elements (if mesh stretching was not used).

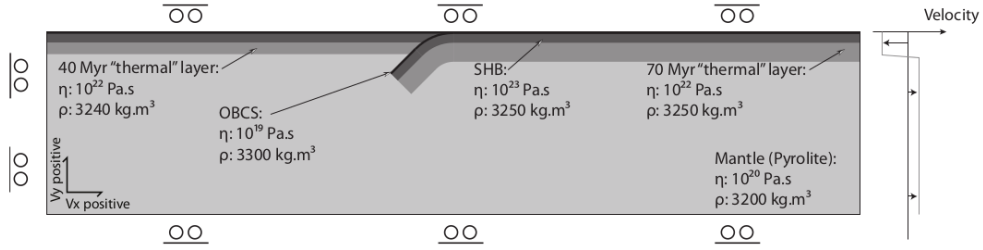


Details of the entrainment and lubrication of the soft surface layer.

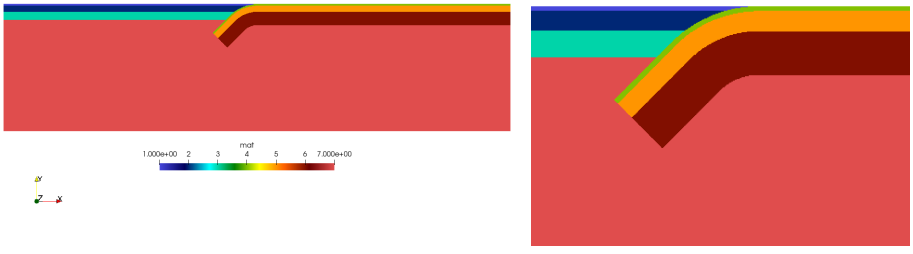
Setup3: Quinquis et al, 2010

This setup is based on a poster by Quinquis et al, presented at EGU 2010, but it was never published in any journal.

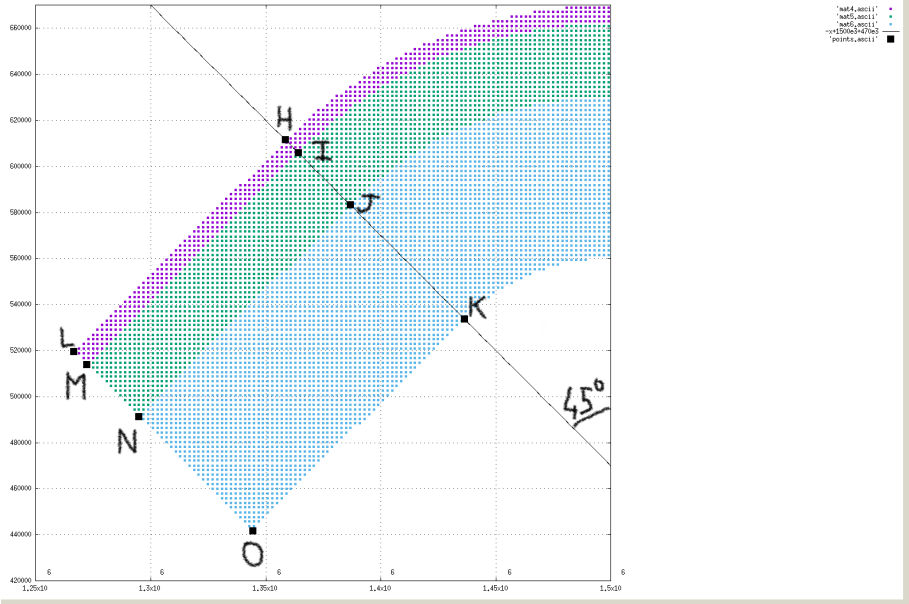
The code relies on $Q_2 \times Q_1$ elements. The domain is 2680×670 km, i.e. the aspect ratio is exactly 4. Boundary conditions are free slip on the left, bottom and top boundaries.



There are 7 materials in the domain:



Depths of interfaces on the left are 7, 39, and 82km and 8, 40, and 110 km on the right. The opening angle of the circular part is 45° and the already subducted slab (i.e. LH=MI=NJ=OK) is 130 km long:



The coordinates of points H to O are then as follows:

```
xH=Lx/2-np.sqrt(2.)/2.*200e3  ; yH=Ly-200e3+np.sqrt(2.)/2.*200e3
xI=Lx/2-np.sqrt(2.)/2.*192e3  ; yI=Ly-200e3+np.sqrt(2.)/2.*192e3
xJ=Lx/2-np.sqrt(2.)/2.*160e3  ; yJ=Ly-200e3+np.sqrt(2.)/2.*160e3
xK=Lx/2-np.sqrt(2.)/2.*90e3   ; yK=Ly-200e3+np.sqrt(2.)/2.*90e3
xL=xH-130e3*np.sqrt(2.)/2.    ; yL=yH-130e3*np.sqrt(2.)/2.
xM=xI-130e3*np.sqrt(2.)/2.    ; yM=yI-130e3*np.sqrt(2.)/2.
xN=xJ-130e3*np.sqrt(2.)/2.    ; yN=yJ-130e3*np.sqrt(2.)/2.
xO=xK-130e3*np.sqrt(2.)/2.    ; yO=yK-130e3*np.sqrt(2.)/2.
```

A horizontal velocity profile is prescribed on the right boundary, but only the horizontal component is prescribed. The velocity profile is described in Section 10.14.2, and we impose $y_1 = L_y - 160$ km and $y_2 = L_y - 128$ km, and a $u_{in} = 5$ cm yr^{-1}.

The mesh is first created so that all elements have the same dimensions $h_x = L_x/nelx$ and $h_y = L_y/nely$. It is then stretched using the two functions *stretch_towards_center* and *stretch_towards_top*.

The structure of the code is as follows:

- initialisation
- generate velocity nodes coordinates arrays
- stretch the mesh horizontaly and verticaly
- generate connectivity arrays for velocity and pressure
- generate pressure nodes coordinates arrays
- particle setup. The ensemble of all *nmarker* particles is called a swarm[1]. Each particle has a position *x*, *y*, a material id *mat*, a *paint* value, reduced coordinates *r*,*s* and an element *iel* it resides in. Markers are placed regularly in the element on a *nmarker_per_dim* X *nmarker_per_dim* grid.
- each particle is then assigned a material id between 1 and 7, based on the geometry of the plates.
- the swarm is then painted, which is a passive field designed to provide visual aid for total deformation
- velocity boundary conditions setup

[1] A mass of people, animals or things in motion

- project material id carried by the particles onto pressure mesh. A $nmat \times NP$ array is needed to store this information:

```
mat_nodal=np.zeros((nmat,NP),dtype=np.float64)
```

We then loop over each particle, compute the values of the 4 Q_1 shape functions inside the element at its location and use these as weights.

```
for im in range(0,nparticle):
    imat=swarm_mat[im]-1
    iel=swarm_iel[im]
    N1=0.25*(1-swarm_r[im])*(1-swarm_s[im])
    N2=0.25*(1+swarm_r[im])*(1-swarm_s[im])
    N3=0.25*(1+swarm_r[im])*(1+swarm_s[im])
    N4=0.25*(1-swarm_r[im])*(1+swarm_s[im])
    mat_nodal[imat,iconP[0,iel]]+=N1
    mat_nodal[imat,iconP[1,iel]]+=N2
    mat_nodal[imat,iconP[2,iel]]+=N3
    mat_nodal[imat,iconP[3,iel]]+=N4
    mat_nodal_counter[iconP[0,iel]]+=N1
    mat_nodal_counter[iconP[1,iel]]+=N2
    mat_nodal_counter[iconP[2,iel]]+=N3
    mat_nodal_counter[iconP[3,iel]]+=N4
mat_nodal/=mat_nodal_counter
```

At this stage *mat_nodal[1,:]* contains values between 0 and 1 on each pressure node which corresponds to how much material 1 is present on the node.

- Now that we know how much of each material is on each node, we can compute the nodal density as follows:

```
rho_nodal=np.zeros(NP,dtype=np.float64)
for i in range(0,NP):
    for imat in range(0,nmat):
        rho_nodal[i]+=mat_nodal[imat,i]*rho_mat[imat]
```

Concerning the viscosity, we have the choice between arithmetic (*avrg=1*), geometric (*avrg=2*) and harmonic (*avrg=3*) averagings, which are implemented as follows:

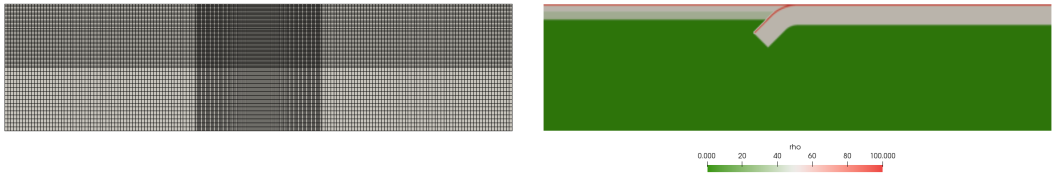
```
if avrg==1:
   for i in range(0,NP):
       for imat in range(0,nmat):
           eta_nodal[i]+=mat_nodal[imat,i]*eta_mat[imat]

if avrg==2:
   for i in range(0,NP):
       for imat in range(0,nmat):
           eta_nodal[i]+=mat_nodal[imat,i]*np.log10(eta_mat[imat])
       eta_nodal[i]=10.**eta_nodal[i]

if avrg==3:
   for i in range(0,NP):
       for imat in range(0,nmat):
           eta_nodal[i]+=mat_nodal[imat,i]/eta_mat[imat]
       eta_nodal[i]=1./eta_nodal[i]
```

- build matrix and rhs. Interpolate density and viscosity onto quadrature points by means of the pressure shape functions (Q_1) so as to avoid unwanted negative values:

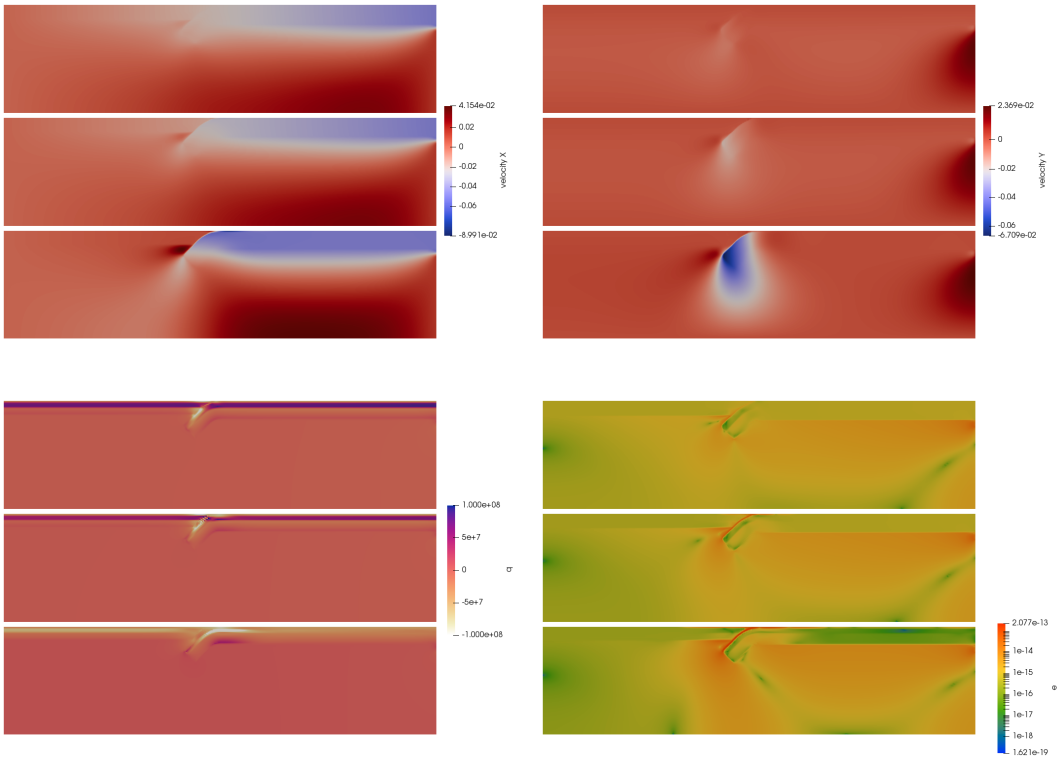
```
NNNP[0:4]=NNP(rq,sq)
[...]
for k in range(0,mP):
    rhoq[counter]+=NNNP[k]*rho_nodal[iconP[k,iel]]
    etaq[counter]+=NNNP[k]*eta_nodal[iconP[k,iel]]
```

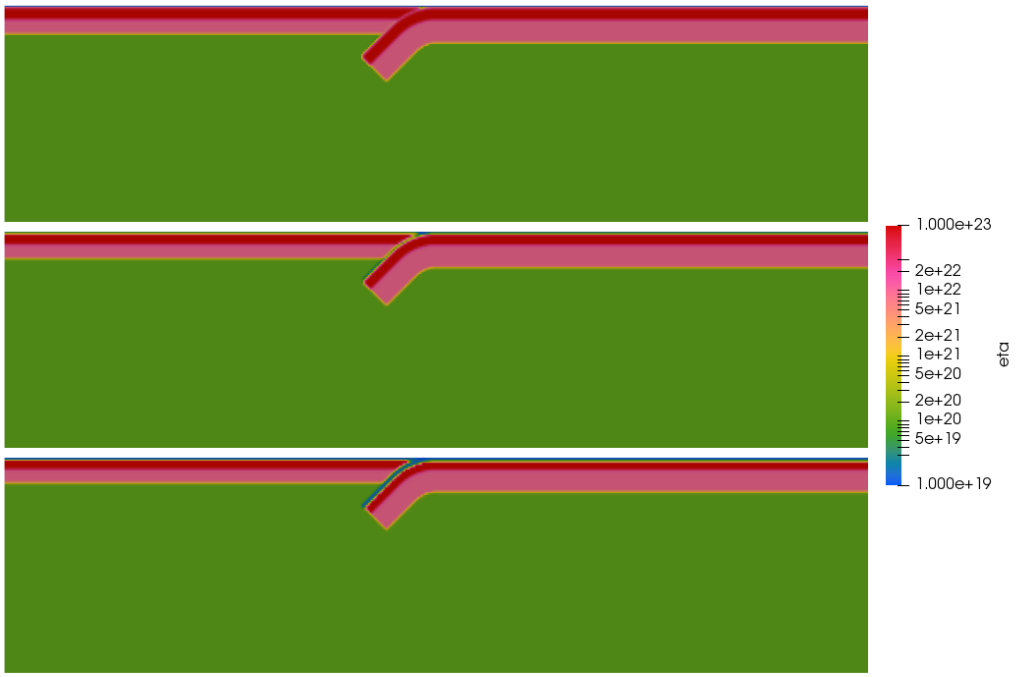
- solve for velocity and pressure
- compute timestep dt

- compute root mean square velocity
- advect markers (Runge-Kutta 1,2,3)
- compute nodal strain rate fields
- interpolate Q_1 fields onto Q_2 mesh
- produce vtu files of the mesh and markers



256x64 elements with mesh stretching



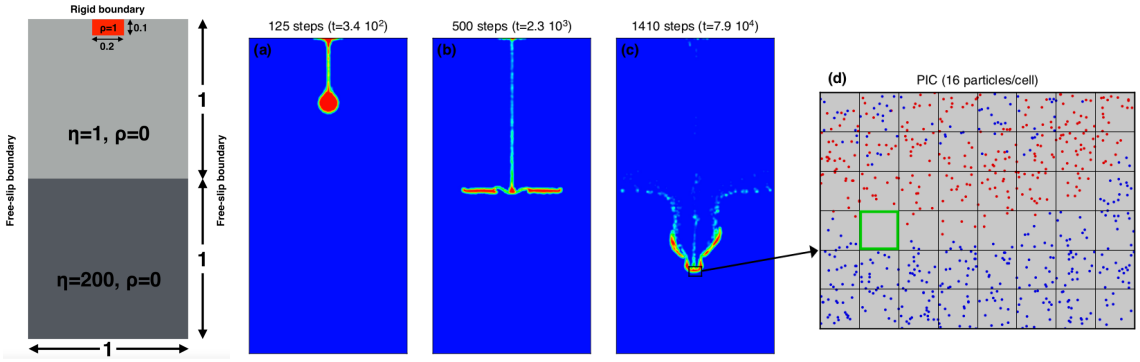


256x64 elements. Top, middle, bottom: arithmetic, geometric, harmonic respectively.

marker localisation by inverse stretching?
COMPUTE ANALYTICAL MASS ?!

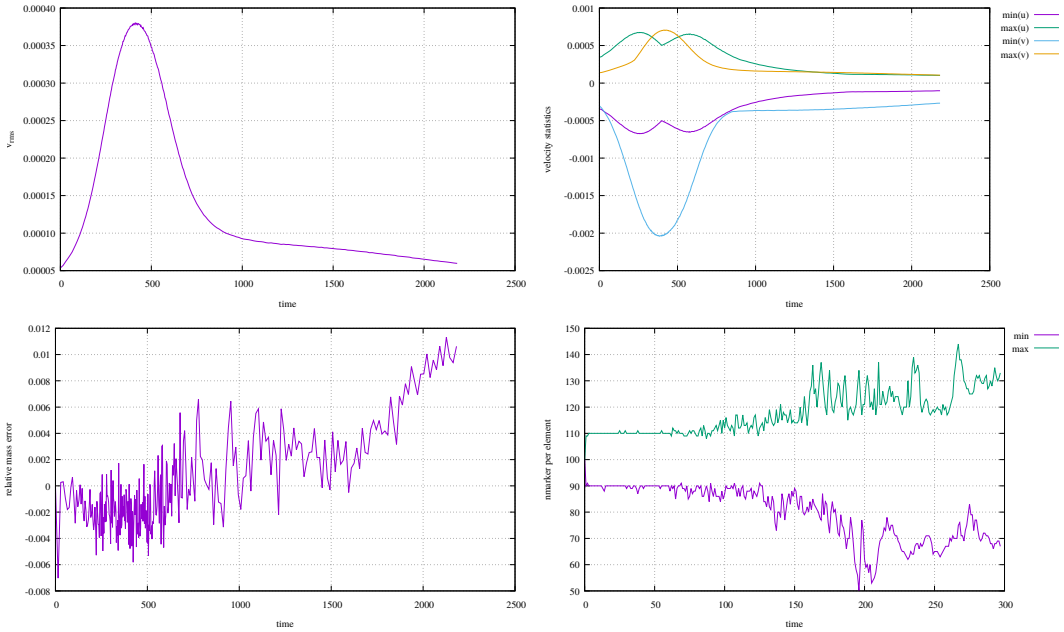
Setup4: Samuel, 2018

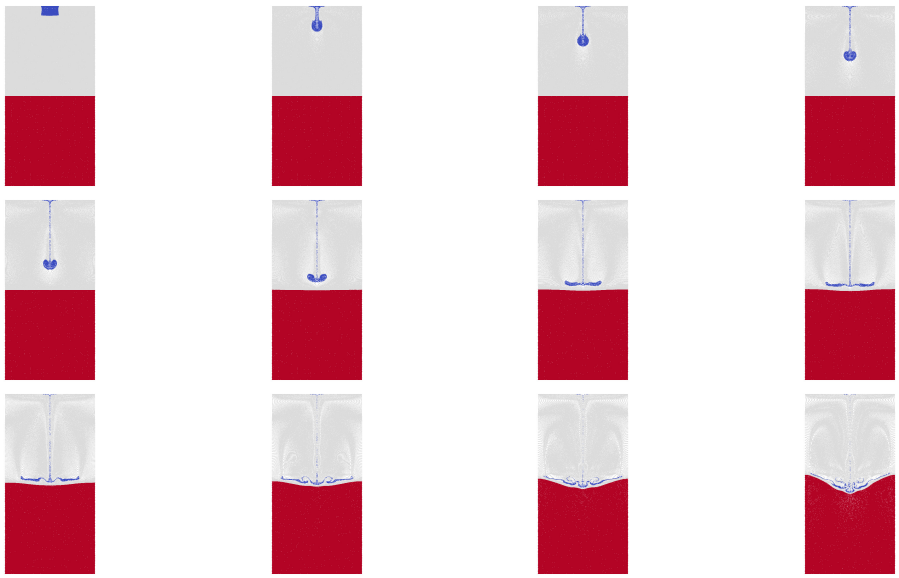
This setup is proposed by Samuel (2018) [276] in the context of his paper on the deformable particle-in-cell method (DPIC). The domain has dimensions 2×1, free slip on the sides, no slip on the top and bottom. Gravity acceleration is set to 1, and the geometry is as follows:

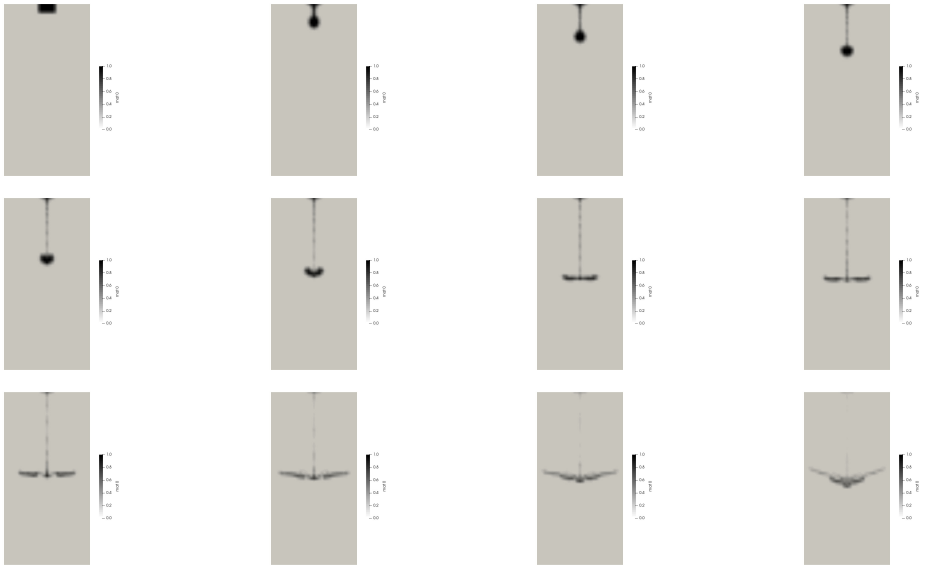


Taken from [276]. Note that the red area also has a unit viscosity.

Samuel states that the $[0, 1] \times [0, 2]$ domain was discretized using either 50×100 or 100×200 square cells and that the incompressible Stokes equations are solved by means of a finite volume code [275, 274]



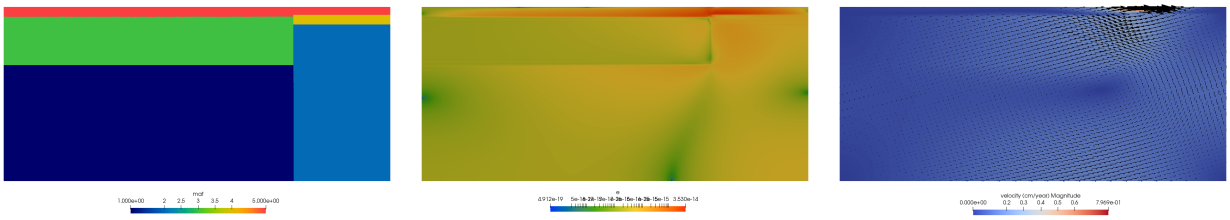




Setup5: subduction initiation

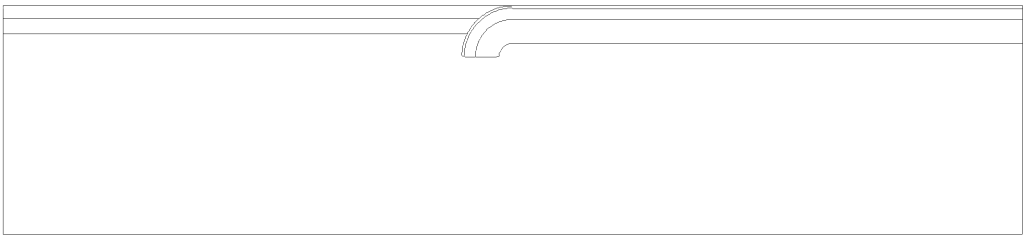
The setup is decribed in STONE 118 and stems from Matsumoto and Tomoda [224] (1983).
total mass is:

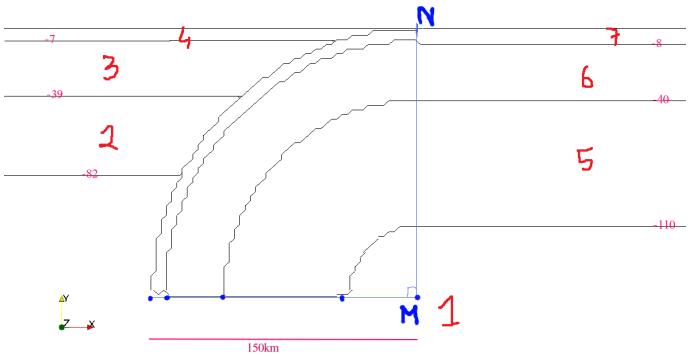
$$M = \frac{3}{4} L_x \left((L_y - D_w - D_l)\rho_a + D_l \rho_l + D_w \rho_w \right) + \frac{1}{4} L_x \left((L_y - d_w - d_l)\rho_a + d_l \rho_l + d_w \rho_w \right) \simeq 223.754 \cdot 10^9$$

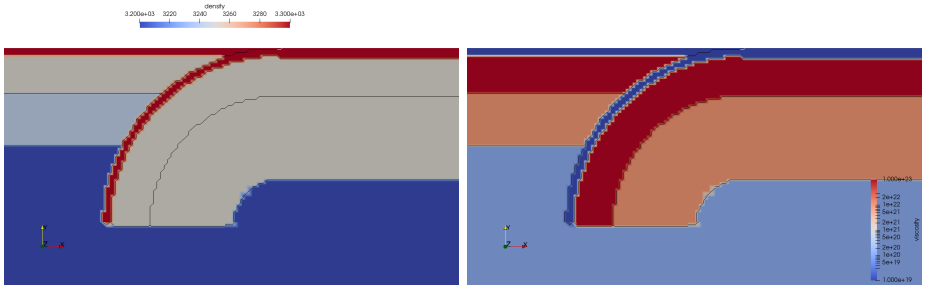


Setup6: another Quinquis-like setup

The domain is 3000 × 670 km. The setup is as follows:







$x_M = x_N = 1500$ km, $y_M = L_y - 150$ km $= 520$ km

Materials are incompressible and with linear viscous rheology. The flow is isothermal. Boundary conditions are free slip on all sides. Gravity is set to -9.81

material	density	viscosity
1	3200	1e20
2	3240	1e22
3	3250	1e23
4	3300	1e19
5	3250	1e22
6	3250	1e23
7	3300	1e19

In aspect input file (`||` is 'or', `&&` is 'and'):

- material 2

```
(x<=L0 && y>= 588e3 && y<= 631e3 && (x-L0)^2+(y-yC)^2>=150e3^2 ) ;
```

- material 3

```
(x<=L0 && y>= 631e3 && y<= 663e3 && (x-L0)^2+(y-yC)^2>=150e3^2 )
```

- material 4

```
(x<=L0 && y>= 663e3 && (x-L0)^2+(y-yC)^2>=150e3^2 )
```

- material 5

```
(x>=L0 && y>= 560e3 && y<= 630e3) ||
((x-L0)^2+(y-yC)^2<110e3^2 && (x-L0)^2+(y-yC)^2>40e3^2 && y>520e3 && x<L0)
```

- material 6

```
(x>=L0 && y>= 630e3 && y<= 662e3) ||
((x-L0)^2+(y-yC)^2<143e3^2 && (x-L0)^2+(y-yC)^2>110e3^2 && y>520e3 && x<L0)
```

- material 7

```
(x>=L0 && y>= 662e3 )
|| ((x-L0)^2+(y-yC)^2<150e3^2 && (x-L0)^2+(y-yC)^2>143e3^2 && y>520e3 && x<L0)
```

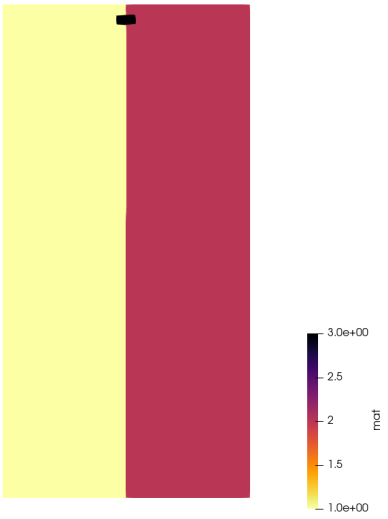
Setup7: sinking of anhydrite blocks in Newtonian salt

This experiment comes from Burchardt, Koyi, Schmeling, and Fuchs [52] (2012). See also other similar articles by the same team such as Burchardt, Koyi, and Schmeling [50] (2011) and Burchardt, Koyi, and Schmeling [51] (2012).

In the paper there are three cases A,B,C with potentially different domain sizes. I choose here experiment C with a domain of size $L_x = 2500$ m and $L_y = 5000$ m. The gravity is not specified but I set it to $g = 9.81\ \mathrm{m\,s^{-2}}$. Boundary conditions are simple: free slip on all four sides. There are three materials in the domain: salt1, salt2 and anhydrite with

- $\eta_{salt1} = 10^{16}$ Pa s, $\rho_{salt1} = 2200\ \mathrm{kg\,m^{-3}}$
- $\eta_{salt2} = 10^{17}$ Pa s, $\rho_{salt2} = 2200\ \mathrm{kg\,m^{-3}}$
- $\eta_{anhyd} = 10^{20}$ Pa s, $\rho_{salt2} = 2900\ \mathrm{kg\,m^{-3}}$

The left half of the domain contains salt1, the right half contains salt2 and a block of anhydrite is placed towards the top of the domain. It is of size $l \times h = 200x100m$ (I choose here a 2:1 aspect ratio but the paper explores many different aspect ratios). Its top edge is 100m below the surface:



The original resolution of the paper is 200x400 cells (FDCON is used), i.e. cells of about 10 m in size. This will be difficult to achieve with this STONE , so we will settle for 20 m instead. Simulations are run up to about 500ka. The authors also mention that an initial distance between markers of 10 m is used, i.e. 250x500=125,000 markers.

The initial mass of the system is given by

$$M_0 = \frac{L_x}{2} L_y \rho_{salt1} + \frac{L_x}{2} L_y \rho_{salt2} + lh(\rho_{anhyd} - \rho_{salt}) = 27,514,000,000$$

Stone 68

the corner flow - Subduction benchmark (2)



2D
box
P2pPm1
sparse_matrix
stokes_equation
energy_equation
thermo−mechanical
time_stepping
use_mesher
viscous_rheology
paraview_output
pressure_scaling

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_68`

This fieldstone was developed in collaboration with Iris van Zelst.

Relevant Literature:

Jilles van den Beukel and Rinus Wortel. "Temperatures and shear stresses in the upper part of a subduction zone". In: *Geophys. Res. Lett.* 14.10 (1987), pp. 1057–1060. DOI: 10.1029/GL014i010p01057

P.E. van Keken et al. "A community benchmark for subduction zone modelling". In: *Phys. Earth. Planet. Inter.* 171 (2008), pp. 187–197. DOI: 10.1016/j.pepi.2008.04.015

E.M. Syracuse, P.E. van Keken, and G.A. Abers. "The global range of subduction zone thermal models". In: *Phys. Earth. Planet. Inter.* 183 (2010), pp. 73–90

PE Van Keken, S Kita, and J Nakajima. "Thermal structure and intermediate-depth seismicity in the Tohoku-Hokkaido subduction zones". In: *Solid Earth* 3.2 (2012), pp. 355–364. DOI: 10.5194/se-3-355-2012

P.E. van Keken, I. Wada, N. Sime, and G.A. Abers. "Thermal structure of the forearc in subduction zones: a comparison of methodologies". In: *Geochem. Geophys. Geosyst.* 20 (2019), pp. 3268–3288. DOI: 10.1029/2019GC008334

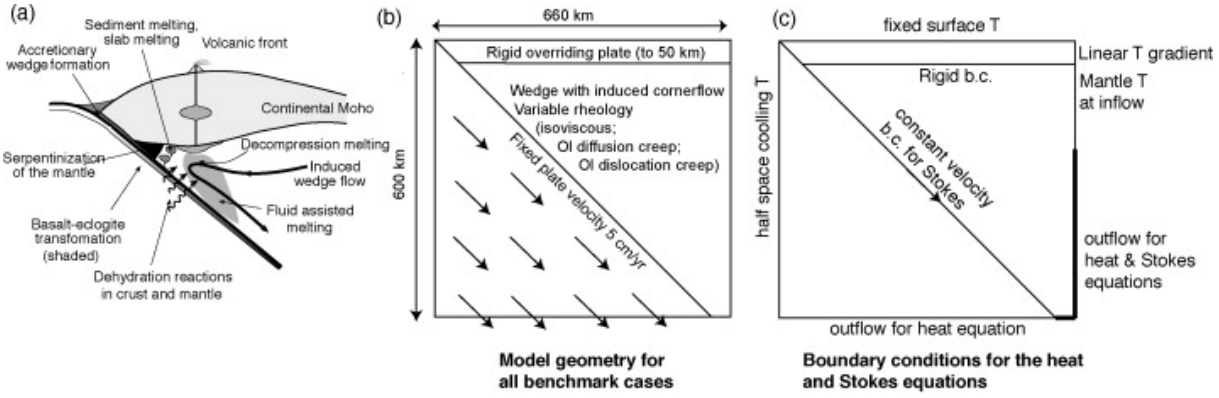
Kazuhiko Ishii and Simon R Wallis. "High-and low-stress subduction zones recognized in the rock record". In: *Earth Planet. Sci. Lett.* 531 (2020), p. 115935. DOI: 10.1016/j.epsl.2019.115935

Nathan Sime and Cian R Wilson. "Automatic weak imposition of free slip boundary conditions via Nitsche's method: application to nonlinear problems in geodynamics". In: *arXiv preprint arXiv:2001.10639* (2020)

Philip C England and Dave A May. "The Global Range of Temperatures on Convergent Plate Interfaces". In: *Geochem. Geophys. Geosyst.* 22 (2021), e2021GC009849. DOI: 10.1029/2021GC009849
Julien Gasc et al. "Deep-focus earthquakes: From high-temperature experiments to cold slabs". In: *Geology* 50 (2022), pp. 1018–1022. DOI: 10.1130/G50084.1

Description of the setup and benchmark cases

The domain is 660 km × 600 km.



Taken from van Keken et al (2008) [339]. (a) Cartoon of the cornerflow model for subduction zone dynamics. (b) Benchmark geometry of a kinematic slab driving flow in the viscous mantle wedge below a rigid overriding plate. (c) Boundary conditions for Stokes and temperature equations.

As shown in the figure above, the inflow boundaries (at both wedge and trench sides) and top of the model have prescribed temperature. The wedge is assumed to be an incompressible fluid that is driven only by the kinematic forcing of the slab. The wedge is confined by the top of the slab and the base of the rigid overriding plate (located at a depth of 50 km). The boundary conditions for the wedge are no-slip below the overriding plate and constant velocity along the top of the slab. The velocity boundary conditions for the boundaries of the wedge are either provided by the Batchelor cornerflow solution (cases 1a and 1b) or based on free inflow/outflow boundaries (cases 1c, 2a, 2b). The velocity field is discontinuous between the slab and the overriding plate. The velocity in the slab is constant (5 $\mathrm{cm\,yr^{-1}}$) and it dips at a 45° angle There is no radiogenic of shear heating. The mantle wedge rheology is either linear viscous (cases 1a,b,c), diffusion creep (case 2a) or dislocation creep (case 2b).

Case 1a The Stokes equations are not solved. Instead the velocity field is prescribed analytically everywhere in the domain, using the corner flow solution in the mantle wedge.

Case 1b - dynamical flow in isoviscous wedge I This case is the same as 1a, except that the solution for the wedge flow is determined by solving the Stokes equations while the Batchelor solution is imposed on the inflow and outflow boundaries. This case tests the ability of the numerical method to accurately reproduce the corner flow solution.

Case 1c - dynamical flow in isoviscous wedge II Same as case 1b, but with stress-free boundary conditions on the mantle wedge.

Case 2a This case is the same as 1c, except that the viscosity of the wedge is given by

$$\eta_{\mathrm{diff,eff}} = \left(\frac{1}{\eta_{\mathrm{diff}}} + \frac{1}{\eta_{\mathrm{max}}} \right)^{-1}$$

and

$$\eta_{\mathrm{diff}} = A_{\mathrm{diff}} \exp\left(\frac{Q_{\mathrm{diff}}}{RT} \right)$$

with $Q_{diff} = 335kJ/mol$ and $A_{diff} = 1.32043 \cdot 10^9$ Pa s, $\eta_{max} = 10^{26}$ Pa s

Case 2b This case is the same as 1c, except that the viscosity of the wedge is given by

$$\eta_{\text{disl,eff}} = \left(\frac{1}{\eta_{\text{disl}}} + \frac{1}{\eta_{\text{max}}} \right)^{-1}$$

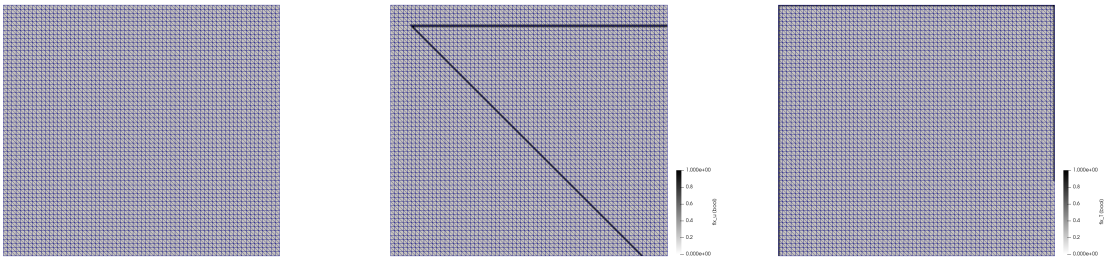
and

$$\eta_{\text{disl}} = A_{\text{disl}} \dot{\varepsilon}^{(1-n)/n} \exp\left(\frac{Q_{\text{disl}}}{nRT} \right)$$

with $Q_{\text{disl}} = 540kJ/mol$, $n = 3.5$ and $A_{\text{disl}} = 28968.6Pa \cdot s^{1/n}$. $\eta_{\text{max}} = 10^{26}$Pa.s

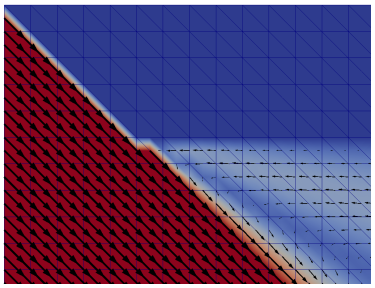
Mesh and boundary conditions

The mesh is composed of square elements which are subdivided in two, as shown here:



Left: mesh composed of 66x60 square elements; Middle: where velocity boundary conditions are prescribed; Right: where temperature boundary conditions are prescribed.

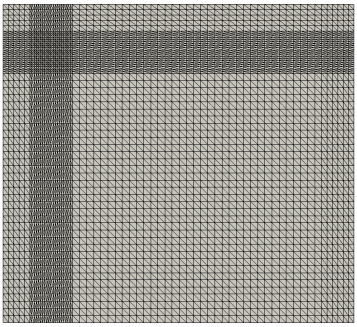
Except for case 1a, the Stokes equation is solved. It is solved on the whole domain but we only care about the solution inside the wedge. The velocity field outside the wedge is overwritten by the desired velocity field before the temperature equation is solved.

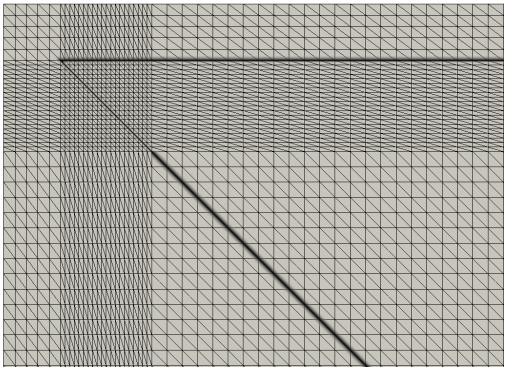


The paper mentions a velocity ramp in section 2.3. While it is not always clear how long it is nor why it should or should not be used in combination with resolution, it nevertheless makes sure that the point at the 'triple junction' has a zero velocity. Through trial an error I was able to arrive to the conclusion that all nodes upstream of this point should too have a zero velocity.

The domain is 660x600km. We can easily generate a mesh made of square elements at any resolution but we need the bottom of the lithosphere to align with a row of nodes, which limits the used resolutions to 66x60 (10km resolution), 132x120 (5km resolution), 264x240 (2.5km resolution) and 330x300 (2km resolution). Higher resolutions require too much memory. The 330x300 mesh was not used for cases 2a and 2b because of the total run time required.

The mesh can be stretched so as to achieve a high(er) resolution in the part of the wedge where some measurements are taken:





Description of the code

Crouzeix-Raviart elements are used for the Stokes equations (see Section **??**) and P_2 elements are used for the temperature.

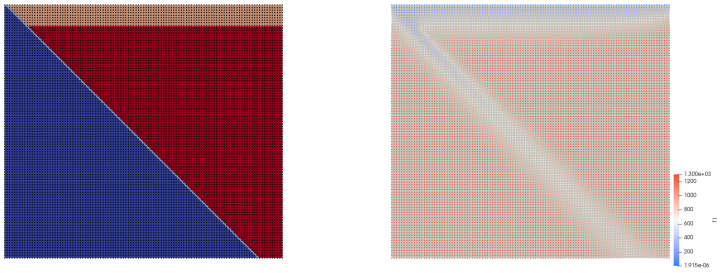
The origin of the coordinate system (x, y) is at the bottom left. The boundary conditions for the wedge are no-slip below the overriding plate and constant velocity along the top of the slab. The velocity boundary conditions for the boundaries of the wedge are either provided by the Batchelor cornerflow solution (case 1b) or based on free inflow/outflow boundaries. The velocity field is discontinuous between the slab and the overriding plate which effectively mimics the fault representative of the seismogenic zone in subduction zones.

For cases 1a,b,c the Stokes equations and the energy equation are not really coupled: the gravity is set to zero, the model is driven by the kinematical boundary conditions and the viscosity is constant, so that temperature does not influence the Stokes solve. Once the velocity field is obtained it is used in the energy equation to compute the steady state temperature field.

For cases 2a,b the viscosity of the mantle wedge is temperature (and strain rate) dependent so that the coupling of the PDEs requires nonlinear iterations. These are as follows:

1. set $\vec{\mathbf{v}}_{old}$=0, $T_{old} = 0$ at all mesh nodes.
2. Solve Stokes equation with $\eta(T_{old}, \dot{\varepsilon}_{old})$ (if iter=0, use $\eta = 10^{21}$ for the whole domain).
3. Relax the velocity solution: $\vec{\mathbf{v}} = relax \cdot \vec{\mathbf{v}} + (1 - relax) \cdot \vec{\mathbf{v}}_{old}$
4. Solve for temperature
5. Relax the temperature solution: $T = relax \cdot T + (1 - relax) \cdot T_{old}$
6. Check for convergence. If $\|\vec{\mathbf{v}} - \vec{\mathbf{v}}_{old}\|_2 < tol\|\vec{\mathbf{v}} + \vec{\mathbf{v}}_{old}\|_2$ i and $\|T - T_{old}\|_2 < tol\|T + T_{old}\|_2$ then exit, otherwise: $T_{old} \leftarrow T$, $\vec{\mathbf{v}} \leftarrow \vec{\mathbf{v}}_{old}$. iter+=1. Go back to 2.

The $relax$ parameter should obviously be between 0 and 1 and set to 0.8. To compare model results each group contributed the temperature field as discrete values T_{ij} on an equidistant grid with 6 km spacing, which is a 111x101 matrix stored row-wise starting in the top left corner.



Measuring grid made of 110x100 points.

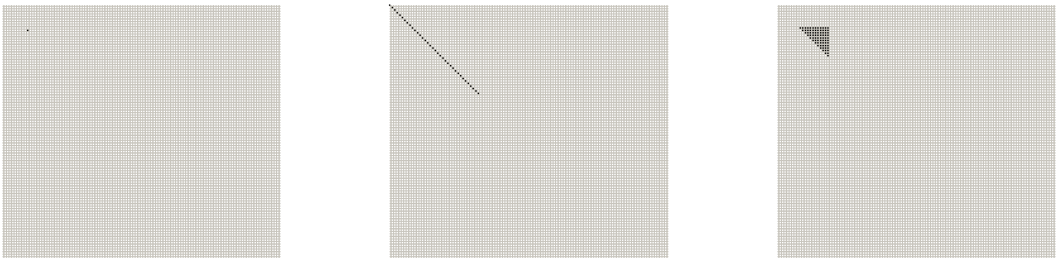
From this grid we have extracted the following measurements for direct comparison:

- the temperature $T_{11,11}$ which is at coordinates (60, 60 km) and just down-stream from the corner point.
- the L2 norm of the slab–wedge interface temperature between 0 and 210 km depth defined by

$$T_{slab} = \sqrt{\frac{1}{36}\sum_{i=1}^{36} T_{ii}^2}$$

- the L2 norm of the temperature in the triangular part of the tip of the wedge, between 54 and 120 km depth:

$$T_{wedge} = \sqrt{\frac{1}{78}\sum_{i=10}^{21}\sum_{j=10}^{i} T_{ij}^2}$$



The black dots indicate the different types of reported quantities in the paper.

In order to interpolate temperature on this grid, we need to find out in which triangle each point of the grid lies. When the mesh is made of irregular triangles, it is a rather costly algorithm. When the mesh is as described above, we first determine in which square element a point lies and then in which of the two triangles.

In general, in order to determine whether a point is inside a triangle, we assume that the reduced coordinates r, s exist and satisfy the following relationships:

$$\begin{aligned} x &= N_1(r,s)x_1 + N_2(r,s)x_2 + N_3(r,s)x_3 \\ y &= N_1(r,s)y_1 + N_2(r,s)y_2 + N_3(r,s)y_3 \end{aligned}$$

where N_i are the linear basis functions inside a triangle, see Section **??**.

We also have the property that $N_1 + N_2 + N_3 = 1$ everywhere inside the element, so that we now have a system of 3 equations for our three unknowns N_1, N_2 and N_3 (having obtained these, we can later easily find r and s). We must then solve:

$$\begin{aligned} x &= N_1x_1 + N_2x_2 + N_3x_3 \\ y &= N_1y_1 + N_2y_2 + N_3y_3 \\ 0 &= N_1 + N_2 + N_3 \end{aligned}$$

which yields

$$N_1 = \frac{(y_2-y_3)(x-x_3)+(x_3-x_2)(y-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)} \qquad N_2 = \frac{(y_3-y_1)(x-x_3)+(x_1-x_3)(y-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)} \qquad N_3 = 1-a-b$$

and then $r = N_2$ and $s = N_3$

To these measurements I have also added the average temperature over the domain

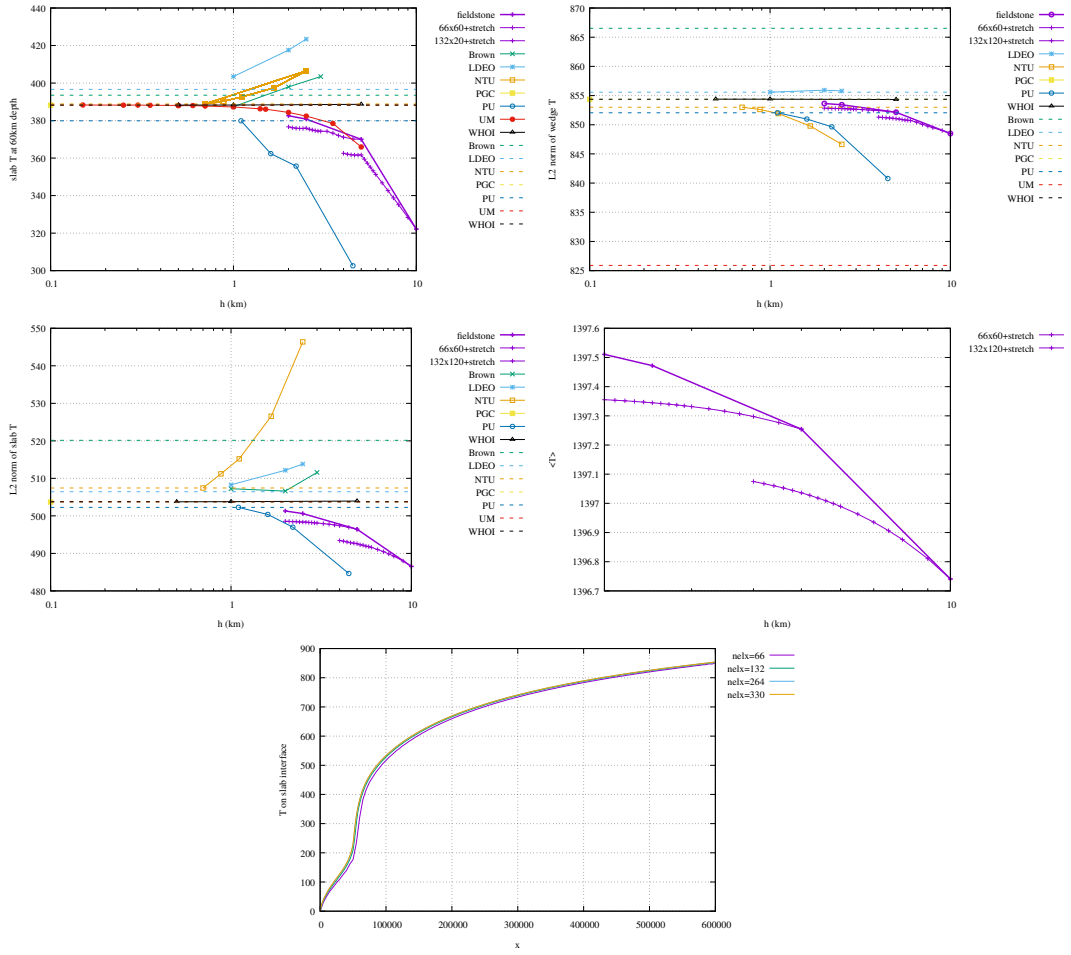
$$T_{avrg} = \frac{1}{L_xL_y}\iint T(x,y)dxdy,$$

the root mean square velocity in the wedge

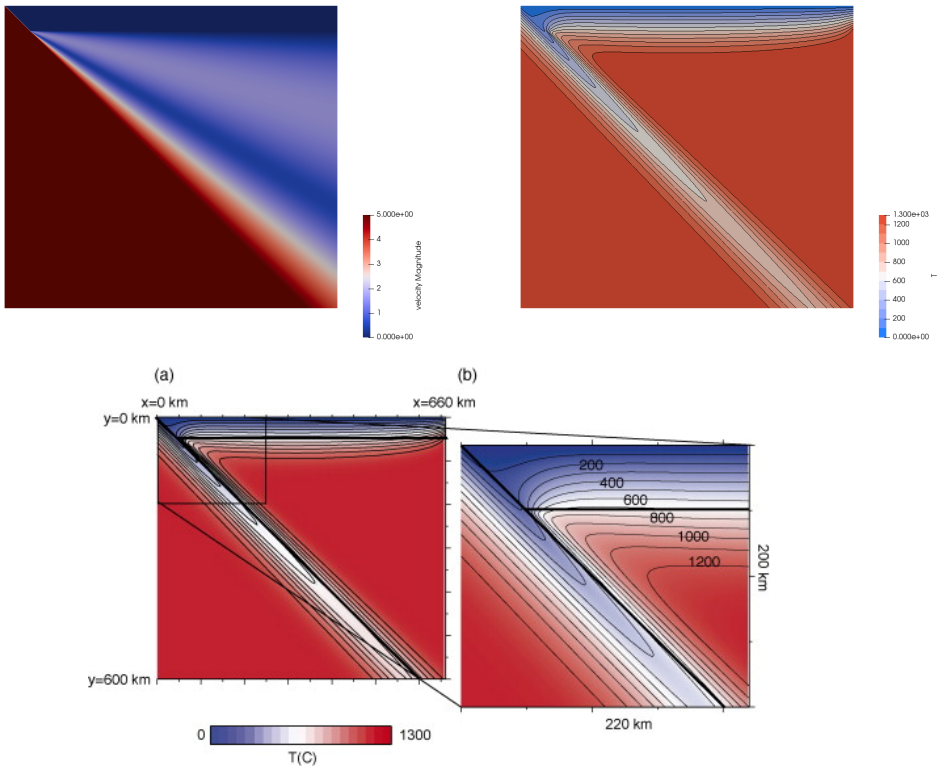
$$v_{rms} = \sqrt{\frac{1}{A_{wedge}}\iint_{wedge}(u^2+v^2)dxdy},$$

and the temperature at the top of the slab (line $y = L_y - x$).

Results - Case 1a

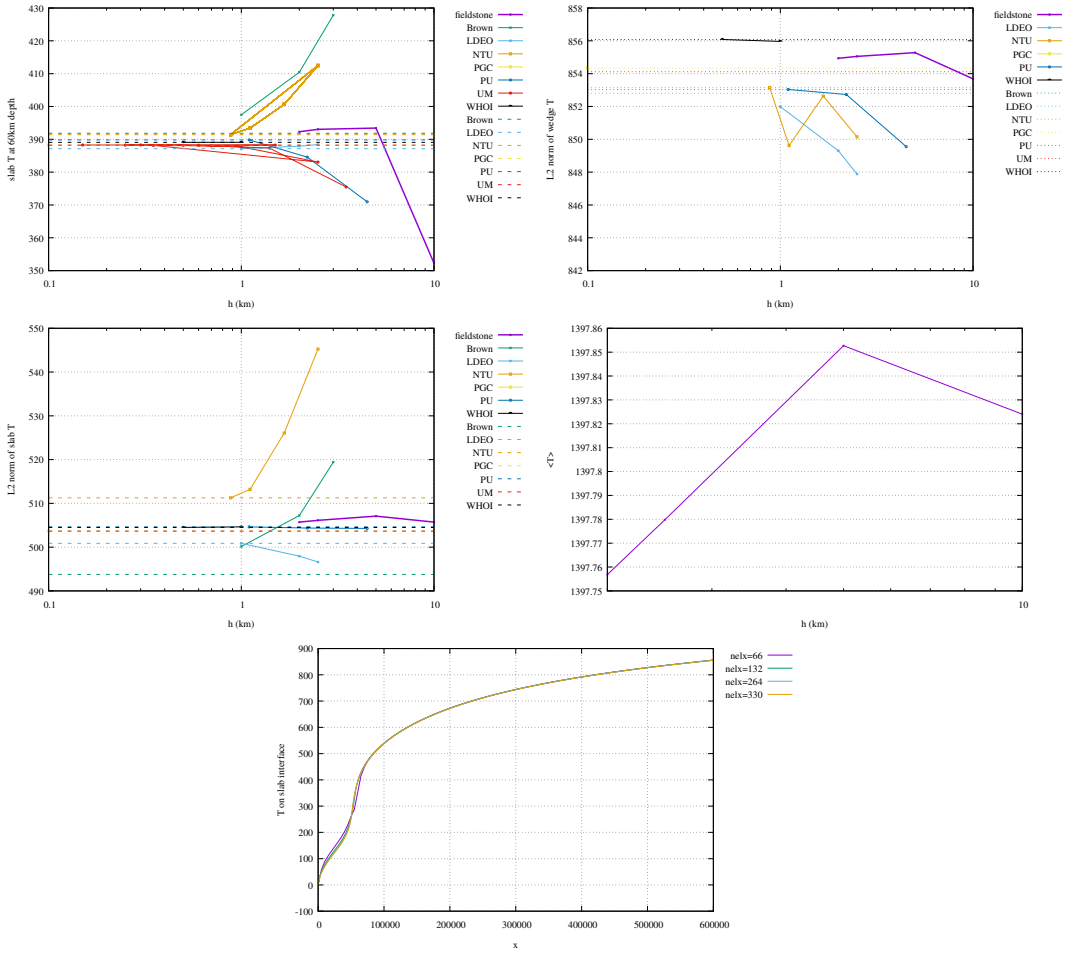


Dashed lines correspond to values reported in Table 2 of [339]. Lines correspond to values communicated to us by P. van Keken.

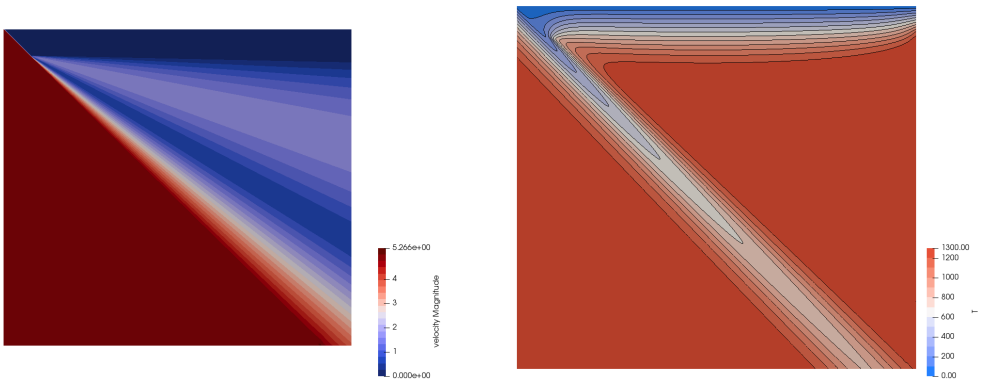


Taken from [339]. Temperature prediction for case 1a. The bold lines indicate the top of the slab and base of the overriding plate. (b) Close up of the top left part of the model.

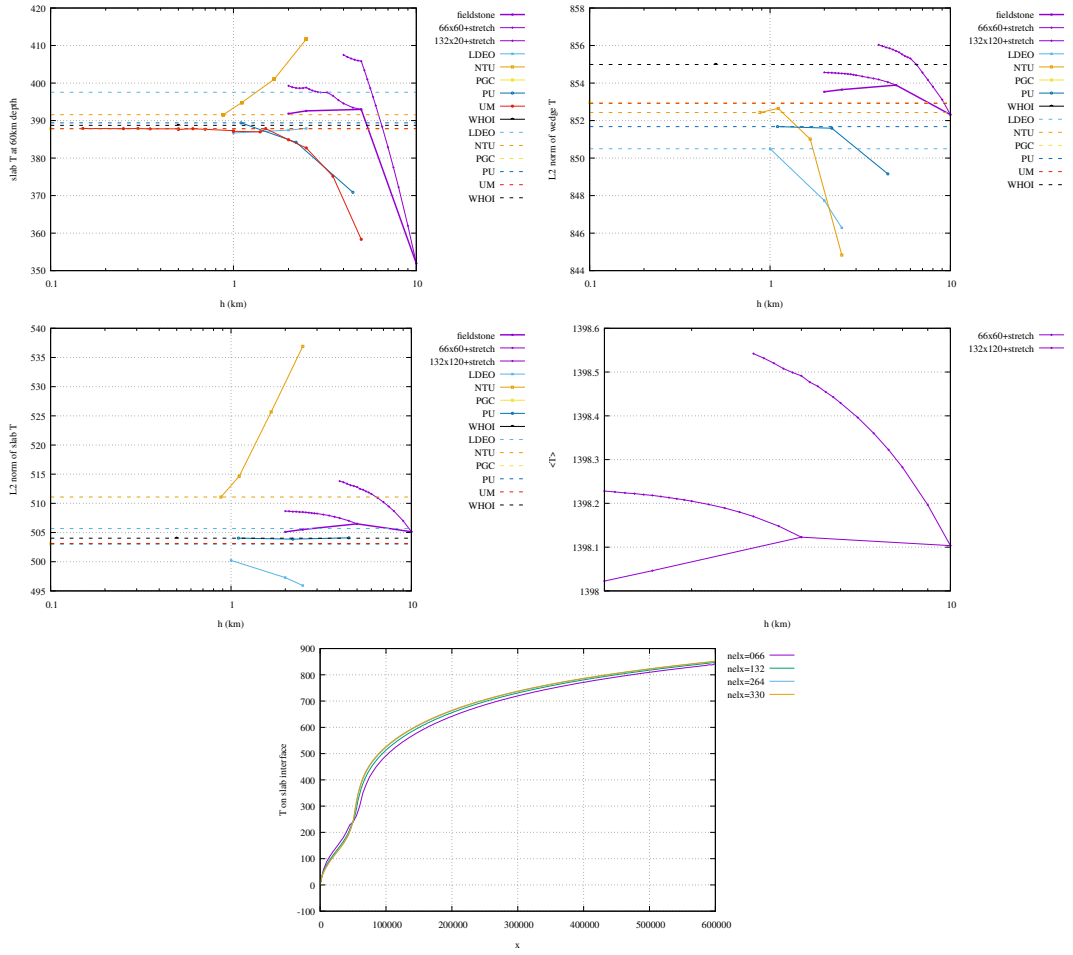
Results - Case 1b



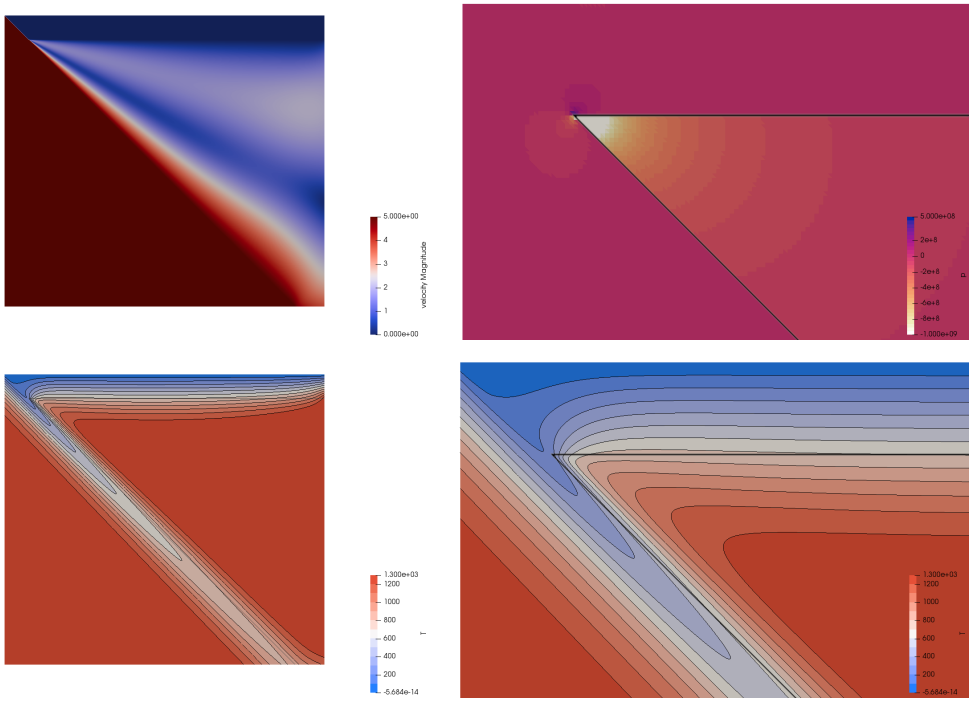
Dashed lines correspond to values reported in Table 2 of [339]. Lines correspond to values communicated to us by P. van Keken.



Results - Case 1c

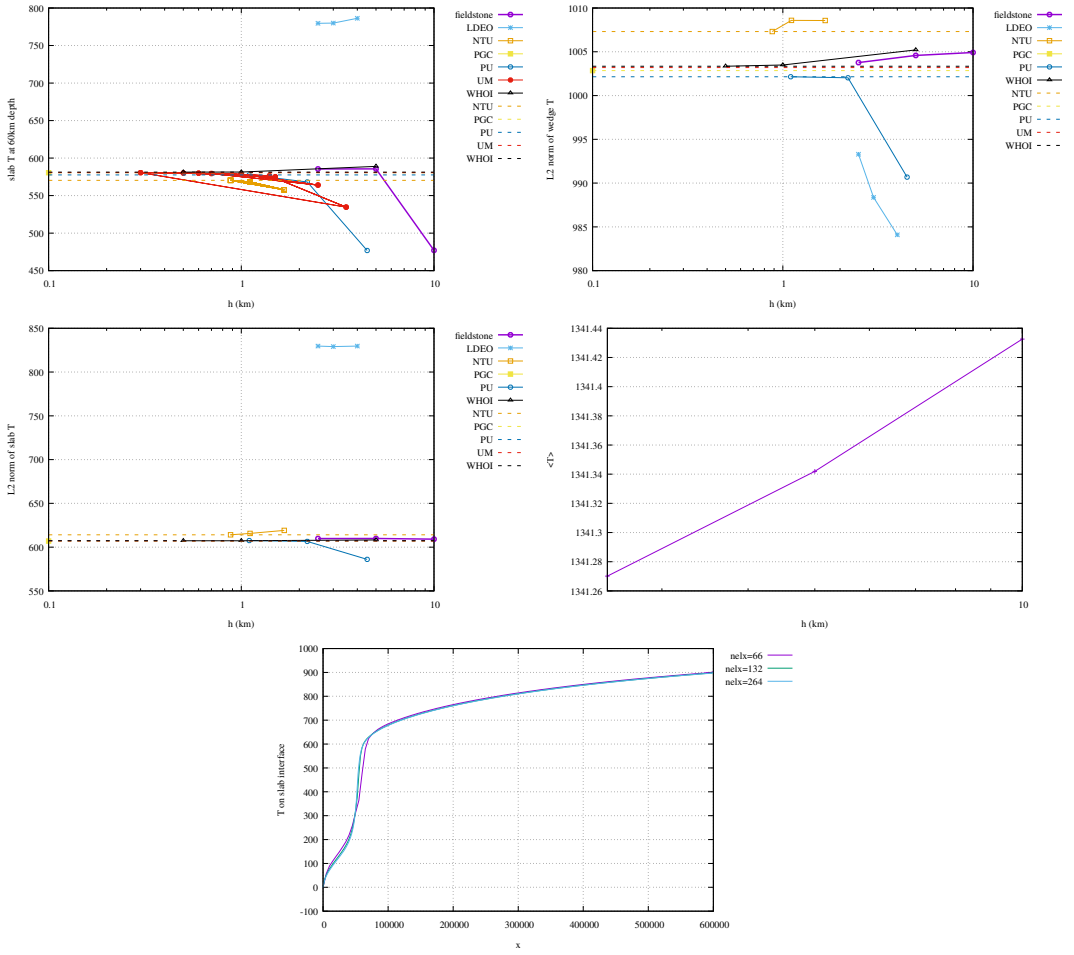


Dashed lines correspond to values reported in Table 2 of [339]. Lines correspond to values communicated to us by P. van Keken.

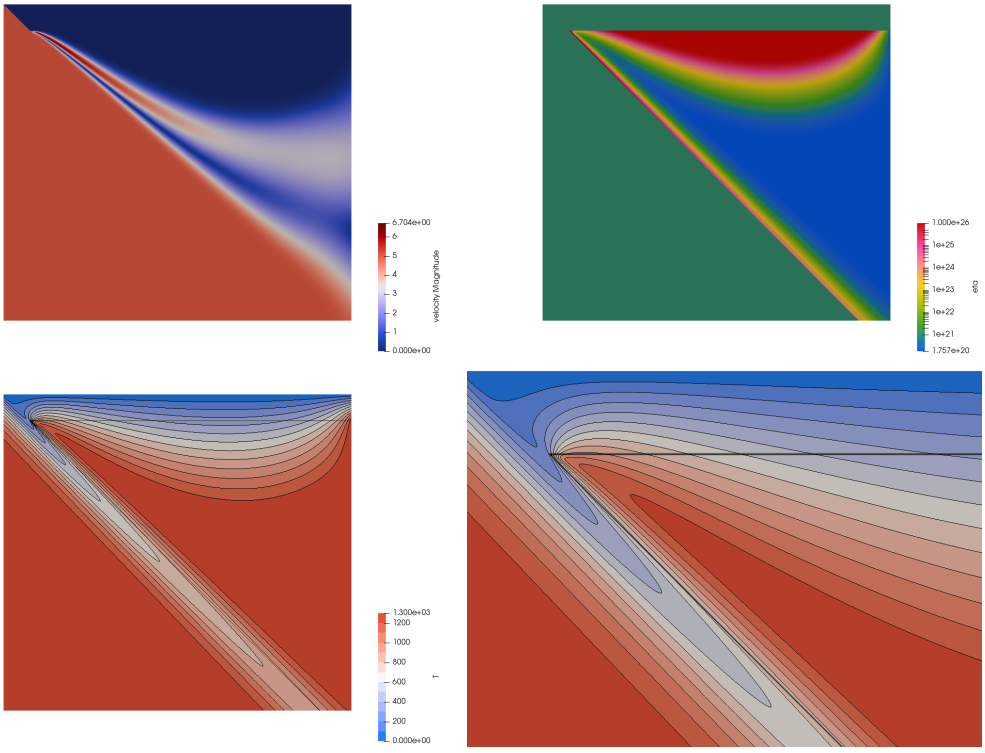


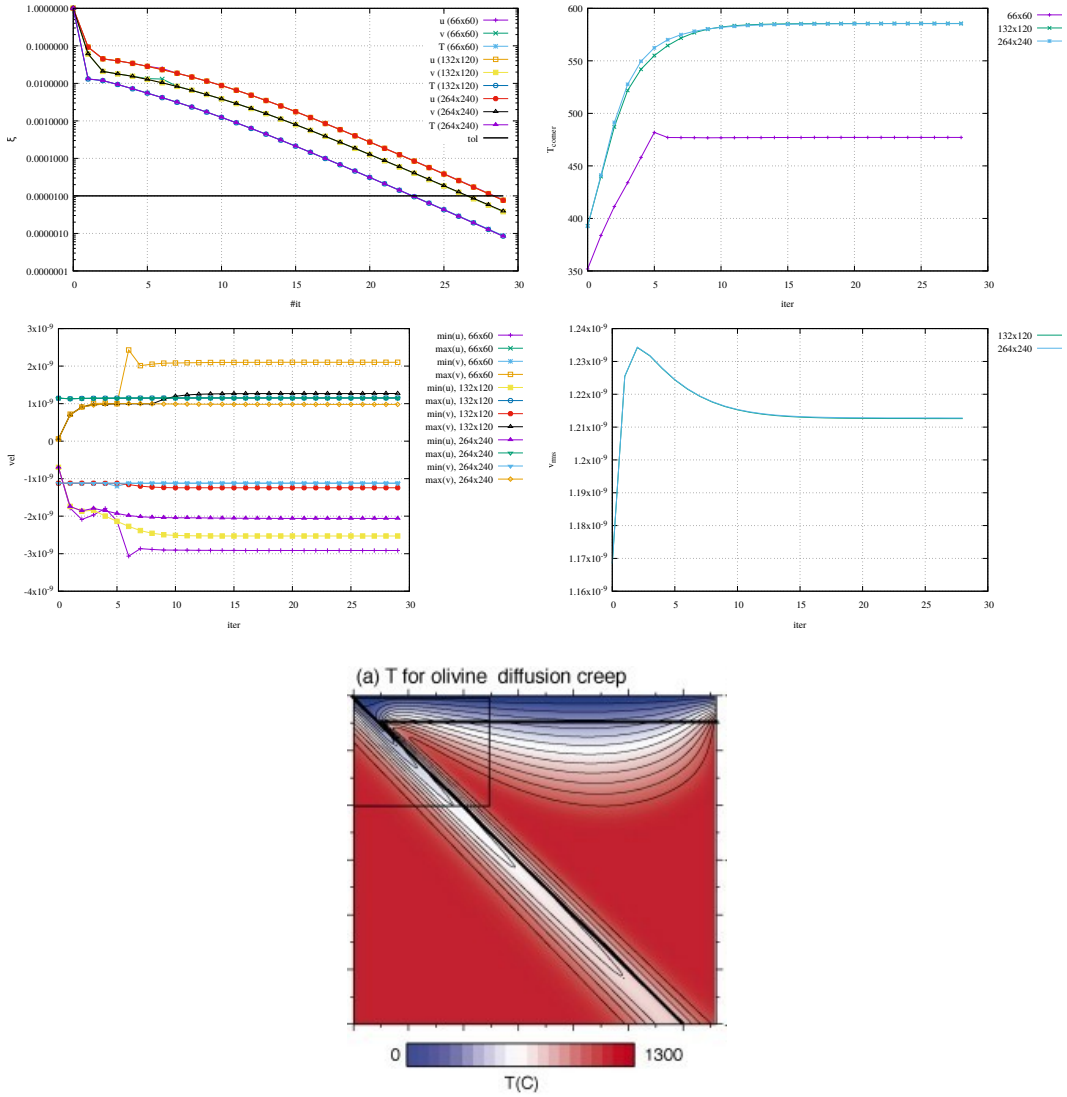
We see that stretching the mesh does not really work since it makes the mesh coarser in the rest of the wedge and therefore alter the global solution.

Results - Case 2a



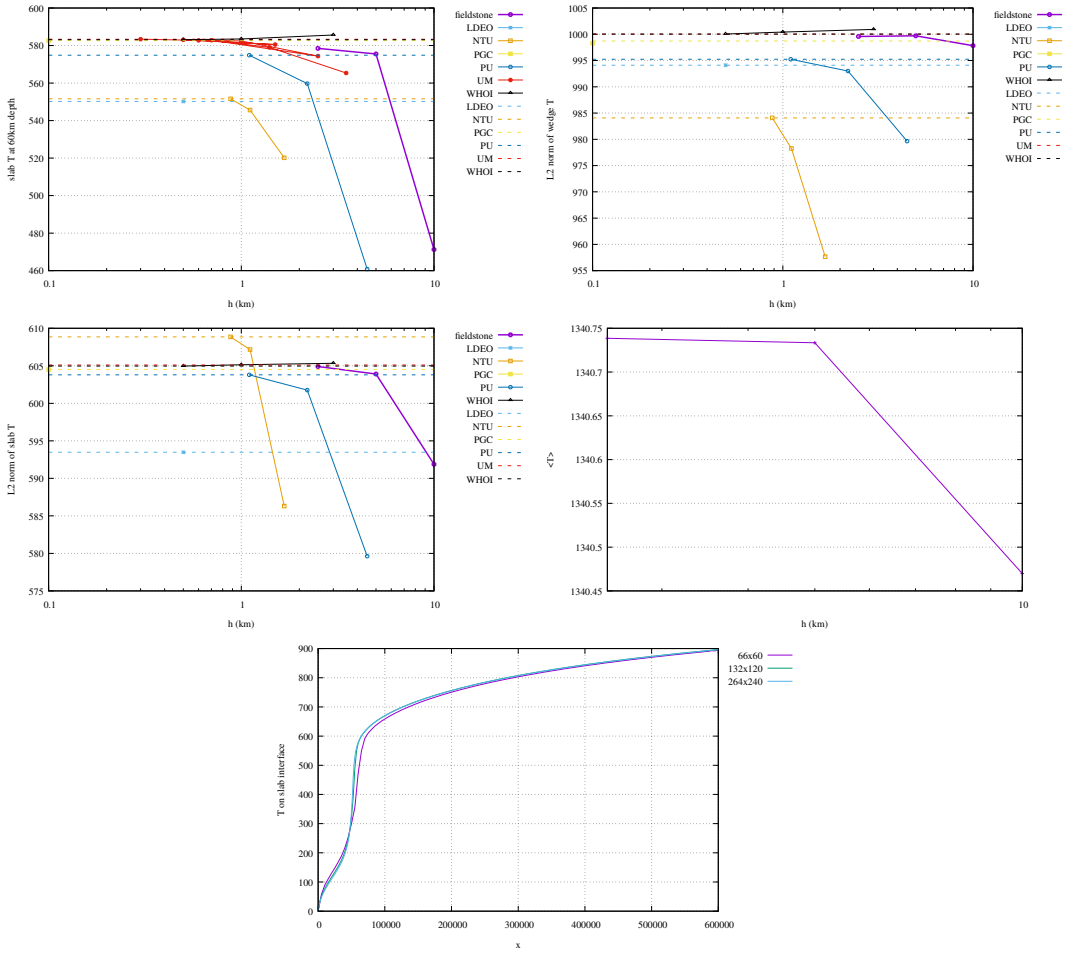
Dashed lines correspond to values reported in Table 3 of [339]. Lines correspond to values communicated to us by P. van Keken.



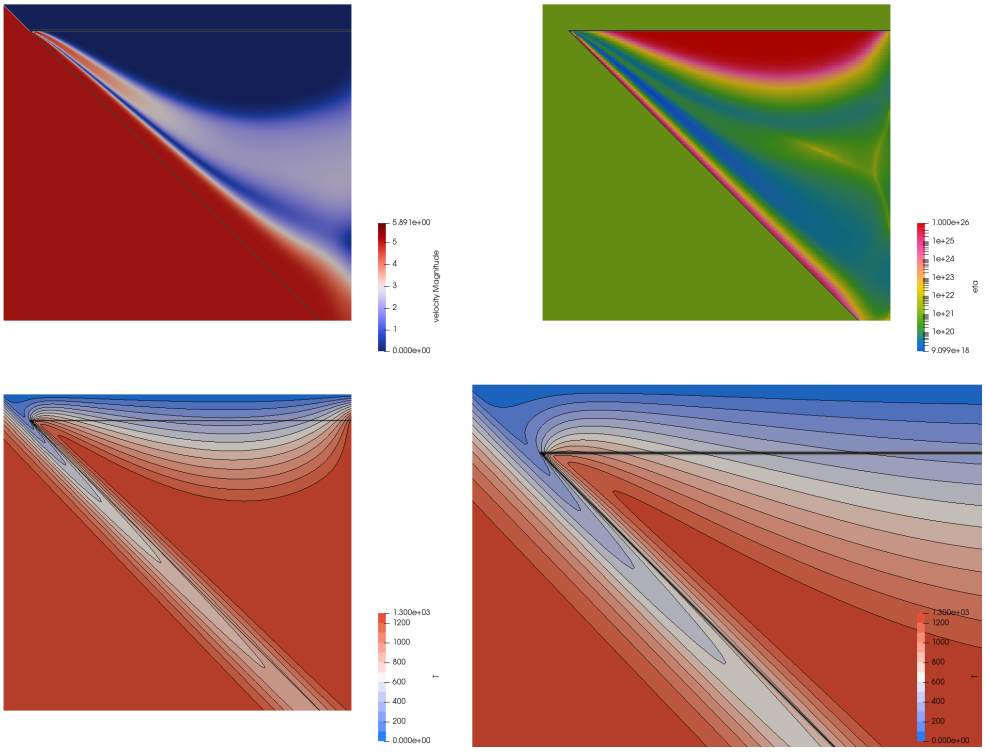


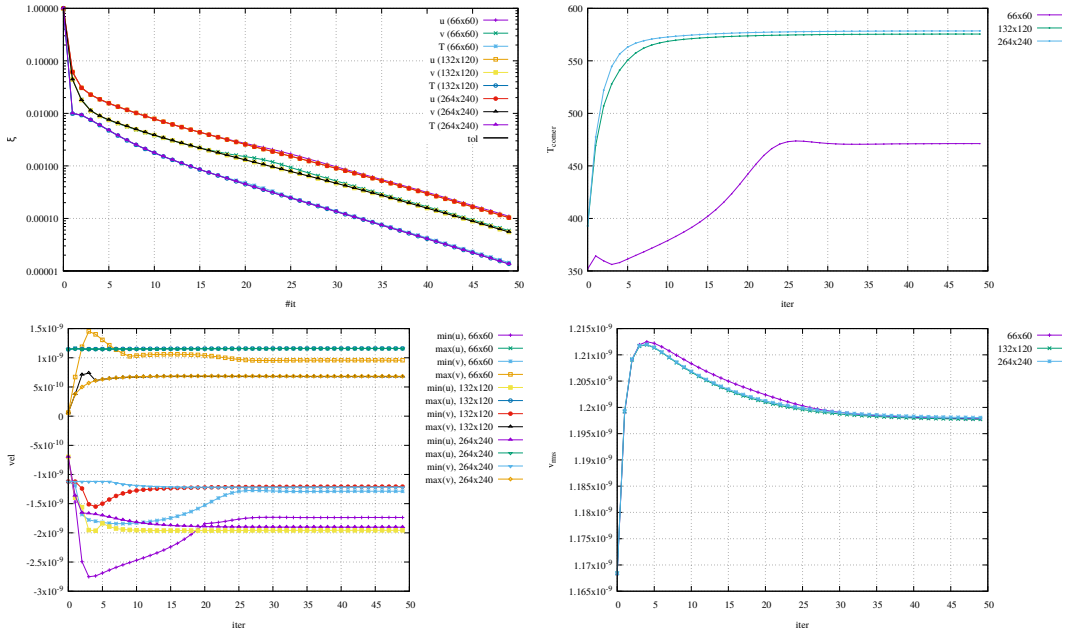
Taken from [339]. (a) Temperature prediction for case 2a with olivine diffusion creep in the mantle wedge. (b) Close up of the top left part of the model, as in Fig. 2b.

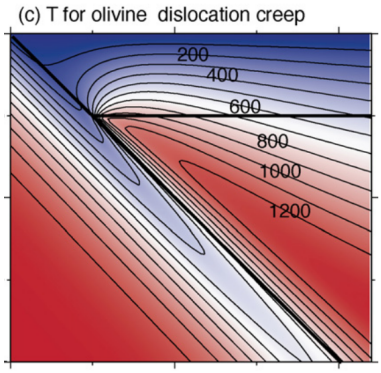
Results - Case 2b



Dashed lines correspond to values reported in Table 3 of [339]. Lines correspond to values communicated to us by P. van Keken.







Taken from [339]. Close up of the model with olivine dislocation creep in the mantle wedge.

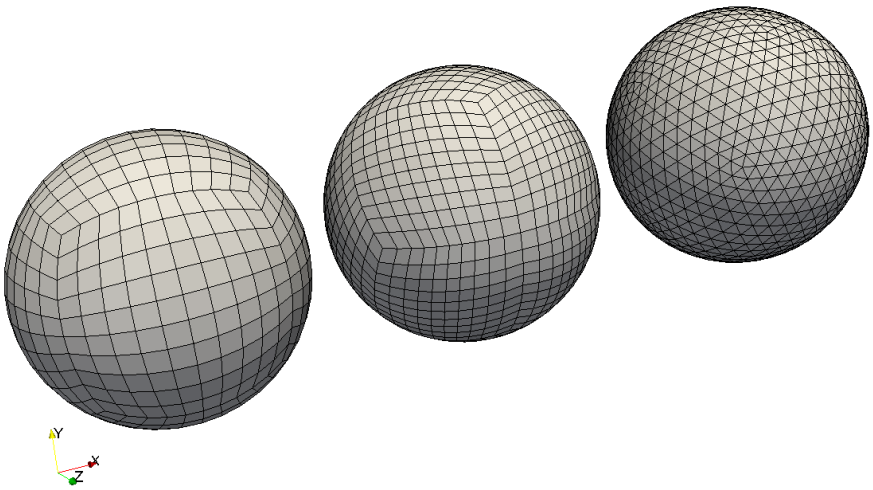
Stone 69

Spherical shell



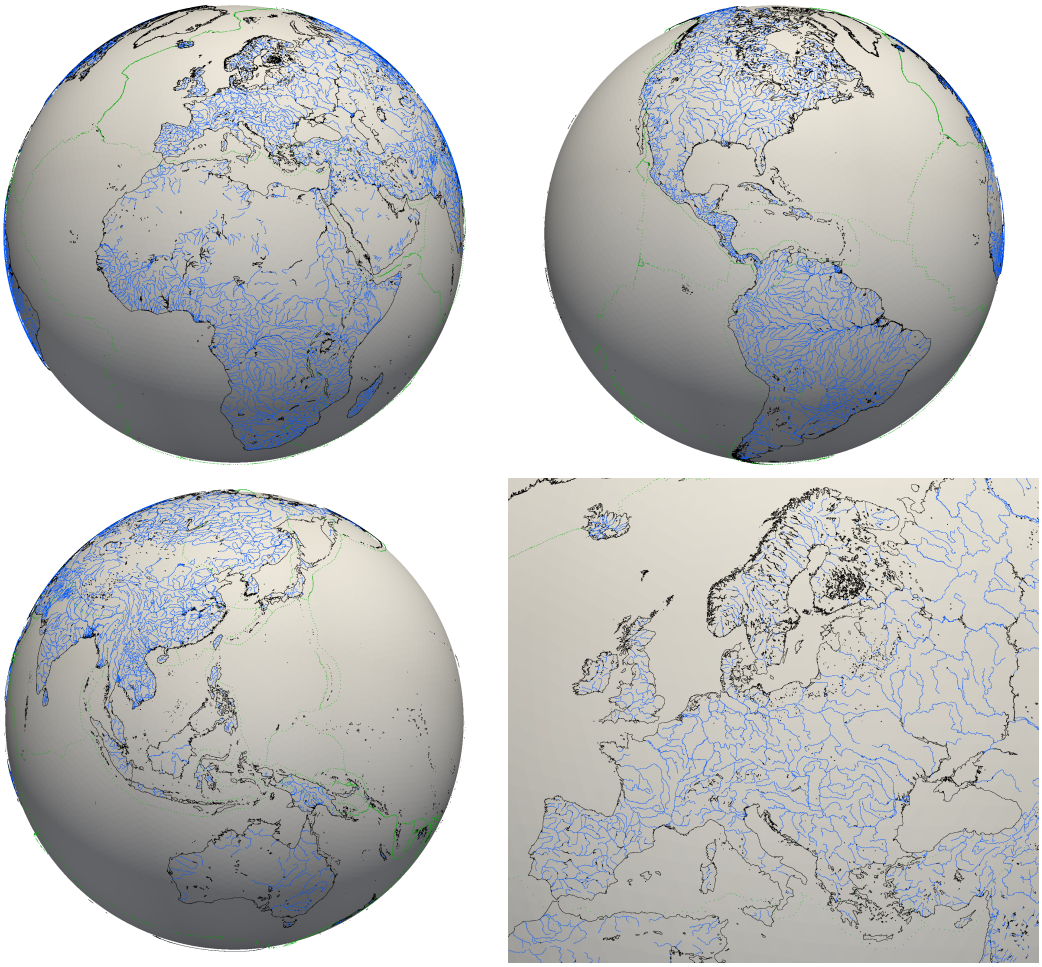
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_69`

I have re-implemented most parts of the GHOST code [327] in python. This stone only generates both a spherical shell and a hollow sphere mesh.

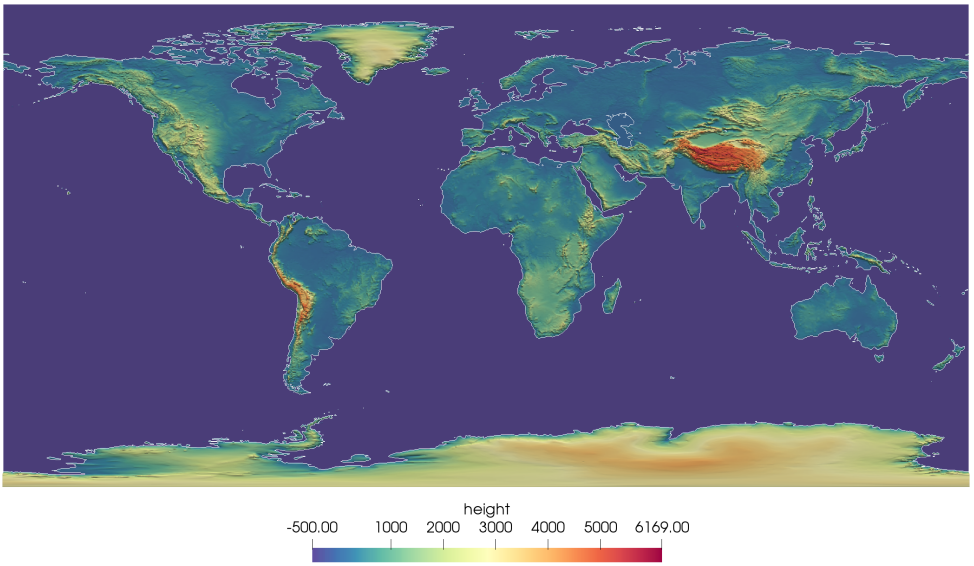


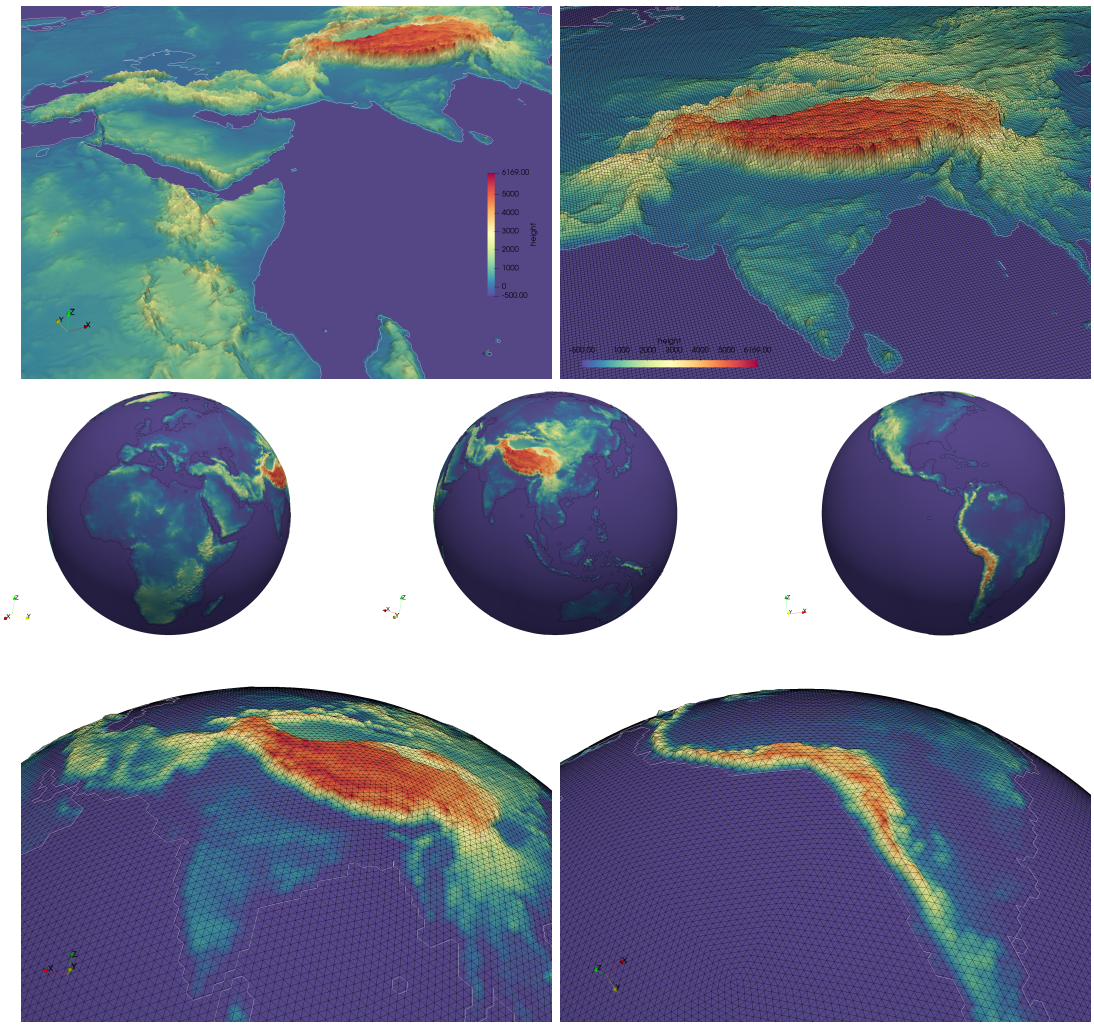
Left: cubed sphere; Middle: Citcom mesh; Right: Icosahedral mesh. All three at level=10.

There are three python codes: *rivers.py*, *coastlines.py* and *plate_boundaries.py* which allow the user to choose a radius on which the data will projected. There are 3 folders *rivers*, *coastlines* and *plate_boundaries* which contain the relevant data per 'continent', i.e. Africa, Asia, Europe, North America and South America in this case. I have downloaded these data off the internet some 15 years ago and I can unfortunately not assign these references, except for the plate boundaries which come from Bird [21] (2003).



The data for the Digital Elevation Model (DEM) is available at `http://www.temis.nl/data/topo/dem2grid.html` which has a 0.25 degree resolution (1440x720 points).





Icosahedral mesh, level=100. Vertical exaggeration=20.

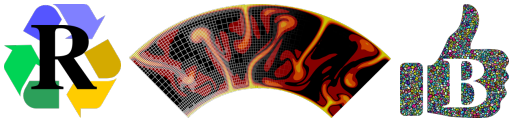


Los Alamos coastlines with grid lines.

Coastlines, plate boundaries, topography, grid lines available at `https://www.earthmodels.org/`.

Stone 70

Visco-Viscoplastic rheology in crustal model ($Q_2 \times Q_1$)



2D
box
Q2Q1
sparse_matrix
numerical_benchmark
stokes_equation
thermo_mechanical
nonlinear
viscous_rheology
plastic_rheology
residual
paraview_output
pressure_scaling
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_70`

The setup is borrowed from Duretz, de Borst, Yamato, and Le Pourhiet [99] (2020). Two major differences with the paper: 1) no elasticity and 2) no Newton solver.

The model consists of a 100×30 km slice of Westerly Granite (Hansen & Carter, 1983), which comprises an imperfection at the center of the domain. This weak inclusion with a 2 km radius serves to initiate localized deformations. The model configuration is thus perfectly symmetric. The model includes a vertical temperature gradient (-15 °C km^{-1}) (starting at 20 °C at the surface), a constant density (2700 kg m^{-3}), and the acceleration of gravity ($g_y = -10$ m s^{-2}). The inclusion is characterised by $\eta = 1 \times 10^{20}$ Pa s. The domain is subjected to kinematic boundary conditions, which cause a pure shear stress state:

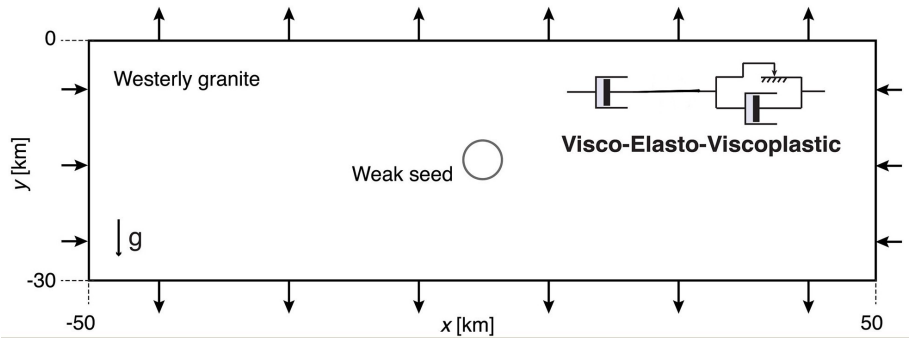


Figure taken from [99]

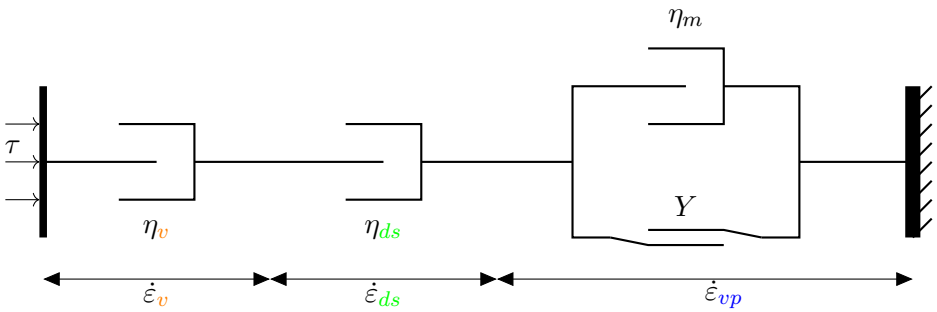
All boundaries are free slip. The material can behave in a ductile manner in the lower, hot part of the domain (displaying temperature-dependent power law creep), and in a viscoplastic manner in the upper, cold part of the domain. The parameters are $A = 3.1623 \cdot 10^{-26} i$ Pa^{-n}/s, $Q = 186.5 \times 10^3$ J mol^{-1}, $n = 3.3$, $c = 50$MPa, $\phi = \arctan(0.6) \simeq 31^\circ$ and $\eta^m = 10^{21}$Pa s (see figure).

The background strainrate is set such that $\dot{\varepsilon}_{bc} = 10^{-15}$ s^{-1}, i.e. $u_{bc} = \pm\dot{\varepsilon}_{bc} L_x/2$ on the sides and $v_{bc} = \pm\dot{\varepsilon}_{bc} L_y/2$ on the top and bottom boundaries. Pressure is normalised such that it is on average zero on the top boundary. Picard iterations are used. The 2-norm of the non-linear residual is monitored, as well as the following quantities (i and $i-1$ are two consecutive nonlinear iterations):

$$\xi_u = \frac{\|u^i - u^{i-1}\|_2}{\|u^i + u^{i-1}\|_2} \qquad \xi_v = \frac{\|v^i - v^{i-1}\|_2}{\|v^i + v^{i-1}\|_2} \qquad \xi_p = \frac{\|p^i - p^{i-1}\|_2}{\|p^i + p^{i-1}\|_2}$$

For the first nonlinear iteration there is no pre-existing velocity or pressure field so we simply use a constant strain rate value of 1×10^{-15} s^{-1}.

As shown on the (modified) figure above, the rheological assemblage is as follows:



The dislocation creep viscosity η_{ds} and effective viscoplastic viscosity η_{vp} are given by

$$\eta_{ds}(\dot{\varepsilon}) = \frac{1}{2} A^{-1/n} \dot{\varepsilon}^{-1+1/n} \exp \frac{Q}{nRT} \qquad \text{and} \qquad \eta_{vp}(\dot{\varepsilon}) = \frac{p \sin\phi + c\cos\phi}{2\dot{\varepsilon}} + \eta_m$$

Note that these expressions are functions of $\dot{\varepsilon}$, the nature of which will be specified later. As documented in Section 5.23, the partitioning of the total strain rate into a viscous creep component and a viscoplastic component is not trivial. The algorithm then works as follows:

1. Assume we know $\dot{\varepsilon}_T$ (from previous iteration), as well as the yield value $Y = p\ \sin\phi + c\ \cos\phi$ and η_m (Here the 'T' stands for 'total').

2. We start by assuming that the plasticity element is not active ($\dot{\varepsilon}_{vp} = 0$, $\dot{\varepsilon}_{ds} + \dot{\varepsilon}_v = \dot{\varepsilon}_T$):

   We use the technique explained here after to compute the partitioning of the strain rate and recover the stress τ and the dislocation creep effective viscosity is given by

$$\begin{aligned} \eta_{ds} &= \frac{1}{2} A^{-1/n} \dot{\varepsilon}_{ds}{}^{-1+1/n} \exp\left(\frac{Q}{nRT}\right) \\ \eta_v &= \frac{\tau}{2\dot{\varepsilon}_v} \end{aligned}$$

```
eta_ds = 0.5*A**(-1./nnn)*np.exp(Q/nnn/Rgas/T)*eps_ds**(1./nnn-1.)
eps_v=tau/2/eta_v
```

   The effective viscosity of both dampers is given by

$$\eta_{ieff} = \frac{1}{1/\eta_v + 1/\eta_{ds}}$$

```
etaeff=1./(1./eta_v+1./eta_ds)
```

3. if $\tau = 2\eta_{eff}\dot{\varepsilon}_T < Y$ the stress is below the yield stress value and the plasticity element is indeed not active. Simply return η_{eff} in the material model.

```
eps_vp=0
```

4. if $\tau = 2\eta_{eff}\dot{\varepsilon}_T > Y$ the stress is above the yield value, which is not allowed. In this case the plastic element must be present and active and the linear viscous and dislocation creep viscous dampers are then in series with the viscoplastic element. They respectively deform with a strain rate $\dot{\varepsilon}_v$, $\dot{\varepsilon}_{ds}$ and $\dot{\varepsilon}_{vp}$ (all under the same tress τ) and we have $\dot{\varepsilon}_T = \dot{\varepsilon}_v + \dot{\varepsilon}_{ds} + \dot{\varepsilon}_{vp}$ so:

$$\begin{aligned}
\dot{\varepsilon}_T - \dot{\varepsilon}_v - \dot{\varepsilon}_{ds}(\tau) &= \dot{\varepsilon}_{vp}(\tau) \\
\dot{\varepsilon}_T - \dot{\varepsilon}_v - \dot{\varepsilon}_{ds}(\tau) &= \frac{\tau}{2\eta_{vp}} \\
\dot{\varepsilon}_T - \dot{\varepsilon}_v - \dot{\varepsilon}_{ds}(\tau) &= \frac{\tau}{2\left(\frac{Y}{2\dot{\varepsilon}_{vp}(\tau)} + \eta_m\right)} \\
\dot{\varepsilon}_T - \dot{\varepsilon}_v - \dot{\varepsilon}_{ds}(\tau) &= \frac{\tau}{2\left(\frac{Y}{2(\dot{\varepsilon}_T - \dot{\varepsilon}_v - \dot{\varepsilon}_{ds}(\tau))} + \eta_m\right)} \\
2[\dot{\varepsilon}_T - \dot{\varepsilon}_v(\tau) - \dot{\varepsilon}_{ds}(\tau)]\left(\frac{Y}{2(\dot{\varepsilon}_T - \dot{\varepsilon}_v - \dot{\varepsilon}_{ds}(\tau))} + \eta_m\right) &= \tau \\
Y + 2(\dot{\varepsilon}_T - \dot{\varepsilon}_v(\tau) - \dot{\varepsilon}_{ds}(\tau))\eta_m &= \tau
\end{aligned}$$

We must then find the zero of the function $\mathcal{F}(\tau)$:

$$\begin{aligned}
\mathcal{F}(\tau) &= Y + 2[\dot{\varepsilon}_T - \dot{\varepsilon}_v(\tau) - \dot{\varepsilon}_{ds}(\tau)]\eta_m - \tau \\
&= Y + 2\left[\dot{\varepsilon}_T - \frac{\tau}{2\eta_v} - A\tau^n \exp\left(-\frac{Q}{RT}\right)\right]\eta_m - \tau
\end{aligned}$$

Because dislocation creep involves the n-th power of the stress (and n need not be an integer!) finding the value of the stress for which $\mathcal{F} = 0$ is not straightforward. This equation can be solved with a Newton-Raphson algorithm and the iterations will be of the form:

$$\tau_{n+1} = \tau_n - \frac{\mathcal{F}(\tau_n)}{\mathcal{F}'(\tau_n)}$$

where the derivative of the function $\mathcal{F}$ with respect to τ reads:

$$\frac{\partial \mathcal{F}}{\partial \tau} = -\frac{\eta_m}{\eta_v} - 2\eta_m n A \tau^{n-1} \exp\left(-\frac{Q}{RT}\right) - 1 \tag{70.1}$$

Iterations require a starting value for τ: we logically use the one obtained just above. Iterations stop when $\mathcal{F} < 10^{-6}$. A maximum of 30 iterations is allowed.

```
while abs(func)>1e-6:
    eps_ds=A*tau**nnn*np.exp(-Q/Rgas/T)
    eps_v=tau/2/eta_v
    func=Y + 2*(E2-eps_ds-eps_v)*eta_m - tau
    funcp=-eta_m/eta_v -2*eta_m*nnn*A*tau**(nnn-1)*np.exp(-Q/Rgas/T)-1
    tau-=func/funcp
    it+=1
    if it>30:
        break
#end while
```

5. Once τ has been found, one can compute

$$\begin{aligned}
\dot{\varepsilon}_{ds}(\tau) &= A\tau^n \exp\left(-\frac{Q}{RT}\right) \qquad \Rightarrow \qquad \eta_{ds}(\dot{\varepsilon}_{ds}) = \frac{1}{2}A^{-1/n}\dot{\varepsilon}_{ds}{}^{-1+1/n} \exp\frac{Q}{nRT} \\
\dot{\varepsilon}_v(\tau) &= \frac{\tau}{2\eta_v}
\end{aligned}$$

```
eps_ds=A*tau**nnn*np.exp(-Q/Rgas/T)
eta_ds = 0.5*A**(-1./nnn)*np.exp(Q/nnn/Rgas/T)*eps_ds**(1./nnn-1.)
eps_v=tau/2/eta_v
```

then $\dot{\varepsilon}_{vp} = \dot{\varepsilon}_T - \dot{\varepsilon}_v(\tau) - \dot{\varepsilon}_{ds}(\tau)$, and then

$$\eta_{vp}(\dot{\varepsilon}_{vp}) = \frac{p\sin\phi + c\cos\phi}{2\dot{\varepsilon}_{vp}} + \eta_m$$

6. Ultimately the effective viscosity at every quadrature point is computed as follows:

$$\eta_{eff} = \left(\frac{1}{\eta_v} + \frac{1}{\eta_{ds}} + \frac{1}{\eta_{vp}}\right)^{-1}$$

```
etaeff=1/(1/eta_v+1/eta_ds+1/eta_vp)
```

By construction, the viscosity is also maintained between $\eta_m \geq 1 \times 10^{19}$ Pa and $\eta_v = 1 \times 10^{25}$ Pa.

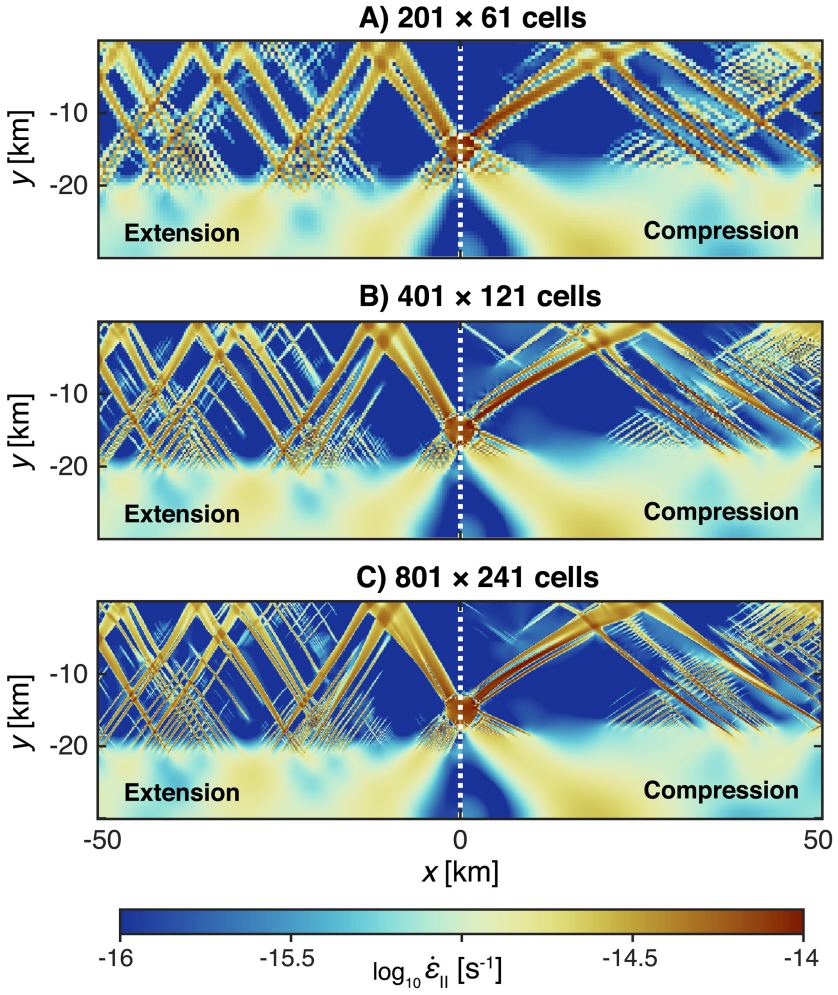
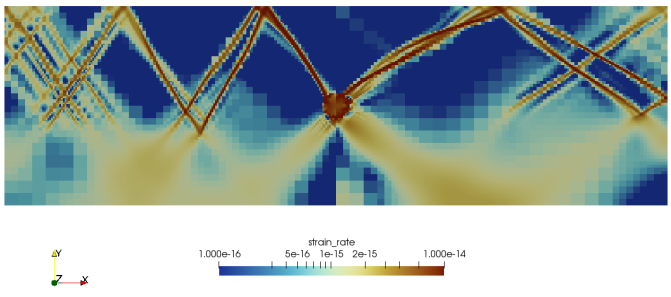
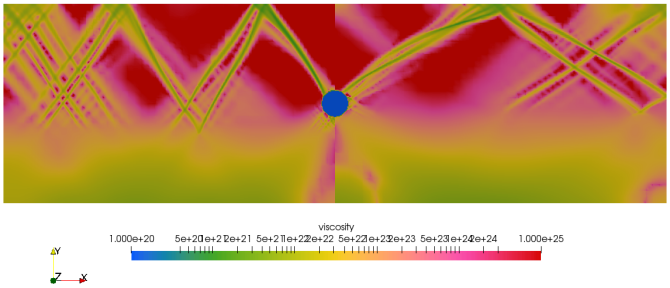


Figure taken from [99]

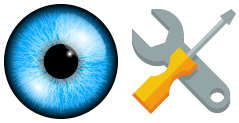




Obtained with ASPECT (see cookbook)

Stone 71

S40RTS in an annulus



2D
annulus
Q2Q1
sparse_matrix
stokes_equation
energy_equation
thermo_mechanical
time_stepping
viscous_rheology
buoyancy_driven
vrms ??
paraview_output
pressure_scaling
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_71`

This stone was developed in collaboration with Rens Elbertsen.

I neeed: - free slip bc - max res ? - crust/lith - rotated plane - compare with submachine

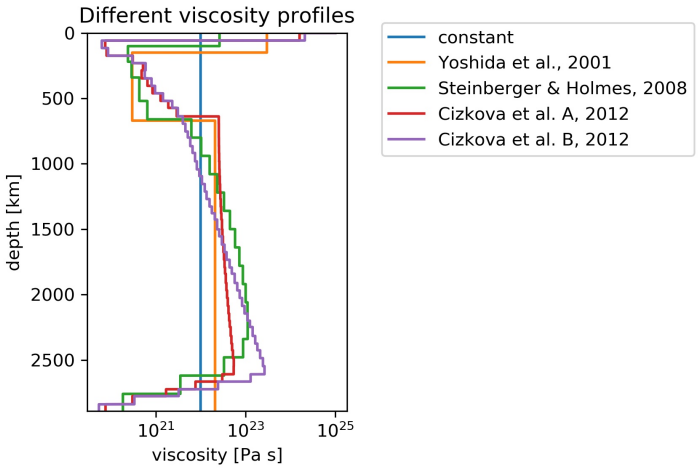
Radial viscosity profiles

In this section we describe the different density profiles used to in the convection model. We assume that viscosity is purely a function of depth. The user can choose a viscosity profile by changing the parameter `case`.

Five radial viscosity profiles are available:

- The first viscosity profile is a constant viscosity for all depths of 10^{22} Pa s. This value is an estimated value of what is normally found in the literature. This is `visc_case = 0`.
- The second viscosity profile comes from Yoshida et al (2001) [357]. It uses three different regions: lithosphere (0 km to 150 km), upper mantle (150 km to 670 km) and lower mantle (670 km to 2900 km). The function uses an if-statement to return the right value for the given depth. This is `case = 1`.
- The third viscosity profile comes from Steinberger & Holmes (2008) [306] which is comparable to [307], but of the latter no available data was available. Data is read from the file `visc_STH008.dat`. This is `case = 2`.

- The fourth and fifth profile come from Ciskova et al (2012) [76]. Data is read from the file `visc_CIVS12.dat`. The paper showcases two main families of radial viscosity profiles in literature. Family A, which has a sharp increase below the 660 km transition zone and remains constant for most of the lower mantle and family B which is much smoother over the transition zone and increases with depth in the lower mantle. Family A can be chosen by `case = 3` and family B can be chosen by `case = 4`.



Cross section of the Earth

Several parameters are implemented which can be modified to get any cross section of the Earth.

The parameter `lon_start` specifies at which longitude we start reading data from the S40RTS model. The model always starts filling the annulus on the right ($x = 1, y = 0$ on the unit circle) continuing in the counter-clockwise direction with increasing longitudes. If we set the parameter `plane_angle` (explained in the next subsection) to 0 we have an equatorial cross section through the Earth that starts at the longitude `lon_start`.

The parameter `plane_angle` indicates the angle the plane of the cross section is making with the equatorial plane. The longitude specified by `lon_start` always is at latitude 0. Latitudes will be increased as a result of the `plane_angle` on the northern hemisphere and decreased on the southern hemisphere. Setting the `plane_angle` to 90 gives us a cross section through both poles and the longitude specified by `lon_start`.

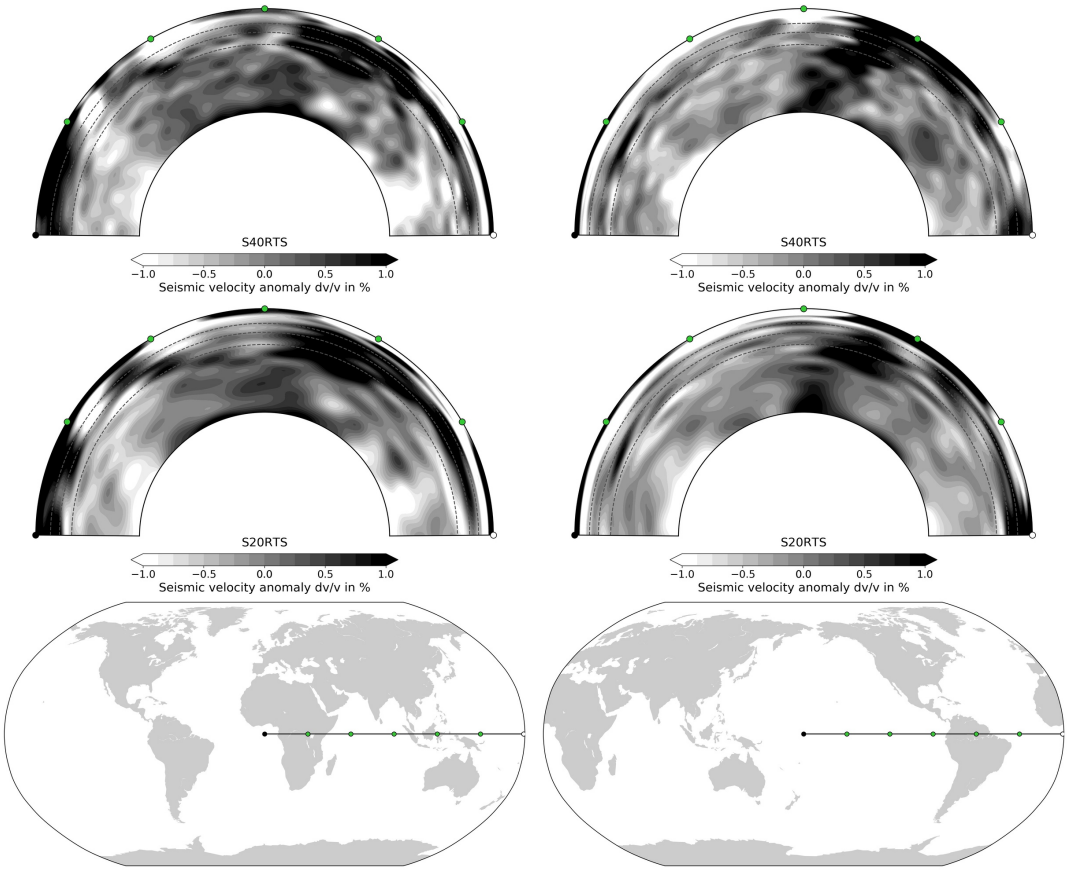
Loading of the Spherical Harmonics, computing $\delta \ln V_s$

In this stone we wish to use the shear wave velocity model S40RTS [264] as the main source of information to build a temperature anomaly/density anomaly model of the mantle.

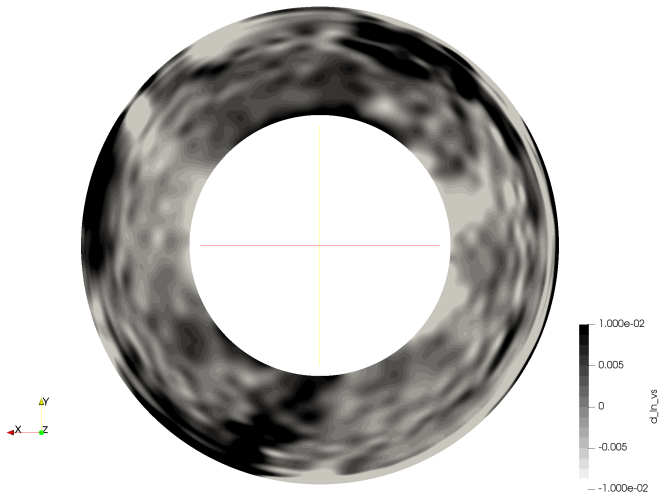
Variations of the shear wave velocity are stored in the form of spherical harmonics coefficients. We make use of an already existing code written by Ross Ronan Maguire (available at `https://github.com/romaguir/sph_models` to read the values of $d \ln V_S$ for a section of the Earth. The code is modified such that we read in $d \ln V_S$ for every nodal point in the annulus domain. Note that the returned value had to be scaled down by a factor of $\sqrt{2}$ to obtain identical figures to those of the original paper or those obtained with SubMachine[1].

There are three different functions in the original code that enables us to read the spherical harmonics data: `read_splines`, `read_sph` and `find_spl_vals`. The function `read_sph` reads the data from the spherical harmonics data file. Thereafter, we loop over all the nodal points. The loop starts at the deepest circle of nodes and loops counter-clockwise over this circle before continuing with the next circle that is one row of nodal points outwards. The function `find_spl_vals` reads the values for a specific depth, meaning that we only have to read these once when we start a new circle of nodal points. This function makes use of the other function `read_splines`. Once we have the correct splines we loop over all nodal points at this depth and calculate the value of $\delta \ln V_s$ for all these points. The value of $\delta \ln V_s$ is stored in the array `d_ln_vs`.

[1] `https://www.earth.ox.ac.uk/~smachine/cgi/index.php`



Taken from SubMachine[2]. Cross sections of the mantle for S20RTS and S40RTS models. Left: latitude=0, longitude between 0 and 179. Right: latitude=0, longitude between -179 and 0.



S40RTS, nelr=50

Converting $\delta \ln V_s$ to $\delta\rho$

We assume that there is a direct scaling from the shear wave velocity anomaly to the density anomaly, i.e. we assume that all anomalies in $\delta \ln V_s$ are the result of temperature perturbations and not of differences in composition. We use the following formula to obtain the $\delta \ln \rho$ values:

$$\delta \ln \rho(r, \theta, \phi) = \xi(r) \cdot \delta \ln V_s(r, \theta, \phi)$$

[2] `https://www.earth.ox.ac.uk/~smachine/cgi/index.php`

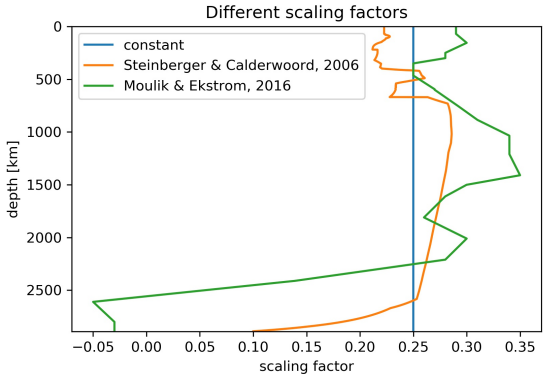
where ξ is the scaling factor that is depth dependent. We will go into more depth for this factor in the following subsections. To get the values for $\delta\rho(r, \theta, \phi)$ we use the following formula:

$$\delta \ln \rho(r, \theta, \phi) = \frac{\delta\rho(r, \theta, \phi)}{\rho_{\text{ref}}}$$

where ρ_{ref} is the reference density, which in our case for the S40RTS model is the PREM model [101]. In the following section we will show which scalings are implemented in the model and how they are implemented. The user can choose a scaling by changing the parameter `xi_case`. The value for ξ is stored in the array `xi_nodal`, the value for ρ_{PREM} is stored in the array `rho_prem_nodal`, the value for $\delta \ln \rho$ is stored in the array `d_ln_rho_nodal` and the value of $\delta\rho$ is stored in the array `d_rho_nodal`.

There are three options in the code to compute the ξ coefficient:

- Constant ξ. The first scaling that is used is a constant value for ξ of 0.25. This is `xi_case = 0`. The factor 0.25 is an estimated average of what is normally found in literature. An overview of the three different cases can be found hereunder:



- The second scaling that is used comes from Steinberger & Calderwood (2006) [307]. Data is read from the file `data/xi/xi_stca06.ascii`.

- The third scaling that is used comes from Moulik & Ekstrom (2016) [239]. Data is read from the file `data/xi/xi_moek16.ascii`. A very noticeable difference between this scaling and the scaling of [307] is that for the upper mantle the value for ξ is slightly higher and for the lowermost lower mantle the value for ξ is negative.

Stone 72

quadrilateral MINI - the $Q_1^+ \times Q_1$ element

```
2D
box
MINI
sparse_matrix
analytical_benchmark
numerical_benchmark
error_convergence
stokes_equation
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_scaling
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_72`

Manufactured solution #1

The analytical solution originates in Lamichhane [205] (2017). The velocity and pressure are given by

$$u(x,y) = -2x^2y(2y-1)(x-1)^2(y-1) \quad (72.1)$$
$$v(x,y) = 2xy^2(2x-1)(x-1)(y-1)^2 \quad (72.2)$$
$$p(x,y) = x(1-x)(1-2y) \quad (72.3)$$

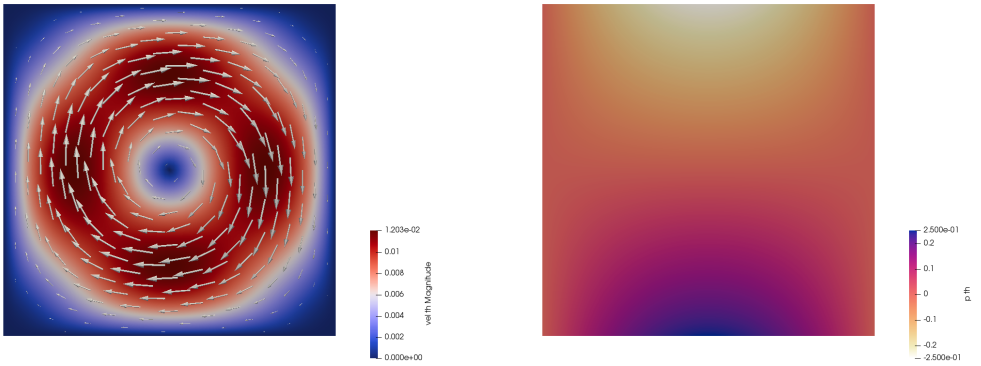
Boundary conditions are no-slip on all sides of the unit square. The corresponding body force terms are derived in Section 18.14.

The quadrilateral MINI element used here also originates in the same article and comes in two flavours with two different bubble functions b_1 and b_2 as explained in Section 8.3.8. The two bubble functions are (in reduced coordinates $-1 \leq r,s \leq 1$):

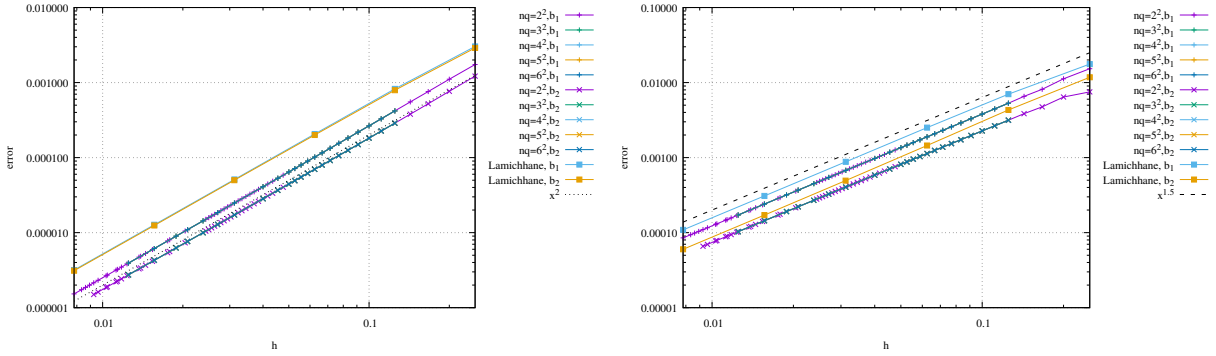
$$b_1(r,s) = (1-r^2)(1-s^2)\cdot(1-r)(1-s) \quad (72.4)$$
$$b_2(r,s) = (1-r^2)(1-s^2)\cdot(1+\beta(r+s)) \quad (72.5)$$

The common term to both bubbles insures that the bubble is exactly zero on all four edges of the element. What differentiates them is the remaining term, which is bilinear (b_1) or linear (b_2). Both also satisfy $b_1(0,0) = b_2(0,0) = 1$. The paper uses $\beta = 1/4$.

The velocity and pressure fields for the benchmark are shown hereunder:

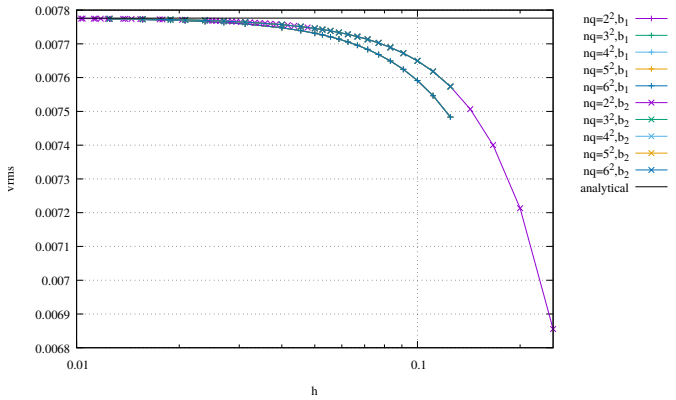


During the debugging process I ended up implementing various Gauss quadratures schemes, from 2^2 to 6^2 points but the results show that 2^2 quadrature points per element are sufficient since the more expensive quadratures yield identical results. The results from the article are somewhat different than mine but I suspect that what the author measured could be different than what I measure (see Table 1,2 in [205]). The trends are similar though, with b_2 performing better than b_1:



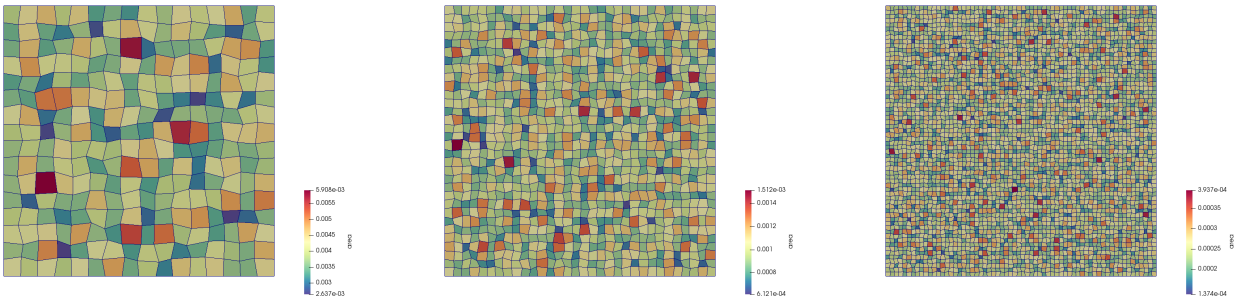
Left: velocity error in L_2 norm; Right: pressure error in L_2 norm. Resolutions from 8×8 until 80×80.

The root mean square velocity is also measured for both bubble functions. As above we see that b_2 performs better than b_1:

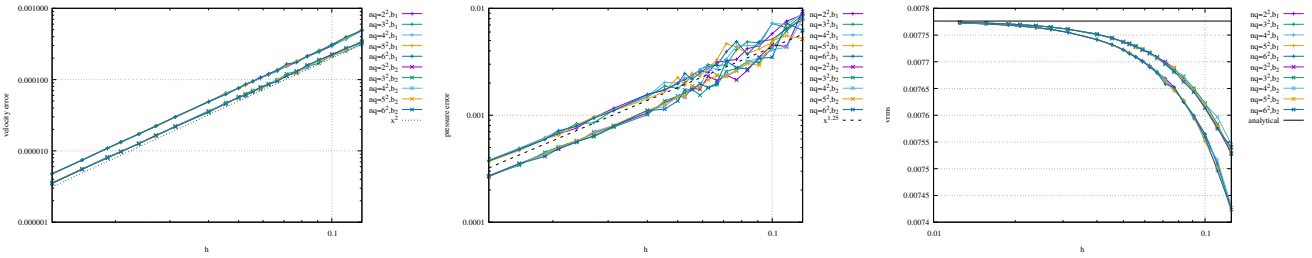


Influence of mesh nodes position:

I have also repeated these experiments with a mesh whose internal nodes have been randomly moved by up to $\pm 20\%$ of h_x or h_y around the initial position.



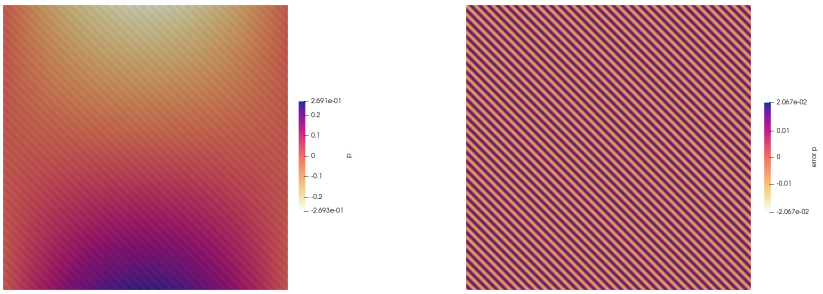
area of elements for randomized meshes. Left to right: 16x16, 32x32, 64x64



We find that the error convergence rate is unchanged for velocities but is now less for pressure (higher than 1, lower than 1.5).

Another bubble?

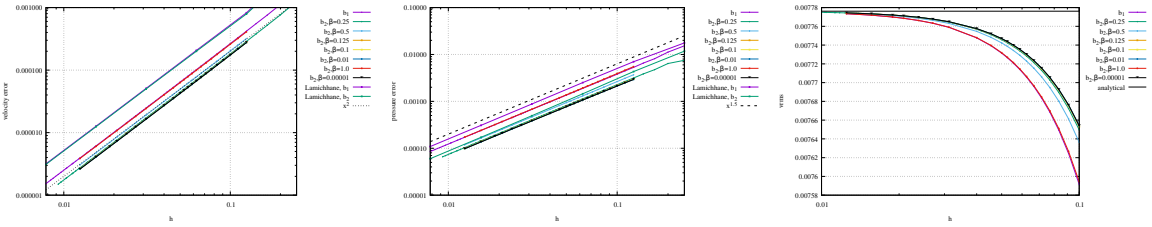
To make the point that the bubble function must contain the (bi-)linear term, I have created a third bubble function which is simply $b_3(r,s) = (1-r^2)(1-s^2)$. Technically it is zero on the sides and 1 in the middle so it fulfills the same requirements as the other 2 bubble functions. However, we see that this function is not sufficient to stabilise the element as the pressure showcases a typical error mode:



Left: pressure field as obtained with b_3; right: pressure error.

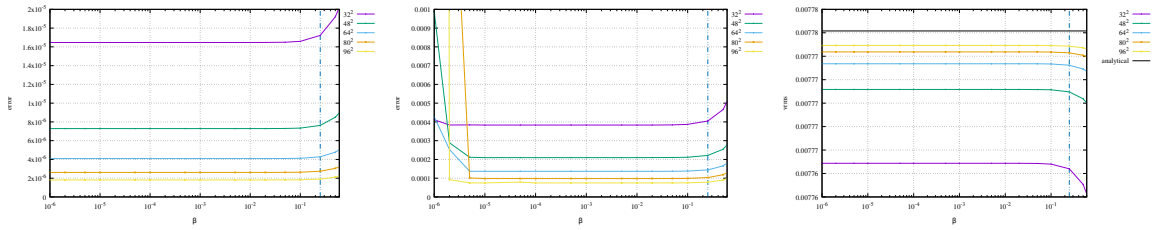
influence of β:

Finally, I use a 2×2 quadrature and look at the errors and the vrms for various values of β:



It looks like $\beta \in [0.0001, 0.01]$ does better than all other higher values. Also, looking at the field in Paraview, no trace of the error modes as we just saw above. The difference between 0.01 and 0.25 is somewhat small, but values of β above 0.5 clearly yield less accurate results.

I can also plot these same results in a different way, i.e. placing β on the horizontal axis:

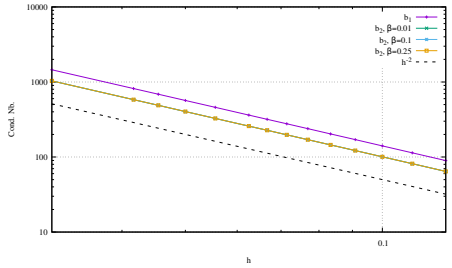


From left to right: velocity error, pressure error and vrms as a function of the parameter β, for 5 different resolutions. The dotted line corresponds to $\beta = 1/4$ as used in [205].

One could conclude that $\beta = 10^{-4} - 10^{-2}$ would be a better choice but the question remains whether these conclusions hold for other benchmarks. We will see later that $\beta = 1/4$ is actually preferable to these very low values.

Looking at the condition number of the $\mathbb{K}$ matrix

(remember: $\mathbb{K}$ is the 1,1 block of the Stokes matrix). I have computed the condition number of the $\mathbb{K}$ matrix for both bubbles and for various mesh resolutions. We see that the bubble 1 yields condition numbers substantially higher than bubble 2, both increasing quadratically with h. This is rather critical is (for instance) a conjugate gradient solver is used is used on this matrix.



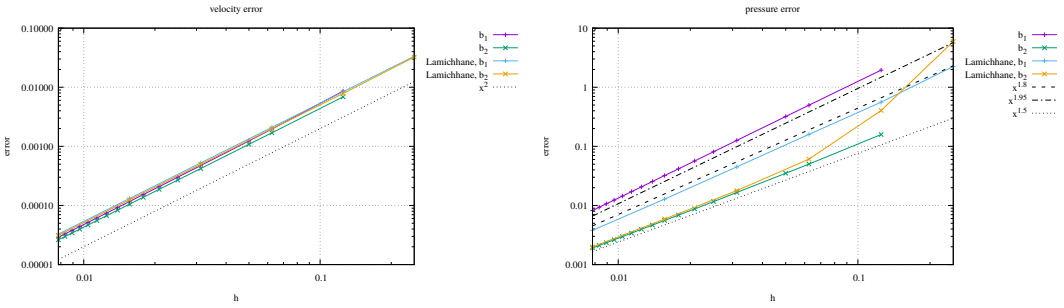
Manufactured solution #2

This is the second manufactured solution mentioned in Lamichhane [205]. It is presented in Section 18.7. It is for a unit square with $\eta = 1$ and the smooth exact solution is

$$u(x,y) = x + x^2 - 2xy + x^3 - 3xy^2 + x^2y \quad (72.6)$$
$$v(x,y) = -y - 2xy + y^2 - 3x^2y + y^3 - xy^2 \quad (72.7)$$
$$p(x,y) = xy + x + y + x^3y^2 - 4/3 \quad (72.8)$$

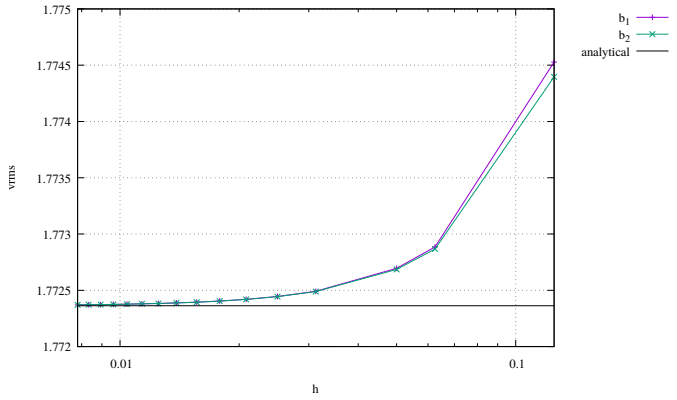
Note that the pressure obeys $\int_\Omega p \, d\Omega = 0$. The analytical velocity is prescribed on the boundary of the domain. The corresponding body force is:

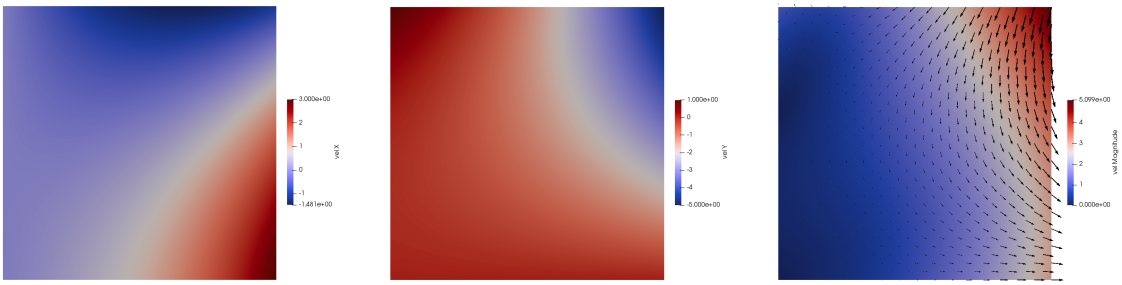
$$b_x = 3x^2y^2 - y - 1 \quad (72.9)$$
$$b_y = 2x^3y + 3x - 1 \quad (72.10)$$



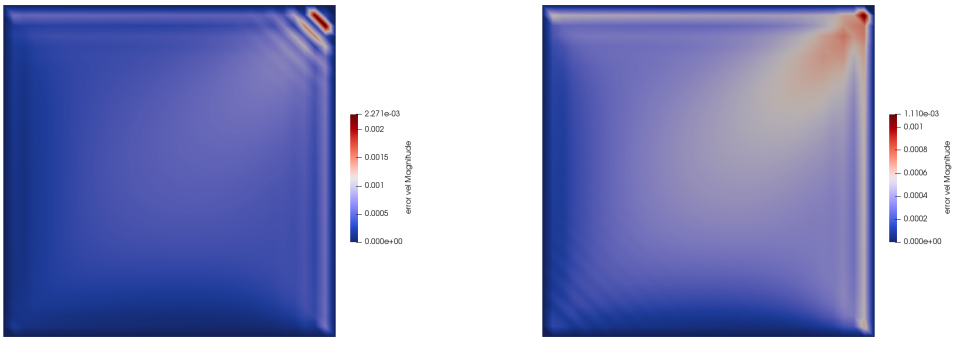
Velocity and pressure error convergence as a function of the mesh size h for meshes 8×8 up to 128×128.

The root mean square velocity is also measured for both bubble functions. As above we see that b_2 performs better than b_1:

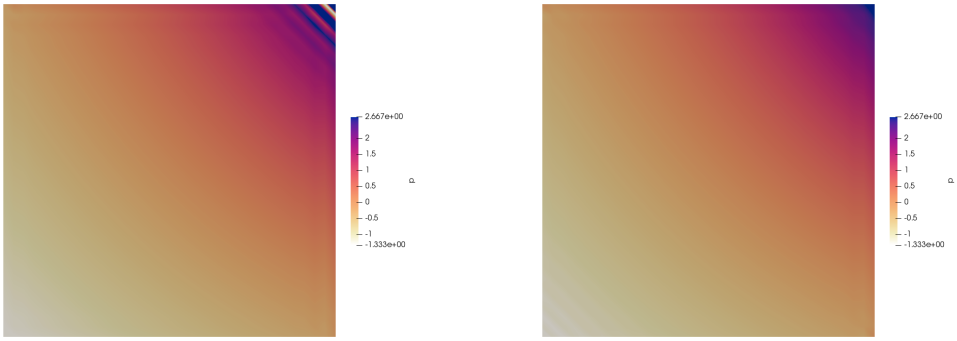




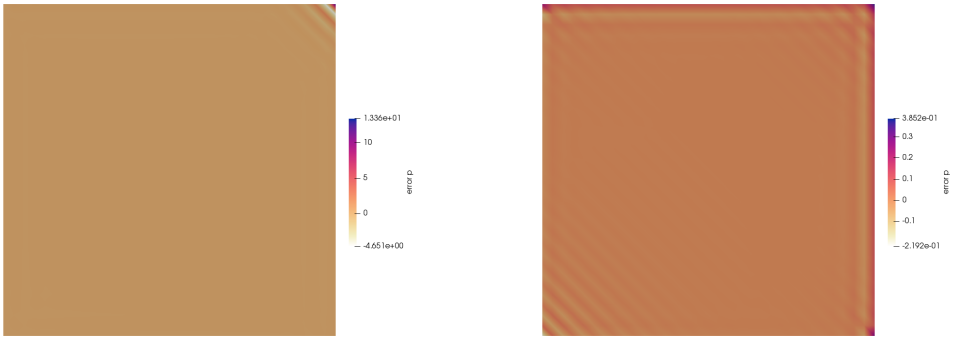
Velocity field as obtained on 32×32 grid. Left: $x-$component, middle: $y-$component, right: velocity vector magnitude.



Velocity error for bubble function 1 (left) and bubble function 2 (right). b_2 maximum error is about half of b_1 error.



Pressure field obtained with bubble function 1 (left) and bubble function 2 (right). Scale has been voluntarily set to analytical values scale for both.

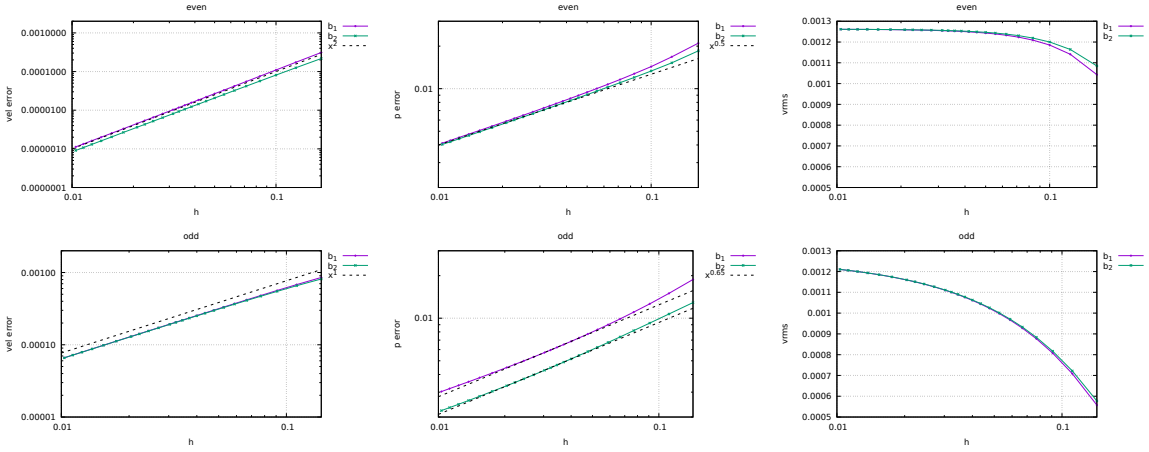


Pressure error field obtained with bubble function 1 (left) and bubble function 2 (right). Notice the different error magnitudes between both!

The conclusions are identical to those obtained with the first manufactured solution. The second bubble function (here with $\beta = 1/4$) performs better. While velocity error convergence rates are quadratic for both bubbles, the pressure error is substantially lower for bubble 2 (although its rate is slightly lower than bubble 1 – both are still superconvergent). This is illustrated above where we see that the pressure field obtained with bubble 1 showcases strong ripples in the upper right corner.

The SolCx benchmark

Because of the viscosity jump at $x = L_x/2$, it has been widely reported that error convergence rates depend on wheter the number of element in the $x-$direction is even or odd.

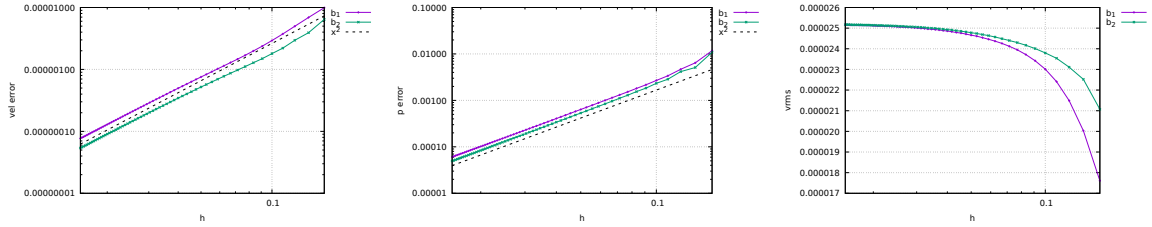


Velocity error convergence rates are 2 for even numbers of elements and 1 for odd numbers, such as for the Q1Q1 stab. Pressure error convergence rate seems to be 0.5 for odd numbers of elements and 0.65 for even numbers ...



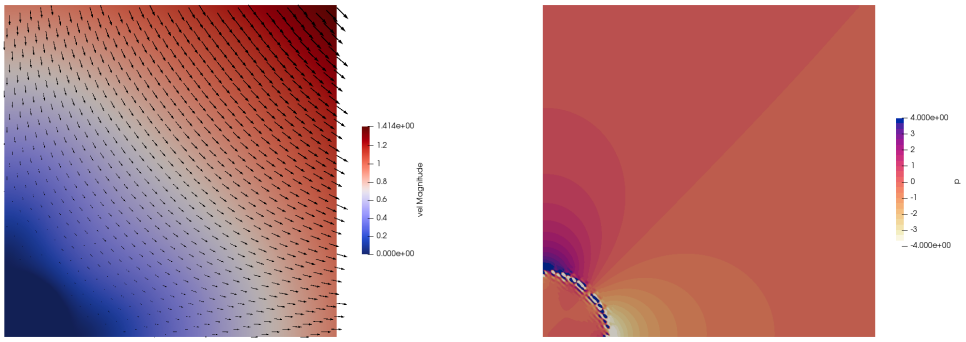
We see that the pressure showcases an strong error (even at high resolution) along the interface.

The SolKz benchmark

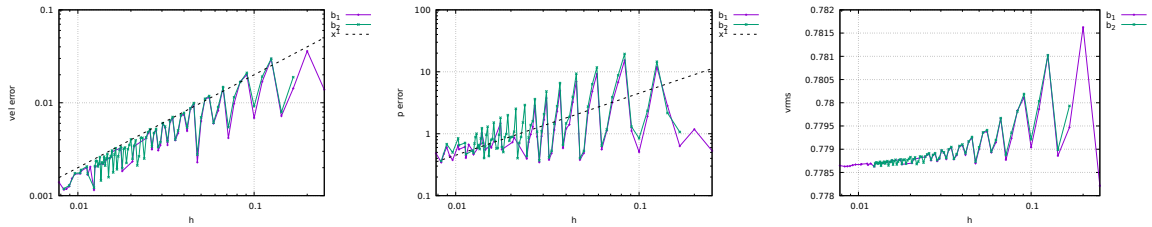


We find that velocity and pressure errors converge quadratically, with once again b_2 better than b_1.

The SolVi benchmark



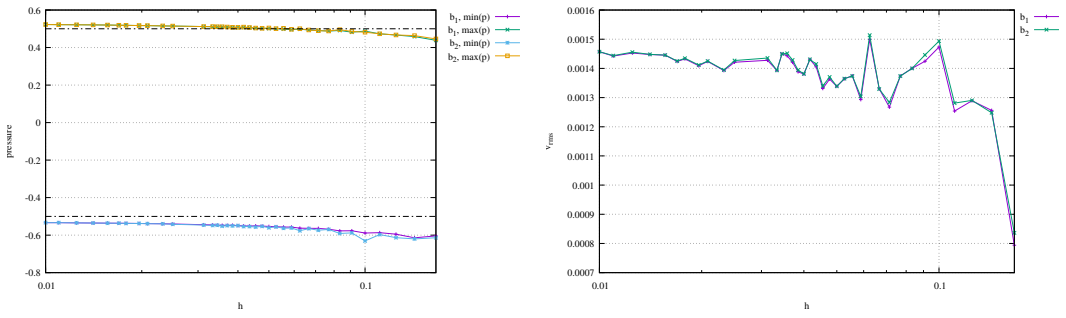
Resolution 128×128 - bubble 1

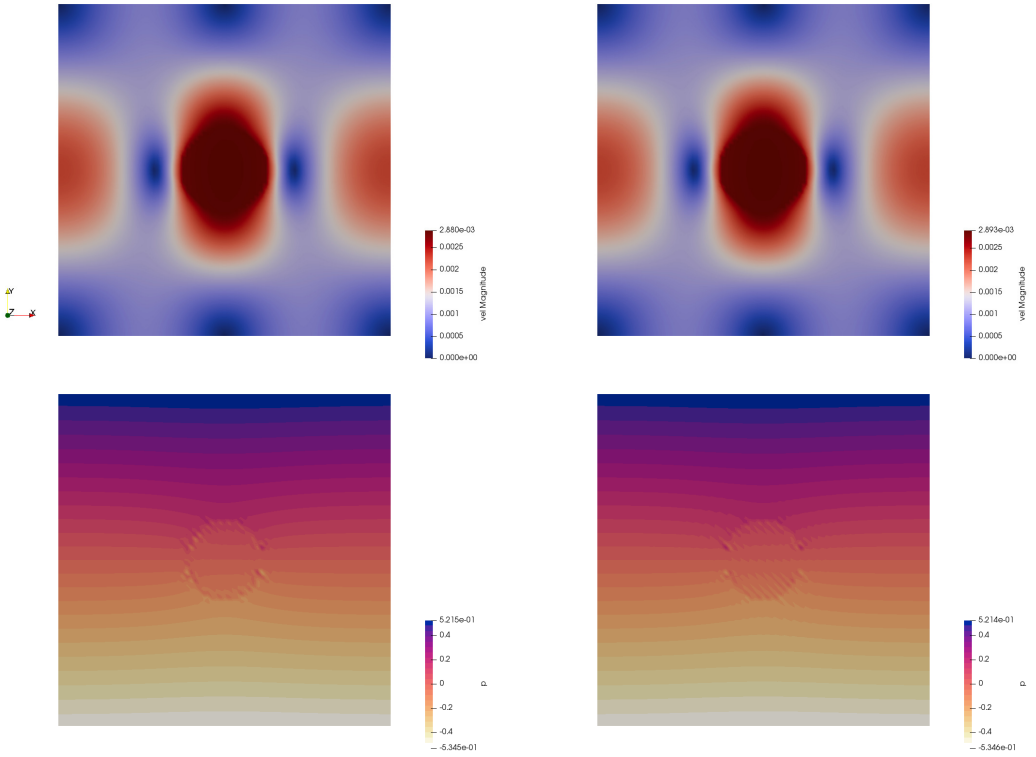


The Stokes Sphere

This experiment is the benchmark of Section 19.24. The density is assigned directly to the quadrature point by means of a function. Inside the disc of radius 0.123456789 centered in the domain the density is set to 1.01 while it is set to 1 outside (viscosities are respectively 1000 and 1). The gravity is $|g_y| = 1$ and free-slip boundary conditions are implemented on all sides.

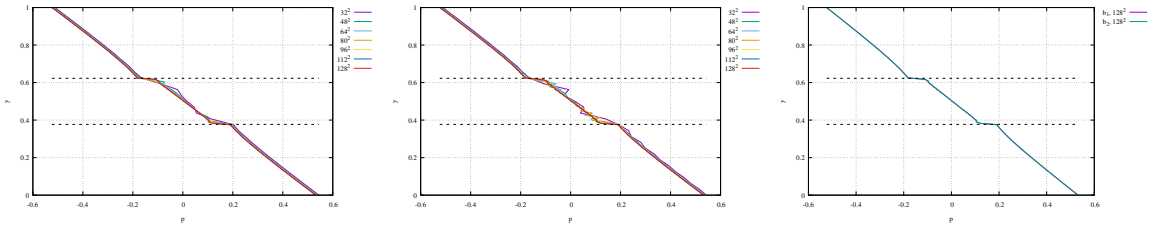
On the following figures the min/max of the pressure in the domain and the vrms are shown for both bubble functions. We see that at low resolution the reported values show some variations but with increasing resolution the quantities converge to a single value:





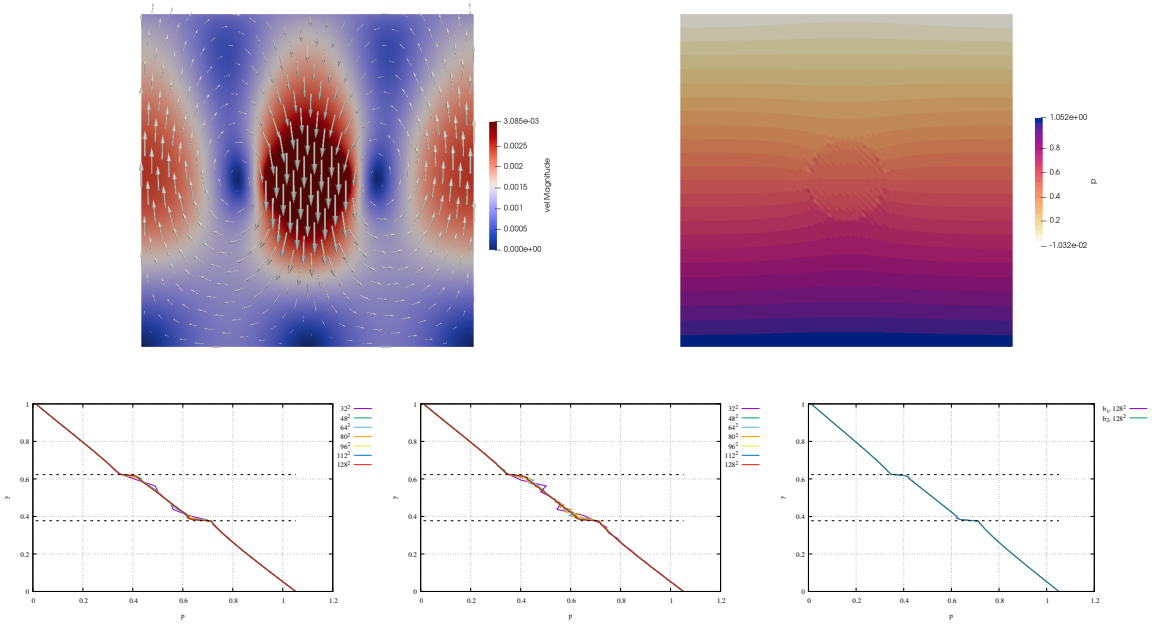
Left is b1, Right is b2.

It looks like b_2 yields more anomalous pressure modes inside the sphere, which is corroborated by the following figures which show the pressure at $x = L_x/2$ in both cases. However we also see that at high resolution both bubbles yield nearly identical pressure profiles and that the ripples have vanished.



pressure profile for b1 (left) and b2 (middle) for different resolutions.

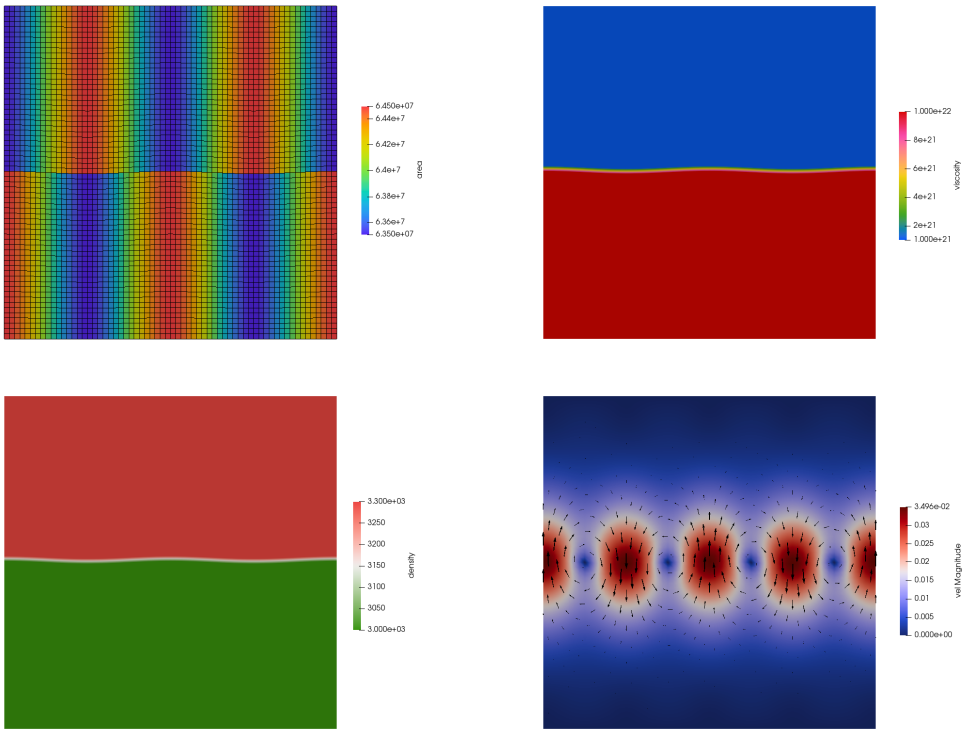
One can also prescribe an open boundary condition at the top:

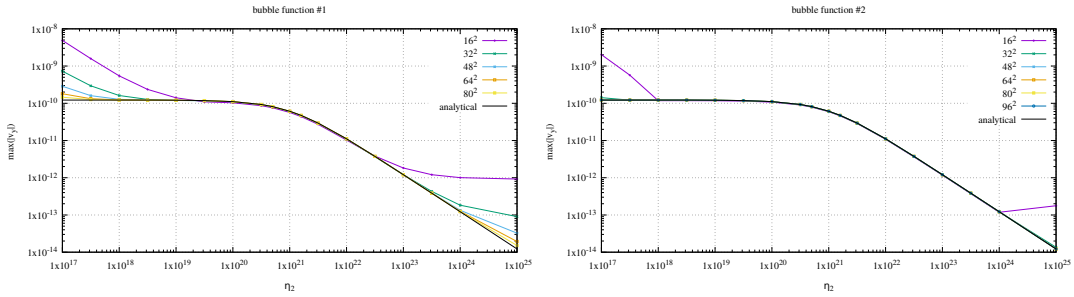


pressure profile for b1 (left) and b2 (middle) for different resolutions.

Rayleigh-Taylor instability

This benchmark is carried out in Appendix D of Thieulot (2011) [326]. It consists of a two-layer system driven by gravity. The mesh is modified so as to align with the sinusoidal perturbation. Resolution and η_2 are varied on the following figures:





Bubble 2 does better than bubble 1. Also, β does not seem to play a role.

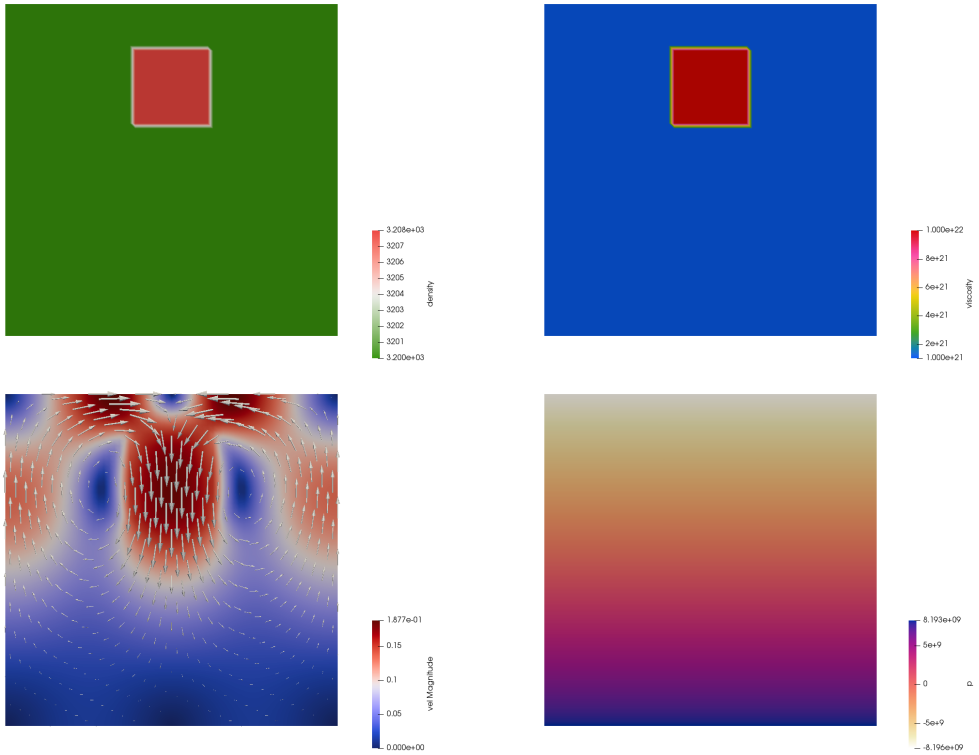
Sinking block

This is the very same experiment as in Stone 53. It consists of a negatively buoyant square object falling in a fluid in a square domain. This particular experiment proved to be the 'downfall' of the $Q_1 \times Q_1 - stab$ element since the stabilisation term is akin to a pressure diffusion and therefore acts on the lithostatic pressure and also tends to smooth the effect of small density variations (Thieulot & Bangerth, in prep.).

Unless specified otherwise bubble 2 has $\beta = 0.25$ by default.

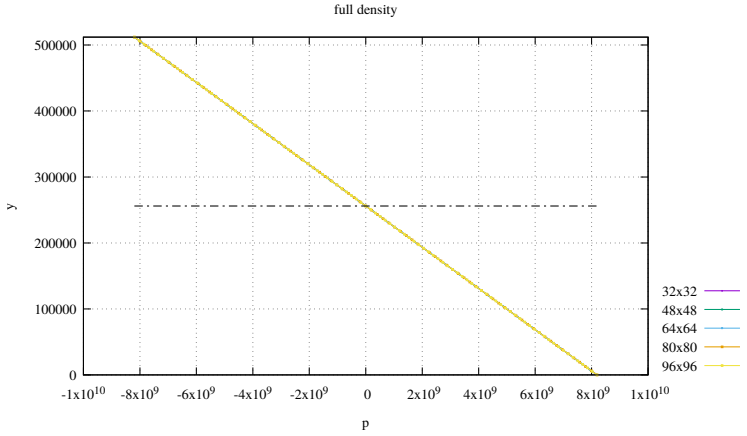
Full density

Results indicate that the element performs adequately, especially the pressure field which looks smooth.



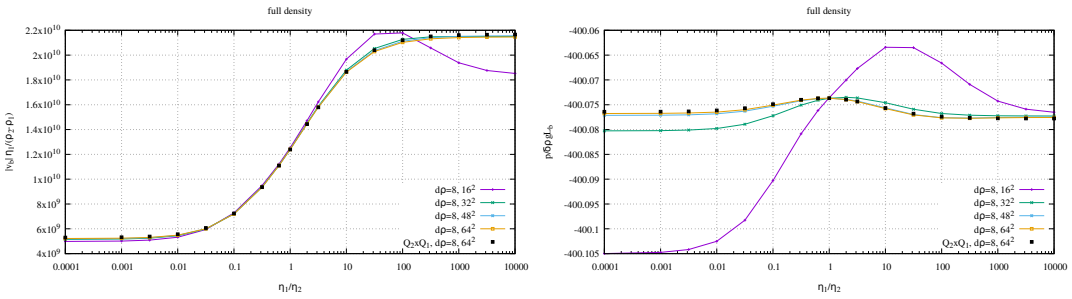
64 × 64. Viscosity ratio is 10, $\delta\rho = 8$, bubble 1.

We see that the element is capable of reprensenting a linear pressure profile (the overpressure signal due to the block is negligible compared to the hydrostatic pressure).

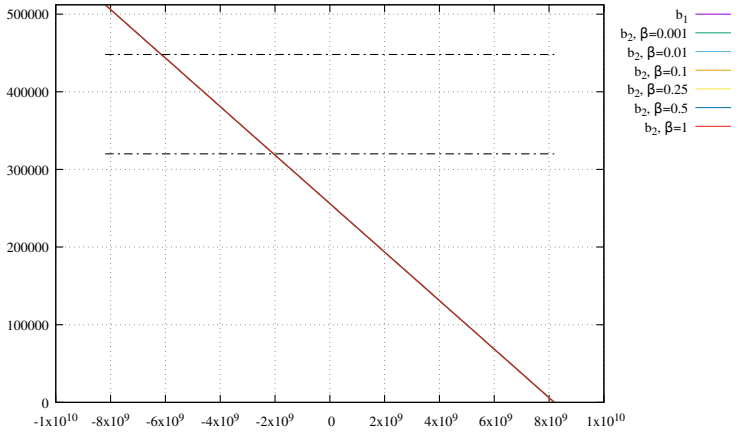


Pressure profile measured at $x = L_x/2$ for various resolutions. Same parameters as previous figure.

As before we produce the characteristic figures of the velocity and pressure in the middle of the block as a function of the viscosity ratio and the density difference. The results obtained with the quadrilateral MINI element agree nicely with those obtained with the $Q_2 \times Q_1$ element:



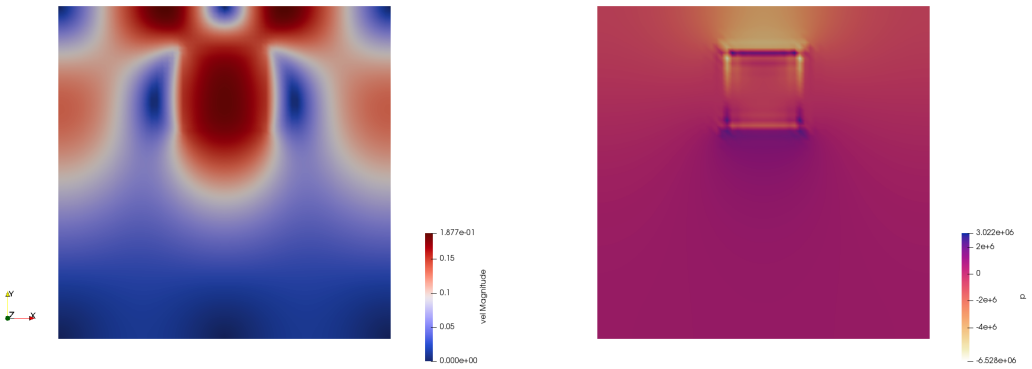
Influence of β parameter for bubble 2: I set $\delta\rho = 8$, resolution 64x64, $\eta^\star = 10$, and record the pressure profile



We can conclude that the bubble type and the value of β do not seem to significantly affect the pressure profile in this case.

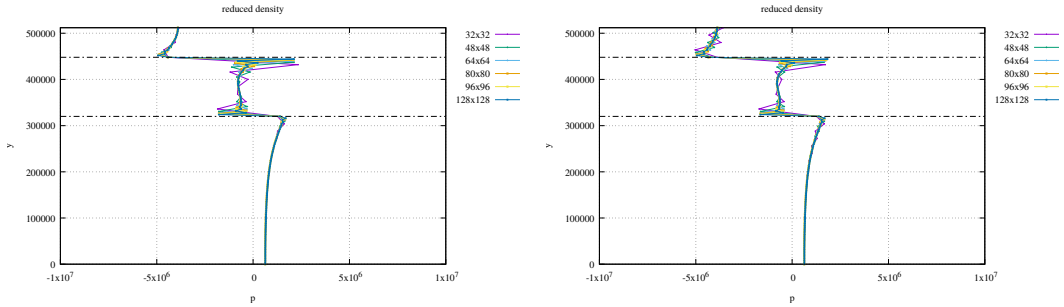
Reduced density

This is the same experiment as above but ρ_1 has been removed from the density everywhere in the domain, so that the surrounding material has zero density and the block has a density $\delta\rho$. The velocity and pressure field are then:



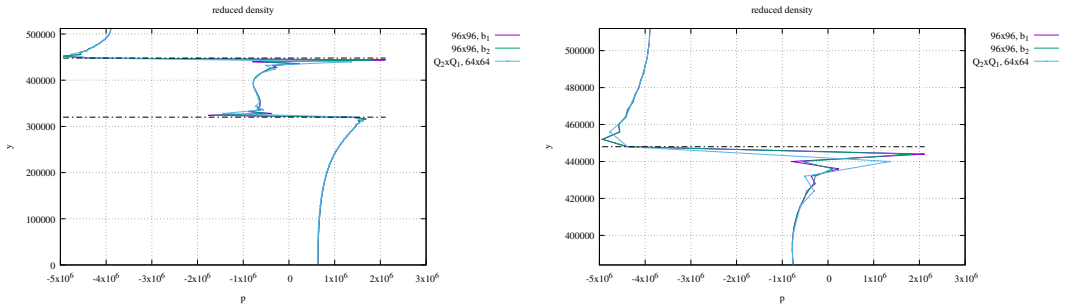
64 × 64. Viscosity ratio is 10, $\delta\rho = 8$. bubble 1

We then turn to the pressure along the vertical line $x = L_x/2$ as obtained with both bubble functions and find that both yield visually similar profiles:

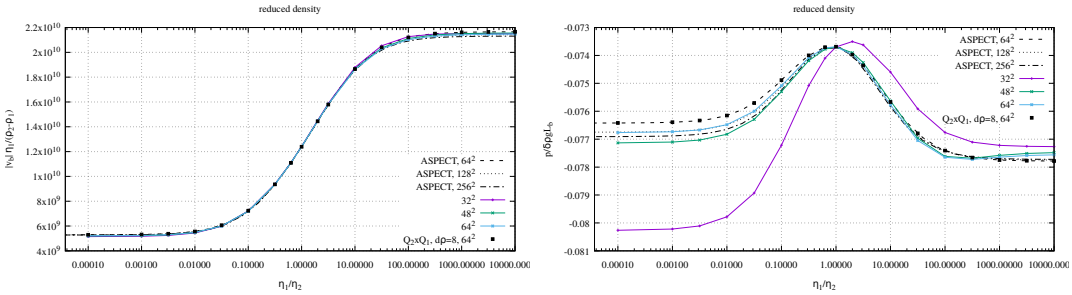


Pressure profile measured at $x = L_x/2$ for various resolutions and for both bubble functions. Left is bubble 1, right is bubble 2.

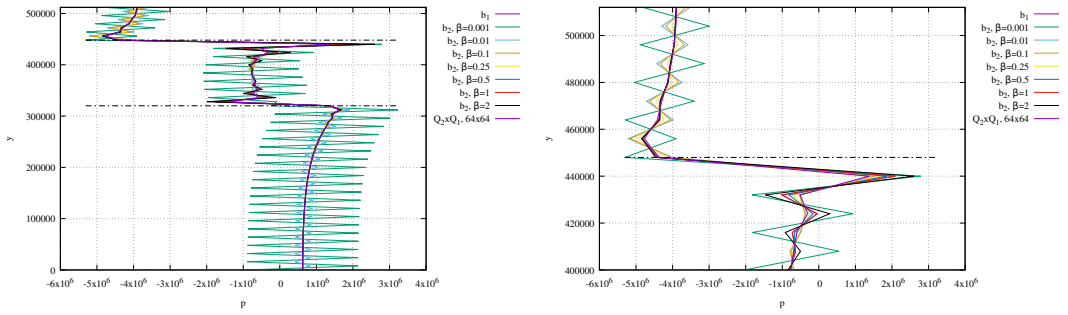
I hereafter plot the pressure profiles for both bubble functions at the highest resolution, i.e. 96×96. We see that differences are somewhat minimal, although bubble 2 yields a pressure above the block which showcases worrying oscillations. Also the results are remarquably similar to those obtained with the Taylor-Hood $Q_2 \times Q_1$ element:



Finally the velocity and pressure inside the block unsurprisingly match nicely with those obtained with the Taylor-Hood $Q_2 \times Q_1$ element:



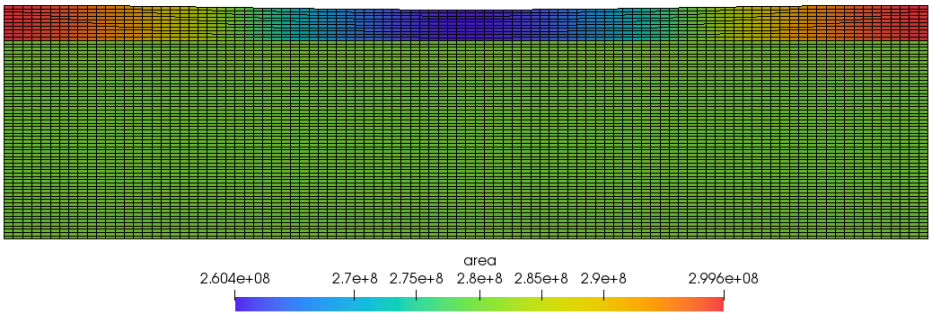
Influence of β parameter for bubble 2: I set $\delta\rho = 8$, resolution 64x64, $\eta^\star = 10$, and record the pressure profile



It is here very obvious that low values of β yield problematic pressure oscillations. It looks like $\beta \geq 0.25$ actually yield near identical results to those obtained with bubble 1, but $\beta = 1$ or $\beta = 2$ also yields unwanted oscillations.

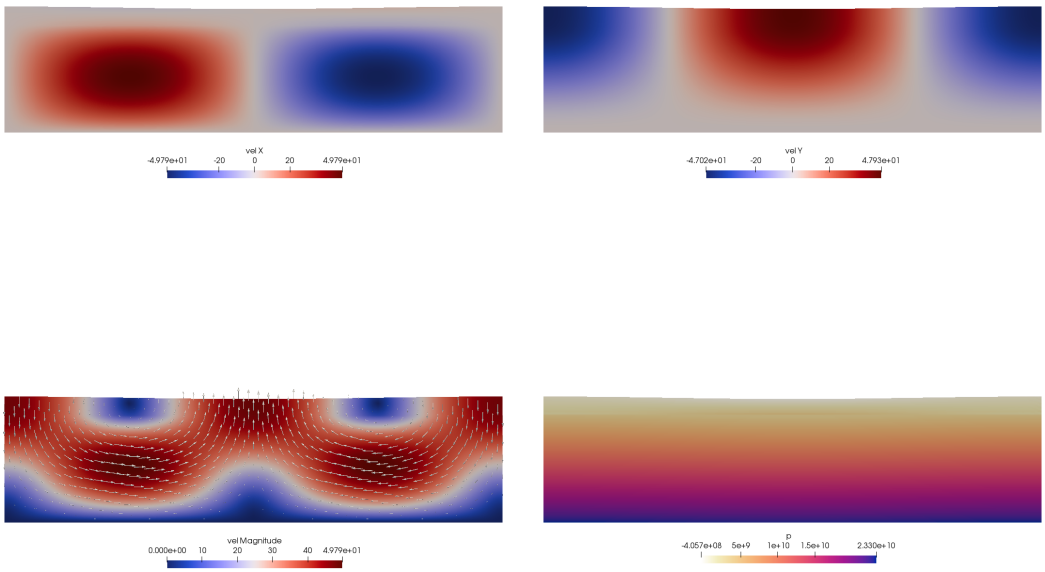
Free surface benchmark

The setup originates in Crameri et al (2012) [80]. It consists of a cosine-shaped layer of homogeneous lithosphere overlaying a homogeneous layer of mantle. The domain is 2800x700 km. The amplitude of the surface perturbation is 7km. Free slip boundary conditions are prescribed on the sides and no-slip at the bottom. The lithosphere is characterised by $\eta = 10^{23}$Pa.s and the mantle by $\eta = 10^{21}$Pa.s, while both have a density of 3330kg/m3.



Setup of the mesh. Area of the elements.

I do not wish to implement a mesh evolution algorithm nor timestepping so I only report the velocity field at $t = 0$:



It is clear that the velocity is (visually) what we expect. The presence of the free surface does not seem to introduce any kind of abnormal behaviour.

NEW stuff

all in `/study` folder.

Following Lamichhane [205] (2017), we consider the case of a 4x4 element mesh (macro-element). We prescribe u and v boundary conditions on all sides (8 nodes) so that 16 lines of the G matrix are zero. Since the original G matrix is (9+4)*2=26 dofs, there are only 26-8*2 lines left, i.e. a 9*10 matrix like the D one of the paper.

bubble 0

The matrix shown in the paper is

$$D = \begin{pmatrix} \frac{2}{9} & \frac{2}{9} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{12} & \frac{1}{12} \\ -\frac{2}{9} & \frac{2}{9} & \frac{2}{9} & \frac{2}{9} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & -\frac{2}{9} & \frac{2}{9} & 0 & 0 & 0 & 0 & -\frac{1}{12} & \frac{1}{12} \\ \frac{2}{9} & -\frac{2}{9} & 0 & 0 & 0 & 0 & \frac{2}{9} & \frac{2}{9} & \frac{1}{3} & 0 \\ -\frac{2}{9} & -\frac{2}{9} & \frac{2}{9} & -\frac{2}{9} & \frac{2}{9} & \frac{2}{9} & -\frac{2}{9} & \frac{2}{9} & 0 & 0 \\ 0 & 0 & -\frac{2}{9} & -\frac{2}{9} & -\frac{2}{9} & \frac{2}{9} & 0 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{2}{9} & -\frac{2}{9} & \frac{1}{12} & -\frac{1}{12} \\ 0 & 0 & 0 & 0 & \frac{2}{9} & -\frac{2}{9} & -\frac{2}{9} & -\frac{2}{9} & 0 & -\frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{2}{9} & -\frac{2}{9} & 0 & 0 & -\frac{1}{12} & -\frac{1}{12} \end{pmatrix}$$

which we can write:

$$D = \frac{1}{9} \begin{pmatrix} 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{3}{4} & \frac{3}{4} \\ -2 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & -2 & 2 & 0 & 0 & 0 & 0 & -\frac{3}{4} & \frac{3}{4} \\ 2 & -2 & 0 & 0 & 0 & 0 & 2 & 2 & 3 & 0 \\ -2 & -2 & 2 & -2 & 2 & 2 & -2 & 2 & 0 & 0 \\ 0 & 0 & -2 & -2 & -2 & 2 & 0 & 0 & -3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & \frac{3}{4} & -\frac{3}{4} \\ 0 & 0 & 0 & 0 & 2 & -2 & -2 & -2 & 0 & -3 \\ 0 & 0 & 0 & 0 & -2 & -2 & 0 & 0 & -\frac{3}{4} & -\frac{3}{4} \end{pmatrix}$$

Here is what I obtain (matrix multiplied by 9). Structure is good. First 8 columns are perfect BUT last two columns are off, and not just by a single constant...rank is found to be 7 as in the paper.

```
[[ 2.    2.   -0.   -0.   -0.   -0.   -0.   -0.    0.25  0.25]
 [-2.    2.    2.    2.   -0.   -0.   -0.   -0.   -0.    2.  ]
 [-0.   -0.   -2.    2.   -0.   -0.   -0.   -0.   -0.25  0.25]
```

```
[ 2.   -2.   -0.   -0.   -0.   -0.    2.    2.    2.   -0.  ]
[-2.   -2.    2.   -2.    2.    2.   -2.    2.   -0.   -0.  ]
[-0.   -0.   -2.   -2.   -2.    2.   -0.   -0.   -2.   -0.  ]
[-0.   -0.   -0.   -0.   -0.   -0.    2.   -2.    0.25 -0.25]
[-0.   -0.   -0.   -0.    2.   -2.   -2.   -2.   -0.   -2.  ]
[-0.   -0.   -0.   -0.   -2.   -2.   -0.   -0.   -0.25 -0.25]]
```

Let us investigate further and go back to the paper:

> Since A_i is an invertible matrix, the rank of the matrix will be unchanged if we replace M_i by the reference element $\hat{S}$, so that we want to investigate the rank of the matrix $D \in \mathbb{R}^{9\times 10}$, where the jth row of D is
>
> $$\left[\int_{\hat{S}} \partial_\xi \hat{\phi}_1 \hat{\varphi}_j \, d\hat{x}, \int_{\hat{S}} \partial_\eta \hat{\phi}_1 \hat{\varphi}_j \, d\hat{x}, \int_{\hat{S}} \partial_\xi \hat{\phi}_2 \hat{\varphi}_j \, d\hat{x}, \int_{\hat{S}} \partial_\eta \hat{\phi}_2 \hat{\varphi}_j \, d\hat{x}, \cdots, \right.$$
> $$\left. \int_{\hat{S}} \partial_\xi \hat{\phi}_5 \hat{\varphi}_j \, d\hat{x}, \int_{\hat{S}} \partial_\eta \hat{\phi}_5 \hat{\varphi}_j \, d\hat{x} \right].$$

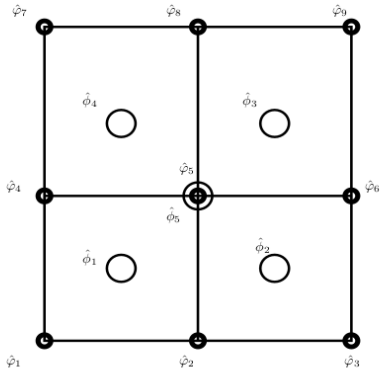


FIGURE 2. The numbering of functions $\hat{\phi}_k$ and $\hat{\varphi}_j$ on the reference square $\hat{S}$.

In my notations:

$$\left[\iint \partial_x b_1 \mathcal{N}_j^p dV, \iint \partial_y b_1 \mathcal{N}_j^p dV, \iint \partial_x b_2 \mathcal{N}_j^p dV, \iint \partial_y b_2 \mathcal{N}_j^p dV, ... \iint \partial_x b_5 \mathcal{N}_j^p dV, \iint \partial_y b_5 \mathcal{N}_j^p dV\right]$$

where the subscript of the bubble refers to the element and the integrals over the domain.

Let us look at $j = 1$ (i.e. first line of the matrix). Since function N_1^p has compact support over element 1 only, we obtain

$$\iint_{domain} \partial_x b_1 \mathcal{N}_1^p dV = \iint_{elt1} \partial_x b_1 \mathcal{N}_1^p dV \tag{72.11}$$

$$= \frac{1}{4} \iint_{refelt} 2\partial_r b_1 \mathcal{N}_1^p dr ds \tag{72.12}$$

$$= \frac{1}{4} 2 \int_{-1}^{+1} \int_{-1}^{+1} \partial_r [(1-r^2)(1-s^2)] \mathcal{N}_1^p dr ds \tag{72.13}$$

$$= \frac{1}{4} 2 \int_{-1}^{+1} \int_{-1}^{+1} -2r(1-s^2) \frac{1}{4}(1-r)(1-s) dr ds \tag{72.14}$$

$$= \frac{1}{4} 2 \int_{-1}^{+1} -2r(1-r) dr \cdot \int_{-1}^{+1} (1-s^2)(1-s) ds \tag{72.15}$$

$$= \frac{1}{4} 2 \frac{4}{3} \frac{4}{3} \tag{72.16}$$

$$= \frac{2}{9} \tag{72.17}$$

Let us now focus on $j=1$ again but this time we use the velocity basis function at node 5 (in the middle of the domain –see fig above). In element 1 this node is node 3 with basis function $\frac{1}{4}(1+r)(1+s)$. Same story as above, pressure basis function compact support on elt 1 only

$$\begin{aligned}
\iint_{domain} \partial_x N_5 \mathcal{N}_1^p dV &= \iint_{elt1} \partial_x N_5 \mathcal{N}_1^p dV && (72.18)\\
&= \frac{1}{4}\iint_{ref} 2\partial_r[\frac{1}{4}(1+r)(1+s)]\frac{1}{4}(1-r)(1-s)dV && (72.19)\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}\int_{-1}^{+1}(1+s)(1-r)(1-s)dV && (72.20)\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}(1-r)dr\int_{-1}^{+1}(1-s^2)ds && (72.21)\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}22\frac{4}{3} && (72.22)\\
&= \frac{1}{12} && (72.23)
\end{aligned}$$

Although it aligns with the values in the paper, I believe it is *wrong*. Indeed, the basis function for node 3 in element 1 is not $\frac{1}{4}(1+r)(1+s)$ but $\frac{1}{4}(1+r)(1+s)-\frac{1}{4}b(r,s)$ (see Section 8.81). In this case,

$$\begin{aligned}
\iint_{domain} \partial_x N_5 \mathcal{N}_1^p dV &= \iint_{elt1} \partial_x N_5 \mathcal{N}_1^p dV && (72.24)\\
&= \frac{1}{4}\iint_{ref} 2\partial_r\left[\frac{1}{4}(1+r)(1+s)-\frac{1}{4}b(r,s)\right]\frac{1}{4}(1-r)(1-s)drds && (72.25)\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}\int_{-1}^{+1}(1+s)(1-r)(1-s)drds - \frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}\int_{-1}^{+1}\partial_r b(r,s)(1-r)(1-s)drds && (72.26)
\end{aligned}$$

and in the case of bubble 0:

$$\begin{aligned}
&\frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}\int_{-1}^{+1}\partial_r b(r,s)(1-r)(1-s)dV\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}\int_{-1}^{+1}\partial_r[(1-r^2)(1-s^2)](1-r)(1-s)drds\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}\int_{-1}^{+1}-2r(1-s^2)(1-r)(1-s)drds\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}2\int_{-1}^{+1}-2r(1-r)dr\int_{-1}^{+1}(1-s^2)(1-s)ds\\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4}2\frac{4}{3}\frac{4}{3}\\
&= \frac{1}{18} && (72.27)
\end{aligned}$$

In the end,

$$\iint_{domain} \partial_x N_5 \mathcal{N}_1^p dV = \frac{1}{12}-\frac{1}{18} = \frac{3}{36}-\frac{2}{36} = \frac{1}{36} = \frac{1}{9}\frac{1}{4}$$

and we recover the value 1/4.

Let us now turn to the second line ($j=2$). The pressure basis function has compact support over elements 1 and 2. In elt 1 it is $\frac{1}{4}(1+r)(1-s)$ and in elt2 it is $\frac{1}{4}(1-r)(1-s)$. The velocity basis fct at node 5 is $\frac{1}{4}(1+r)(1+s)-\frac{1}{4}b$ in element 1 and $\frac{1}{4}(1-r)(1+s)-\frac{1}{4}b$ in elt 2. In the end:

$$
\begin{aligned}
&\iint_{domain} \partial_x N_5 \mathcal{N}_2^p dV \\
&= \iint_{elt1} \partial_x N_5 \mathcal{N}_2^p dV + \iint_{elt2} \partial_x N_5 \mathcal{N}_2^p dV \\
&= \frac{1}{4} \iint_{elt1} 2\partial_r [\frac{1}{4}(1+r)(1+s) - \frac{1}{4}b] \frac{1}{4}(1+r)(1-s) drds + \frac{1}{4} \iint_{elt2} 2\partial_r [\frac{1}{4}(1-r)(1+s) - \frac{1}{4}b] \frac{1}{4}(1-r)(1-s) drds \\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4} 2 \left[\iint_{elt1} \partial_r [(1+r)(1+s) - b](1+r)(1-s) drds + \iint_{elt2} \partial_r [(1-r)(1+s) - b](1-r)(1-s) drds \right] \\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4} 2 \left[\int_{-1}^{+1} \int_{-1}^{+1} [(1+s) - \partial_r b](1+r)(1-s) drds + \int_{-1}^{+1} \int_{-1}^{+1} [-(1+s) - \partial_r b](1-r)(1-s) drds \right] \\
&= \frac{1}{4}\frac{1}{4}\frac{1}{4} 2 \left[\int_{-1}^{+1} \int_{-1}^{+1} [(1+s) + 2r(1-s^2)](1+r)(1-s) drds + \int_{-1}^{+1} \int_{-1}^{+1} [-(1+s) + 2r(1-s^2)](1-r)(1-s) drds \right] \\
&= 4.4444... - 4.44444 \\
&= 0
\end{aligned}
\tag{72.28}
$$

(thank you Wolfram Alpha)

Note that in the case of a pressure field, its gradient should be zero. This means that the sum of all numbers on a given column of this matrix should be zero, which is the case for both my and his matrix.

Interestingly, we can compute the null space of the D^T matrix (.i.e $\mathbb{G}$). We obtain

```
[[-0.11773066 -0.43143886]
 [ 0.48236331 -0.13162688]
 [-0.11773066 -0.43143886]
 [ 0.48236331 -0.13162688]
 [-0.11773066 -0.43143886]
 [ 0.48236331 -0.13162688]
 [-0.11773066 -0.43143886]
 [ 0.48236331 -0.13162688]
 [-0.11773066 -0.43143886]]
```

It is of size two, which is problematic since the only vector in the null space should be the vector of constant pressures.

bubble 1

$$D = \begin{bmatrix} \frac{4}{15} & \frac{4}{15} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{12} & \frac{1}{12} \\ -\frac{4}{15} & \frac{8}{45} & \frac{4}{15} & \frac{4}{15} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & -\frac{4}{15} & \frac{8}{45} & 0 & 0 & 0 & 0 & -\frac{1}{12} & \frac{1}{12} \\ \frac{8}{45} & -\frac{4}{15} & 0 & 0 & 0 & 0 & \frac{4}{15} & \frac{4}{15} & \frac{1}{3} & 0 \\ -\frac{8}{45} & -\frac{8}{45} & \frac{8}{45} & -\frac{4}{15} & \frac{4}{15} & \frac{4}{15} & -\frac{4}{15} & \frac{8}{45} & 0 & 0 \\ 0 & 0 & -\frac{8}{45} & -\frac{8}{45} & -\frac{4}{15} & \frac{8}{45} & 0 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{8}{45} & -\frac{4}{15} & \frac{1}{12} & -\frac{1}{12} \\ 0 & 0 & 0 & 0 & \frac{8}{45} & -\frac{4}{15} & -\frac{8}{45} & -\frac{8}{45} & 0 & -\frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{8}{45} & -\frac{8}{45} & 0 & 0 & -\frac{1}{12} & -\frac{1}{12} \end{bmatrix}$$

$\boldsymbol{G}$ matrix obtained with the $(1-x^2)(1-y^2)(1-x)(1-y)$ bubble

after multiplying by 27/4

```
[[ 2.      2.     -0.     -0.     -0.     -0.     -0.     -0.      0.0625  0.0625]
 [-2.      1.      2.      2.     -0.     -0.     -0.     -0.     -0.      1.5   ]
 [-0.     -0.     -2.      1.     -0.     -0.     -0.     -0.     -0.0625  0.3125]
 [ 1.     -2.     -0.     -0.     -0.     -0.      2.      2.      1.5     -0.    ]
 [-1.     -1.      1.     -2.      2.      2.     -2.      1.     -0.      -0.    ]
 [-0.     -0.     -1.     -1.     -2.      1.     -0.     -0.     -1.5     -0.    ]
 [-0.     -0.     -0.     -0.     -0.     -0.      1.     -2.      0.3125 -0.0625]
 [-0.     -0.     -0.     -0.      1.     -2.     -1.     -1.     -0.      -1.5   ]
 [-0.     -0.     -0.     -0.     -1.     -1.     -0.     -0.     -0.3125 -0.3125]]
```

For reasons being made evident in the bubble 0 case, I cannot obtain the same matrix as he does in the paper. I should still at some point investigate in detail why and do a few calculations to be sure I understand how he obtained his coeffs.

rank is 8

We can compute the null space of the D^T matrix (.i.e $\mathbb{G}$). We obtain

```
[[-0.33333333]
 [-0.33333333]
 [-0.33333333]
 [-0.33333333]
 [-0.33333333]
 [-0.33333333]
 [-0.33333333]
 [-0.33333333]
 [-0.33333333]]
```

It is of size one and is the vector of constant pressures, as expected.

bubble 2

$$D = \begin{bmatrix} \frac{19}{90} & \frac{19}{90} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{12} & \frac{1}{12} \\ -\frac{19}{90} & \frac{7}{30} & \frac{19}{90} & \frac{19}{90} & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & -\frac{19}{90} & \frac{7}{30} & 0 & 0 & 0 & 0 & -\frac{1}{12} & \frac{1}{12} \\ \frac{7}{30} & -\frac{19}{90} & 0 & 0 & 0 & 0 & \frac{19}{90} & \frac{19}{90} & \frac{1}{3} & 0 \\ -\frac{7}{30} & -\frac{7}{30} & \frac{7}{30} & -\frac{19}{90} & \frac{19}{90} & \frac{19}{90} & -\frac{19}{90} & \frac{7}{30} & 0 & 0 \\ 0 & 0 & -\frac{7}{30} & -\frac{7}{30} & -\frac{19}{90} & \frac{7}{30} & 0 & 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{7}{30} & -\frac{19}{90} & \frac{1}{12} & -\frac{1}{12} \\ 0 & 0 & 0 & 0 & \frac{7}{30} & -\frac{19}{90} & -\frac{7}{30} & -\frac{7}{30} & 0 & -\frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{7}{30} & -\frac{7}{30} & 0 & 0 & -\frac{1}{12} & -\frac{1}{12} \end{bmatrix}.$$

$\boldsymbol{G}$ matrix obtained with the $(1-x^2)(1-y^2)(1+x/4+y/4)$ bubble

after multiplying by 18*3

```
[[ 11.    11.    -0.    -0.    -0.    -0.    -0.    -0.     1.75   1.75]
 [-11.    13.    11.    11.    -0.    -0.    -0.    -0.    -0.    12.  ]
 [ -0.    -0.   -11.    13.    -0.    -0.    -0.    -0.    -1.75   1.25]
 [ 13.   -11.    -0.    -0.    -0.    -0.    11.    11.    12.    -0.  ]
 [-13.   -13.    13.   -11.    11.    11.   -11.    13.    -0.    -0.  ]
 [ -0.    -0.   -13.   -13.   -11.    13.    -0.    -0.   -12.    -0.  ]
 [ -0.    -0.    -0.    -0.    -0.    -0.    13.   -11.     1.25  -1.75]
 [ -0.    -0.    -0.    -0.    13.   -11.   -13.   -13.    -0.   -12.  ]
 [ -0.    -0.    -0.    -0.   -13.   -13.    -0.    -0.    -1.25  -1.25]]
```

For reasons being made evident in the bubble 0 case, I cannot obtain the same matrix as he does in the paper.

rank is 8

We can compute the null space of the D^T matrix (.i.e $\mathbb{G}$). We obtain

```
[[0.33333333]
 [0.33333333]
 [0.33333333]
 [0.33333333]
 [0.33333333]
 [0.33333333]
 [0.33333333]
 [0.33333333]
 [0.33333333]]
```

It is of size one and is the vector of constant pressures, as expected.

let us reflect....

Bubbles are postulated to be of the form

$$b(r,s) = (1-r^2)(1-s^2)(1+ar+bs+crs)$$

Bubble 0 corresponds to $a = b = c = 0$. Bubble 1 corresponds to $a = -1$, $b = -1$, $c = 1$ Bubble 2 corresponds to $a = \frac{1}{4}$, $b = \frac{1}{4}$, $c = 0$

Bubble 0 does not yield a stable formulation, while bubbles 1&2 do. The question is then: how many a, b, c combinations yield stable formulations?

In order to answer this, I need to compute the D matrix of the paper as a function of a, b, c. Then I need to establish its rank as a function of a, b, c...

For example, I can run `script_run_beta` for $\beta \in [0..100]$. I seems that any nonzero value of β yields a rank equal to 8.

```
beta= 0.0 rank= 7
beta= 1e-06 rank= 8
beta= 2e-06 rank= 8
beta= 5e-06 rank= 8
beta= 1e-05 rank= 8
beta= 5e-05 rank= 8
beta= 0.0001 rank= 8
beta= 0.0005 rank= 8
beta= 0.001 rank= 8
beta= 0.005 rank= 8
beta= 0.01 rank= 8
beta= 0.02 rank= 8
beta= 0.05 rank= 8
beta= 0.1 rank= 8
beta= 0.25 rank= 8
beta= 0.5 rank= 8
beta= 1.0 rank= 8
beta= 2.0 rank= 8
beta= 5.0 rank= 8
beta= 10.0 rank= 8
beta= 100.0 rank= 8
```

Stone 73

using python functionalities ($Q_1 \times P_0$+ penalty)

```
2D
box
Q1P0
penalty
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
vrms
paraview_output
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_73`

This stone was developed in collaboration with Bob Myhill.

Building the mesh

The mesh counts $nnp = nnx \times nny$ points. In fieldstone 1 we have seen that we can build the node coordinates as follows:

```
x = np.empty(nnp,dtype=np.float64)  # x coordinates
y = np.empty(nnp,dtype=np.float64)  # y coordinates
counter = 0
for j in range(0, nny):
    for i in range(0, nnx):
        x[counter]=i*Lx/float(nelx)
        y[counter]=j*Ly/float(nely)
        counter += 1
```

The new approach taken here is

```
xs = np.linspace(0., 1., nnx, dtype=np.float64)
ys = np.linspace(0., 1., nny, dtype=np.float64)
xv, yv = np.meshgrid(xs, ys)
x = xv.flatten()
y = yv.flatten()
xy = np.vstack((x, y)).T
```

np.linspace returns evenly spaced numbers over a specified interval, in this case nnx points between 0 and 1.

Example :

```
>>> nnx, nny = (3, 2)
>>> x = np.linspace(0, 1, nnx)
>>> y = np.linspace(0, 1, nny)
>>> xv, yv = np.meshgrid(x, y)
>>> xv
array([[0. , 0.5, 1. ],
       [0. , 0.5, 1. ]])
>>> yv
array([[0.,  0.,  0.],
       [1.,  1.,  1.]])
x = xv.flatten()
y = yv.flatten()
>>> x
[0.  0.5 1.  0.  0.5 1. ]
>>> y
[0. 0. 0. 1. 1. 1.]
xy = np.vstack((x, y)).T
[[0.  0.5 1. ]
 [0.  0.5 1. ]]
[[0. 0. 0.]
 [1. 1. 1.]]
[0.  0.5 1.  0.  0.5 1. ]
[0. 0. 0. 1. 1. 1.]
[[0.  0. ]
 [0.5 0. ]
 [1.  0. ]
 [0.  1. ]
 [0.5 1. ]
 [1.  1. ]]
>>> xy.shape
(6, 2)
```

Building the connectivity array

The connectivity array is of size $m \times nel$ where m is the number of nodes per element, and nel is the number of element in the mesh:

```
icon = np.zeros((m, nel), dtype=np.int32)
```

The original version is a simple double for loop which for each element iel finds its 4 corners and stores them in $icon[iel, 0:3]$:

```
counter = 0
for j in range(0, nely):
    for i in range(0, nelx):
        icon[0, counter] = i + j * (nelx + 1)
        icon[1, counter] = i + 1 + j * (nelx + 1)
        icon[2, counter] = i + 1 + (j + 1) * (nelx + 1)
        icon[3, counter] = i + (j + 1) * (nelx + 1)
        counter += 1
```

In the new version the same is achieved using linspace and meshgrids again:

Bob: say more?

```
xis = np.linspace(0., nelx-1, nelx, dtype=np.int32)
yis = np.linspace(0., nely-1, nely, dtype=np.int32)
xiv, yiv = np.meshgrid(xis, yis)
icon = np.array([xiv + yiv * nnx,
                 (xiv + 1) + yiv * nnx,
                 (xiv + 1) + (yiv + 1) * nnx,
                 xiv + (yiv + 1) * nnx]).reshape((m, nel))
```

Building the boundary conditions

```
bc_fix = np.zeros(Nfem, dtype=bool)  # boundary condition, yes/no
bc_val = np.zeros(Nfem, dtype=np.float64)  # boundary condition, value
for i in range(0, nnp):
    if x[i]<eps:
        bc_fix[i*ndof]   = True ; bc_val[i*ndof]   = 0.
        bc_fix[i*ndof+1] = True ; bc_val[i*ndof+1] = 0.
    if x[i]>(Lx-eps):
        bc_fix[i*ndof]   = True ; bc_val[i*ndof]   = 0.
        bc_fix[i*ndof+1] = True ; bc_val[i*ndof+1] = 0.
    if y[i]<eps:
        bc_fix[i*ndof]   = True ; bc_val[i*ndof]   = 0.
        bc_fix[i*ndof+1] = True ; bc_val[i*ndof+1] = 0.
    if y[i]>(Ly-eps):
        bc_fix[i*ndof]   = True ; bc_val[i*ndof]   = 0.
        bc_fix[i*ndof+1] = True ; bc_val[i*ndof+1] = 0.
```

bc_fix has been replaced by bc_inds bc_val has been replaced by bc_vals:

Bob: say more?

```
raw_b_inds = np.where(np.logical_or.reduce((x < eps, x > Lx-eps,
                                            y < eps, y > Ly-eps)))[0]
# the [0] index above is necessary because numpy.where returns a tuple
# with len(number of dimensions), which is here equal to one.

bc_inds = np.sort(np.hstack((raw_b_inds*ndof, raw_b_inds*ndof+1)))
bc_vals = np.array([0. for idx in bc_inds])
```

Building the FE matrix

The following arrays are unchanged between both versions:

```
a_mat = np.zeros((Nfem,Nfem),dtype=np.float64)  # matrix of Ax=b
b_mat = np.zeros((3,ndof*m),dtype=np.float64)   # gradient matrix B
rhs   = np.zeros(Nfem,dtype=np.float64)         # right hand side of Ax=b
N     = np.zeros(m,dtype=np.float64)            # shape functions
u     = np.zeros(nnp,dtype=np.float64)          # x-component velocity
v     = np.zeros(nnp,dtype=np.float64)          # y-component velocity
k_mat = np.array([[1,1,0],[1,1,0],[0,0,0]],dtype=np.float64)
c_mat = np.array([[2,0,0],[0,2,0],[0,0,1]],dtype=np.float64)
```

Only the arrays containing the shape function derivarives in r,s and x,y coordinates have been changed from

```
dNdx  = np.zeros(m,dtype=np.float64)
dNdy  = np.zeros(m,dtype=np.float64)
dNdr  = np.zeros(m,dtype=np.float64)
dNds  = np.zeros(m,dtype=np.float64)
```

to this:

```
dNdxy = np.zeros((2, m), dtype=np.float64)
dNdrs = np.zeros((2, m), dtype=np.float64)
```

i.e,

$$\texttt{dNdrs} = \begin{pmatrix} \partial_r N_1 & \partial_r N_2 & \dots & \partial_r N_m \\ \partial_s N_1 & \partial_s N_2 & \dots & \partial_s N_m \end{pmatrix} \qquad \texttt{dNdxy} = \begin{pmatrix} \partial_x N_1 & \partial_x N_2 & \dots & \partial_x N_m \\ \partial_y N_1 & \partial_y N_2 & \dots & \partial_y N_m \end{pmatrix}$$

We see that the loop over elements and quadrature points have disappeared in the new version. What follows is specific to the pythonic version.

```
ijq = np.array([[iq, jq] for iq in [-1, 1] for jq in [-1, 1]]).T
rsq = ijq/sqrt3
```

Concretely ijq is a 2×4 array:

```
[[-1 -1  1  1]
 [-1  1 -1  1]]
```

and so is rsq:

```
[[-0.57735027 -0.57735027  0.57735027  0.57735027]
 [-0.57735027  0.57735027 -0.57735027  0.57735027]]
```

This in fact corresponds of the explicit double for loop of the original version:

```
for iq in [-1, 1]:
    for jq in [-1, 1]:
        rq=iq/sqrt3
        sq=jq/sqrt3
```

Two arrays are then declared which will contain the values of the shape function N and its derivatives $\partial_r N, \partial_s N$ at the nqel quadrature points of an element:

```
Nq = np.zeros((m, nqel), dtype=np.float64)
dNdrsq = np.zeros((2, m, nqel), dtype=np.float64)

Nq[0, :] = 0.25*(1.-rsq[0])*(1.-rsq[1])
Nq[1, :] = 0.25*(1.+rsq[0])*(1.-rsq[1])
Nq[2, :] = 0.25*(1.+rsq[0])*(1.+rsq[1])
Nq[3, :] = 0.25*(1.-rsq[0])*(1.+rsq[1])

dNdrsq[:, 0, :] = [-0.25*(1.-rsq[1]), -0.25*(1.-rsq[0])]
dNdrsq[:, 1, :] = [+0.25*(1.-rsq[1]), -0.25*(1.+rsq[0])]
dNdrsq[:, 2, :] = [+0.25*(1.+rsq[1]), +0.25*(1.+rsq[0])]
dNdrsq[:, 3, :] = [-0.25*(1.+rsq[1]), +0.25*(1.-rsq[0])]
```

The Jacobian matrix of the transformation is then built, as well as its inverse and its determinant. The Jacobian is built with the einsum function which evaluates the Einstein summation convention on the operands:

Bob: I n your help here. how did u arr to kej-¿e

```
jcb = np.einsum('ikq,kej->eqij', dNdrsq, xy[icon])
jcob = np.linalg.det(jcb)
jcbi = np.linalg.inv(jcb)
```

Then the coordinates xq,yq and the derivatives of the shape functions are computed at all quadrature points

```
xyeq = np.einsum('kq,kej->jeq', Nq, xy[icon])
dNdxyeq = np.einsum('eqij,jkq->ikeq', jcbi, dNdrsq)
```

We find that xyeq has shape $2 \times nel \times 4$ while dNdxyeq has shape $2 \times 4 \times nel \times 4$.

At that stage thi is magic me. I ca picture 4 arrays, n do I kno whether first four correspo to nqel o m.

The $\boldsymbol{B}$ matrix is then built, and stored for every element, hence its shape $3 \times ndof * m \times nel$.

```
meq_null = np.zeros((m, nel, nqel), dtype=np.float64)
b_mat = np.array([[dNdxyeq[0], meq_null],
                  [meq_null,   dNdxyeq[1]],
                  [dNdxyeq[1], dNdxyeq[0]]]).reshape(3, ndof*m, nel, nqel,order='F')
```

In the original version it reads:

```
for i in range(0, m):
    b_mat[0:3, 2*i:2*i+2] = [[dNdx[i],0.     ],
                             [0.     ,dNdy[i]],
                             [dNdy[i],dNdx[i]]]
```

so we see that the array *meq_null* stands for a zero (which explains its name).

The elemental matrix and rhs for all elements are then computed with einsum again:

happy it works bu 'jieq, jk, kleq, eq- is magic once aga and defi- nitely no readable.

```
a_el = (np.einsum('jieq,jk,kleq,eq->eil', b_mat, c_mat, b_mat, jcob)
        * viscosity * weightqq)
b_el = np.einsum('iq,jeq,eq->eji', Nq, body_force(xyeq), jcob)*weightqq
```

We find that *a_el* has shape $nel \times 8 \times 8$ while *b_el* has shape $nel \times 2 \times 4$.

The same process as above is repeated for the one-point integration of the penalty term.

Finally all the elemental matrices and vectors need to be assembled into the global FE matrix.

Bob: hel

```
m_indices = ((ndof*icon).T[:, np.newaxis, :]
             + np.indices((ndof,))[0, np.newaxis, :, np.newaxis]) # iel, k1, i1

mkk_indices = m_indices.reshape(nel, ndof*m, order='F') # iel, 1kk
mm_indices = (mkk_indices[:, :, np.newaxis] +
              0*mkk_indices[:, np.newaxis, :]) # iel, 1kk, 2kk
mm_indices = (mm_indices, np.einsum('ijm->imj', mm_indices))

np.add.at(rhs, m_indices, b_el)
np.add.at(a_mat, mm_indices, a_el)
```

	Pros	Cons
old	readable	slow
	easy to debug	
new	faster	memory usage
		not so readable
		loops not visible

Stone 74

the annulus benchmark with $Q_1^+ \times Q_1$ elements (MINI)

2D
annulus
MINI
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
vrms
paraview_output
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_74`

The benchmark is described fully in Section 19.18. The following results have been obtained with $k = 4$. MINI $Q_1^+ \times Q_1$ elements are used with an isoparametric mapping.

The layout of the points is borrowed from STONE 9 which showcases $Q_1 \times P_0$ elements. Special care should be taken with the placement of the bubble node. If placed at

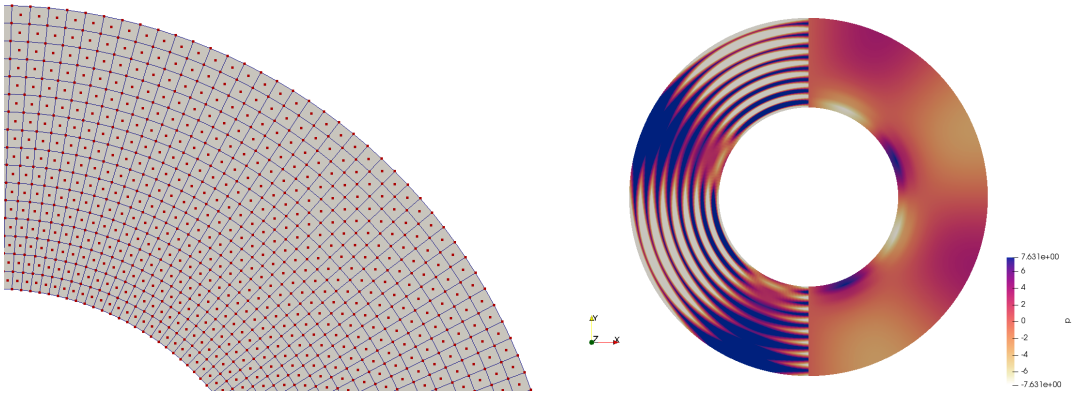
$$x_4 = \frac{1}{4}(x_0 + x_1 + x_2 + x_3) \qquad y_4 = \frac{1}{4}(y_0 + y_1 + y_2 + y_3)$$

this yields a pressure field with large oscillations. In order to correct the problem, I proceed as follows:

$$r_4 = \frac{1}{4}(r_0 + r_1 + r_2 + r_3) \qquad \theta_4 = \frac{1}{4}(\theta_0 + \theta_1 + \theta_2 + \theta_3)$$

and then $\vec{r}_4 = (x_4, y_4) = (r_4 \cos\theta_4, r_4 \sin\theta_4)$.

```
for iel in range(0,nel):
    r[iconV[4,iel]]=r[iconV[0,iel]]+dr/2
    theta[iconV[4,iel]]=theta[iconV[0,iel]]-dtheta/2
    xV[iconV[4,iel]]=r[iconV[4,iel]]*np.cos(theta[iconV[4,iel]])
    yV[iconV[4,iel]]=r[iconV[4,iel]]*np.sin(theta[iconV[4,iel]])
```

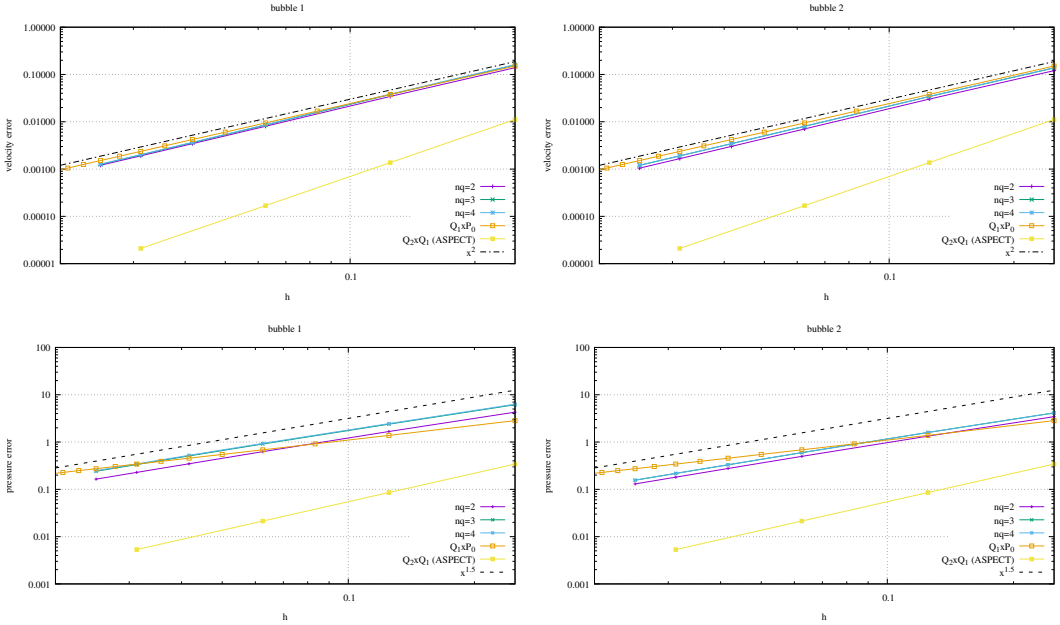


Left: Velocity nodes in the mesh. Right: Difference in pressure field when the bubble is placed at the wrong location (left half of the annulus) and at the right location (right half).

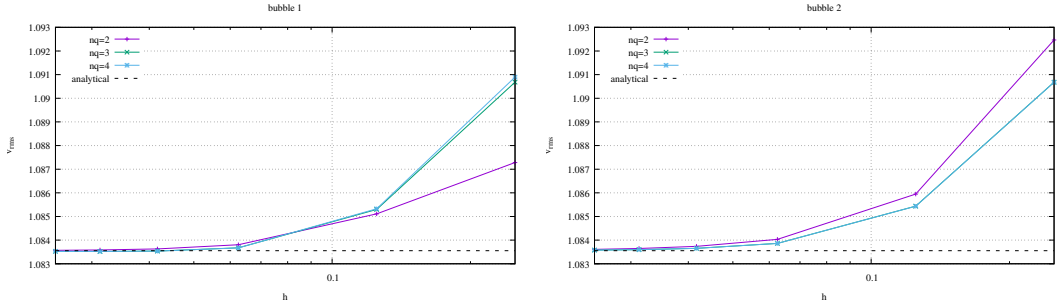
The pressure nullspace is removed by means of a Lagrange multiplier.

As in STONE 21, I here compute the velocity gradient tensor first $\boldsymbol{L}(\vec{\mathrm{v}}) = \vec{\nabla}\vec{\mathrm{v}}$. I loop over elements. For each element I loop over its velocity nodes and use the Q_2 shape function derivatives to compute the four components $L_{xx}, L_{xy}, L_{yx}, L_{yy}$ at the node and add this value to the node (while keeping track of how many times a value has been added to the node). In the end I simply compute the average of the values on each node. From $\boldsymbol{L}(\vec{\mathrm{v}})$ I then compute $\dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}})$.

We recover a quadratic convergence for the velocity error and a 1.5 index error rate for the pressure (which is between the quadratic convergence of the Taylor-Hood element and the $Q_1 \times P_0$). Note that $h = dr = (R_2 - R_1)/nelr = 1/nelr$.



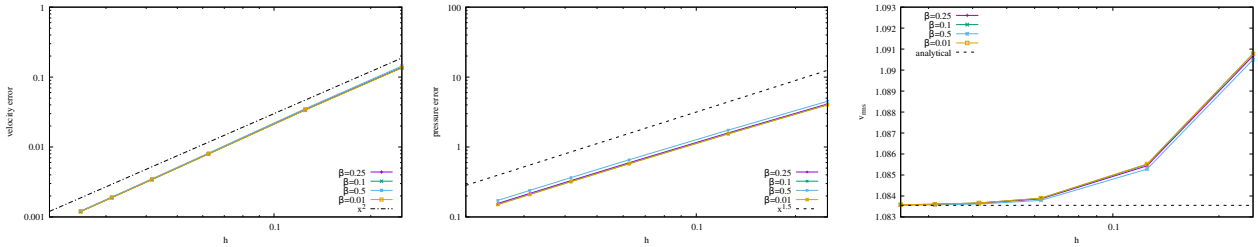
Also the root mean square velocity is logically found to converge to its expected analytical value.



This shows that a 3×3 quadrature is to be preferred over a 2×2. Higher quadratures do not yield any improvement.

Influence of β for bubble 2:

I fix the quadrature at 3×3 points.



We see that the value of β does not really matter, although 0.5 is probably too large.

Stone 75

the 3D stokes sphere with $Q_1^+ \times Q_1$ elements (MINI)

```
3D
box
MINI
sparse_matrix
analytical_benchmark
numerical_benchmark
error_convergence
stokes_equation
buoyancy_driven
vrms
paraview_output
pressure_scaling
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_75`

In STONE 72 and STONE 74 I have shown that this element seems usable in the context of geodynamics. However, the original paper and my two stones focus on 2D geometries for which one could simply use stable higher order elements.

As explained in Section **??**, we could extend the bubbles of Lamichhane (2017) [205] to 3D in order to enrich the Q_1 space:

$$b^{(1)}(r,s,t) = (1-r)(1-s)(1-t)\cdot(1-r^2)(1-s^2)(1-t^2) \quad (75.1)$$

$$b^{(2)}(r,s,t) = (1+\beta(r+s+t))\cdot(1-r^2)(1-s^2)(1-t^2) \quad (75.2)$$

In what follows I set $\beta = 1/4$ as I have shown in the 2D case that it does not really matter. If successful, the 3D version of this element could be rather useful.

The 'Burstedde' benchmark

It is called like this in the ASPECT manual but it originates in Dohrmann & Bochev (2004) [93]. It is carried out with $Q_2 \times Q_1$ elements in STONE 17.

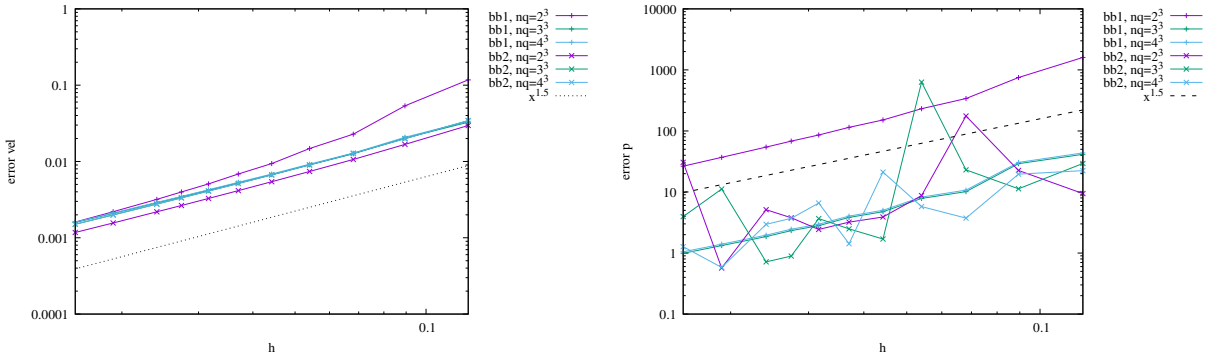
The polynomial solution to the 3D Stokes equation are postulated:

$$\vec{\mathbf{v}} = \begin{pmatrix} x + x^2 + xy + x^3 y \\ y + xy + y^2 + x^2 y^2 \\ -2z - 3xz - 3yz - 5x^2 yz \end{pmatrix} \quad (75.3)$$

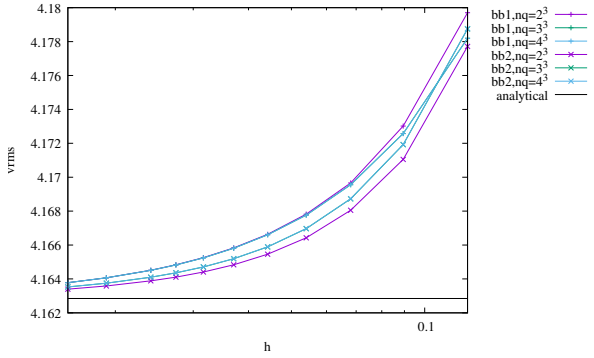
and

$$p = xyz + x^3y^3z - 5/32 \tag{75.4}$$

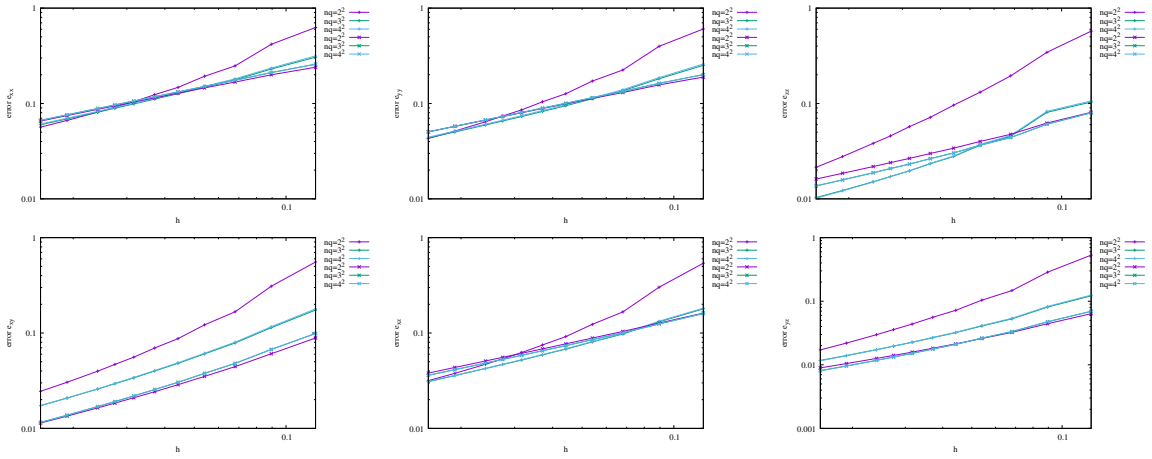
The body force is obtained by inserting the expressions above in the Stokes equations (see Section 18.9) and the viscosity is set to 1.



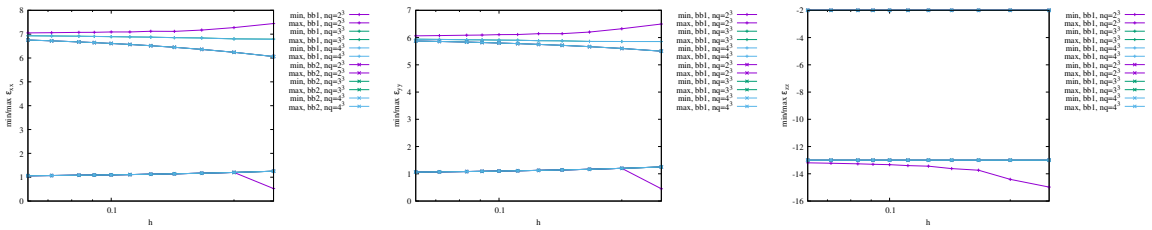
Velocity and pressure error convergence for both bubbles and with varying number of quadrature points nq.

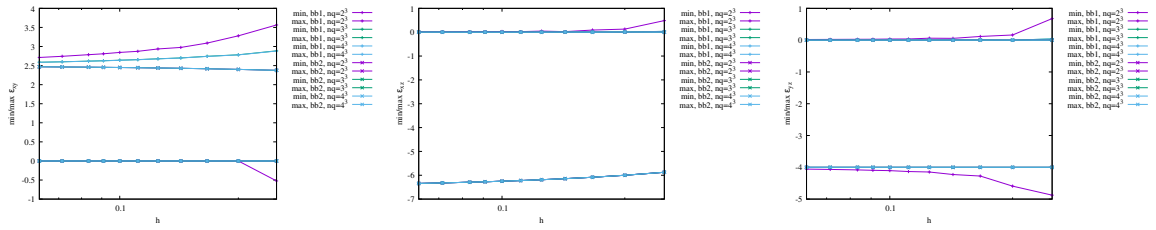


Root mean square velocity for both bubbles and with varying number of quadrature points nq.

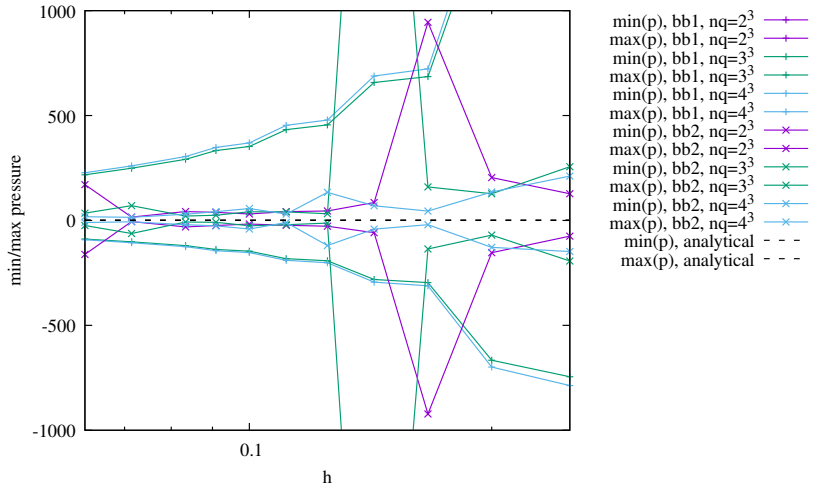


Error convergence for the 6 components of the strain rate tensor.

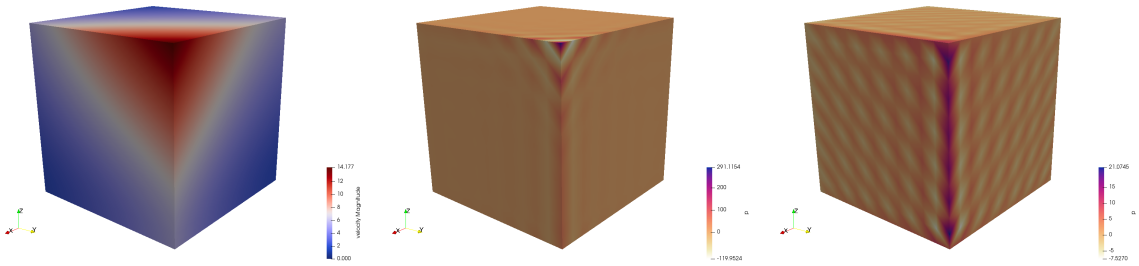




min/max statistics of the 6 components of the strain rate tensor as a function of the mesh size h.



min/max statistics of the pressure as a function of the mesh size h. Note that the black dashed lines represent the analytical solution!



From left to right: velocity field; pressure field obtained with bubble 1; pressure field obtained with bubble 2. Resolution is 12x12x12 elements.

We see that in this case the bubble #2 is way worse than bubble #1. Both yield very bad pressure fields, although the pressure min/max value seem to converge towards the analytical values for bubble #1. Note that the current code is limited to fairly low resolutions, i.e. about 24^3 elements so that I would need to implement a better Stokes solver (e.g. CG on Schur complement) instead of building the Stokes system and passing it to the solver as a whole as I do now.

It appears that 3^3 quadrature points always yield better results than 2^3 points so I set $nq = 3^3$ in what follows.

I have done a fair bit of exploration when it comes to bubbles. In the end, and given our experience with the 2D ones, we expect them to be the natural extension of the 2D bubbles:

$$b^0(r,s,t) = (1-r^2)(1-s^2)(1-t^2)$$

$$b^1(r,s,t) = (1-r^2)(1-s^2)(1-t^2)(1-r)(1-s)(1-t)$$

$$b^2(r,s,t) = (1-r^2)(1-s^2)(1-t^2)(1+\beta(r+s+t))$$

As I have done in STONE 72, I have created in `study` a simple code that creates a $2 \times 2 \times 2$ mesh (a macro-element) and looks at the null space and rank of the $\mathbb{G}$ matrix for these bubble functions.

The issue is that the mesh counts 3*3*3+8 nodes, i.e. NfemV=105 and NfemP=27 so that printing the $\mathbb{G}$ matrix is cumbersome. There are 26 nodes on the sides/edges so that 26*3=78 lines of the $\mathbb{G}$ matrix will be entirely zero. In the end the $\mathbb{G}^\star$ matrix is of size 27×27...

bubble 0

In this case the nullspace is given by

```
[[-6.55615919e-02  1.44031044e-01  0.00000000e+00  0.00000000e+00  2.97645930e-01]
 [-6.76523164e-03 -2.60178381e-02  1.24864351e-01 -4.15781369e-01 -1.43523567e-02]
 [-6.55615919e-02  1.44031044e-01 -3.94785116e-16 -8.39823003e-16  2.97645930e-01]
 [-1.91177759e-01  2.51551355e-01 -1.98170968e-01 -1.20271118e-01 -1.89288260e-01]
 [-2.62530447e-01  1.06706899e-02  3.71268341e-01  2.24840853e-01 -1.20858682e-02]
 [-1.91177759e-01  2.51551355e-01 -1.98170968e-01 -1.20271118e-01 -1.89288260e-01]
 [-6.55615919e-02  1.44031044e-01 -2.02474756e-17  4.16767315e-16  2.97645930e-01]
 [-6.76523164e-03 -2.60178381e-02  1.24864351e-01 -4.15781369e-01 -1.43523567e-02]
 [-6.55615919e-02  1.44031044e-01 -8.93653641e-17 -3.63641409e-16  2.97645930e-01]
 [ 2.44846258e-02  2.60307508e-01  3.14515329e-01  2.81444969e-02 -1.46022049e-01]
 [-4.78192832e-01  1.91453680e-03 -1.41417956e-01  7.64252380e-02 -5.53520784e-02]
 [ 2.44846258e-02  2.60307508e-01  3.14515329e-01  2.81444969e-02 -1.46022049e-01]
 [-2.93780305e-01 -2.75654656e-01  1.81617363e-01 -2.19085013e-01  1.19583825e-01]
 [ 2.21453481e-01  3.93667862e-01 -5.67530121e-02 -1.96696356e-01  1.63709749e-01]
 [-2.93780305e-01 -2.75654656e-01  1.81617363e-01 -2.19085013e-01  1.19583825e-01]
 [ 2.44846258e-02  2.60307508e-01  3.14515329e-01  2.81444969e-02 -1.46022049e-01]
 [-4.78192832e-01  1.91453680e-03 -1.41417956e-01  7.64252380e-02 -5.53520784e-02]
 [ 2.44846258e-02  2.60307508e-01  3.14515329e-01  2.81444969e-02 -1.46022049e-01]
 [-6.55615919e-02  1.44031044e-01 -4.55825698e-16  2.59774841e-16  2.97645930e-01]
 [-6.76523164e-03 -2.60178381e-02  1.24864351e-01 -4.15781369e-01 -1.43523567e-02]
 [-6.55615919e-02  1.44031044e-01 -8.03014339e-16 -8.03014339e-16  2.97645930e-01]
 [-1.91177759e-01  2.51551355e-01 -1.98170968e-01 -1.20271118e-01 -1.89288260e-01]
 [-2.62530447e-01  1.06706899e-02  3.71268341e-01  2.24840853e-01 -1.20858682e-02]
 [-1.91177759e-01  2.51551355e-01 -1.98170968e-01 -1.20271118e-01 -1.89288260e-01]
 [-6.55615919e-02  1.44031044e-01 -6.81597248e-16  5.15429713e-16  2.97645930e-01]
 [-6.76523164e-03 -2.60178381e-02  1.24864351e-01 -4.15781369e-01 -1.43523567e-02]
 [-6.55615919e-02  1.44031044e-01 -7.82793969e-16 -3.43692089e-16  2.97645930e-01]]
```

and the rank is 5.

bubble 1

In this case the nullspace is given by

```
[[ 0.         -0.23884962  0.          0.          0.08080294]
 [ 0.16934348  0.02392195 -0.03938398  0.22747521  0.14630426]
 [-0.21471563 -0.27040088 -0.02233311 -0.06461868  0.09425184]
 [-0.10673752  0.02892976  0.07100044  0.24642258  0.16110711]
 [-0.06591281 -0.23602745 -0.14788432 -0.17954949  0.18312833]
 [ 0.06791595  0.01165658  0.24348047  0.38573023  0.08079715]
 [ 0.13038563 -0.27666064 -0.16031363 -0.08830289  0.07574828]
 [ 0.17144633  0.00977865  0.20208631  0.37862496  0.07524609]
 [-0.18274347 -0.19525262 -0.43607008 -0.21124923  0.17994838]
 [-0.03432163  0.06536259 -0.13910722  0.08303938  0.26880082]
 [-0.17453665 -0.2906767   0.16727717  0.06552531  0.02158776]
 [ 0.23085171  0.09363046 -0.22926176  0.01811802  0.32310801]
 [ 0.23958486 -0.29818842  0.00170054  0.03710426 -0.00061651]
 [ 0.00764748  0.04207031 -0.04093494  0.19051424  0.18979327]
 [ 0.0629548  -0.24369011 -0.0715382   0.07091686  0.00812761]
 [-0.28680018  0.1030201  -0.02229098  0.05364434  0.35086335]
 [-0.09234076 -0.24087321 -0.00944696  0.08157475  0.01645422]
 [ 0.05486823 -0.06197475  0.11312173 -0.12609726  0.3621444 ]
 [ 0.03986577 -0.32220169  0.10232094  0.11592612 -0.05886886]
 [ 0.30722613  0.07809022 -0.19186555  0.07228145  0.2771718 ]
 [-0.38641316 -0.29771997  0.15485771  0.24826604 -0.12294019]
 [-0.31395614  0.08935779  0.0564994   0.11491304  0.31047821]
 [-0.05160682 -0.22037974 -0.12763252 -0.0103283   0.07703193]
 [-0.00623268 -0.09271495  0.29040007  0.01175732  0.27127783]
 [ 0.39006467 -0.31180443 -0.15559848  0.19497656 -0.1645732 ]
 [ 0.22671067 -0.09694029  0.19726322 -0.00422952  0.25878793]
 [-0.18660551  0.08617576 -0.54478139  0.54324418 -0.14236849]]
```

and rank is 22

bubble 2

$\beta = 1/4$

In this case the nullspace is given by

```
[[ 0.          0.26976967  0.        ]
 [ 0.27392841 -0.01117426 -0.09223388]
 [-0.00458301  0.25379735 -0.11621606]
 [ 0.27392841 -0.01117426 -0.09223388]
 [ 0.0093026   0.30219028  0.23589544]
 [ 0.27790582  0.00268746  0.00862534]
 [-0.00458301  0.25379735 -0.11621606]
 [ 0.27790582  0.00268746  0.00862534]
 [-0.01089738  0.23179104 -0.27633605]
 [ 0.27392841 -0.01117426 -0.09223388]
 [ 0.0093026   0.30219028  0.23589544]
 [ 0.27790582  0.00268746  0.00862534]
 [ 0.0093026   0.30219028  0.23589544]
 [ 0.26255856 -0.05079946 -0.38055052]
 [ 0.00606537  0.29090817  0.15380569]
 [ 0.27790582  0.00268746  0.00862534]
 [ 0.00606537  0.29090817  0.15380569]
 [ 0.28365743  0.0227325   0.1544749 ]
 [-0.00458301  0.25379735 -0.11621606]
 [ 0.27790582  0.00268746  0.00862534]
 [-0.01089738  0.23179104 -0.27633605]
 [ 0.27790582  0.00268746  0.00862534]
 [ 0.00606537  0.29090817  0.15380569]
 [ 0.28365743  0.0227325   0.1544749 ]
 [-0.01089738  0.23179104 -0.27633605]
 [ 0.28365743  0.0227325   0.1544749 ]
 [-0.01972214  0.20103577 -0.50011426]]
```

and rank is 24

This is the 'best'. changing beta value does not change anything.

further systematic testing

From the short study above we can conclude that the bubbles obtained by extending the 2D ones are not sufficient to stabilise the element in 3D.

We then postulate the bubble functions to be of the form

$$b(r,s,t) = (1-r^2)(1-s^2)(1-t^2)\mathcal{P}(r,s,t)$$

where $\mathcal{P}(r,s,t)$ is a low order polynomial, say

$$\mathcal{P}(r,s,t) = 1+a_1r+b_1s+c_1t+a_2r^2+b_2s^2+c_2t^2+d_2rs+e_2st+f_2rt+a_3r^3+b_3s^3+c_3t^3+d_3r^2s+e_3r^2t+f_3rs^2+g_3s^2t+h_3rt^2+$$

The question is then: is there such a polynomial that yields a stable element pair?

I have therefore designed a bash script `script_bubblesearch` that performs a large number of rank/nullspace calculations for many combinations of the polynomial coefficients. There are 19 parameters. Even if I only try 2 values for each, I'll need to run the code 2^{19} times...

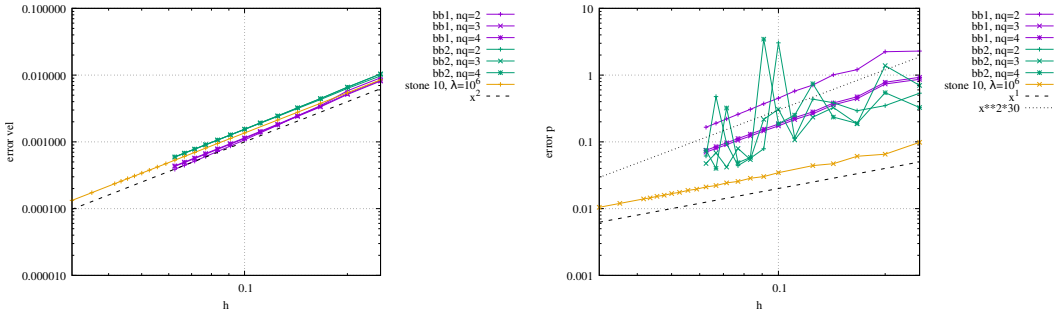
Let us start with the simple case $a_3 = b_3 = ... = j_3 = 0$. Then the search space is much smaller. We choose $a_1, b_1, c_2 \in \{0, 0.25, 1\}$, $a_2, ..f_2 \in \{-1, 0, 1\}$ and $a_3 = ...j_3 = 0$. We find that the size of the nullspace is always 3, 4 or 5 but not smaller.

Thought: is 2D, the macro element has 10 Vdofs left and 9 Pdfofs, the matrix is not square. in 3D, the macro element has 27 Vdofs left and 27 Pdofs, i.e. matrix is square.

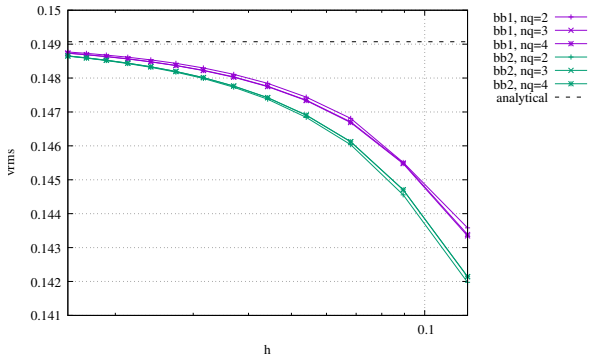
The generic mms3D

In order to verify that the above results are robust (or that I have not made a mistake) I have tried another manufactured solution, as defined in Section 18.30. Python file on github is stone_mms3D.py

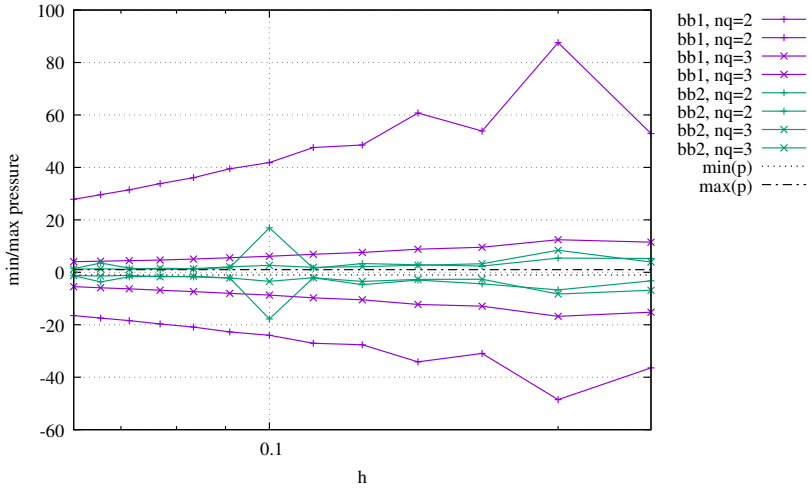
$$u(x,y,z) = x(1-x)(1-2y)(1-2z) \tag{75.5}$$
$$v(x,y,z) = (1-2x)y(1-y)(1-2z) \tag{75.6}$$
$$w(x,y,z) = -2(1-2x)(1-2y)z(1-z) \tag{75.7}$$
$$p(x,y,z) = (2x-1)(2y-1)(2z-1) \tag{75.8}$$



↑ Velocity and pressure error convergence for both bubbles



↑ Root mean square velocity for both bubbles



↑ min/max statistics of the pressure as a function of the mesh size h.

Once again, the conclusion is clear: none of the two bubbles is capable of generating a smooth pressure field. Bubble 1 is objectively better than bubble 2. It could be that my conclusions are limited by the lack of high resolution measurements but the 2D experiments with the equivalent bubbles worked fine at low resolution...

The nail in the coffin comes from Remark 1 of Karabelas, Haase, Plank, and Augustin [176]: "Contrary to the two-dimensional case studied in Bai [7] (1997) and Lamichhane [205] (2017) it is not sufficient to enrich the standard isoparametric finite element space for hexahedrons with only one bubble function. In this case both the spaces V_{0,M_i} and P_{M_i} have a dimension of 27, however, matrix D has only rank 24."

The answer to how to continue further is in STONE 82.

Stone 76

mms and stokes sphere with $Q_2 \times P_{-1}$ elements



2D
box
Q2Pm1
sparse_matrix
compressible
analytical_benchmark
numerical_benchmark
error_convergence
stokes_equation
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_scaling
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_76`

This stone is an example of $Q_2 \times P_{-1}$ implementation. The 2D Q_2 shape functions have been derived in Section 8.3.2 The -1 subscript indicates that the pressure field is linear (but not bilinear!).

Actually the story is a bit more complicated than that, as explained for example in Boffi and Gastaldi [26] (2002): "Two possible choices are given for the definition of the pressure space: one can either use a global pressure approximation (that is on each quadrilateral the finite element space is spanned by 1 and by the global co-ordinates x and y) or a local approach (consisting in generating the local space by means of the constants and the local curvilinear co-ordinates on each quadrilateral r and s). [...] Numerical results actually show that the second choice (local or mapped pressure approximation) is suboptimally convergent."

In other words, the mapped version consists of using the P_{-1} basis functions defined in Section 8.3.11: inside the element the pressure is given by $p^h(r,s) = a + br + cs$

The unmapped version is a bit less straightforward and is as follows: let us assume there are three pressure nodes inside the element. Inside the element, the pressure is defined by a linear field

$$p^h(x,y) = a + bx + cy$$

We must then compute the coefficients a, b, c. We know that the expression above evaluated at the pressure

nodes with coordinates (x_i, y_i) should be equal to p_i. Then we have the following three equations:

$$\begin{aligned} a + bx_1 + cy_1 &= p_1 \\ a + bx_2 + cy_2 &= p_2 \\ a + bx_3 + cy_3 &= p_3 \end{aligned}$$

or,

$$\begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

Then

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix}^{-1} \cdot \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \boldsymbol{M}^{-1} \cdot \vec{p}$$

with

$$\Delta = x_2y_3 - x_3y_2 - x_1y_3 - x_3y_1 + x_1y_2 - x_2y_1$$

and

$$\boldsymbol{M}^{-1} = \frac{1}{\Delta} \begin{pmatrix} x_2y_3 - x_3y_2 & x_3y_1 - x_1y_3 & x_1y_2 - x_2y_1 \\ y_2 - y_3 & y_3 - y_1 & y_1 - y_2 \\ x_3 - x_2 & x_1 - x_3 & x_2 - x_1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$$

$$\begin{aligned} a &= m_{11}p_1 + m_{12}p_2 + m_{13}p_3 \\ b &= m_{21}p_1 + m_{22}p_2 + m_{23}p_3 \\ c &= m_{31}p_1 + m_{32}p_2 + m_{33}p_3 \end{aligned}$$

so

$$\begin{aligned} p^h(x,y) &= a + bx + cy \\ &= m_{11}p_1 + m_{12}p_2 + m_{13}p_3 + (m_{21}p_1 + m_{22}p_2 + m_{23}p_3)x + (m_{31}p_1 + m_{32}p_2 + m_{33}p_3)y \\ &= \underbrace{(m_{11} + m_{21}x + m_{31}y)}_{N_1(x,y)} p_1 + \underbrace{(m_{12} + m_{22}x + m_{32}y)}_{N_2(x,y)} p_2 + \underbrace{(m_{13} + m_{23}x + m_{33}y)}_{N_3(x,y)} p_3 \end{aligned} \tag{76.1}$$

The basis functions are then evaluated at the (real) coordinates of the quadrature point (x_q, y_q).

Note that I use isoparametric elements where the Q_2 basis functions are used for the mapping. In the case of rectangular elements the approaches above (mapped and unmapped) yield the same pressure basis functions values at the quadrature points so the measured errors and vrms are identical.

In both cases the pressure is discontinuous from an element to another and this poses a little problem in terms of exporting it to vtu (and plotting with paraview). I then simply choose to export the value of the pressure in the middle of each element.

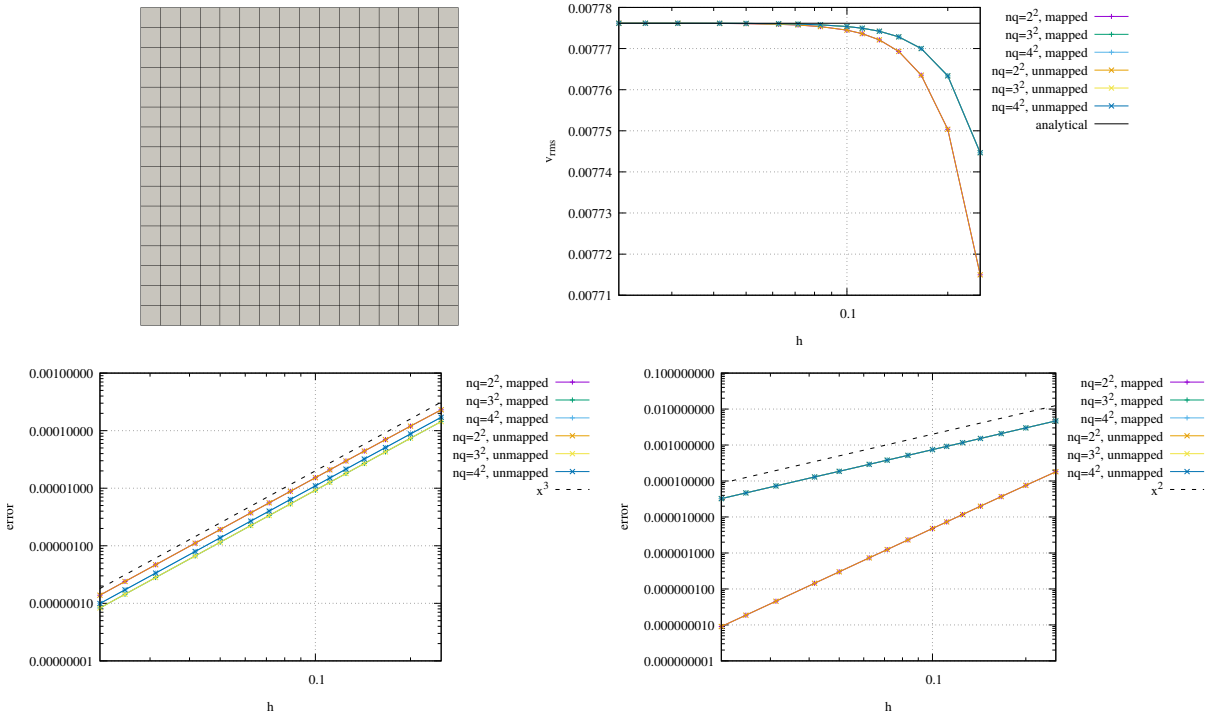
When mesh randomization is added to all nodes, I make sure that the velocity nodes 4,5,6,7 in the middle of the edges are really put back between nodes 0-1, 1-2, 2-3, and 3-0 respectively. The node in the middle of the element is also then moved accordingly.

I also explore the effect of the number of quadrature points on the solution, so I allow for 2^2, 3^2 and 4^2 quadrature points.

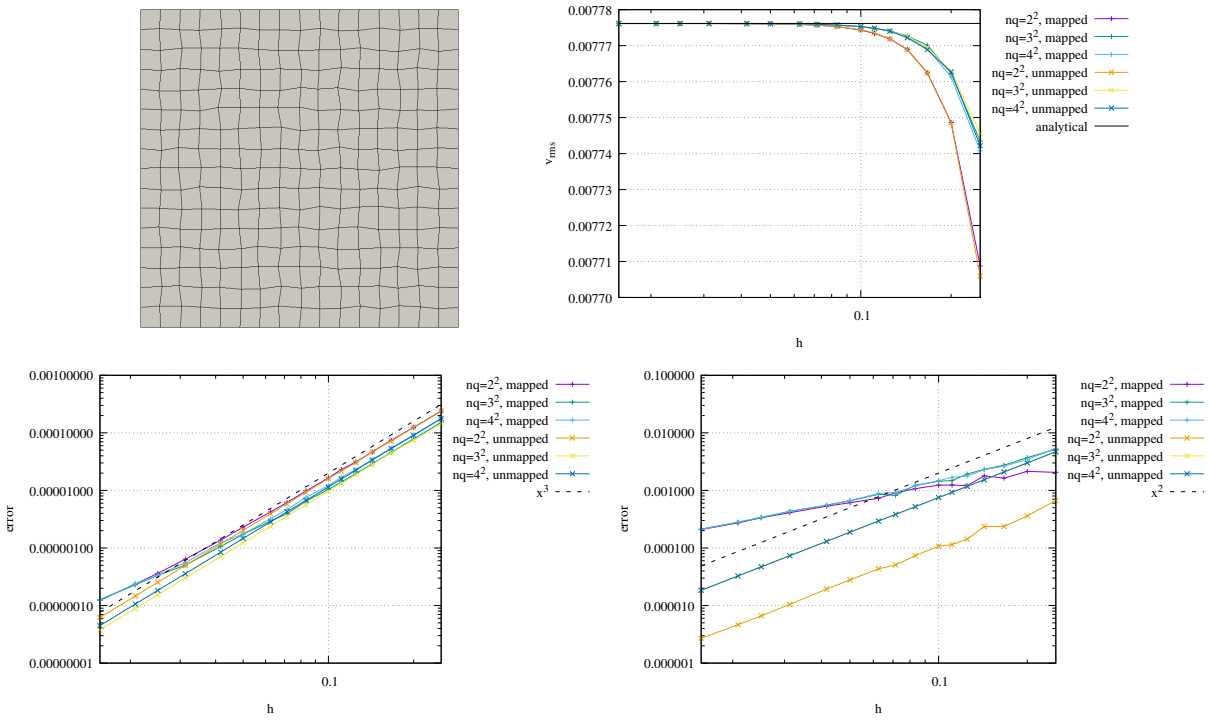
I carry out three benchmarks based on manufactured solutions and recover the expected cubic convergence for the velocity error and a quadratic error convergence for the pressure *only* for the unmapped approach in the case of non-rectangular elements.

Manufactured solution of Donea & Huerta (`bench=3`)

This is the Donea & Huerta benchmark, see Section 18.2.



Regular square mesh



Randomized $\xi = 0.1$ mesh

The observed super-convergence of the pressure for $n_q = 2^2$ is probably due to the pressure field being too 'simple' in this case.

Manufactured solution #1 (`bench=1`)

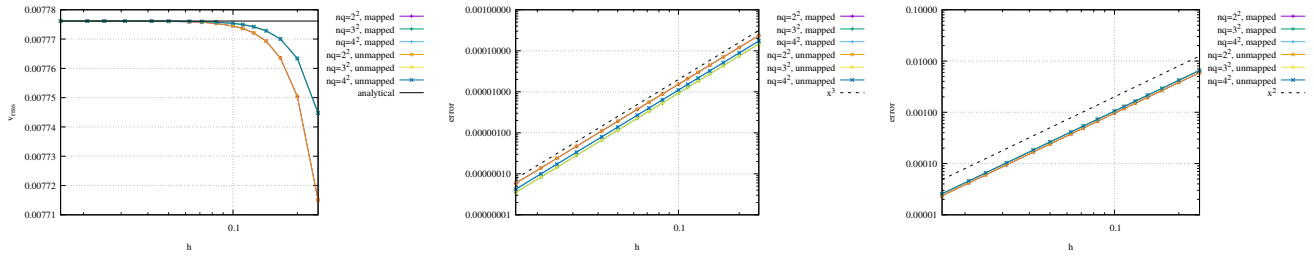
The analytical solution originates in Lamichhane (2017) [205]. The velocity and pressure are given by

$$u(x,y) = -2x^2y(2y-1)(x-1)^2(y-1) \tag{76.2}$$

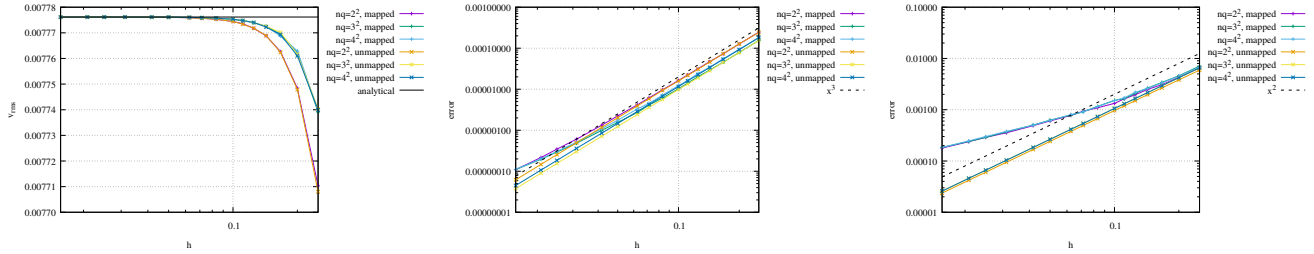
$$v(x,y) = 2xy^2(2x-1)(x-1)(y-1)^2 \tag{76.3}$$

$$p(x,y) = x(1-x)(1-2y) \tag{76.4}$$

Boundary conditions are no-slip on all sides of the unit square. The corresponding body force terms are derived in Section 18.14.



Regular square mesh



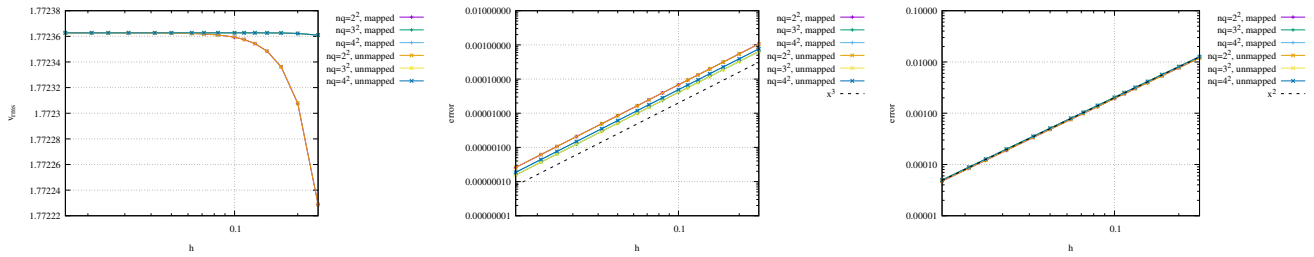
Randomized $\xi = 0.1$ mesh

Manufactured solution #2 (`bench=9`)

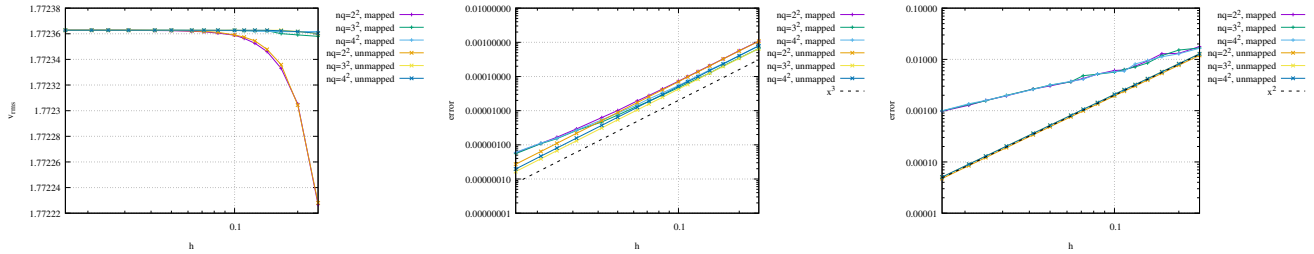
This is the second manufactured solution mentioned in Lamichhane [205]. It is presented in Section 18.7. It is for a unit square with $\eta = 1$ and the smooth exact solution is

$$u(x,y) = x + x^2 - 2xy + x^3 - 3xy^2 + x^2y \tag{76.5}$$
$$v(x,y) = -y - 2xy + y^2 - 3x^2y + y^3 - xy^2 \tag{76.6}$$
$$p(x,y) = xy + x + y + x^3y^2 - 4/3 \tag{76.7}$$

Note that the pressure obeys $\int_\Omega p \; dV = 0$. The analytical velocity is prescribed on the boundary of the domain.



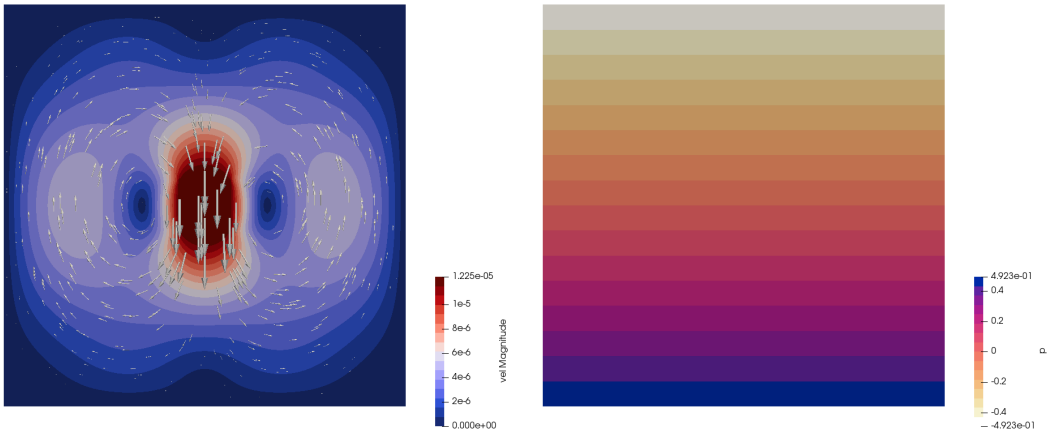
Regular square mesh



Randomized $\xi = 0.1$ mesh

Instantaneous sinking block

It is fully described in Section 19.22



Stone 77

Rotated $Q_1 \times P_0$ element & DSSY element

```
2D
box
disk
Q1P0RT
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
buoyancy_driven
vrms
paraview_output
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_77`

I have implemented the two variants of the Rannacher-Turek element presented in Section 10.4.9, the mid-value (MV) and the mid-point (MP). After communicating with Prof. Turek, I also implemented the Laplace formulation of the Stokes equation, i.e. we assume η to be constant and therefore for an incompressible flow the momentum conservation equation becomes

$$-\vec{\nabla}p + \eta\Delta\vec{\mathrm{v}} + \rho\vec{g} = \vec{0}$$

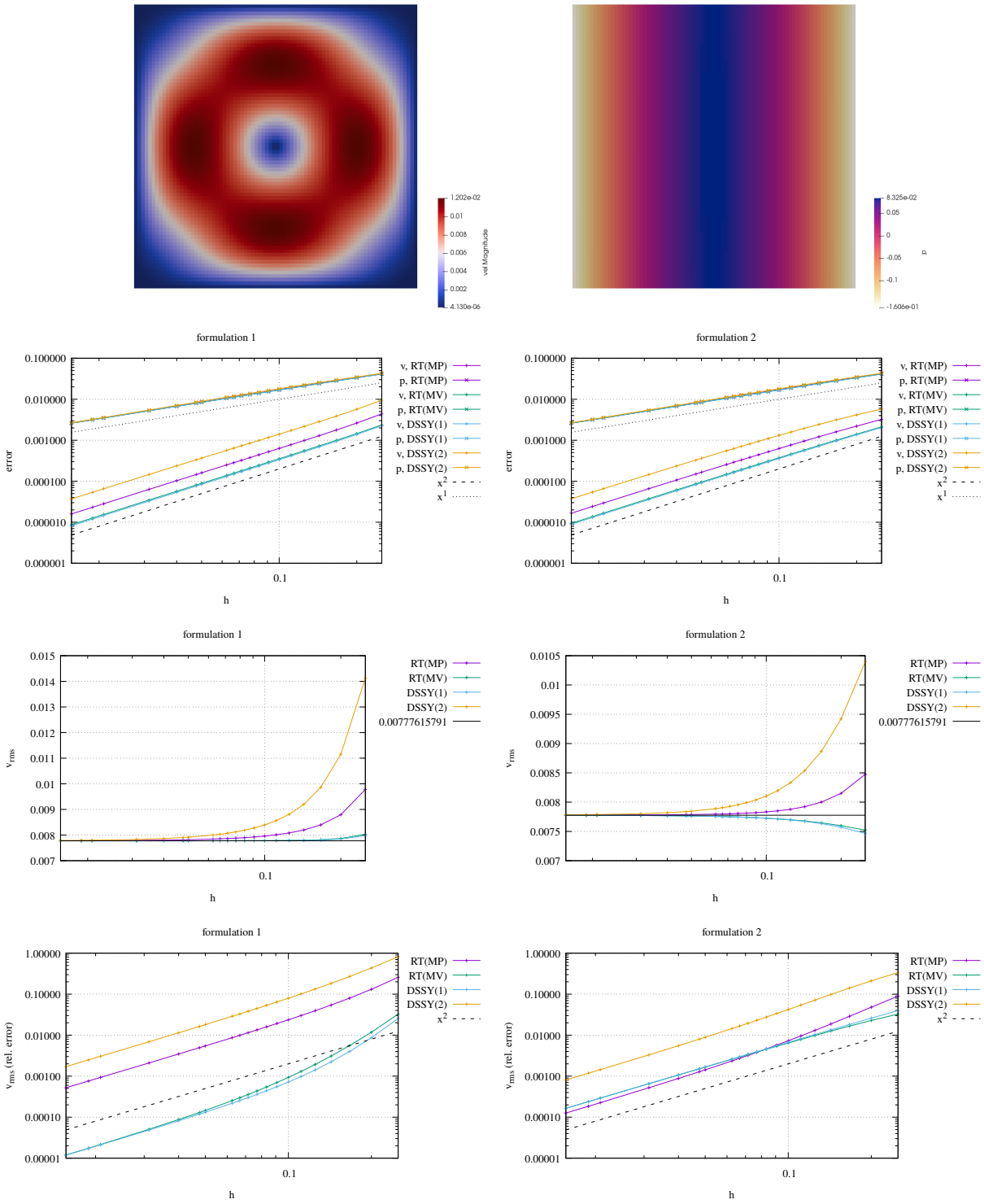
This yields a different form of the viscous block of the Stokes system as explained in Section 10.11. I use an isoparametric mapping, but I have tried a Q_1 mapping and given that all elements are square it does not change anything.

Following a discussion with W. Bangerth, I have also implemented the DSSY element (see Section **??**) which 'lives' on the same four nodes placed at the mid-edges. DSSY(1) corresponds to the basis functions based on θ_1 while DSSY(2) corresponds to the basis functions based on θ_2 [96]

In what follows formulation 1 stands for the regular form of the Stokes equation, while formulation 2 stands for the Laplace one.

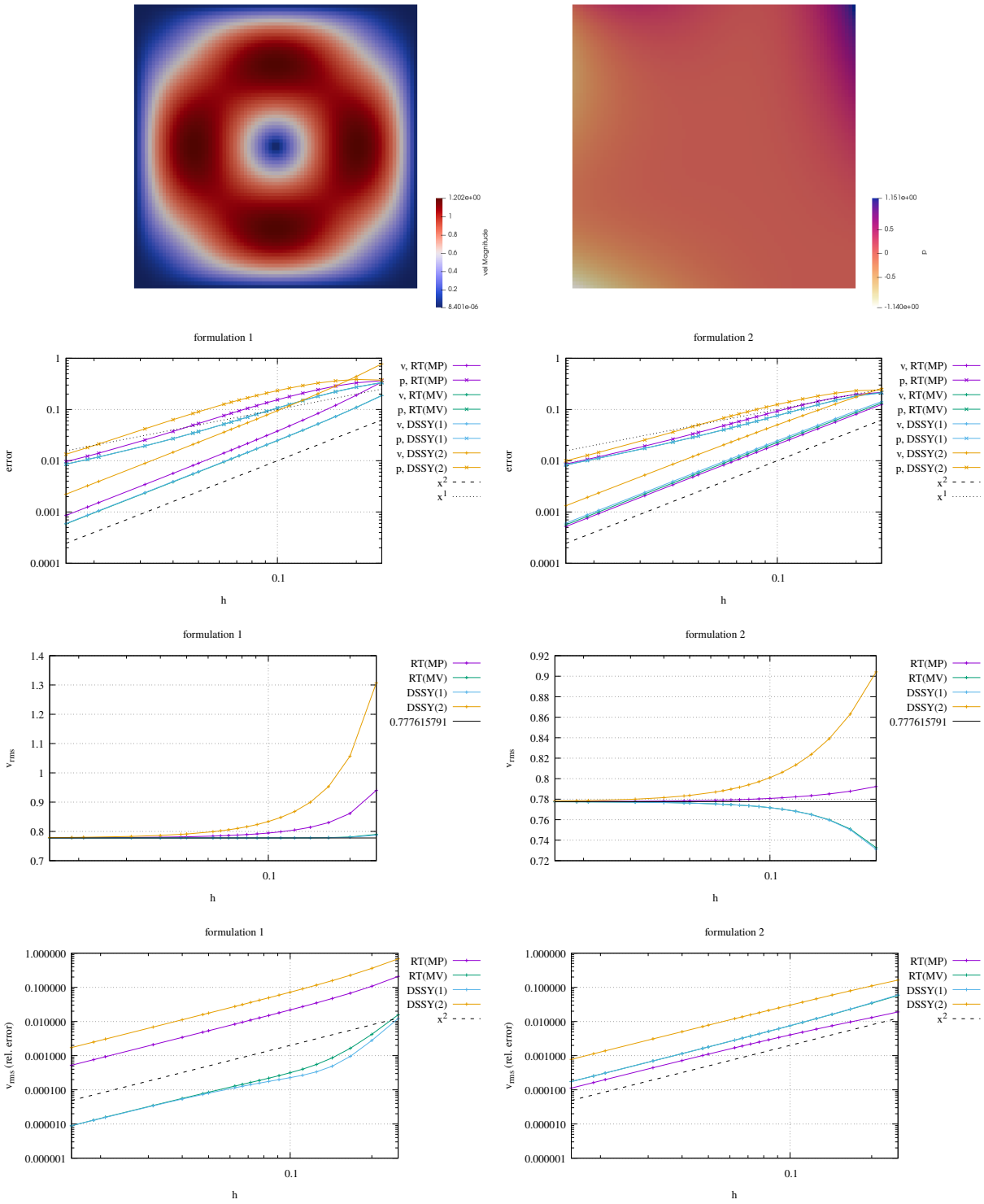
I have implemented four benchmarks based on manufactured solutions:

- Donea-Huerta, see Section 18.2



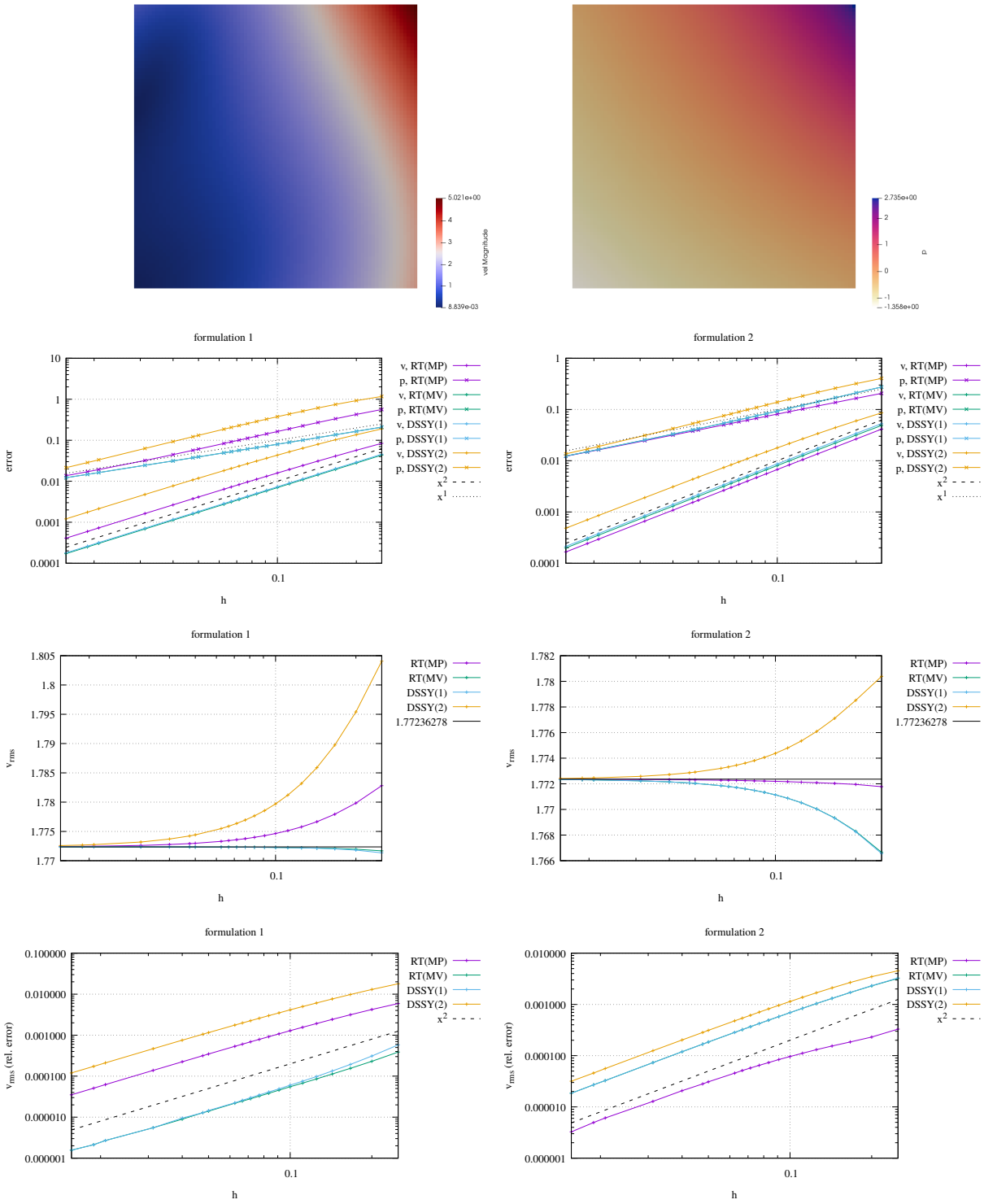
Top row: Velocity and pressure error convergence; Middle row: root mean square velocity. Bottom row: root mean square velocity relative error.

- Volker John benchmark [172], see Section 18.36:



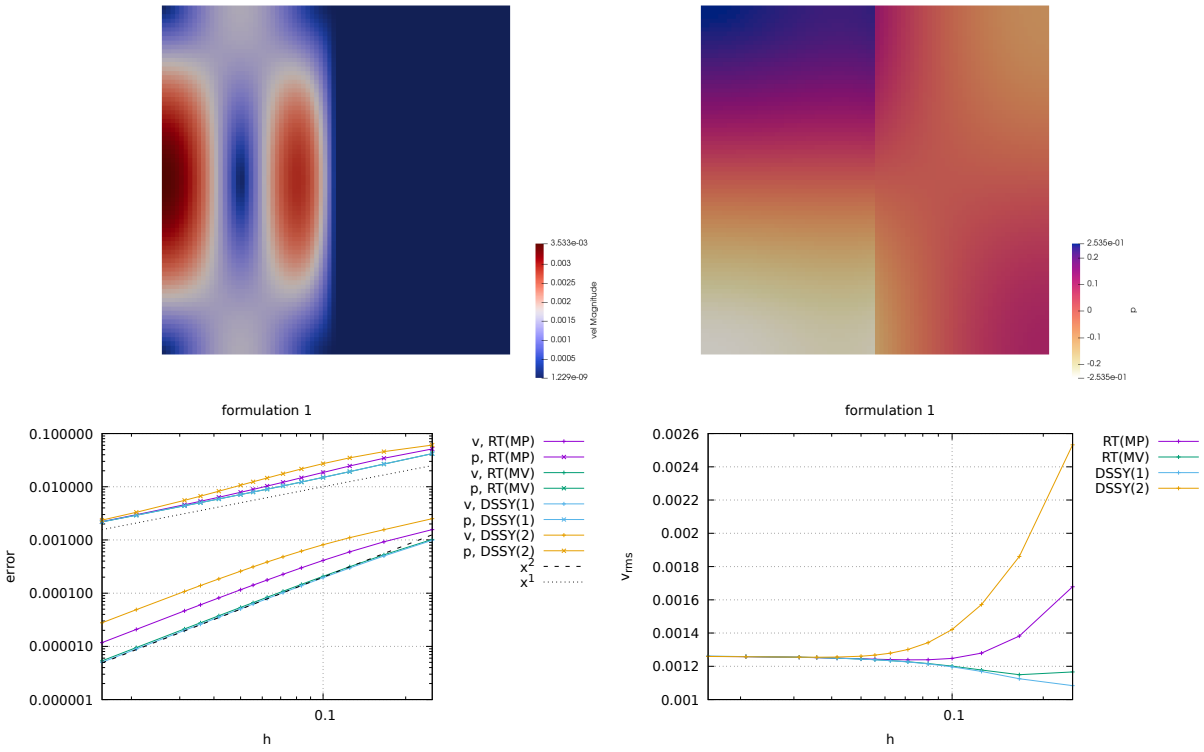
Top row: Velocity and pressure error convergence; Middle row: root mean square velocity. Bottom row: root mean square velocity relative error.

- Dohrmann & Bochev 2D benchmark [93, 24], see Section 18.7:



Top row: Velocity and pressure error convergence; Middle row: root mean square velocity. Bottom row: root mean square velocity relative error. Formulation 1 stands for the regular form of the Stokes equation, while formulation 2 stands for the Laplace one.

- SolCx benchmark, see STONE 5 (only formulation 1):



Only even numbers of elements. Note that the paraview visualisation is based on quadrilaterals obtained by joining the nodes.

Looking at these 4 analytical benchmarks (and focusing on the standard formulation only), we can conclude that RT(MV) and DSSY(1) are the two best elements and that they are very close to each other.

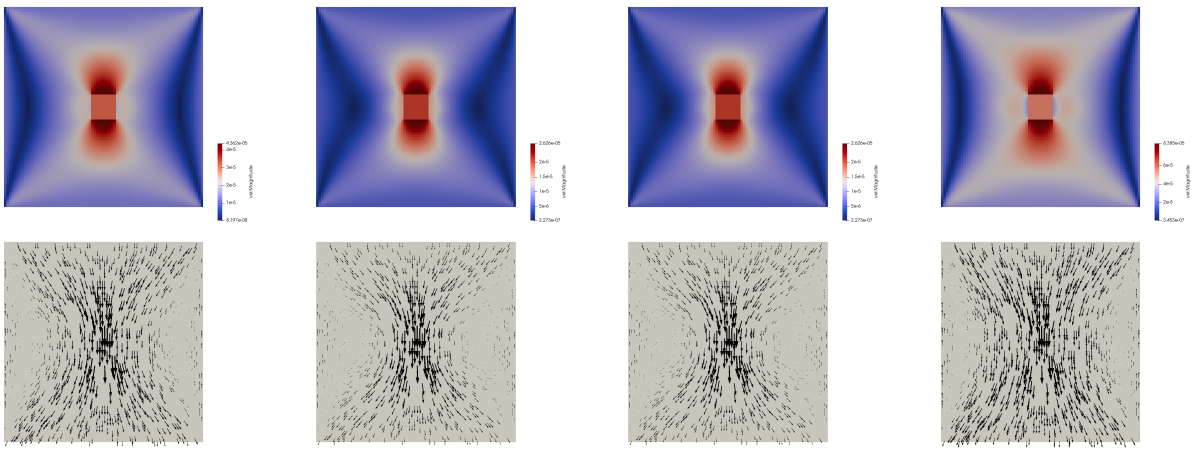
Buoyancy-driven flow

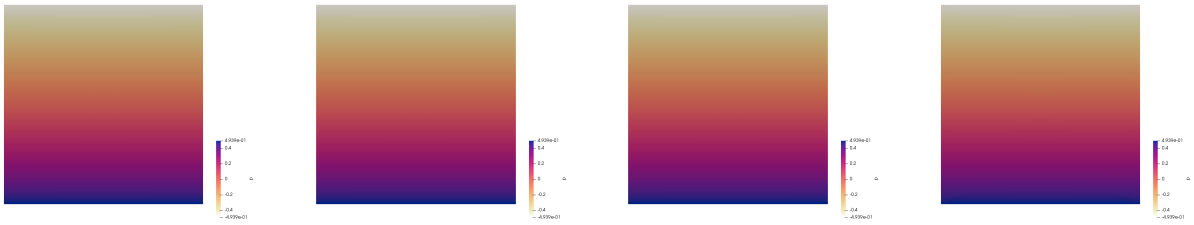
We see that both RT(MV) and DSSY(1) exhibit a quadratic convergence for the velocity error and a linear convergence for the pressure error. Also the root mean square velocity measurements logically converge to their analytical values.

Since these elements have been proven to be LBB stable one could think that it should then replace the standard $Q_1 \times P_0$: near identical cost, but LBB stable.

However, there is a problem. I have also implemented another experiment: a unit cube domain filled with a fluid of density $\rho_0 = 1$ and a cube of size 0.125×0.125 centered in the domain with density $\rho_0 + \delta\rho$ with $\delta\rho = 0.01$. The fluid has a viscosity $\eta = 1$ while the block has a viscosity $\eta = 1000$. Boundary conditions are no slip on all sides, gravity points downwards with $|\vec{g}| = 1$.

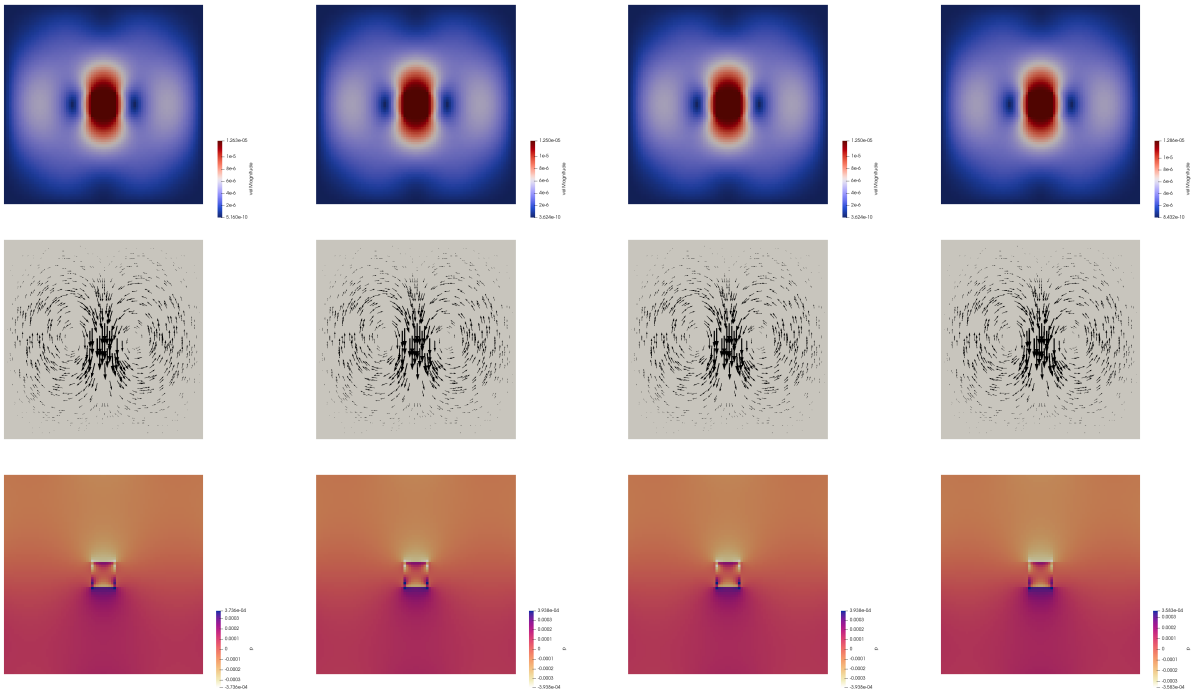
Results for all four basis functions are shown here:





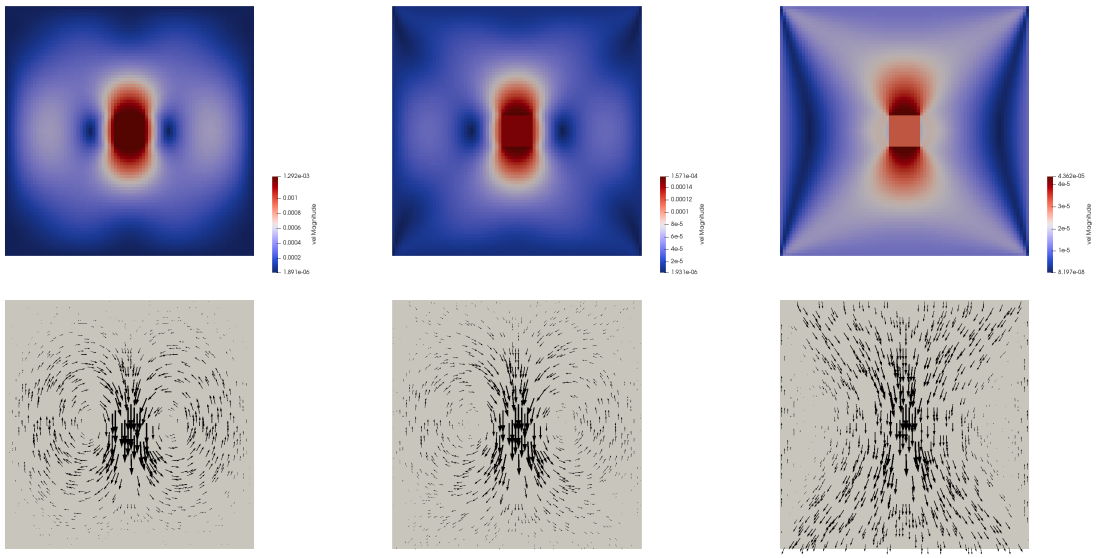
Full density. Left to right: RT(MP), RT(MV), DSSY(1), DSSY(2). 64×64 elements

We see that the pressure field is lithostatic and chequerboard-free but the velocity field is abnormal (we expect a convection cell on each side of the cube). If I now re-run these experiments in reduced density ($\rho_0 = 0$) then we recover the expected velocity field[1]:

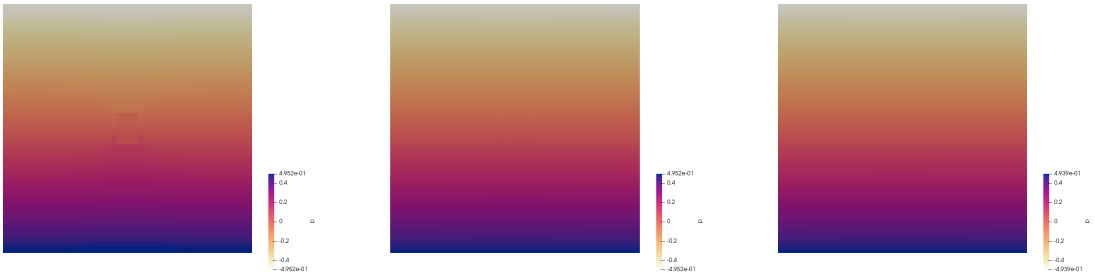


Reduced density. Left to right: RT(MP), RT(MV), DSSY(1), DSSY(2). 64×64 elements

If I now conduct a short study on the value of $\delta\rho$:



[1]Not saying it is 'the' solution but it at least makes sense



From left to right: $\delta\rho = 1, 0.1, 0.01$. 80×80 elements.

The conclusion is clear: in its current form, and unless there is a fundamental flaw in my implementation[2], these elements are not capable to deal with buoyancy-driven flows where $\delta\rho/\rho < 1 - 10\%$ which is unfortunately the type of simulations that are carried out in mantle dynamics modelling. The reasons are (partially) discussed in Section 10.4.9, but I am not sure how to go further. If an additional jump term is needed in the weak form in order to fix this problem, this makes the element not as simple as advertised.

[2]but then how could the anlalytical benchmarks work so well?

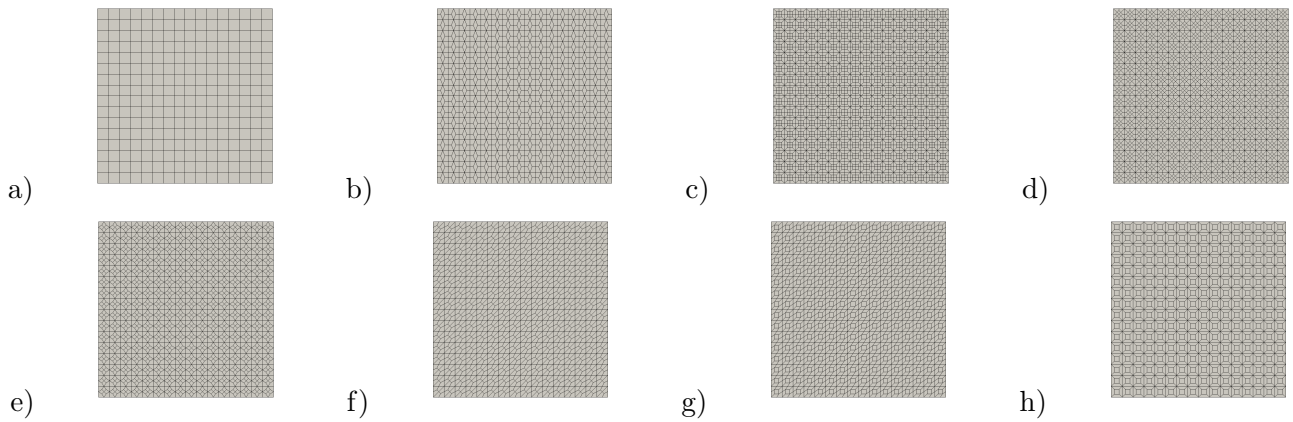
Stone 78

$Q_1 \times P_0$ macro-element

```
2D
box
Q1P0
sparse_matrix
analytical_benchmark
numerical_benchmark
error_convergence
stokes_equation
buoyancy_driven
vrms
paraview_output
pressure_scaling
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_78`

Although the $Q_1 \times P_0$ is not LBB-stable (see Section 10.2) it has been proven that some spatial arrangements of this element can be, such as the so-called Stenberg macro-element described in Section 10.12. On the following a mesh is shown which consists of 16×16 of such macro-elements:



opla

The meshes are created by importing the corresponding files:

```
import letallec
import stenberg
import qinzhang
import regular
```

Inside each of these files a function called *mesher* is defined:

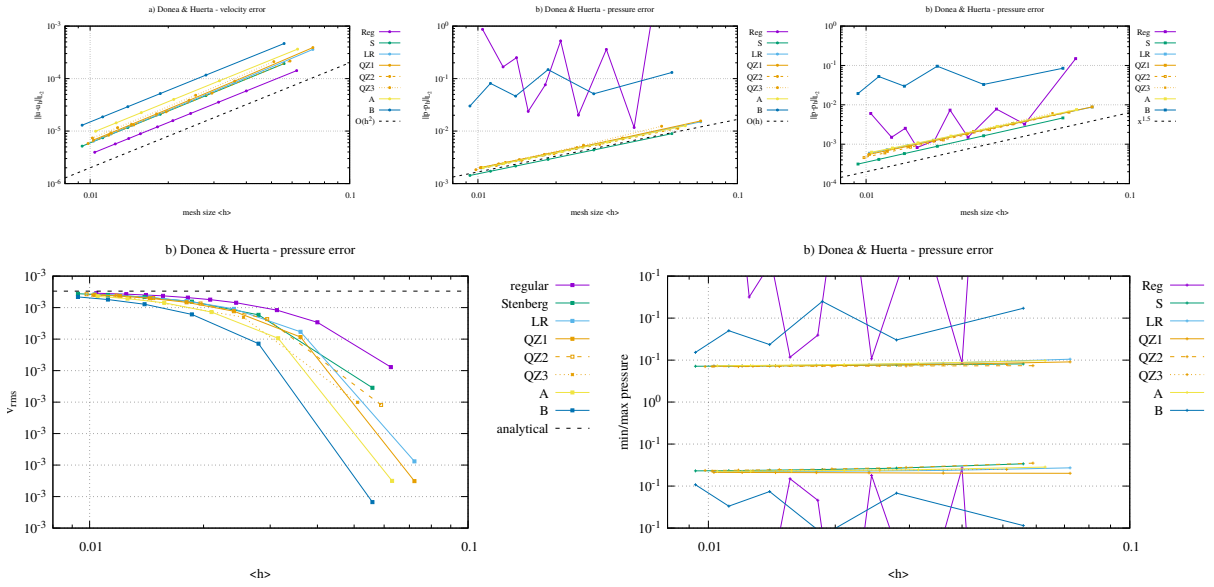
```
def mesher(Lx,Ly,nelx,nely,nel,NV,mV):
```

The arguments are the domain size L_x and L_y, the number of macroelements in each direction $nelx$ and $nely$, the precomputed total number of $Q_1 \times P_0$ elements nel and nodes NV and the number of nodes per element m_ν.

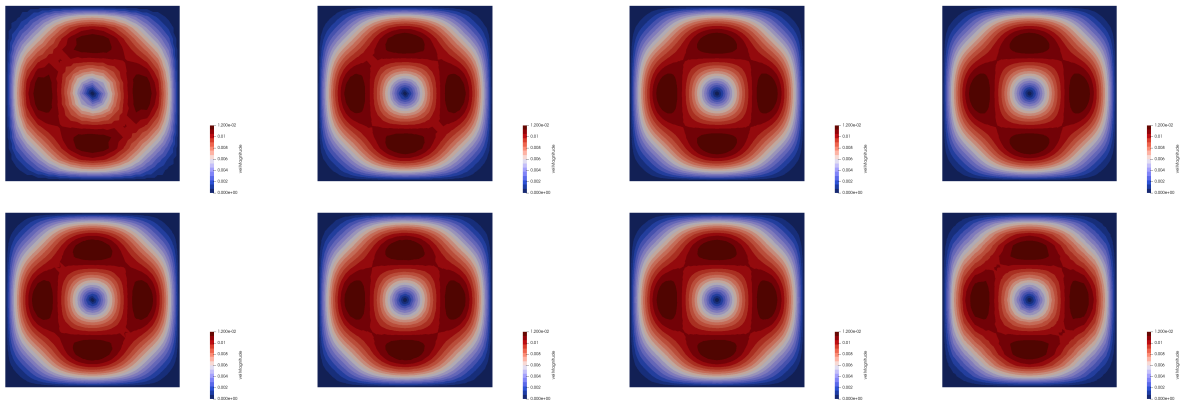
Pressure normalisation is achieved by setting $p = 0$ on the last element and then renormalising the pressure field so that $\int p dV = 0$. In what follows p is the 'raw' pressure field while q denotes its projection onto the nodes.

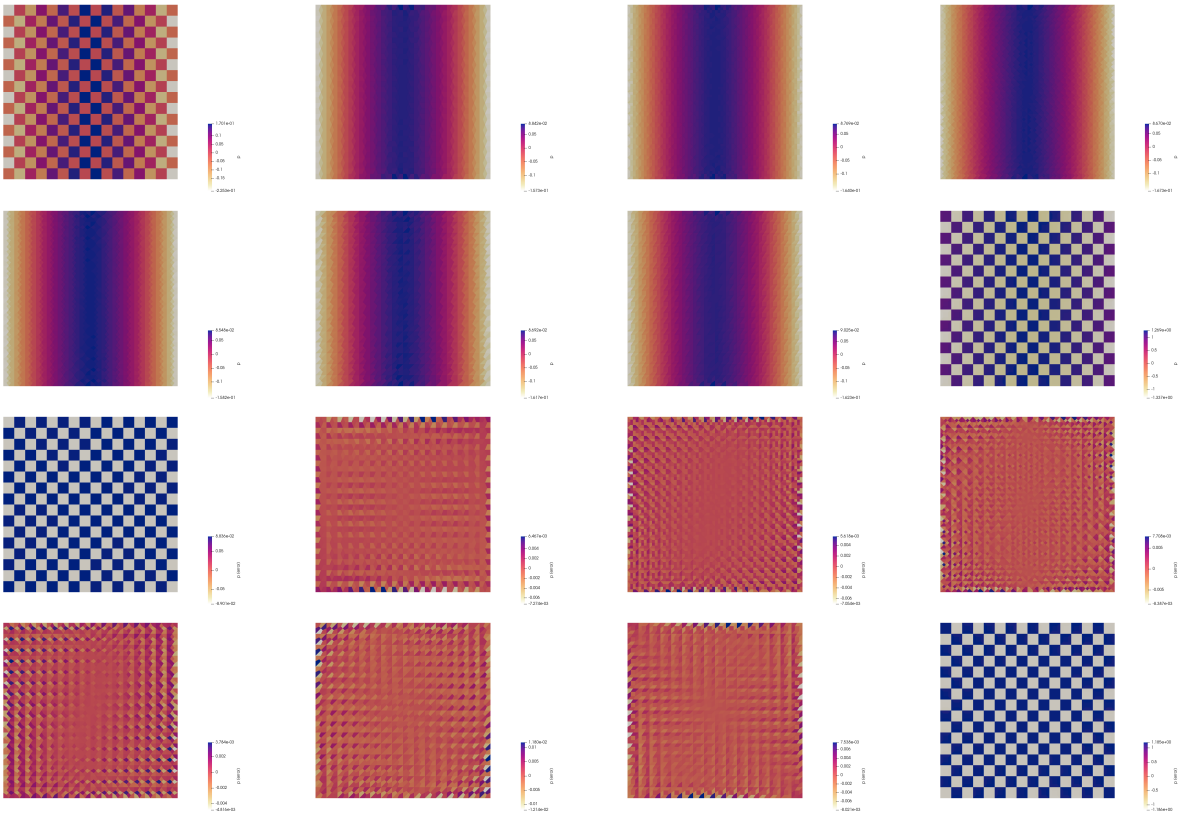
note that the viscosity is evaluated in the middle of each element, not at the quadrature points. This can/will have an effect on non-isoviscous models.

Results - mms We start with the Donea & Huerta manufactured solution (see Section 18.2) and proceed to compute the velocity and pressure error convergence as a function of the element size which is taken to be $h = \sqrt{L_x L_y / nel}$. We see that the errors converge as expected, quadratically for the velocity and linearly for the pressure. Rather interestingly the projection of the pressure onto the nodes has a convergence rate higher than the raw elemental pressure. As predicted in Qin & Zhang, the Stenberg macro-element yields the best results, followed by the Le Tallec and then the one they propose (this conclusion is logically supported by looking at root mean square velocity measurements). Finally, the presence of the checkerboard for the regular structure mesh case makes it painfully clear that it is the worst mesh topology and the pressure error does not converge.



On the following figures the pressure is plotted against the analytical solution and we see that there is no checkerboarding occurring: Rather interestingly the pressure error is the largest next to the boundaries:

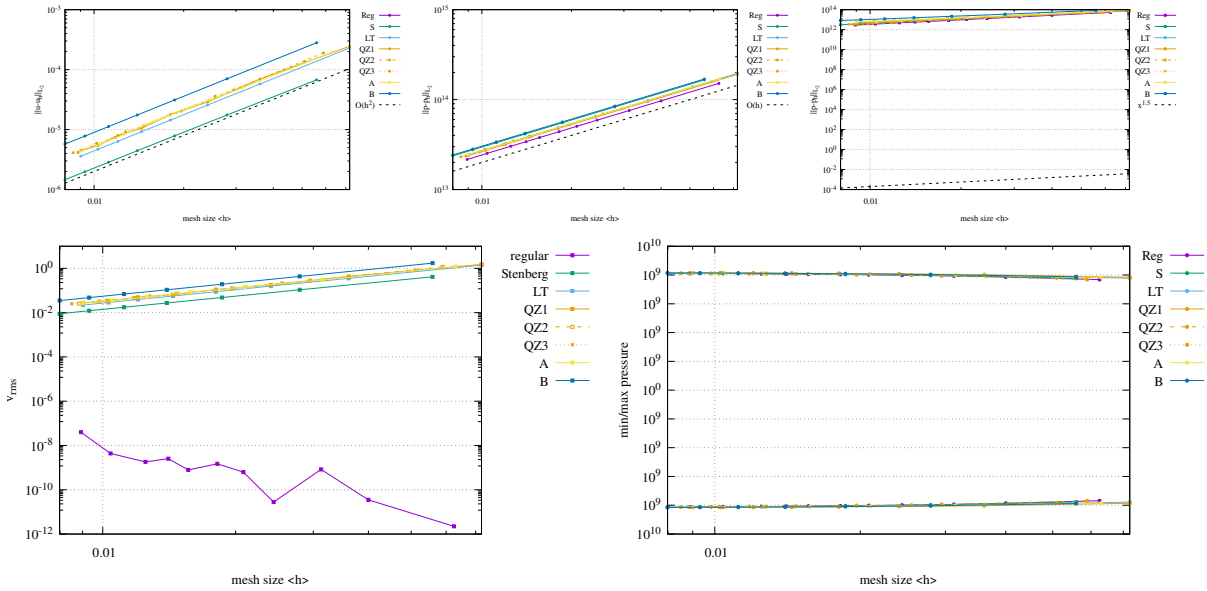




Meshes made of 16x16 (macro-)elementscount.

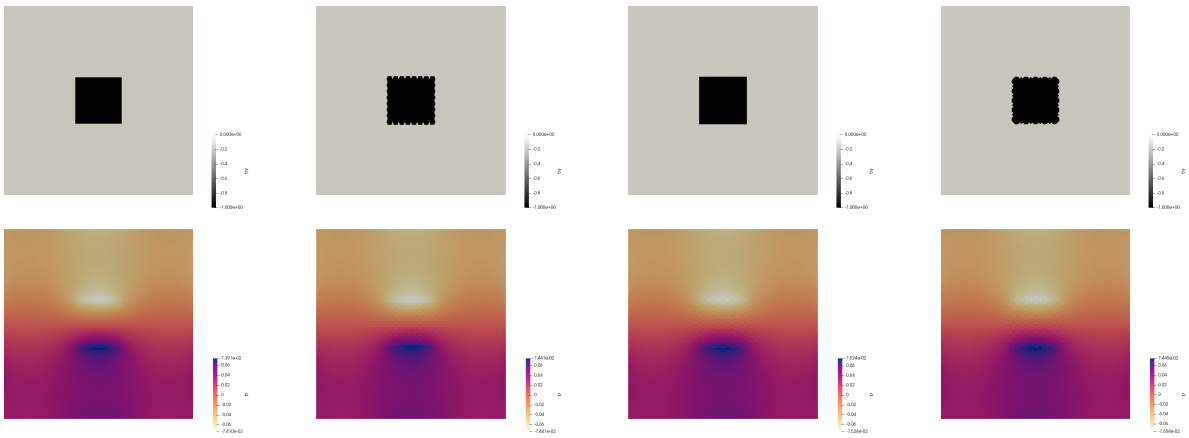
Discussion: what is the real advantage of such a macro-element? it is LBB stable, so iterative solver will work optimally, and the pressure has no checkerboard. Also the number of non-zeros per line of the matrix is small. On the other hand it is anisotropic since the 'diamonds are vertical'. Also if one would consider a macro-element as an element, it counts 10 velocity nodes and 5 pressures, which makes it much more expensive than a $Q_2 \times Q_1$ element of the same size...

Results - Aquarium



We see that the Stenberg macro-element is the best, with the lowest velocity and pressure errors.

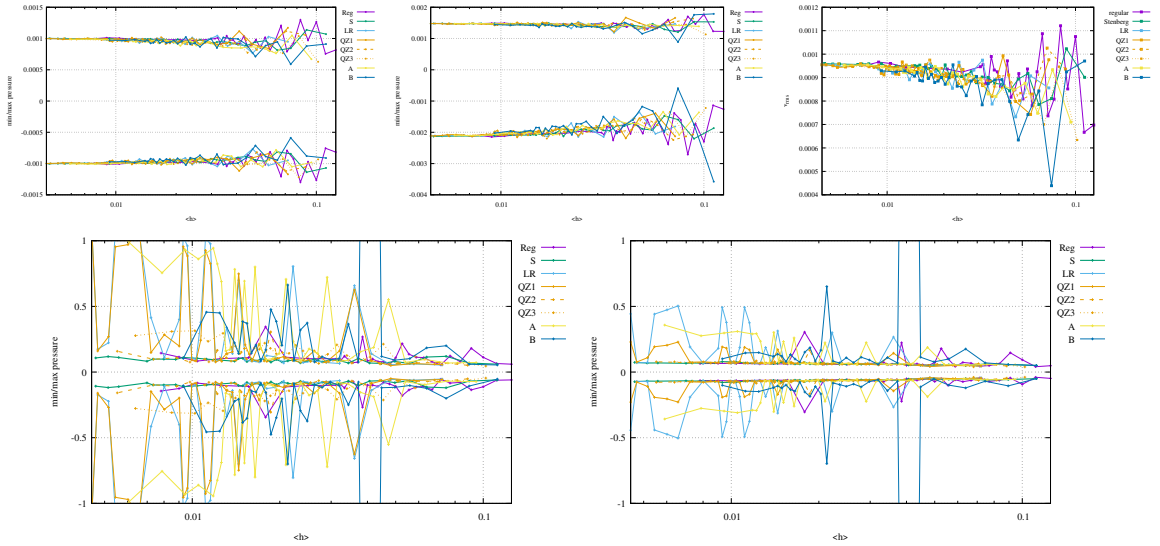
Results - sinking block - reduced density The block has size 0.25×0.25, centered in the domain. No-slip boundary conditions are imposed on all sides. The buoyancy force ρg_y is -1 in the block and zero elsewhere. Viscosity is constant and equal to 1 everywhere.



Velocity field (top row) and pressure field (bottom row) for regular, Stenberg, Le Tallec, Qin & Zhang.

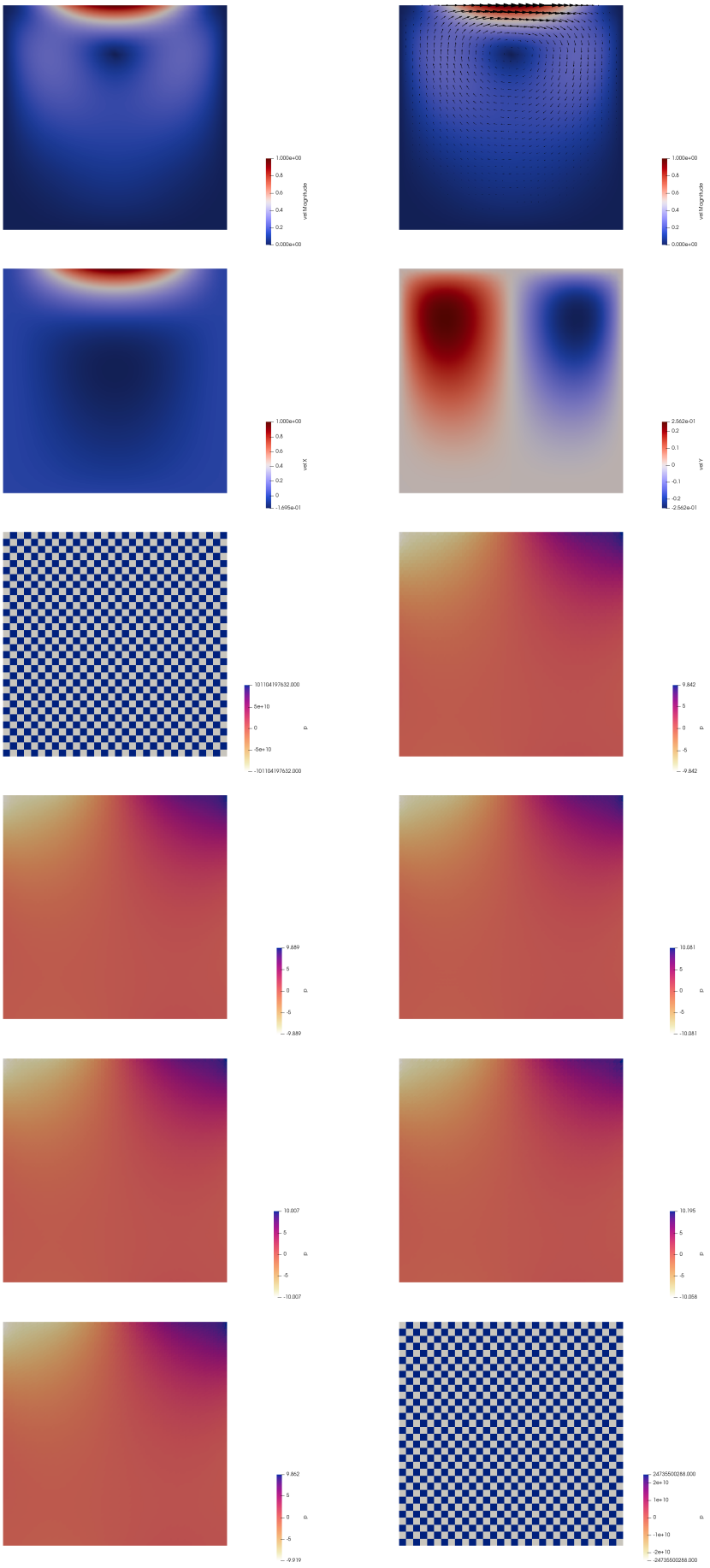
Results - sinking block - full density We then proceed with $\rho g_y = -1$ in the surrounding fluid and $\rho g_y = -(1 + \delta)$ in the block.

Results - Stokes sphere This is the same experiment as above, except for the geometry of the object which is now a sphere of radius 0.125 so that the interface between both fluids is never aligned with the mesh/element edges. The sphere has density is 1 while the surrounding is 0. Its viscosity is 10^4 while the surrounding is 1.

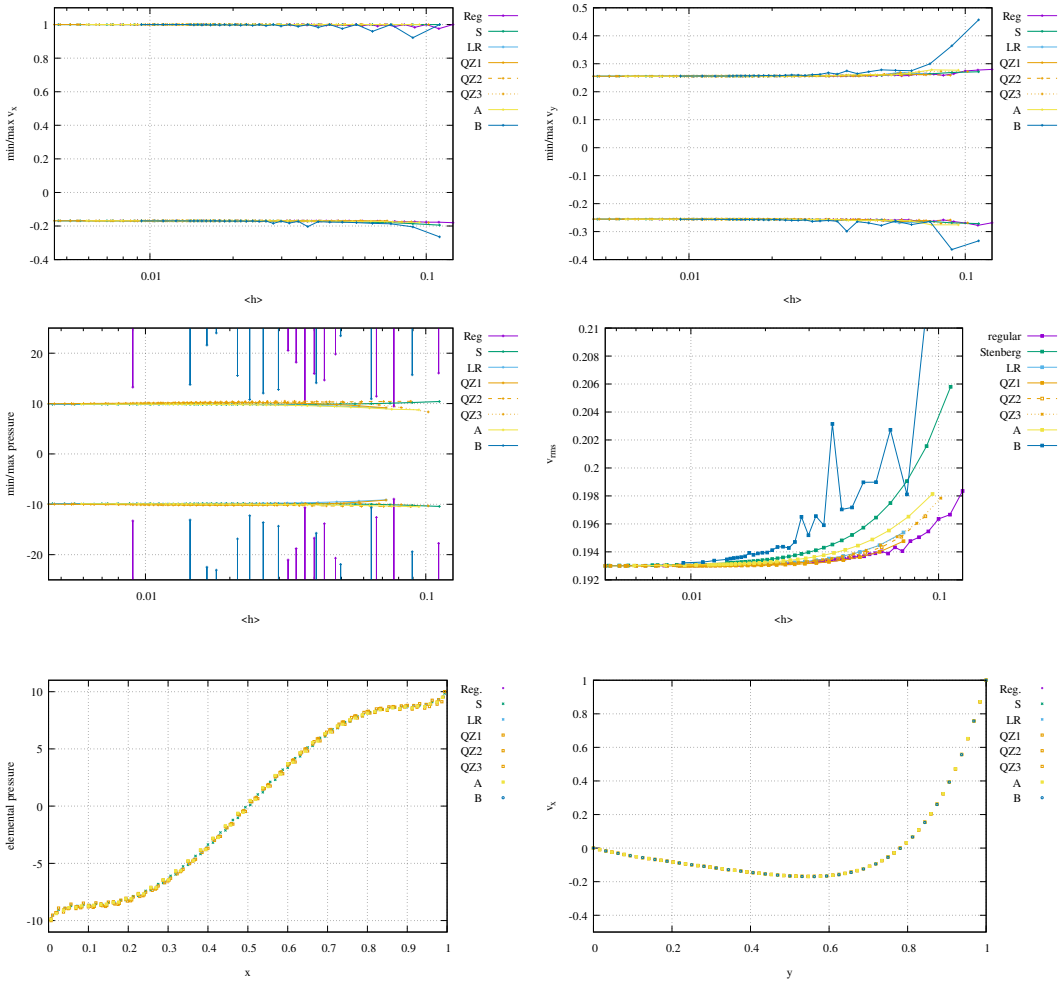


All (macro-)elements converge to the same vrms and show very similar velocity and pressure statistics. Stenberg is undeniably the best in terms of pressure under/over shoot.

Results - Regularised lid driven cavity Domain is 1x1. Viscosity is 1 and density is zero. No-slip prescribed at sides and bottom and $\vec{\mathbf{v}} = (16x^2(1-x^2), 0)$ prescribed on the top.

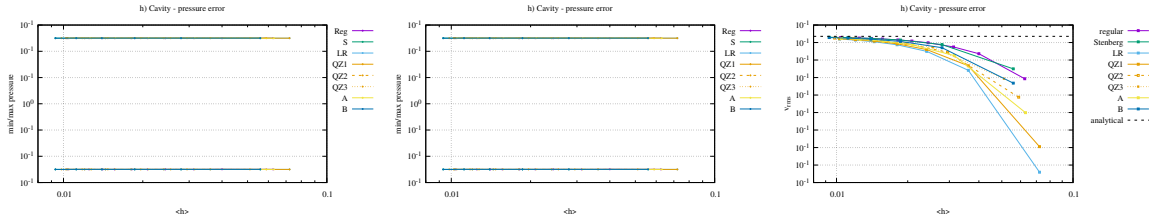


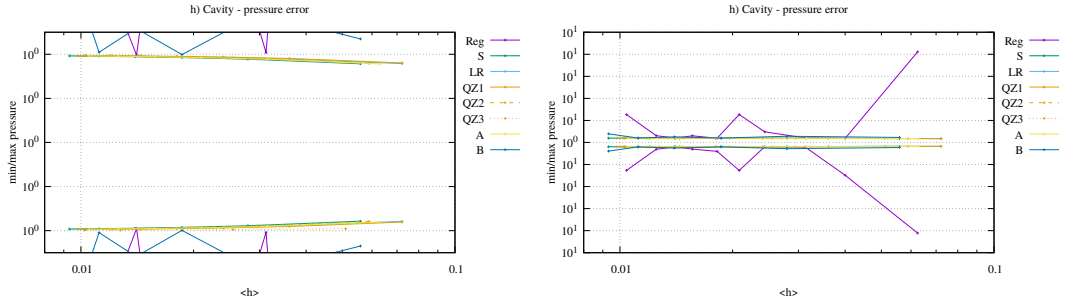
32x32 mesh. Note the insane amplitude of the checkerboard modes!

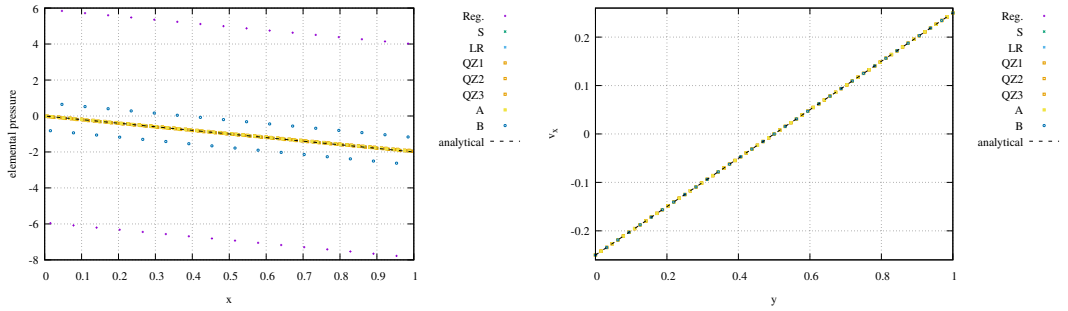


Left: Pressure at the surface; Right: velocity on vertical line in the middle

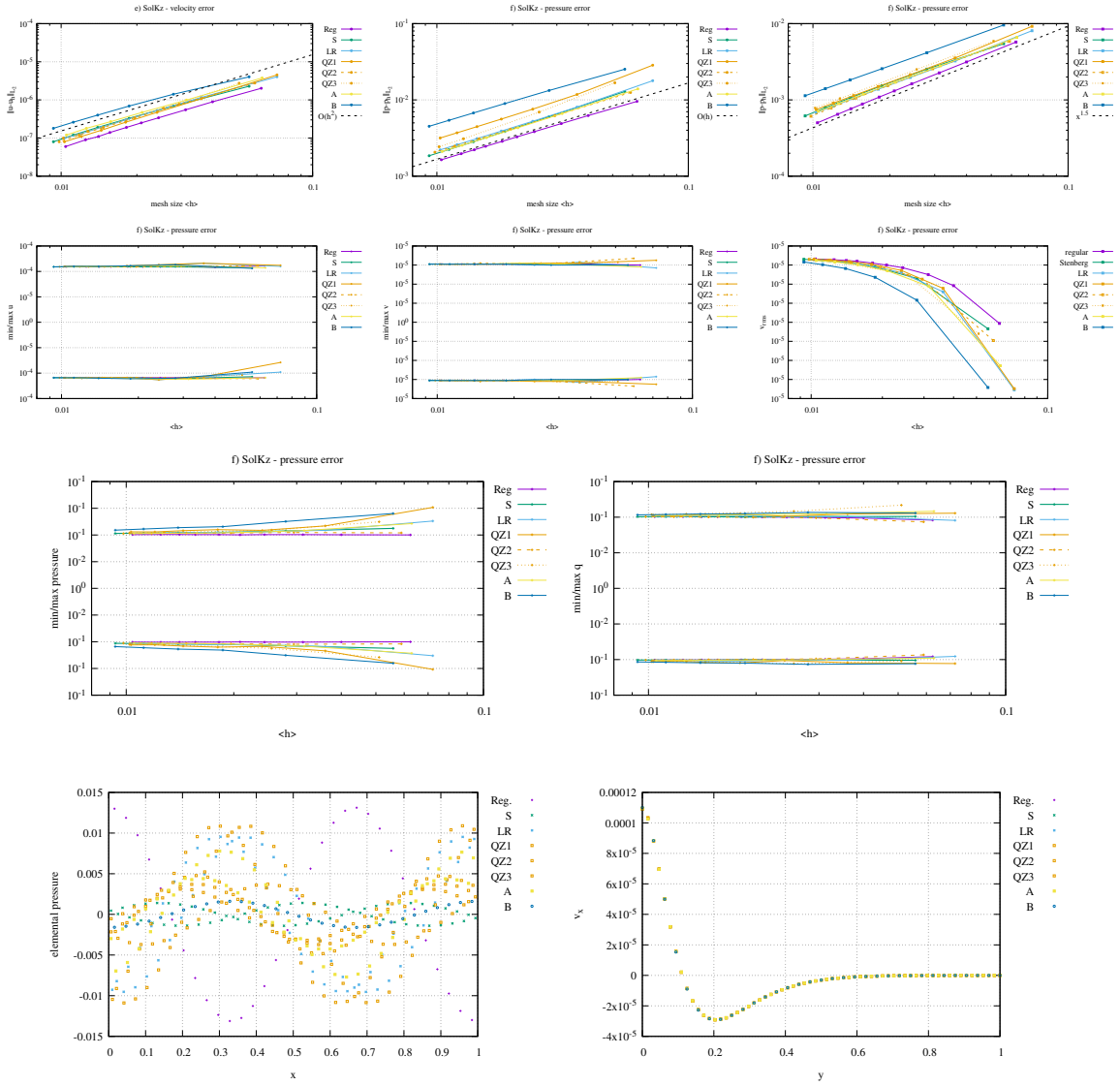
Results - analytical cavity





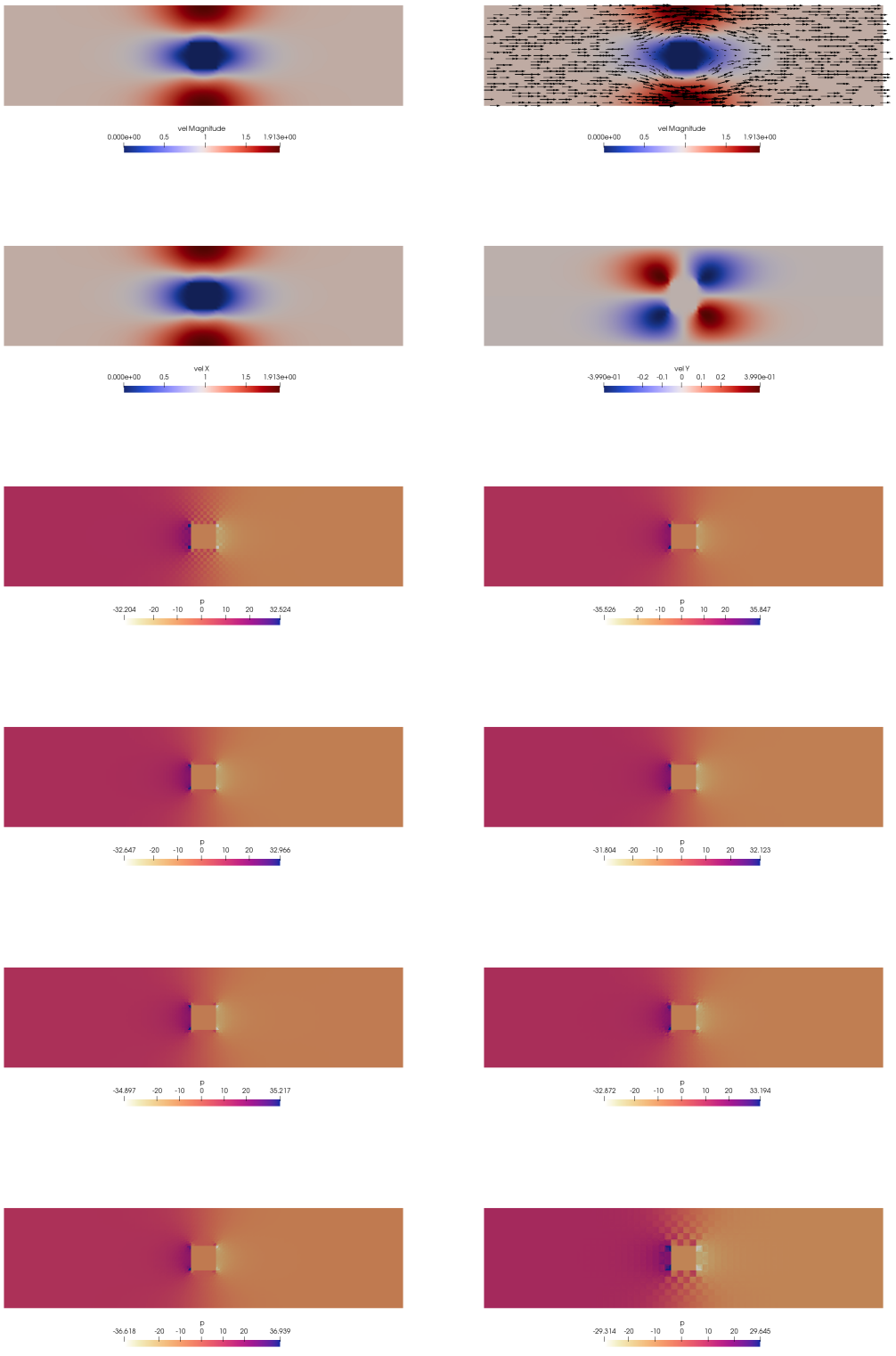


Results - SolKz



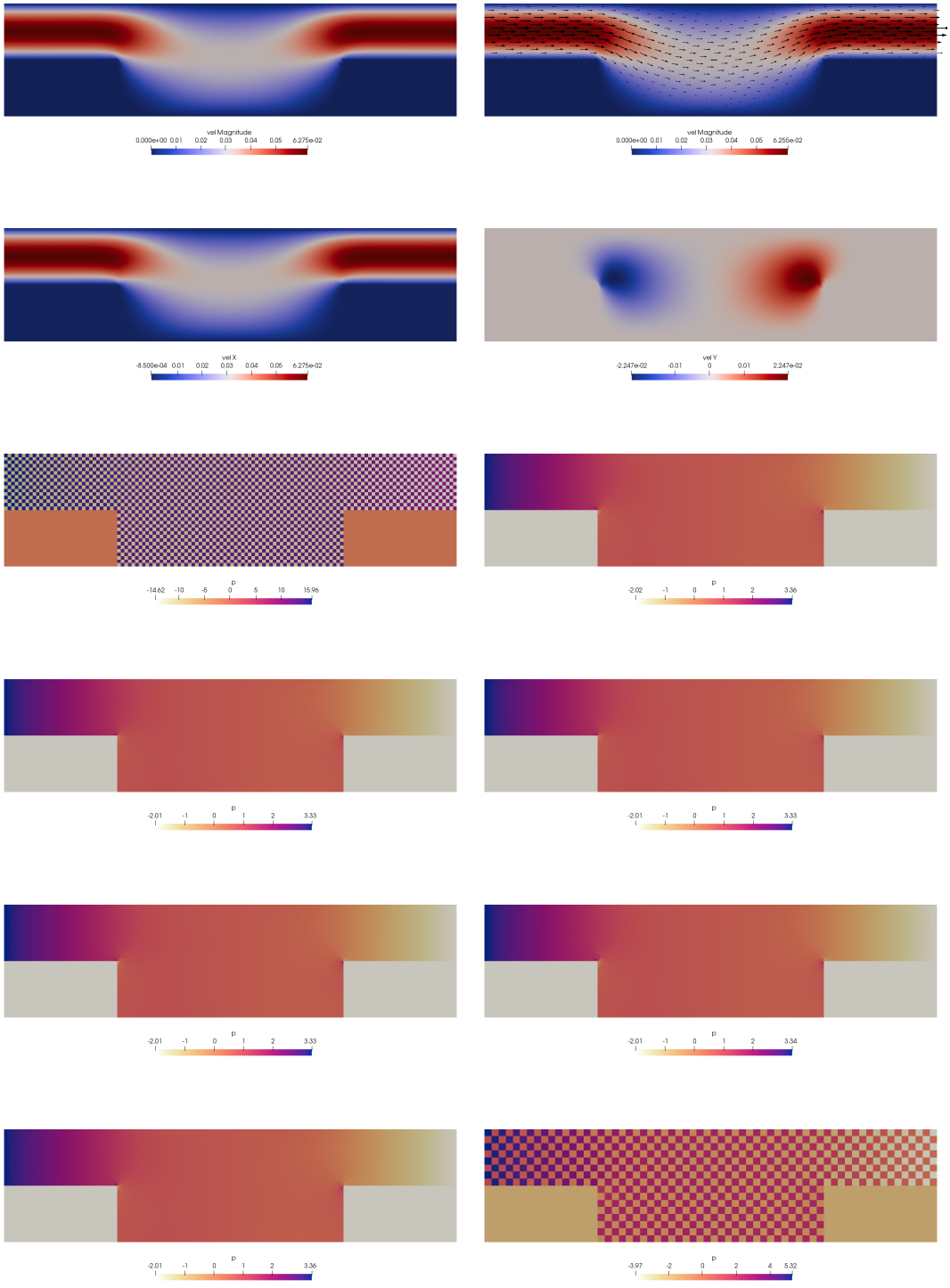
Results - square cylinder This experiment is inspired by the very common one found in CFD: a laminar flow is forced to go around a cylinder (circular or square cross section) often generating turbulence in its wake for high-enough Reynolds number Navier-Stokes flow.

The domain is 4x1. Free slip boundary conditions are imposed top and bottom. A horizontal flow $\vec{v} = (0, 1)$ is prescribed on the left boundary, and the right boundary is left open. All the nodes on a square centered in the middle of the domain of size 0.125x0.125 are set to no-slip boundary conditions. Viscosity is one and gravity is switched off.



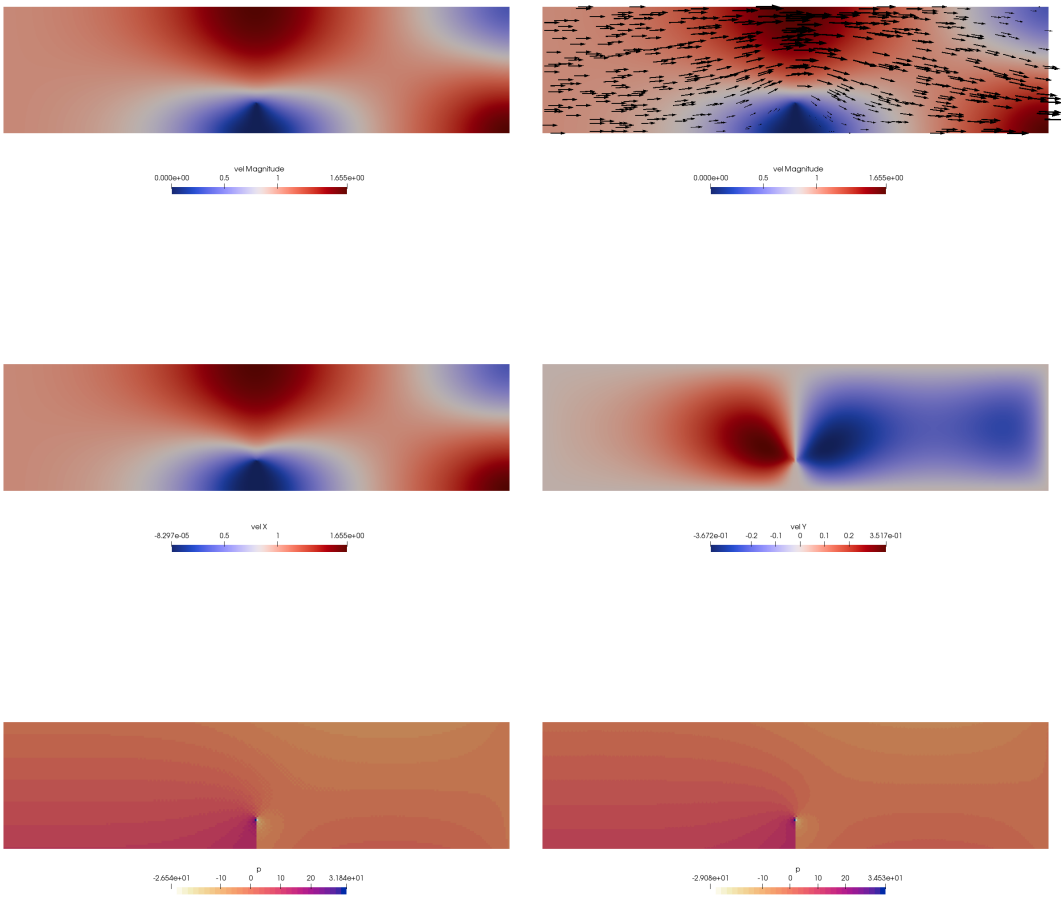
run with script_cylinder

Results - flow over a cavity (experiment=11) The domain is 4x1. No slip boundary conditions are imposed on all walls. The cavity is formed by prescribing $\vec{\nu} = \vec{0}$ on the right nodes inside the domain. $\vec{\nu} = -(y - L_y)(y - L_y/2)$ is prescribed on the left inflow boundary while $\nu_y = 0$ is prescribed on the outflow boundary. Viscosity is one and gravity is switched off.



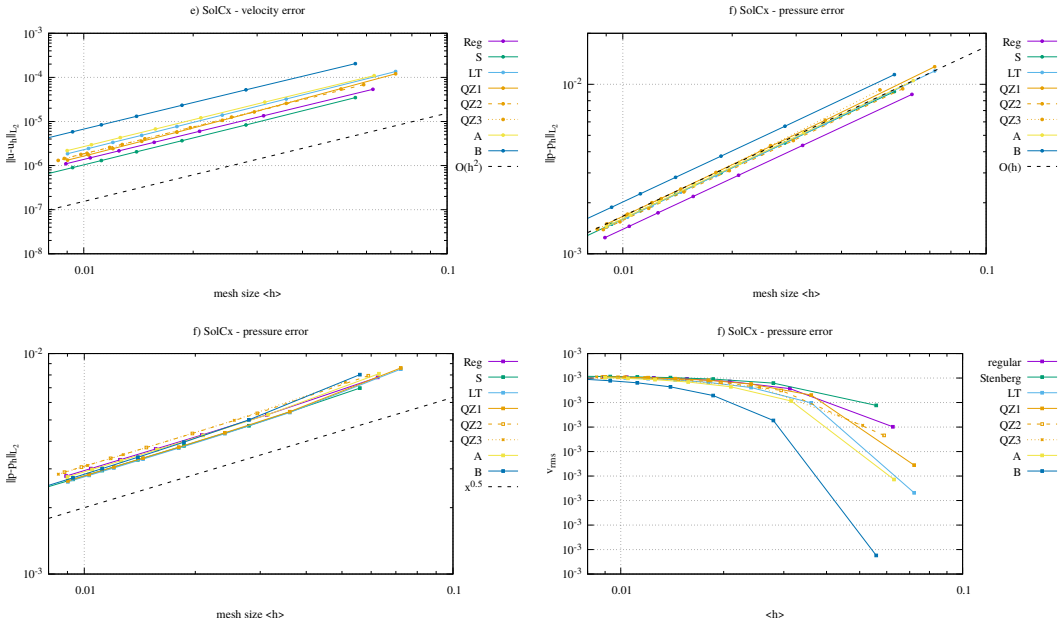
run with script_cavity

Results - flow over an obstacle (experiment=12) Domain is 4x1. Viscosity is 1, $rhog = -1$. Free slip top and bottom. Obstacle is no slip and is composed of nodes at $x = L_x/2$ and $y \leq L_y/2$. $\vec{\nu} = (1, 0)$ prescribed on the left, $\nu_y = 0$ on the right.

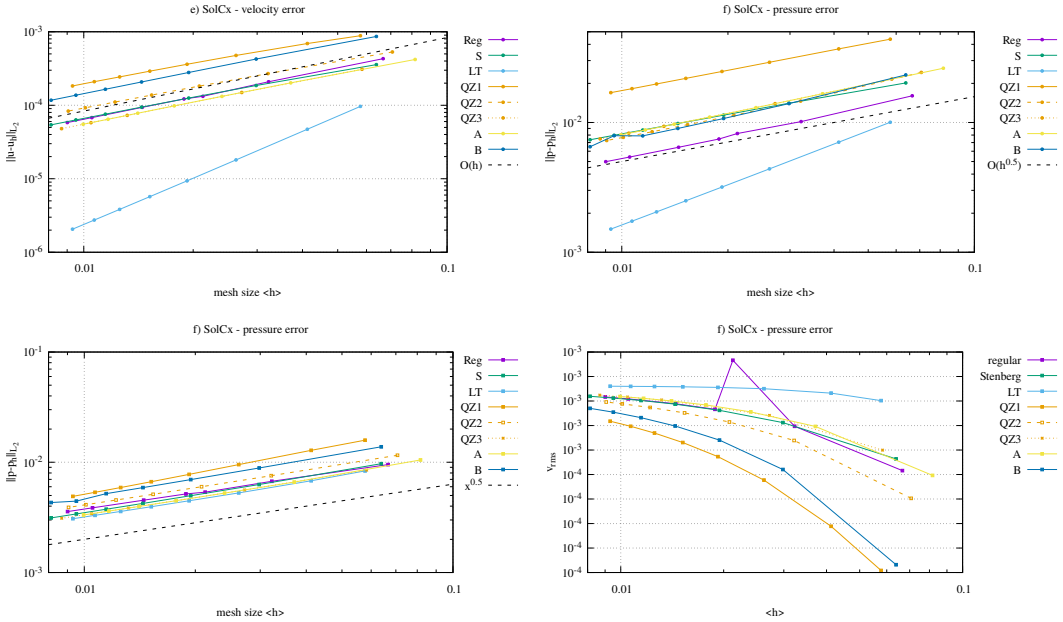


run with script_obstacle

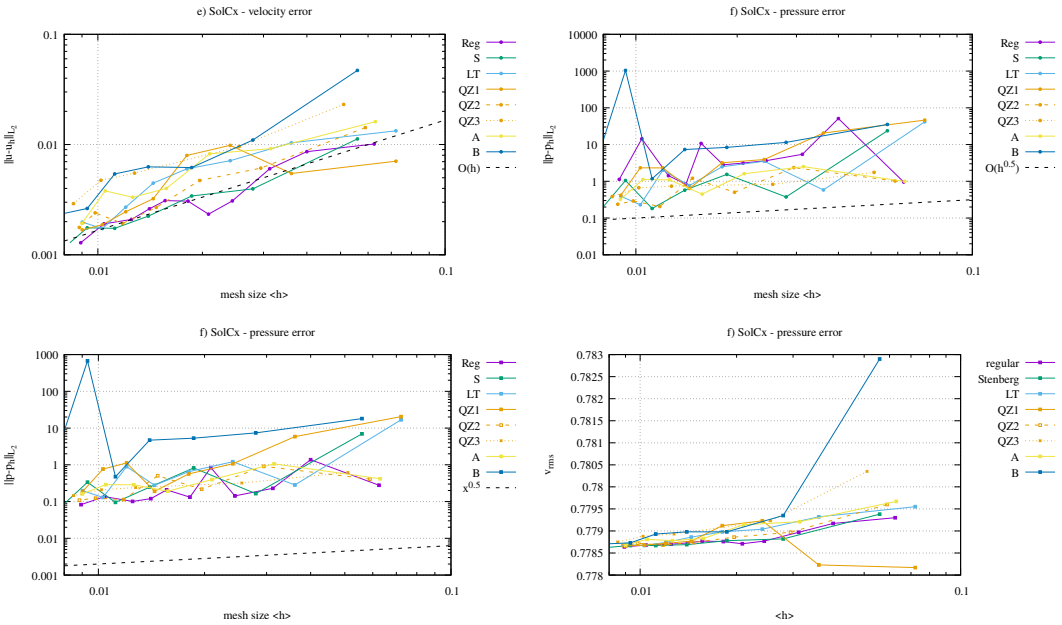
Results - SolCx mms (experiment=13) Even number of elements:



Odd number of elements:



Results - Solvi mms (experiment=14)



Stone 79

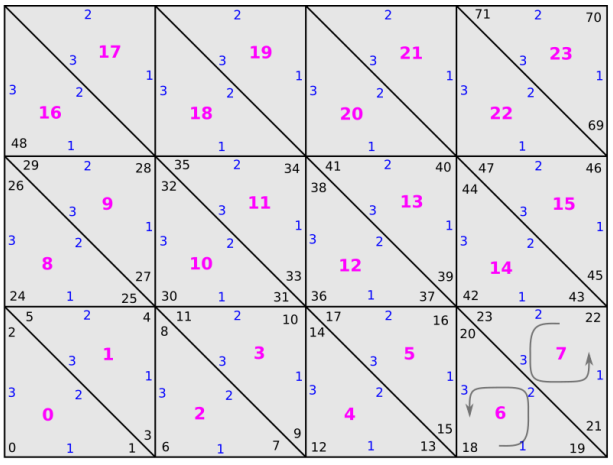
DG-FEM: 2D steady state diffusion

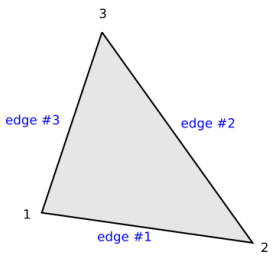
2D
box
P1
Q1
analytical_benchmark
numerical_benchmark
error_convergence
energy_equation
residual
paraview_output
steady_state
discontinuous_galerkin

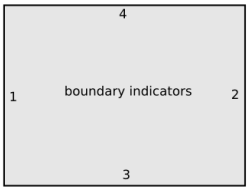
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_79`

This stone was developed in collaboration with Jort Jansen.

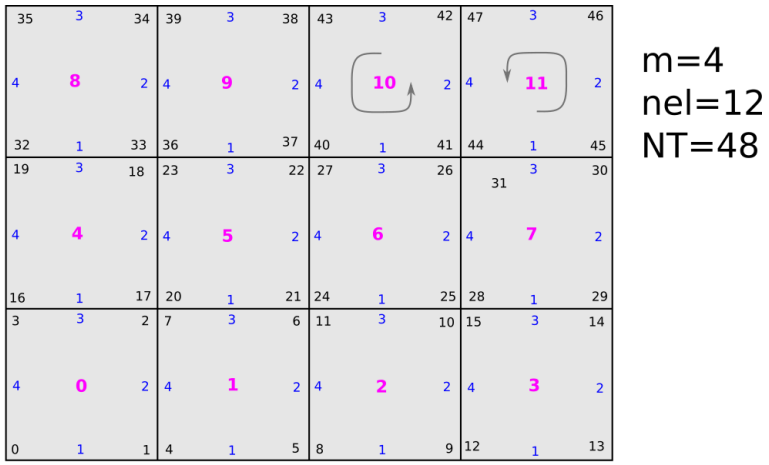
Before reading what follows I urge you to go to Chapter 13. The derivations presented therein are long and complex, but absolutely necessary to make sense of the code. The code has been written for both bilinear (Q_1) quadrilateral elements and and linear (P_1) triangular elements. Meshes are then as follows:

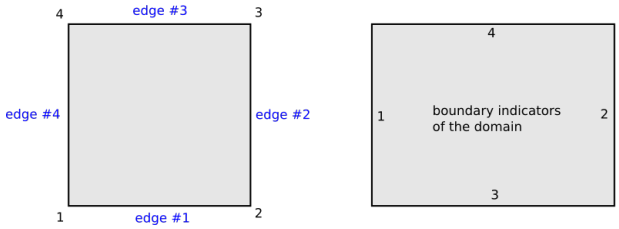






Example of a 4 × 3 triangular mesh.





Example of a 4 × 3 quadrilateral mesh.

Contrarily to standard continuous finite elements nodes/vertices are not shared across elements so that the node layout is entirely based on the already built connectivity array `icon`. Node coordinates are stored in `xT` and `yT`.

Six ascii output files are opened at the beginning of the code, and in these various statistics will be written out. These are *T_stats.ascii*, *qx_stats.ascii*, *qy_stats.ascii*, *residual_T_stats.ascii*, *residual_qx_stats.ascii*, and *residual_qy_stats.ascii*.

Then a large number of arrays is declared. Then contain information that is needed when building the matrices and vectors later on and it is convenient to store this information beforehand. They all start with `edge` and record the normal to the edge, its center coordinates, its length, whether it is on the domain boundary, etc ... Because this is an educative code with a simple mesh topology we can directly ascribe the normals to each side. If a mesher, like Triangle [293], was used then we would need to compute the normals to each side by means of simple geometrical considerations.

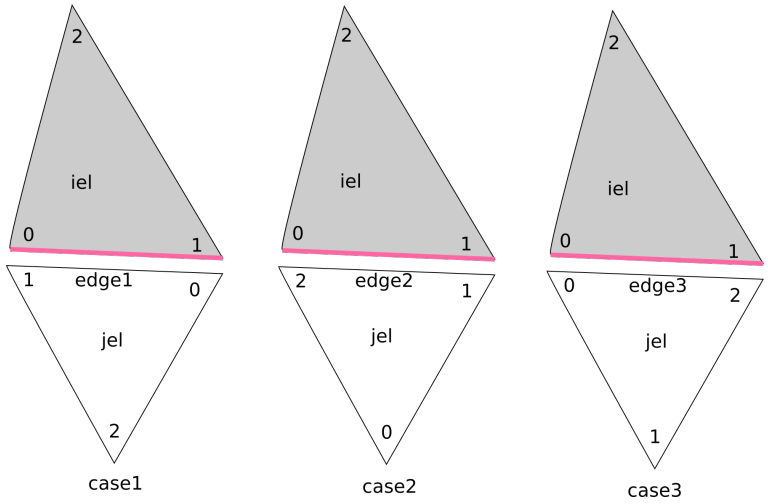
We continue to build the necessary arrays pertaining to neighbours. We build the `edgeX_neighb[iel]` arrays which contain the identity of the element on the other size of edge X of element iel. For instance, looking at the example above of a triangular mesh, we have

```
edge1_neighb[0]=-1
edge2_neighb[0]=1
edge3_neighb[0]=-1
...
edge1_neighb[10]=3
edge2_neighb[10]=11
edge3_neighb[10]=9
```

where the -1 value indicates that there is no element neighbour.

We then build the arrays `edgeX_neighbedge[iel]` (where X stand for 1, 2, 3, or 4) which stores the identity/number of the edge on the element across the common face (if there is a neighbour). For instance:

```
edge1_neighbedge[10]=2
edge2_neighbedge[10]=3
edge3_neighbedge[10]=1
```



When two triangles are neighbours (here along edge 1 of grey triangle) three configurations can occur.

Iterative process Although we are solving the steady-state diffusion equation, we still need to carry out iterations (this is a consequence of the DG formulation itself and it is not the case for 'regular' finite elements). These iterations are implemented as follows:

```
for iter in range(0,niter):
```

where `niter` is set to 100. Vector residuals are computed for T, q_x and q_y. When the infinite-norm of all three is below the tolerance `tol` then the system is considered converged and iterations are stopped. For each iteration the code prints a number of helpful statistics in the terminal:

```
iter=   0 : T,qx,qy (m/M)= 6.944e-03 2.053e+00 | -1.067e+01 2.054e+01 | -1.067e+01 2.054e+01 | max(resT)= 5.700e-01 (tol= 1.0e-09)
iter=   1 : T,qx,qy (m/M)= 9.982e-04 2.053e+00 | -3.991e+00 8.163e+00 | -3.991e+00 8.163e+00 | max(resT)= 7.515e-01 (tol= 1.0e-09)
iter=   2 : T,qx,qy (m/M)= 9.982e-04 2.702e+00 | -2.278e+00 3.863e+00 | -2.278e+00 3.863e+00 | max(resT)= 5.316e-01 (tol= 1.0e-09)
...
iter=  41 : T,qx,qy (m/M)= 3.473e-13 2.000e+00 | 1.000e+00 1.000e+00 | 1.000e+00 1.000e+00 | max(resT)= 1.537e-09 (tol= 1.0e-09)
iter=  42 : T,qx,qy (m/M)= 3.473e-13 2.000e+00 | 1.000e+00 1.000e+00 | 1.000e+00 1.000e+00 | max(resT)= 9.357e-10 (tol= 1.0e-09)
```

Each iteration consists in a sweep through elements and we implement a forward-backward scheme: even-numbered iterations got from 0 to nel-1, while odd-numbered ones go from nel-1 down to 0:

```
if iter%2==0:
    start=0
    end=nel
    step=1
else:
    start=nel-1
    end=-1
    step=-1
```

Explore/implement
source term randomised nodes so no right angles

Experiment #1 - testing signs

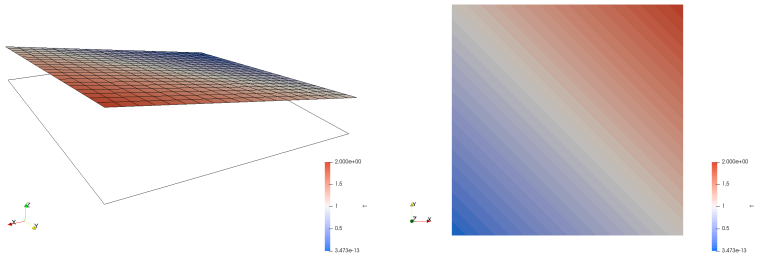
We define the simple temperature field

$$T(x, y) = x + y$$

over a unit square domain, so that (taking $k = 1$) we have

$$q_x(x, y) = -1 \qquad q_y(x, y) = -1$$

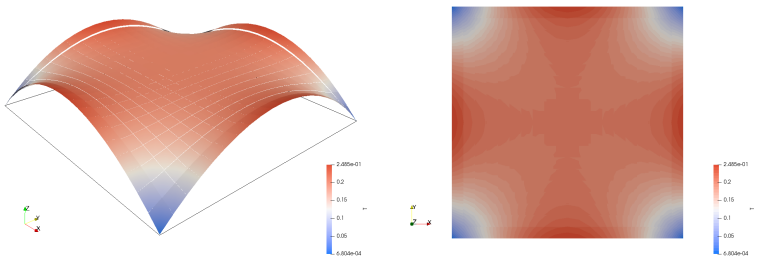
I prescribe this temperature on all four sides of the domain.



VERIFY: minus sign problem for heat fluxes ?

Experiment #2 - testing symmetry

On all four sides of the domain the temperature $T(x, y) = x(L_x - x) + y(L_y - y)$ is prescribed. There is no analytical solution to this problem (?) but it allows us to test that the solution is symmetric as expected.



Stone 80

Fortin's $Q_1^+ \times P_0$ in 2D

```
2D
3D
box
Q1P0
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
vrms
paraview_output
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_80`

We here consider the enriched $Q_1 \times P_0$ element introduced first by Fortin [110] (1981). The shape functions in 2D and 3D are in derived in Section 10.4.6 and Section 10.4.7 respectively. We here only focus on the 2D variant.

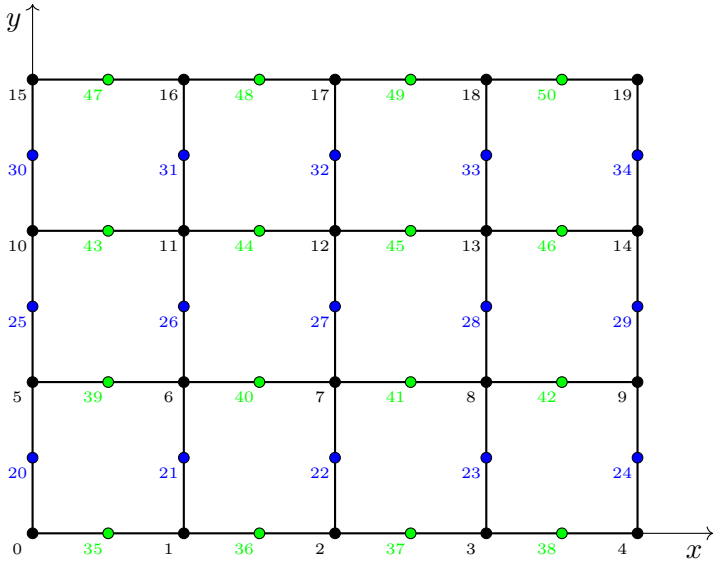
The list of velocity degrees of freedom per element is as follows:

$$\vec{\mathcal{V}} = (u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4, u_5, v_5, u_6, v_6)$$

with the following internal numbering

```
u dofs      v dofs

4-----3     4--6--3
|     |     |     |
5     6     |     |
|     |     |     |
1-----2     1--5--2
```

We start with a simple 4×3 element mesh. Bubble u dofs are represented in blue and bubble v dofs are shown in green.



For this mesh we have

```
nel=nelx*nely (=12)
NP=nel (=12)
NV=nnx*nny+nnx*nely+nny*nelx (=20+16+15=51)
```

The total number of velocity dofs is

```
NfemV=ndofV*nnx*nny + nnx*nely + nny*nelx (=71)
```

while the total number of pressure dofs remains

```
NfemP=NP*ndofP (=nel)
```

What makes this element pair rather awkward to implement is the fact that there are two connectivity arrays `iconu` and `iconv`, both of size $mV \times nel$, where $mV = 6$ is the number of nodes linked to an element for each velocity component. For each element, and depending on whether we are considering the polynomial approximation $u^h(x, y)$ or $v^h(x, y)$ in the element, there are the standard 4 Q_1 nodes and 2 additional dofs, so $mV = 6$.

- content of `iconu`

```
elt
0 | [ 0  1  6  5 20 21]
1 | [ 1  2  7  6 21 22]
2 | [ 2  3  8  7 22 23]
3 | [ 3  4  9  8 23 24]
4 | [ 5  6 11 10 25 26]
5 | [ 6  7 12 11 26 27]
6 | [ 7  8 13 12 27 28]
7 | [ 8  9 14 13 28 29]
8 | [10 11 16 15 30 31]
9 | [11 12 17 16 31 32]
10 | [12 13 18 17 32 33]
11 | [13 14 19 18 33 34]
```

- content of `iconv`

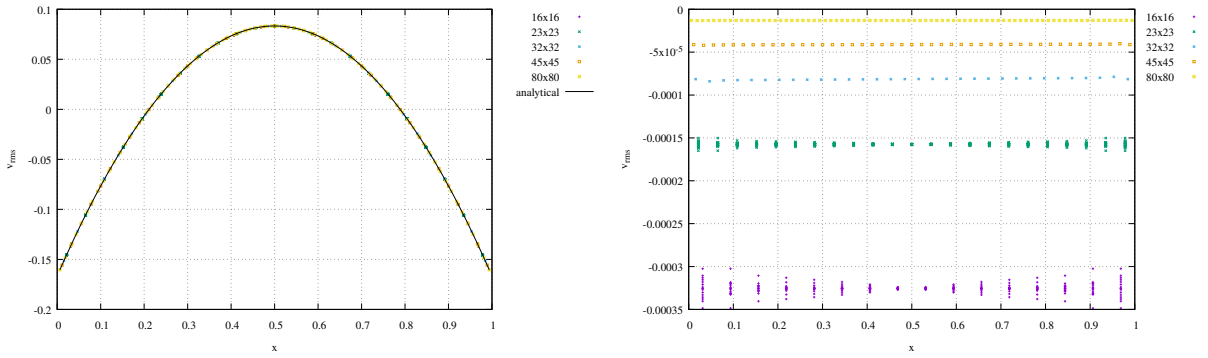
```
elt
 0 | [ 0  1  6  5 35 39]
 1 | [ 1  2  7  6 36 40]
```

```
 2 | [ 2  3  8  7 37 41]
 3 | [ 3  4  9  8 38 42]
 4 | [ 5  6 11 10 39 43]
 5 | [ 6  7 12 11 40 44]
 6 | [ 7  8 13 12 41 45]
 7 | [ 8  9 14 13 42 46]
 8 | [10 11 16 15 43 47]
 9 | [11 12 17 16 44 48]
10 | [12 13 18 17 45 49]
11 | [13 14 19 18 46 50]
```

The pressure field is normalised by imposing $\int_\Omega p dV = 0$ so that there is no nullspace.

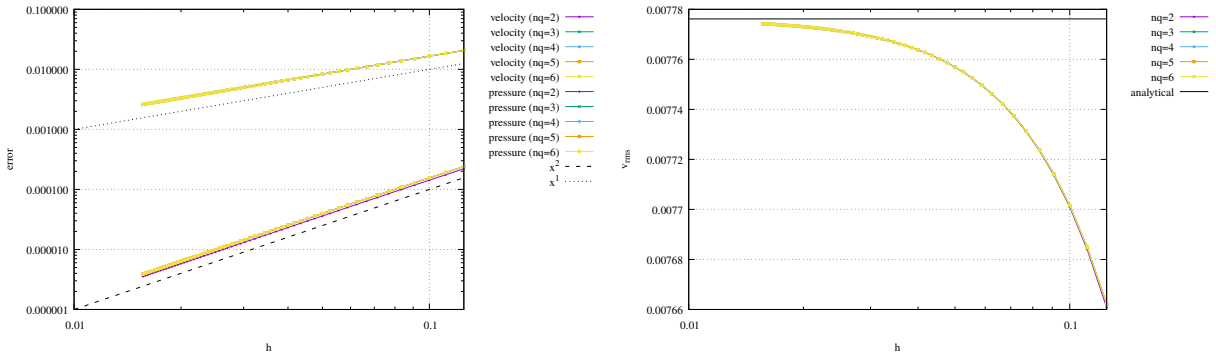
Donea & Huerta manufactured solution benchmark

The Donea & huerta benchmark of Section 18.2 is implemented. It requires no-slip boundary conditions on all sides. This means that $u = 0$ must be prescribed on the lateral boundaries, while $v = 0$ is prescribed at the top and bottom, i.e. boundary conditions are imposed on the bubble nodes too.

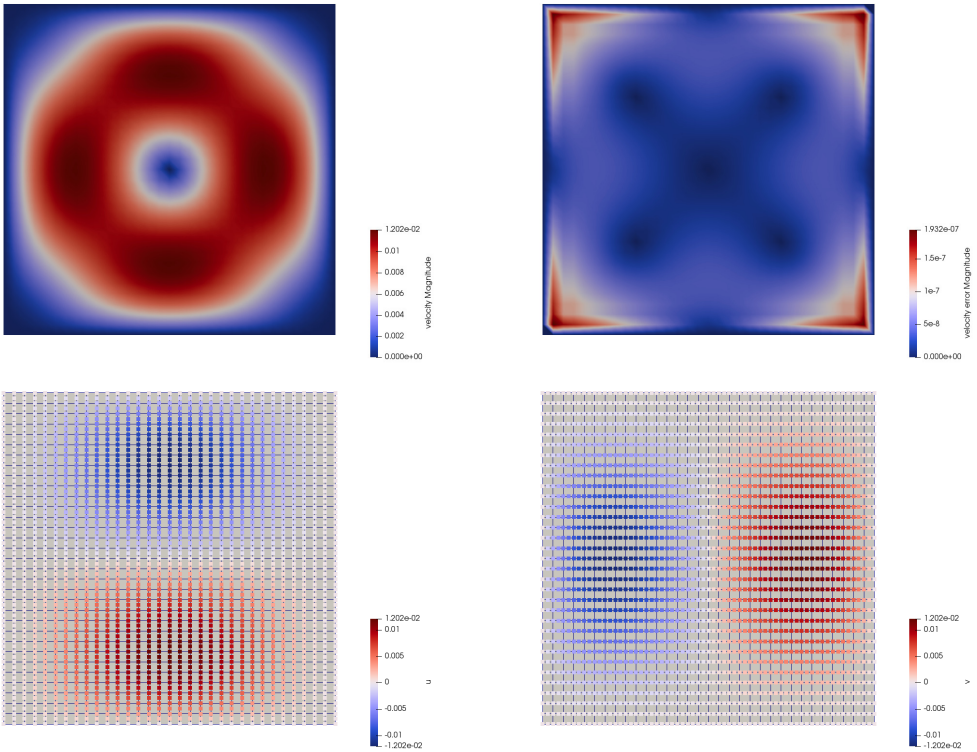


opla

Looking at the velocity and pressure error convergence, we see that the number of quadrature points does not matter much, but surprisingly the lowest number of quad points seems to yield slightly better results...



Left: Velocity and pressure error convergence as a function of resolution for different quadrature rules. Right: root mean square velocity.

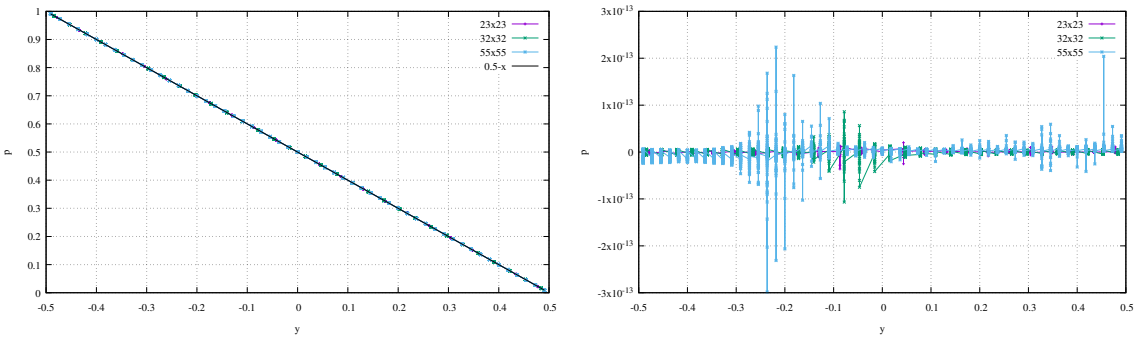


Top row: velocity and velocity error. Bottom row: u and v plotted on their respective meshes.

The aquarium

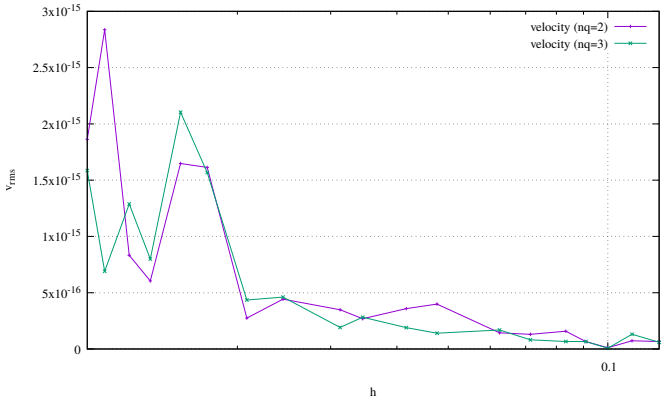
No-slip boundary conditions are prescribed on all sides. Density and viscosity are 1. Gravity is $\vec{g} = -\vec{e}_y$.

We recover a linear pressure profile as expected, that is very accurate:



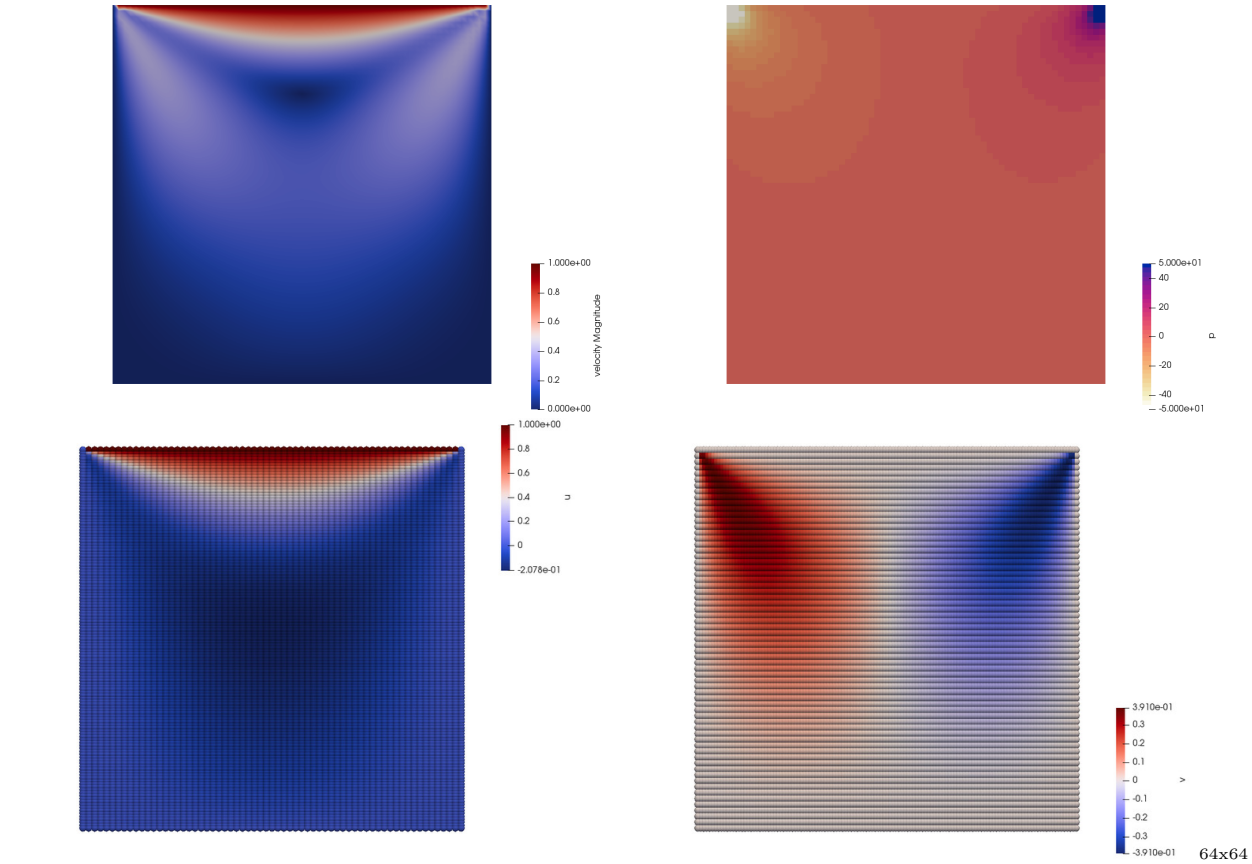
Left: pressure profile; Right: pressure profile error

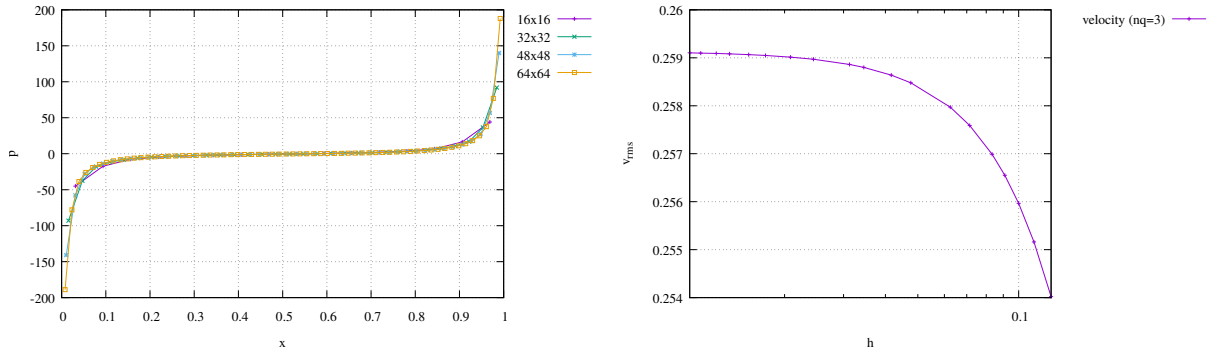
We can also record the root mean square velocity as a function of the resolution h and we find that the velocity is (as expected) zero (down to machine precision):



The lid driven cavity

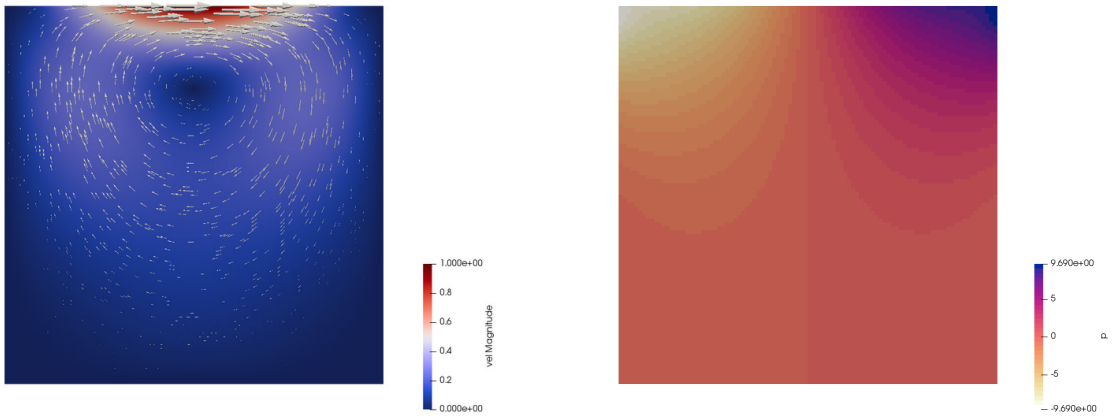
This is the non leaky lid driven cavity, i.e. the velocity at the top left and right corner is set to zero. No slip are imposed on all three other boundaries. The pressure field does not showcase any checkerboard mode.





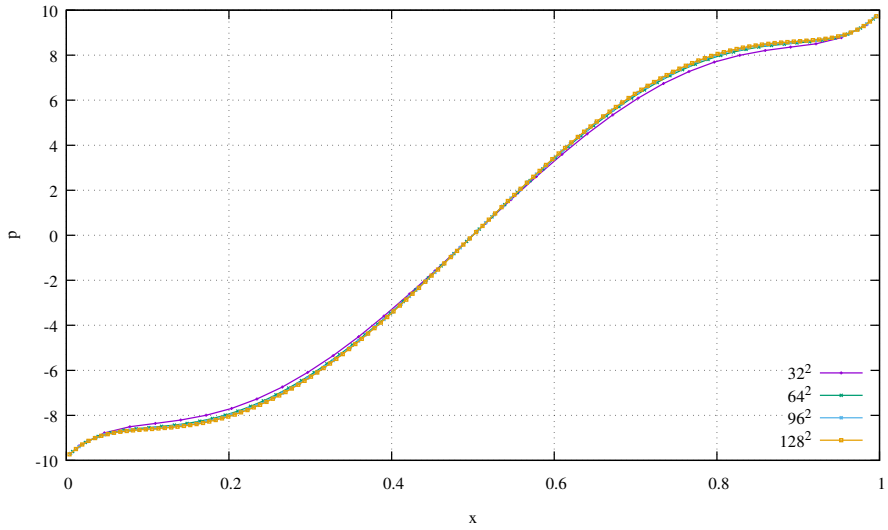
The regularised lid driven cavity

The velocity at the top is $u(x, y) = 16x^2(1 - x)^2$.



Resolution 96x96

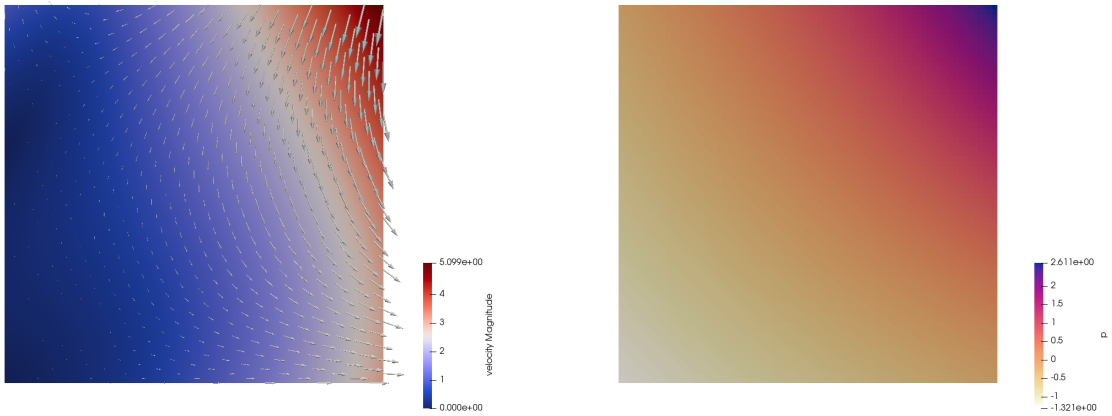
The pressure field converges to a unique solution and no visible checkerboard signal.

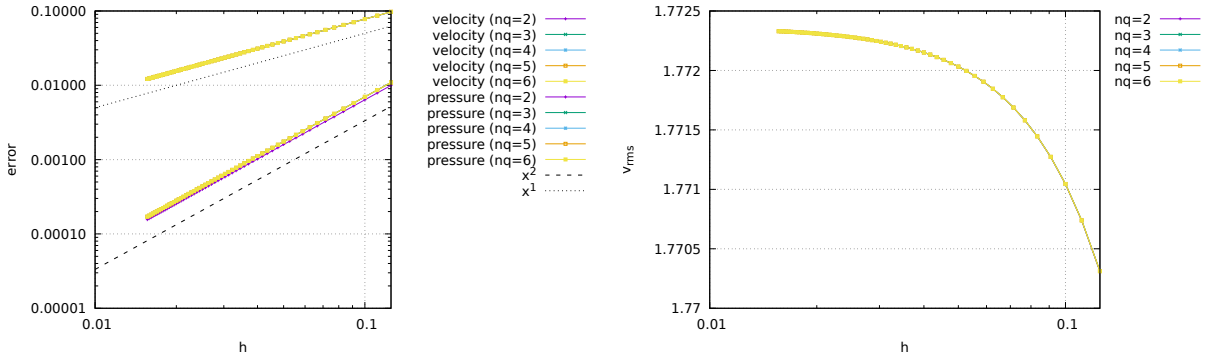


Elemental pressure at the top of the domain.

Dohrmann & Bochev manufactured solution

This benchmark is defined in Section 18.7.

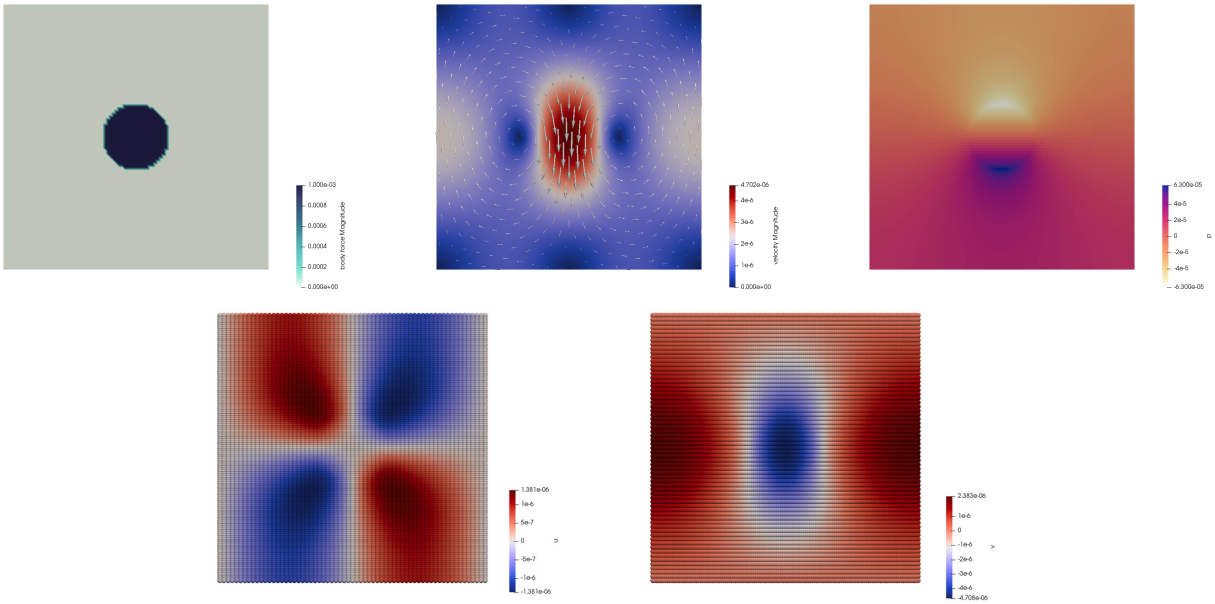




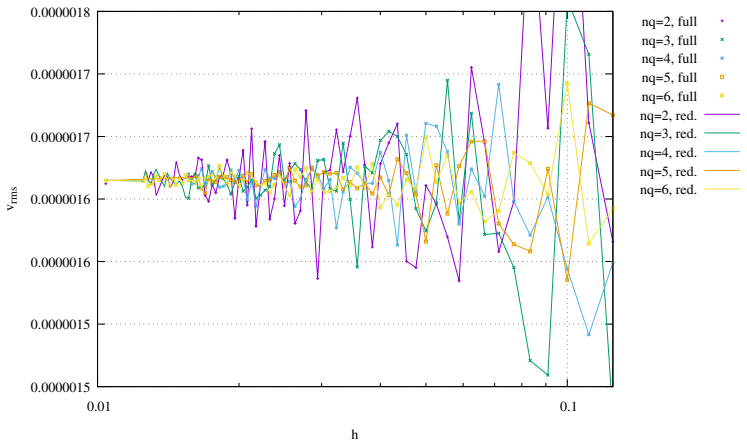
Left: Velocity and pressure error convergence; Right: root mean square velocity as a function of mesh size.

Stokes sphere

Sphere is placed in the middle with radius 0.123, free slip boundary conditions on all sides. Viscosity is constant in the domain.

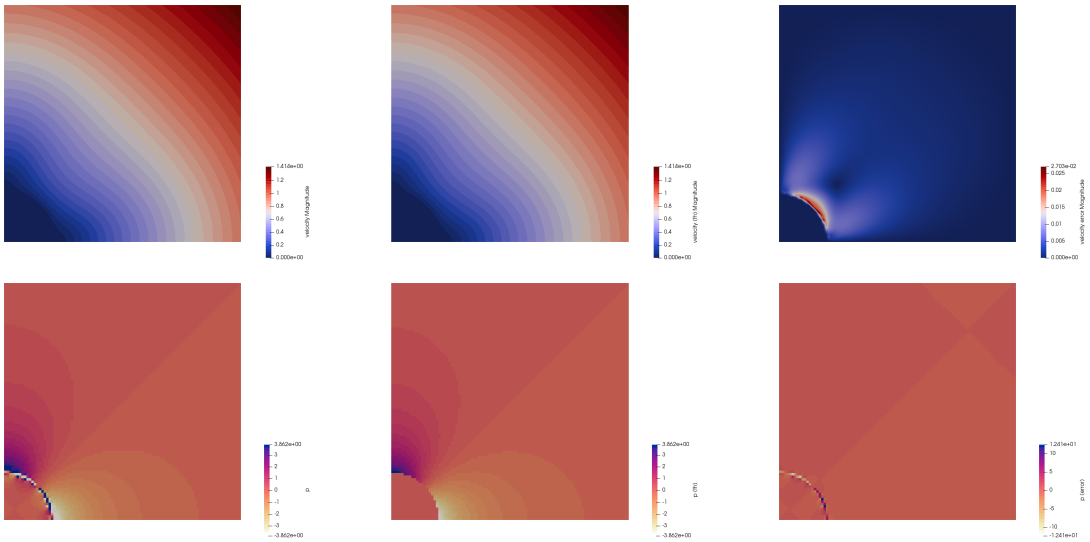


velocity and pressure fields for $\delta\rho = 0.001$. 96x96 elements.

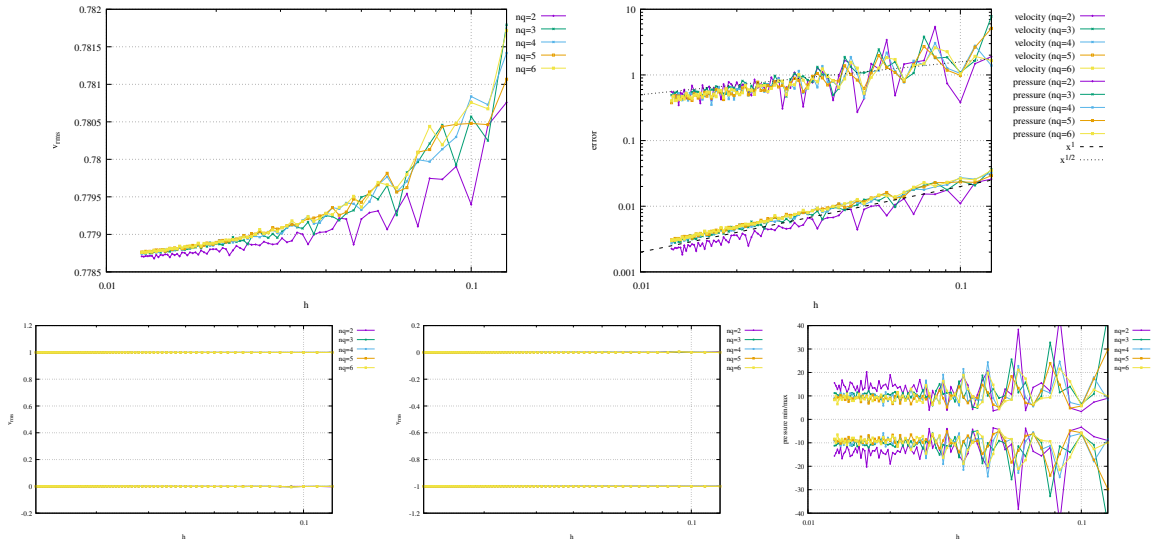


Root mean square velocity for $\delta\rho = 0.001$, for both full density and reduced density models.

SolVi



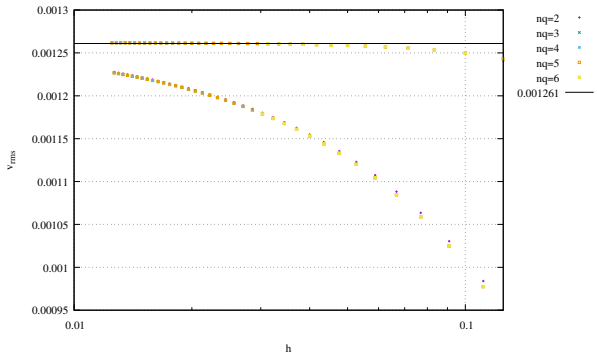
96x96 elements, nqperdim=5. Left to right: computed field, analytical solution, error between both. Top row field is velocity, bottom row field is pressure.

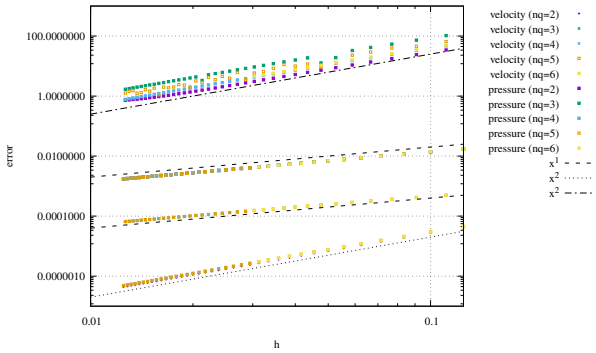


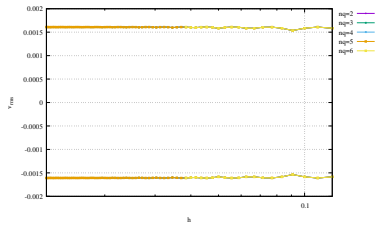
What is surprising is the fact that L2-norm pressure error is best for nq=2 but the min/max values of pressure are best for $nq > 2$.

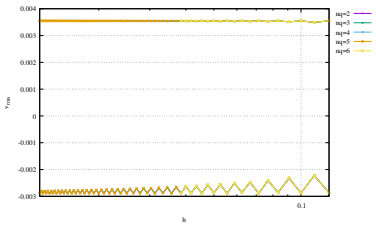
SolCx

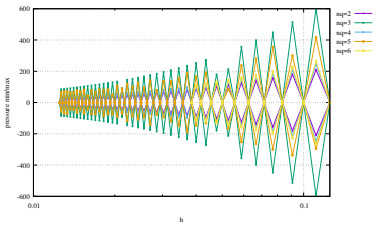
See Stone 5 and Section 19.





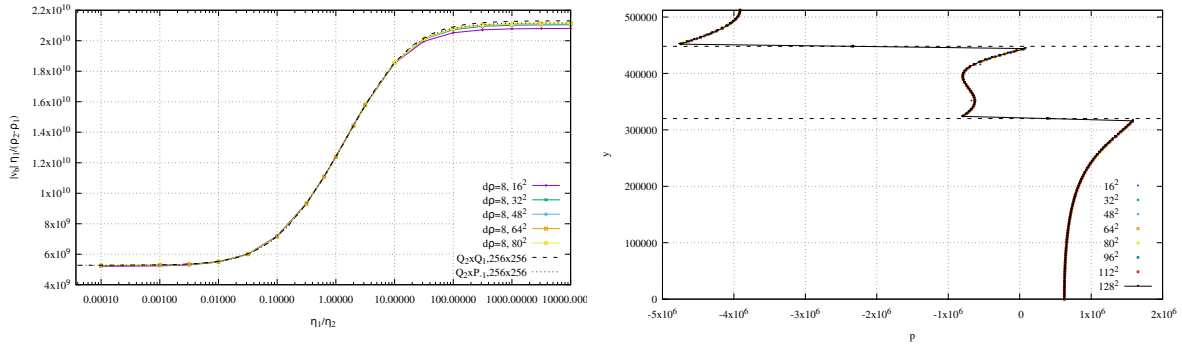




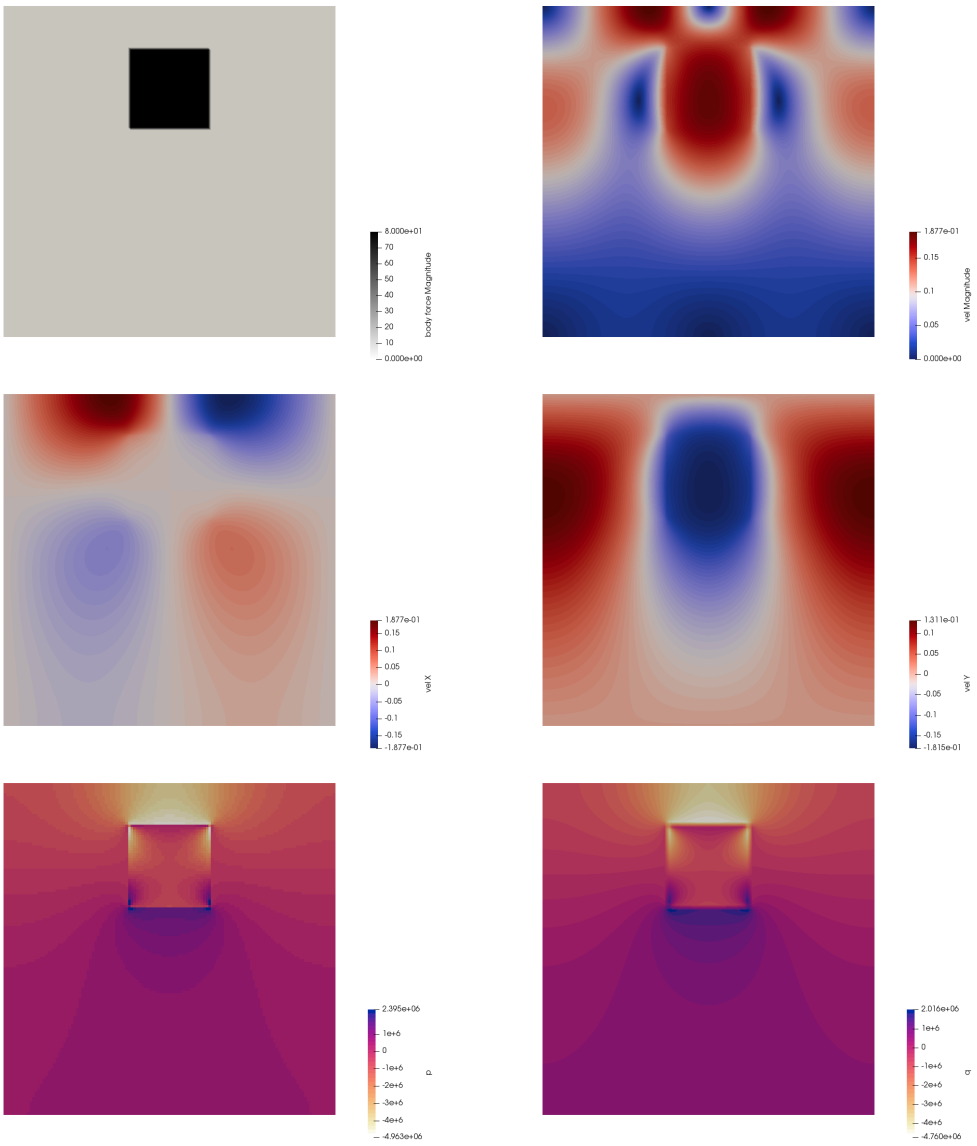


Sinking block

This experiment is fully described in Section 19.8. It is run at resolutions such that the boundaries of the elements are aligned with the sides of the block.



Left: normalised velocity as a function of the viscosity ratio. Right: nodal pressure profile on the $x = L_x/2$ line.



Resolution 128x128

The elemental pressure does not showcase any sign of checkerboard and converges to a unique solution.

Conclusion

This element is very much usable: it is accurate for both buoyancy-driven problems and analytical solutions. However, one last set of tests should be carried out with regards to its implementation and usability when the mesh deforms, or when an annulus geometry is used.

Stone 81

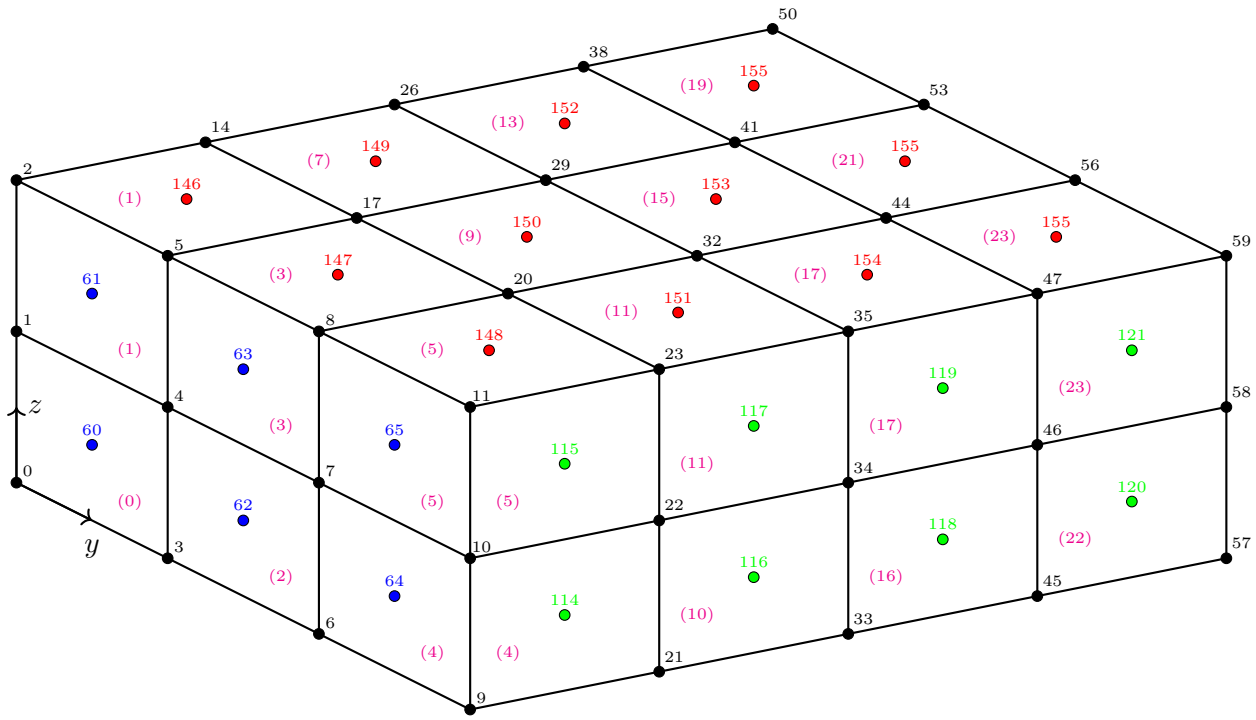
Fortin's $Q_1^+ \times P_0$ in 3D



3D
box
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
vrms
paraview_output
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_81`

This element is presented in Fortin [110] (1981). It is also mentioned in Dhatt and Hubert [89] (1986) and called 'H8N' and proved to be LBB-stable in the appendix.



Example of a simple 4x3x2 mesh. Additional face dofs are indicated in colour.

Each element has

- 8+2 u dofs
- 8+2 v dofs
- 8+2 w dofs

Setting nelx=4, nely=3 and nelz=2, we can print the three connectivity arrays[1]:

- iconu:

```
0 [ 0 12 15  3  1 13 16  4 60 66]
1 [ 1 13 16  4  2 14 17  5 61 67]
2 [ 3 15 18  6  4 16 19  7 62 68]
3 [ 4 16 19  7  5 17 20  8 63 69]
4 [ 6 18 21  9  7 19 22 10 64 70]
5 [ 7 19 22 10  8 20 23 11 65 71]
6 [12 24 27 15 13 25 28 16 66 72]
7 [13 25 28 16 14 26 29 17 67 73]
8 [15 27 30 18 16 28 31 19 68 74]
9 [16 28 31 19 17 29 32 20 69 75]
10 [18 30 33 21 19 31 34 22 70 76]
11 [19 31 34 22 20 32 35 23 71 77]
12 [24 36 39 27 25 37 40 28 72 78]
13 [25 37 40 28 26 38 41 29 73 79]
14 [27 39 42 30 28 40 43 31 74 80]
15 [28 40 43 31 29 41 44 32 75 81]
16 [30 42 45 33 31 43 46 34 76 82]
17 [31 43 46 34 32 44 47 35 77 83]
18 [36 48 51 39 37 49 52 40 78 84]
19 [37 49 52 40 38 50 53 41 79 85]
20 [39 51 54 42 40 52 55 43 80 86]
21 [40 52 55 43 41 53 56 44 81 87]
22 [42 54 57 45 43 55 58 46 82 88]
23 [43 55 58 46 44 56 59 47 83 89]
```

- iconv:

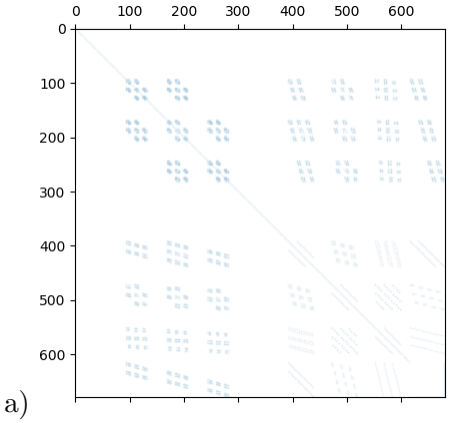
```
0 [ 0 12 15  3  1 13 16  4 90 98]
1 [ 1 13 16  4  2 14 17  5 91 99]
2 [  3  15  18   6   4  16  19   7  98 106]
3 [  4  16  19   7   5  17  20   8  99 107]
4 [  6  18  21   9   7  19  22  10 106 114]
5 [  7  19  22  10   8  20  23  11 107 115]
6 [ 12  24  27  15  13  25  28  16  92 100]
7 [ 13  25  28  16  14  26  29  17  93 101]
8 [ 15  27  30  18  16  28  31  19 100 108]
9 [ 16  28  31  19  17  29  32  20 101 109]
10 [ 18  30  33  21  19  31  34  22 108 116]
11 [ 19  31  34  22  20  32  35  23 109 117]
12 [ 24  36  39  27  25  37  40  28  94 102]
13 [ 25  37  40  28  26  38  41  29  95 103]
```

[1] it is the simplest way I found to account for the additional dofs in the three dimensions: a connectivity array per dimension.

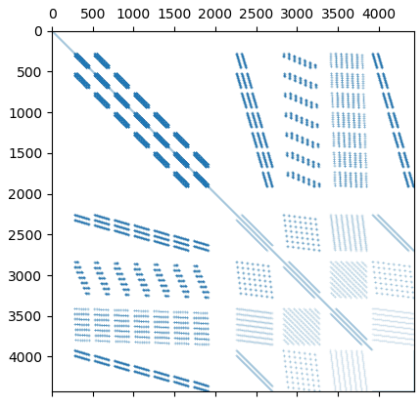
```
14 [ 27  39  42  30  28  40  43  31 102 110]
15 [ 28  40  43  31  29  41  44  32 103 111]
16 [ 30  42  45  33  31  43  46  34 110 118]
17 [ 31  43  46  34  32  44  47  35 111 119]
18 [ 36  48  51  39  37  49  52  40  96 104]
19 [ 37  49  52  40  38  50  53  41  97 105]
20 [ 39  51  54  42  40  52  55  43 104 112]
21 [ 40  52  55  43  41  53  56  44 105 113]
22 [ 42  54  57  45  43  55  58  46 112 120]
23 [ 43  55  58  46  44  56  59  47 113 121]
```

- iconw:

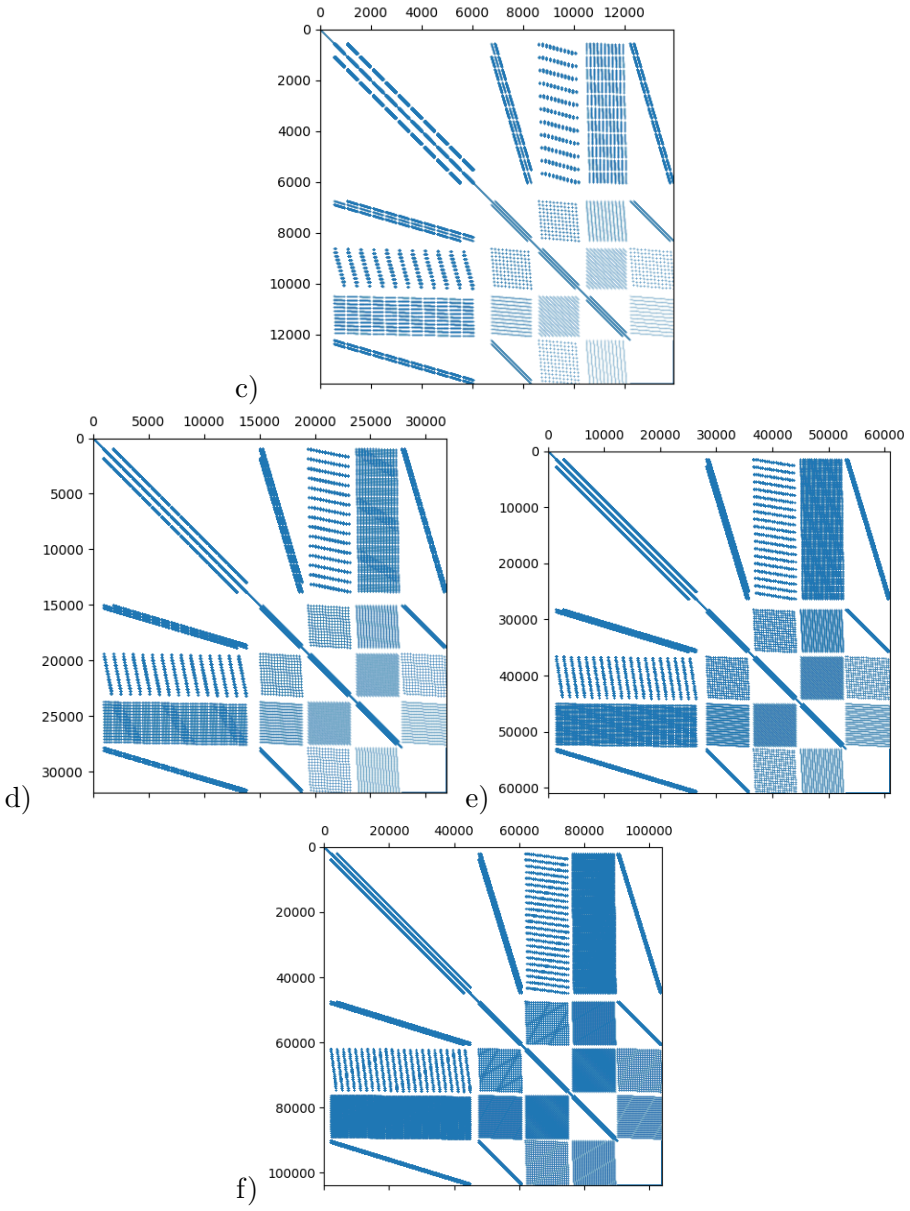
```
0 [  0  12  15   3   1  13  16   4 122 134]
1 [  1  13  16   4   2  14  17   5 134 146]
2 [  3  15  18   6   4  16  19   7 123 135]
3 [  4  16  19   7   5  17  20   8 135 147]
4 [  6  18  21   9   7  19  22  10 124 136]
5 [  7  19  22  10   8  20  23  11 136 148]
6 [ 12  24  27  15  13  25  28  16 125 137]
7 [ 13  25  28  16  14  26  29  17 137 149]
8 [ 15  27  30  18  16  28  31  19 126 138]
9 [ 16  28  31  19  17  29  32  20 138 150]
10 [ 18  30  33  21  19  31  34  22 127 139]
11 [ 19  31  34  22  20  32  35  23 139 151]
12 [ 24  36  39  27  25  37  40  28 128 140]
13 [ 25  37  40  28  26  38  41  29 140 152]
14 [ 27  39  42  30  28  40  43  31 129 141]
15 [ 28  40  43  31  29  41  44  32 141 153]
16 [ 30  42  45  33  31  43  46  34 130 142]
17 [ 31  43  46  34  32  44  47  35 142 154]
18 [ 36  48  51  39  37  49  52  40 131 143]
19 [ 37  49  52  40  38  50  53  41 143 155]
20 [ 39  51  54  42  40  52  55  43 132 144]
21 [ 40  52  55  43  41  53  56  44 144 156]
22 [ 42  54  57  45  43  55  58  46 133 145]
23 [ 43  55  58  46  44  56  59  47 145 157]
```



a)



b)



non-zero pattern of the assembled Stokes matrix for 8^3 (a), 12^3 (b), 16^3 (c) and 20^3 (d) meshes.

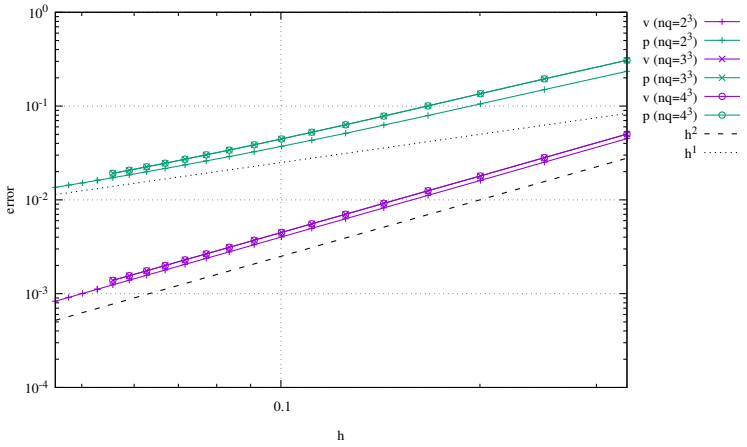
This is bad news: a renumbering of the nodes/dofs would be beneficial so as to reduce the skyline of the matrix.

Manufactured solution (`bench=1`)

This benchmark is presented in Section 18.9 and is called the 'Burstedde' benchmark in the ASPECT manual after the paper by Burstedde et al. [57] (2013), although it originates in Dohrmann and Bochev [93] (2004). It was run with 3 different quadrature levels, $nq = 2^3, 3^3, 4^3$. Unfortunately a resolution $24 \times 24 \times 24$ is the maximum which can be run as demonstrated by the following table:

resolution	Nfem	build (s)	solve(s)
12x12x12	12207+1728	21	5.4
16x16x16	27795+4096	49	40
20x20x20	52983+8000	98.3	414
21x21x21	61050+9261	130	753
22x22x22	69897+10648	144	1213
24x24x24	90075+13824	173	2481

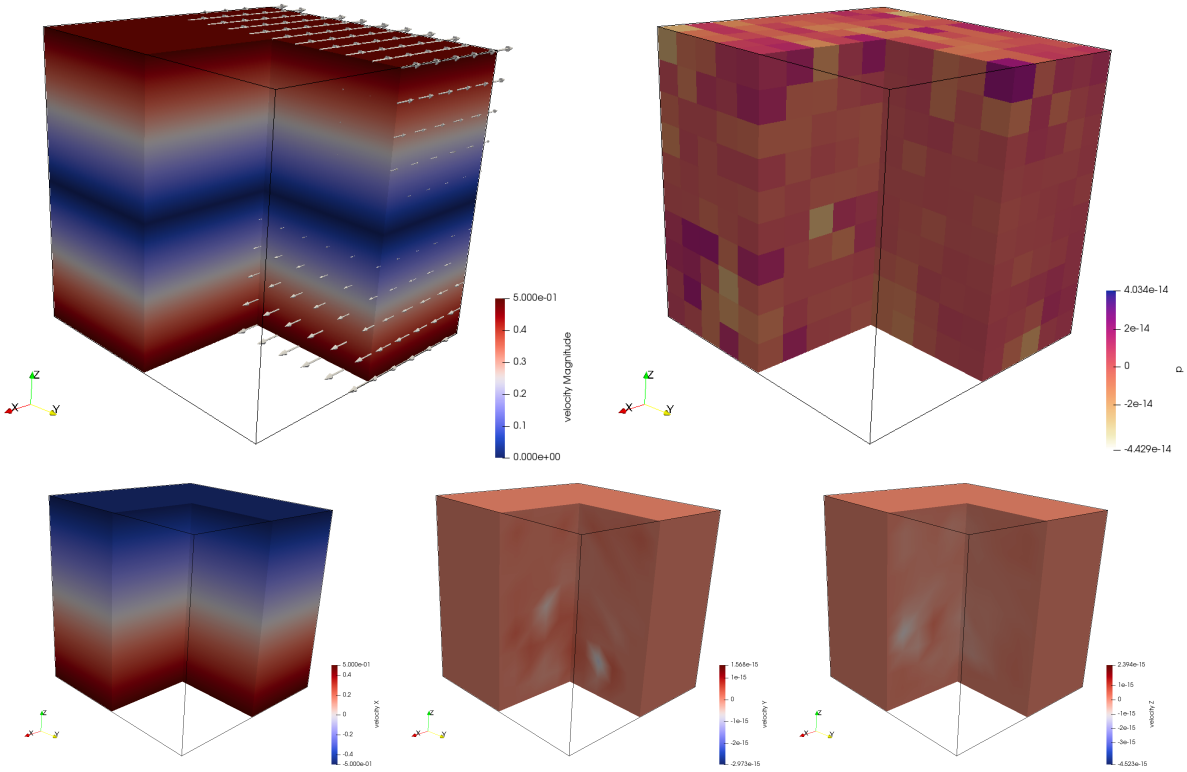
We recover a quadratic convergence for the velocity error and a linear convergence for the pressure error, similarly to the $Q_1 \times P_0$ element:



It appears that a $2 \times 2 \times 2$ is sufficient and yields the best results.

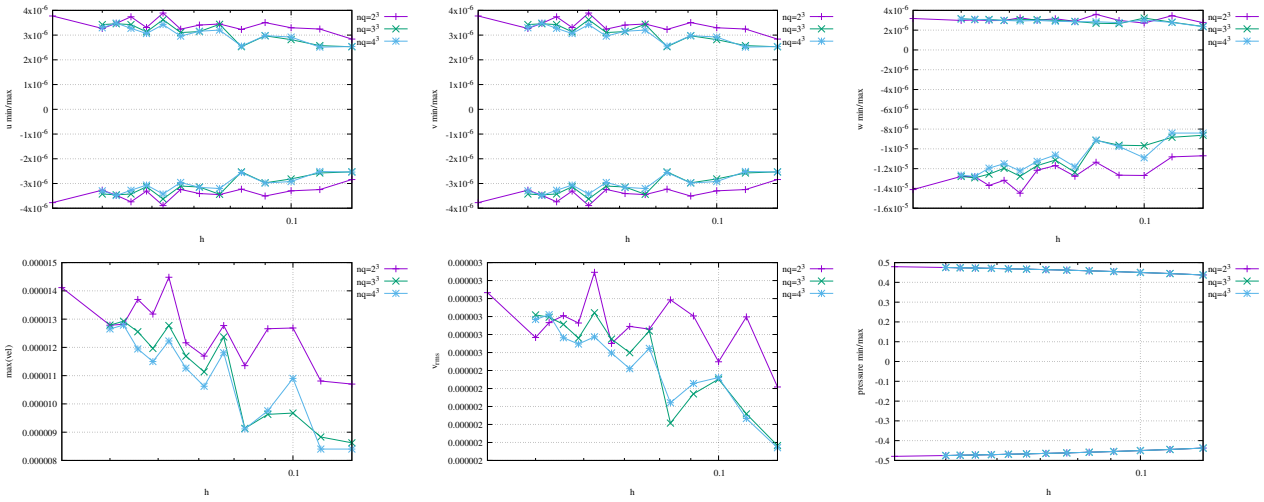
Horizontal shear (`bench=2`)

Gravity is set to zero so that density values are irrelevant. Viscosity is set to 1. Velocity boundary conditions $\frac{1}{2} - z, 0, 0$ are prescribed at the top and bottom, while the side boundaries are left free. We expect this same velocity field everywhere in the domain and given the pressure normalisation a zero pressure in the domain.



Stokes sphere benchmark (`bench=3`)

It is described in Section 19.23. We here focus on the 'NS' case, i.e. no slip boundary conditions are prescribed on all six faces. Unfortunately the resolution is so limited that it is hard to really see a convergence in the results.



Remark: I am using mV everywhere in the code. Technically it should be mU, mV, mW.

Stone 82

The 3D MINI element with 2 bubble functions ($Q_1^{++} \times Q_1$)



python_codes/fieldstone_82/text.tex

```
3D
box
analytical_benchmark
stokes_equation
paraview_output
pressure_normalisation
```

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_82`

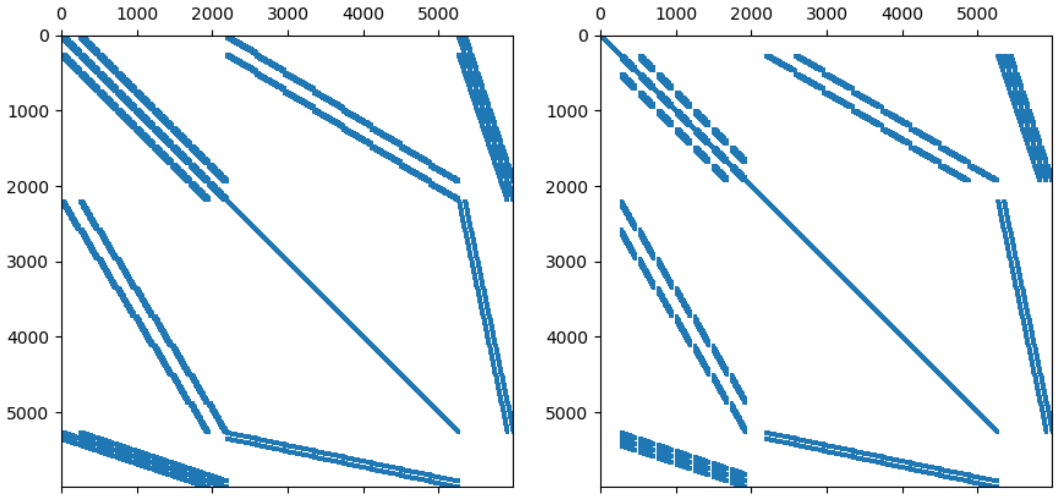
Purpose: implement and document the $Q_1^{++} \times Q_1$ element.

The basis functions for this element is presented in Section 10.4.17. This element is presented in Karabelas, Haase, Plank, and Augustin [176] (2020).

There are two bubble functions/nodes per element so we then have

```
NV=nnx*nny*nnz+2*nel
NP=nnx*nny*nnz
```

For a $8 \times 8 \times 8 = 512$ element grid, there are $9^3 = 729$ nodes, and 1024 bubble nodes. We then have $3 \cdot 729 = 2157$ Q_1 velocity dofs and 3072 velocity dofs attached to the bubbles. Since the bubble nodes are stored last the structure of the matrix is as follows:



Sparsity pattern of the matrix for a 8x8x8 grid. NV= 1753, NP=729.

Left: before boundary conditions are applied. Right: after boundary conditions are applied.

Manufactured solution (bench=1)

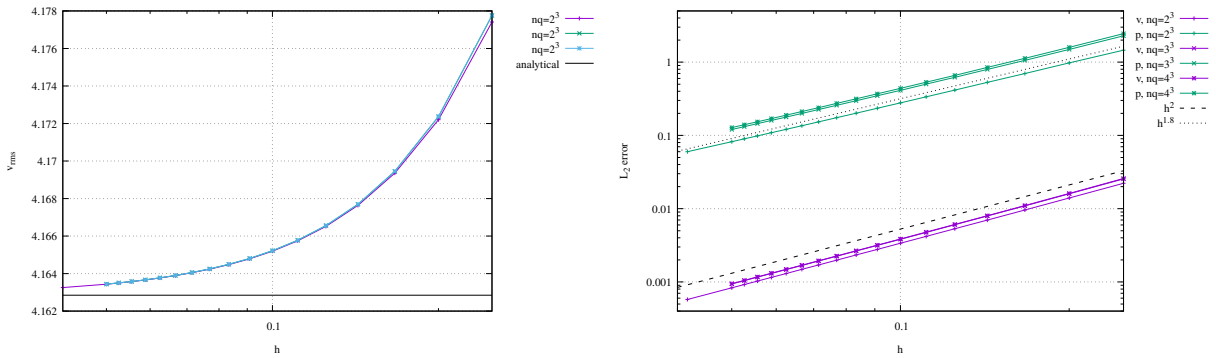
This benchmark begins by postulating a polynomial solution to the 3D Stokes equation (see Dohrmann & Bochev (2004) [93]):

$$\vec{\mathbf{v}}(x,y,z) = \begin{pmatrix} x + x^2 + xy + x^3 y \\ y + xy + y^2 + x^2 y^2 \\ -2z - 3xz - 3yz - 5x^2 yz \end{pmatrix} \tag{82.1}$$

and

$$p(x,y,z) = xyz + x^3 y^3 z - 5/32 \tag{82.2}$$

The corresponding right-hand side is computed in Section 18.9. The domain is a unit cube and velocity boundary conditions simply use Eq. (82.1). Note that the pressure fulfills $\int_\Omega p(x,y,z)dV = 0$.

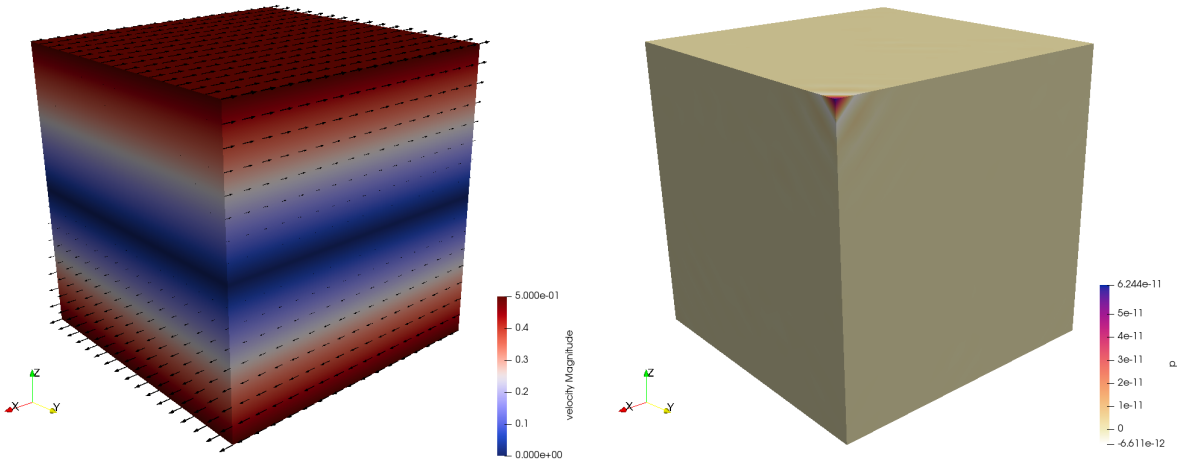


Results obtained for three levels of quadrature, with $\beta = 0$ (i.e. viscosity is constant and equal to 1).

Somewhat surprisingly, the pressure convergence is not exactly quadratic but seems to be around 1.8. Also, the $2 \times 2 \times 2$ quadrature yields better results than the $3 \times 3 \times 3$ or $4 \times 4 \times 4$.

Horizontal shear (bench=2)

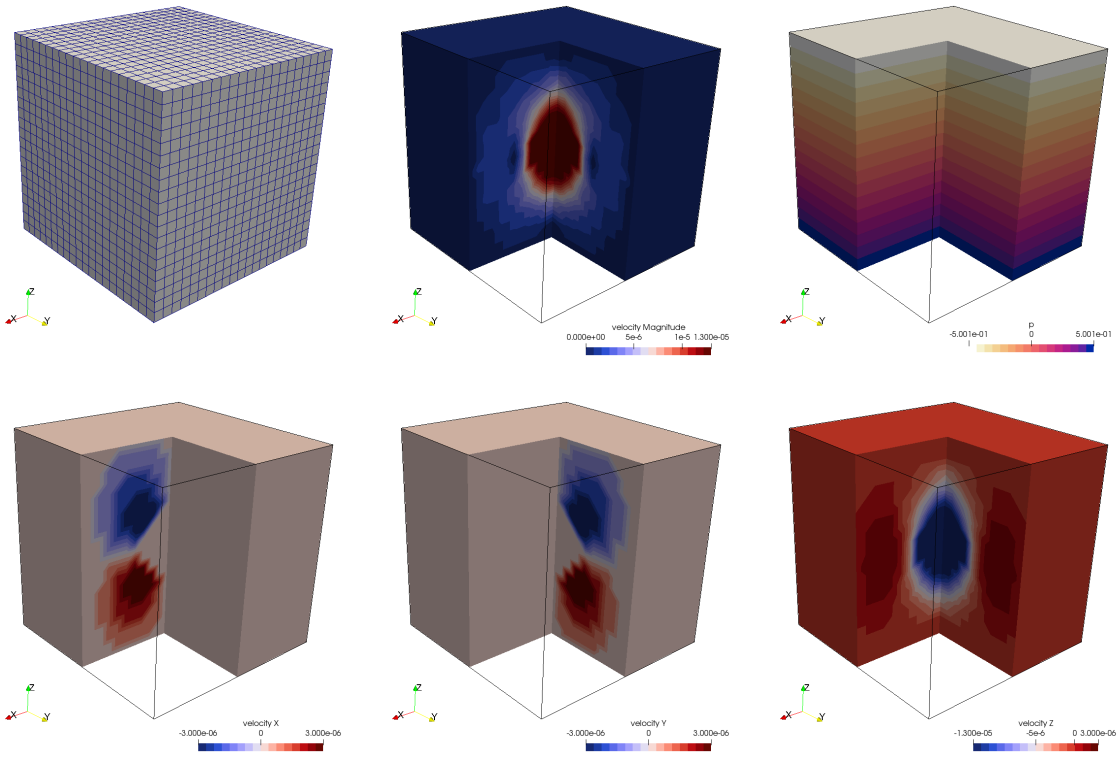
This is a simple kinematical test: gravity is set to zero, no boundary conditions on the sides and $\vec{\mathbf{v}} = (-1, 0, 0)$ prescribed on the top, and $\vec{\mathbf{v}} = (+1, 0, 0)$ prescribed on the bottom.



$20 \times 20 \times 20$ mesh.

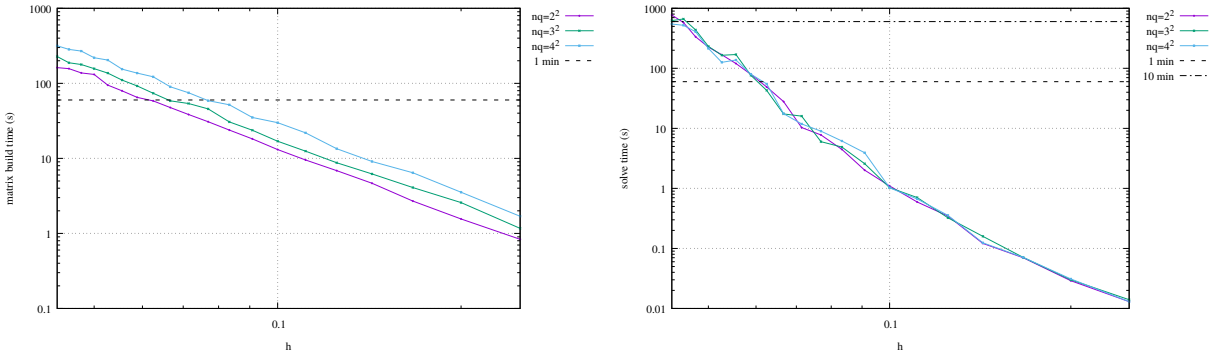
Stokes sphere (bench=3)

This is the benchmark of Section 19.23. A sphere of radius 0.123456789 is placed in the middle of the domain. It has a density excess $\delta\rho = 0.01$ with respect to the surrounding fluid which has a density $\rho_{fluid} = 1$. The sphere viscosity is 1000 while the fluid viscosity is 1. Boundary conditions are no slip on all sides. Gravity is $\vec{g} = -\vec{e}_z$.



Resolution $20 \times 20 \times 20$. Quadrature $2 \times 2 \times 2$.

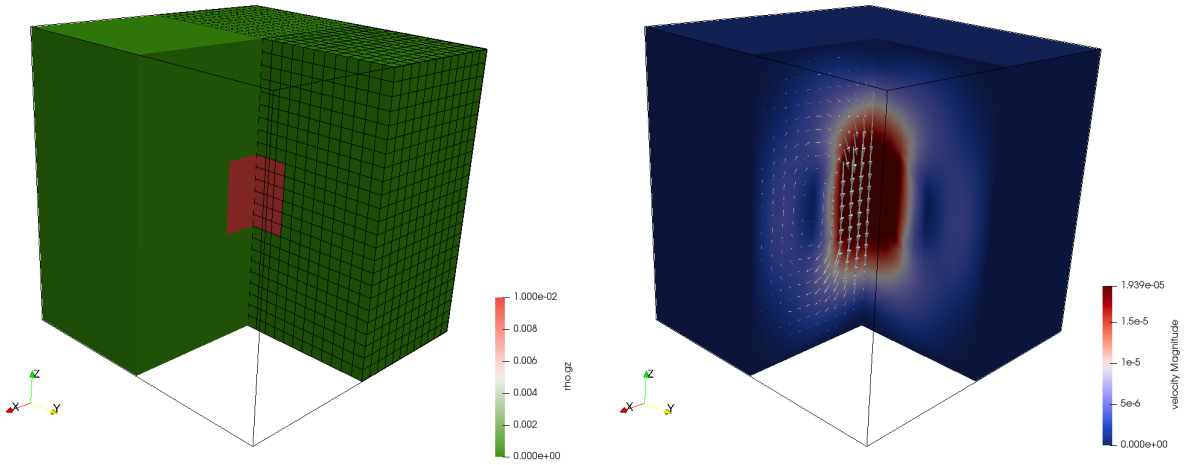
All velocity and pressure statistics are presented in Section 19.23.



Matrix build and solve time as a function of h.

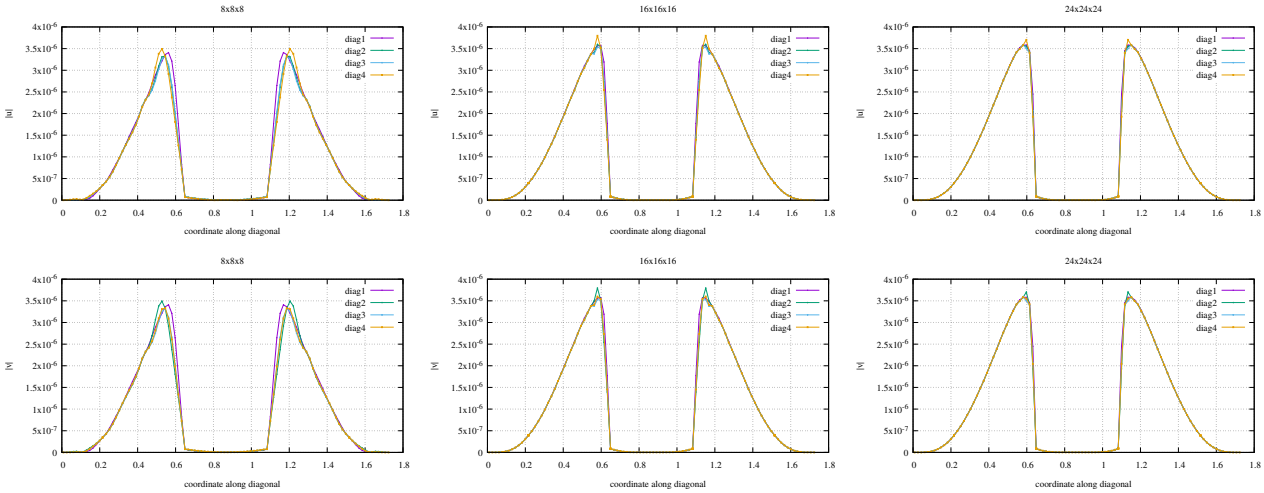
Sinking cube (bench=4)

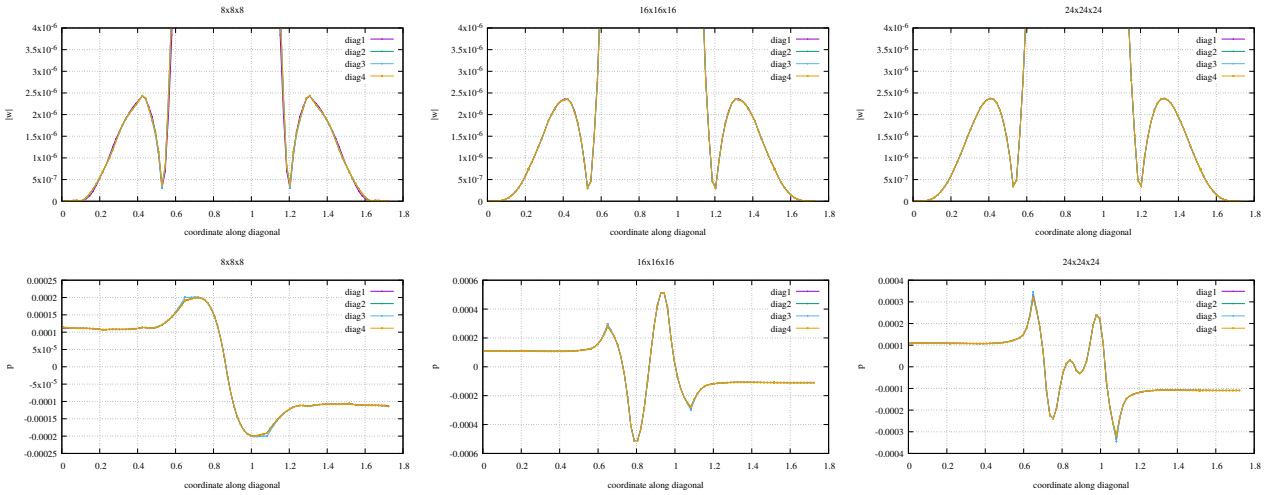
This is the same experiment as the Stokes sphere with the exception of the geometry of the sinker which is $0.25 \times 0.25 \times 0.25$ in size so that its edges align with boundary edges for $8 \times 8 \times 8$, $16 \times 16 \times 16$ and $24 \times 24 \times 24$ meshes. Reduced density is used in order to remove the lithostatic pressure signal.



resolution $24 \times 24 \times 24$ elements.

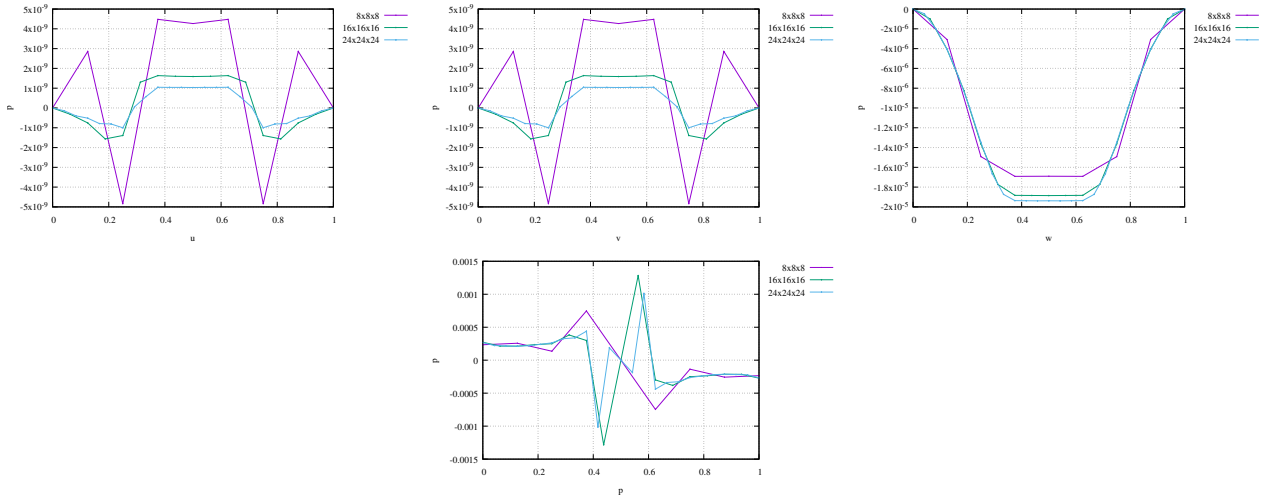
The velocity and pressures are interpolated on 100 points on the four diagonals of the cube as a way to verify the symmetry of the solution (the bubbles rely on the basis functions of nodes 0 and 6 afterall!). 2^3 quadrature points per element are used.





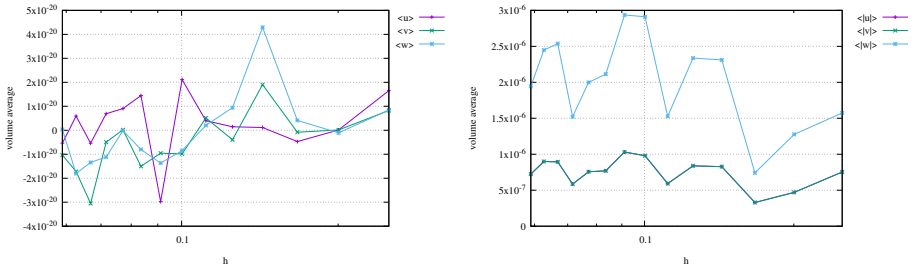
Left to right: increasing resolution

We can also plot the velocity and pressure fields on a vertical line passing through the middle of the block:



We see that unfortunately the resolution $24 \times 24 \times 24$ is most certainly not high enough to speak of capturing the true solution of the problem...

Also, one can verify that the solution is symmetric by looking at the averages of the velocity components (and their absolute value) as a function of the mesh size:



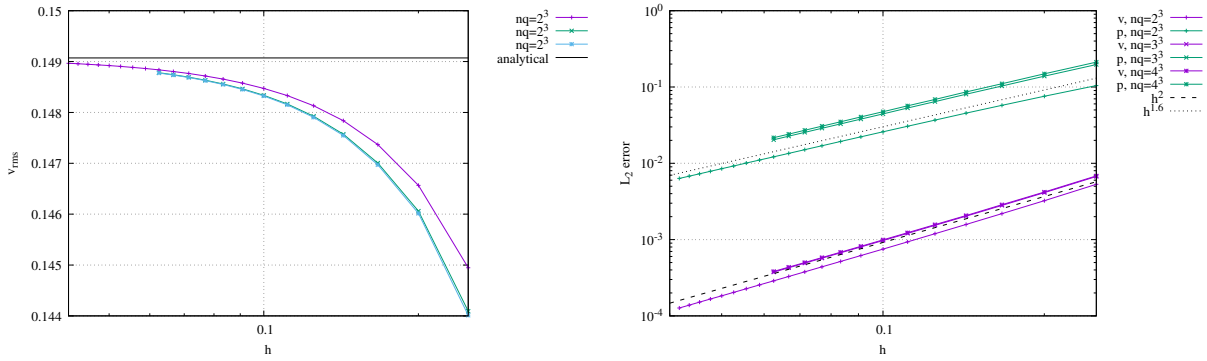
It seems that the somewhat anisotropic placement of the bubbles does not introduce a form of asymmetry in the solution.

Manufactured solution (bench=5)

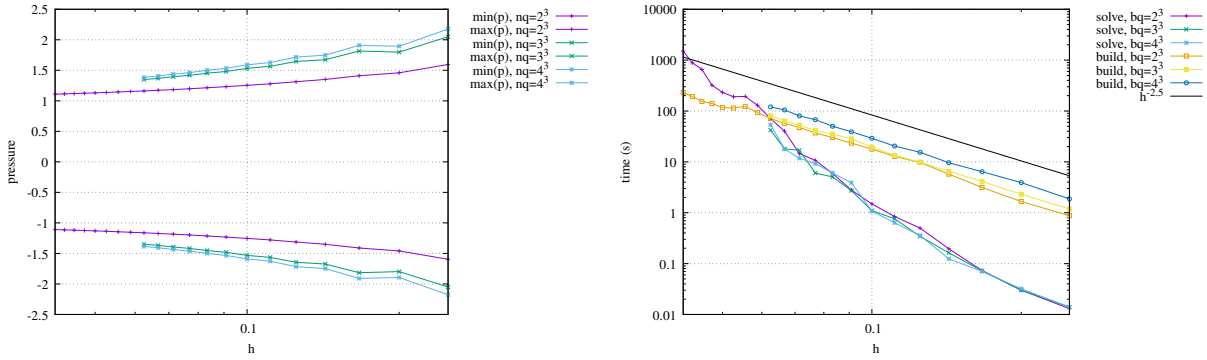
We here consider the manufactured solution which is introduced in Section 18.30. The velocity and pressure fields are:

$$u(x,y,z) = x(1-x)(1-2y)(1-2z) \tag{82.3}$$
$$v(x,y,z) = (1-2x)y(1-y)(1-2z) \tag{82.4}$$
$$w(x,y,z) = -2(1-2x)(1-2y)z(1-z) \tag{82.5}$$
$$p(x,y,z) = (2x-1)(2y-1)(2z-1) \tag{82.6}$$

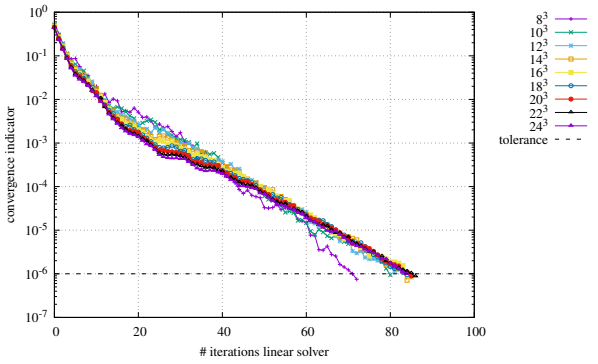
This flow field has the built-in property that there is no flux through the boundaries.

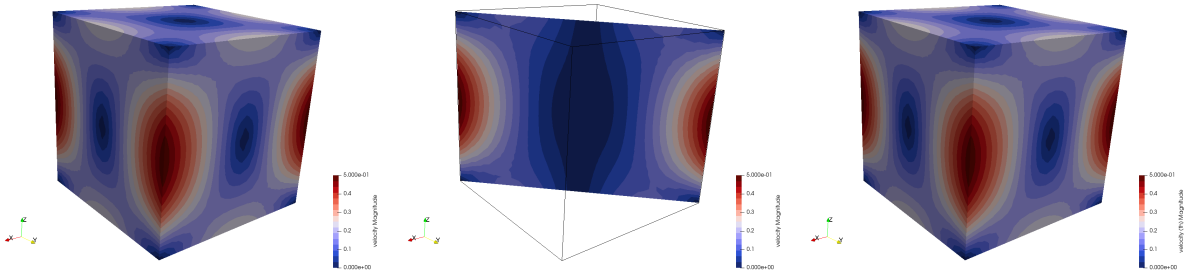


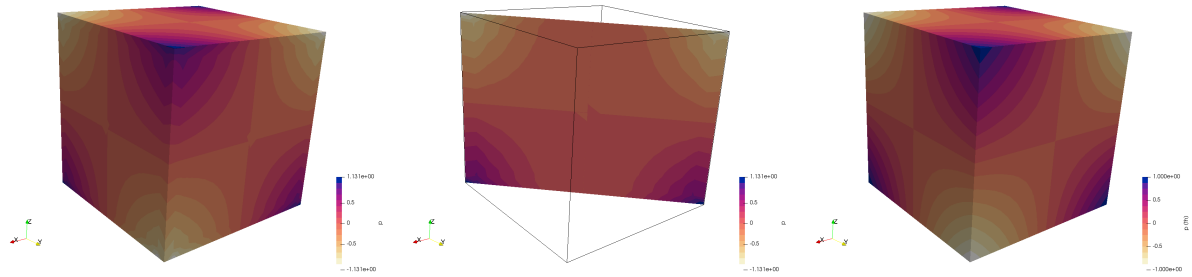
Results obtained for three levels of quadrature.



When using the Preconditioned Conjugate Gradients algorithm on the Schur complement we find that the number of iterations is rather constant across the range of resolutions that can be used:







bench=-1: computing Ker($\mathbb{G}$) on a $2 \times 2 \times 2$ mesh

We consider a mesh composed of $2 \times 2 \times 2$ elements. We prescribe no slip boundary conditions on all sides, so that only the node in the middle has 3 velocity dofs left, in addition to the 8*2=16 bubbles counting in total 48 velocity dofs, so 51 vdofs are left.

The matrix $\mathbb{G}$ is [(3*3*3)+(8*2)]*3=43*3=129 vdofs X (3*3*3=27) pdofs. Imposing the boundary conditions will zero 26*3=78 lines so that the leftover matrix $\mathbb{G}$ has size 51×27. If this element is stable its nullspace should be of size 1.

Given the ordering of the nodes, the middle node has global number 13, so that its vdofs are 39, 40, and 41. The bubbles occupy rows 81 to 128. We create a matrix $\mathbb{G}_2$ which contains only the lines of $\mathbb{G}$ which contain non-zero terms:

```
G2 = np.zeros((51,NfemP),dtype=np.float64)
G2[0:3,:]=G_mat[39:42,:]
G2[3:,:]=G_mat[81:,:]
```

We then proceed to find the kernel/nullspace of matrix $\mathbb{G}_2$:

```
ns = null_space(G2)
```

and we find that its size is just 1, as expected.

Stone 83

Thermal structure of oceanic and continental lithosphere (Q_1, Q_2)



2D
box
Q1
Q2
analytical_benchmark
numerical_benchmark
energy_equation
time_stepping
nonlinear
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_83`

This stone was developed in collaboration with Iris van Zelst.

To accurately study the thermal structure of the oceanic lithosphere, it is important to take into account the temperature dependence of thermal parameters such as the thermal conductivity k, heat capacity C_p, and the density ρ. Classic (analytical) models, such as the half-space cooling model and the plate model, typically ignore this complexity and assume constant parameters.

Here, we model the thermal structure of oceanic lithosphere with temperature-dependent thermal conductivity, heat capacity, and density according to McKenzie, Jackson, and Priestley [232] and Richards, Hoggard, Cowton, and White [262].

We use a 2D approach[1], although there is no lateral variation in the x-direction, which essentially makes this a 1D model of cooling oceanic lithosphere with time. We implement various options for the temperature-dependence of the parameters (including the constant case) based on experimental findings and different rock compositions Parsons and Sclater [248] (model=0), Berman 1988, Berman and Aranovich 1996, Hofmeister 1999, Xu et al., 2004. We then compare our results to those of Richards, Hoggard, Cowton, and White [262].

We start from Eq. (1) of McKenzie, Jackson, and Priestley [232] (2005):

$$\frac{\partial}{\partial t}\left(\rho(T)C_p(T)T\right) = \vec{\nabla}\cdot\left(k(T)\vec{\nabla}T\right)$$

[1] This allows us to reuse and adapt an existing 2D code

Using a simple Euler scheme we have

$$\frac{(\rho(T)C_p(T)T)^n - (\rho(T)C_p(T)T)^{n-1}}{\delta t} = \vec{\nabla}\cdot(k(T^n)\vec{\nabla}T^n)$$

or, simply reorganising the terms:

$$\rho(T^n)C_p(T^n)T^n - \delta t\vec{\nabla}\cdot(k(T^n)\vec{\nabla}T^n) = \rho(T^{n-1})C_p(T^{n-1})T^{n-1}$$

However, since we wish to compute $\vec{T}^n$ then $\rho(T^n)$, $C_p(T^n)$ and $k(T^n)$ are not available to us so we resort to using the ones from the previous timestep[2], or

$$\rho(T^{n-1})C_p(T^{n-1})T^n - \delta t\vec{\nabla}\cdot(k(T^{n-1})\vec{\nabla}T^n) = \rho(T^{n-1})C_p(T^{n-1})T^{n-1}$$

We left multiply by a shape function $\mathcal{N}_i$ and integrate over the whole domain Ω:

$$\int_\Omega \mathcal{N}_i\rho(T^{n-1})C_p(T^{n-1})T^n dV - \delta t\int_\Omega \mathcal{N}_i\vec{\nabla}\cdot(k(T^{n-1})\vec{\nabla}T^n)dV = \int_\Omega \mathcal{N}_i\rho(T^{n-1})C_p(T^{n-1})T^{n-1}dV$$

Using the basis functions, we can write

$$T^h(x,y) = \vec{\mathcal{N}}(x,y)\cdot\vec{T}$$

where $\vec{T}$ is the vector of nodal temperatures, and this ultimately yields[3]

$$\mathbb{M}^n\cdot\vec{T}^n + \delta t\mathbb{K}^n\cdot\vec{T}^n = \vec{f}^{n-1}$$

with

$$\begin{aligned}
\mathbb{M}^n &= \int_V \rho(T^{n-1})C_p(T^{n-1})\vec{\mathcal{N}}^T\vec{\mathcal{N}}dV \\
\mathbb{K}^n &= \int_V \boldsymbol{B}^T k(T^{n-1})\boldsymbol{B}dV \\
\vec{f}^{n-1} &= \int_V \vec{\mathcal{N}}^T\rho(T^{n-1})C_p(T^{n-1})dV
\end{aligned}$$

The second term has been integrated by parts (and the surface term disregarded: Dirichlet boundary conditions top and bottom, and zero heat flux on the sides).

The density is given by Eq. (9) of McKenzie, Jackson, and Priestley [232]:

$$\rho(T) = \rho_0\exp-\left[\alpha_0(T-273) + \frac{\alpha_1}{2}(T^2-273^2)\right] \tag{83.1}$$

with $\rho_0 = 3330\text{kg}\,\text{m}^{-3}$.

The heat capacity is given by Eq. (10) of McKenzie, Jackson, and Priestley [232]:

$$C_p(T) = k_0 + k_1T^{-1/2} + k_3T^{-3} \tag{83.2}$$

with $k_0 = 233.18$, $k_1 = -1801.6$ and $k_3 = -26.794\cdot10^7$ for fosterite and $k_0 = 252$, $k_1 = -2013.7$ and $k_3 = -6.219\cdot10^7$ for fayalite. We assumed that the molar fraction of fayalite in the mantle is 11%.

The heat conductivity is given by Eq.(4) of McKenzie, Jackson, and Priestley [232]:

$$k(T) = \frac{b}{a+cT} + \sum_{m=0}^{3} d_m(T+273)^m \tag{83.3}$$

with $b = 5.3$, $c = 0.0015$, $d_0 = 1.753\cdot10^{-2}$, $d_1 = -1.0365\cdot10^{-4}$, $d_2 = 2.2451\cdot10^{-7}$ and $d_3 = -3.4071\cdot10^{-11}$

Because the coefficients $\rho(T)$, $C_p(T)$ and $k(T)$ depend on the temperature, which is the unknown of the PDE, this PDE is highly nonlinear and a dedicated algorithm *should* then be designed to handle the nonlinearity. For now the code disregards those considerations and its structure is identical to the linear equation case. We come back to this point hereafter.

Following Richards et al, we run the model for 170 Myr. Boundary conditions are $T = 0$ °C at the top and $T = 1350$ °C at the bottom. The functions for $k(T)$, $C_p(T)$ and $\rho(T)$ are in a separate file temperature_dependent_variables.py.

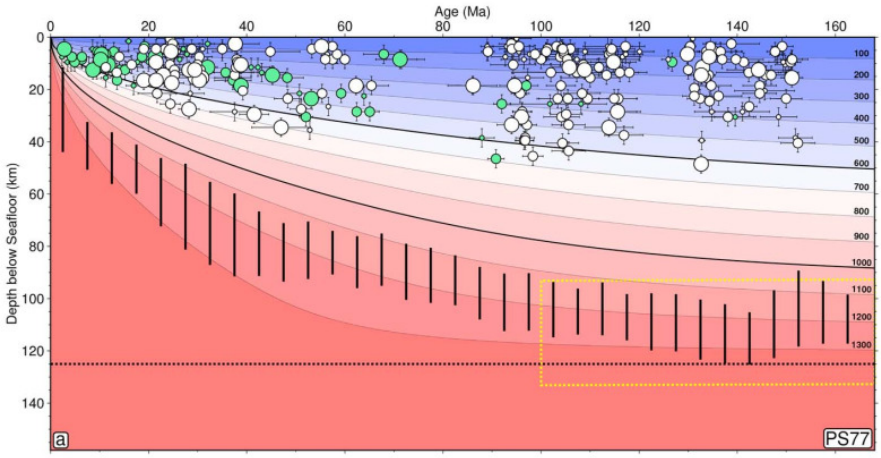
In what follows quadratic elements are used (although the code can also be used with linear ones).

[2] As noted later we should in fact implement a nonlinear loop and these would become the values of the previous iteration.

[3] see Section 9.3 for the derivations

A simple case: Constant parameters

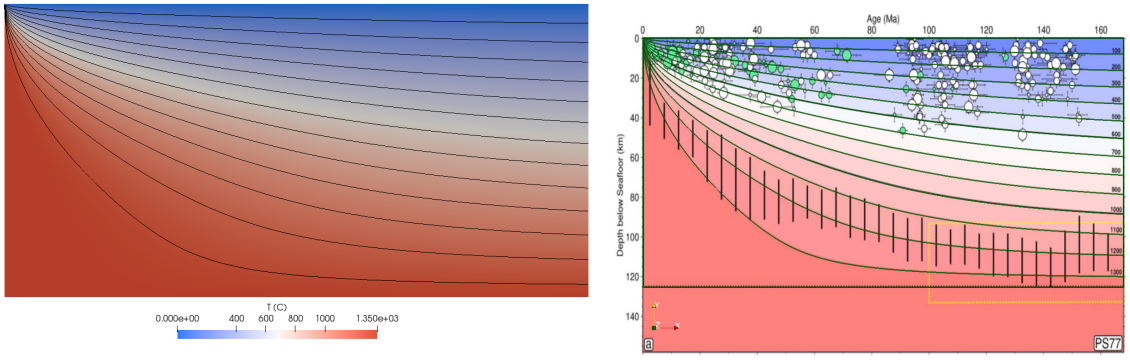
In this case we set $k = 3.138\ \mathrm{W\,m^{-1}\,K^{-1}}$, $C_p = 1171.52\ \mathrm{J\,kg^{-1}\,K^{-1}}$ and $\rho = 3330\ \mathrm{kg\,m^{-3}}$.



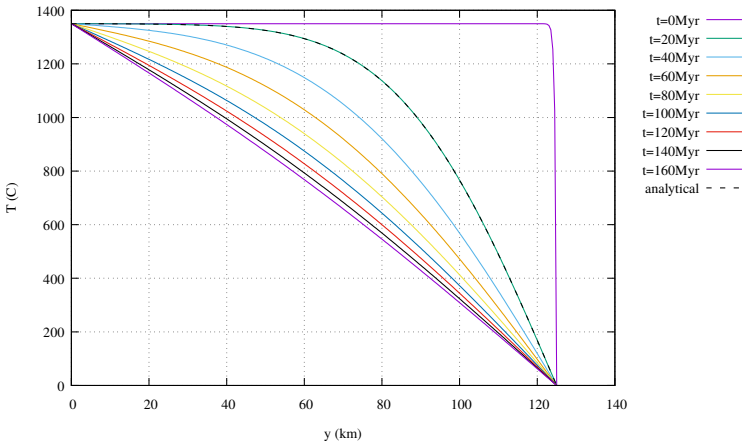
Taken from Richards, Hoggard, Cowton, and White [262]. Thermal structure of oceanic lithosphere. (a) Simple analytical plate model using the published values reported by Parsons and Sclater [248]; numbered contours = isothermal surfaces plotted in °C; green and white circles with error bars = oceanic intraplate and outer rise earthquakes from Craig et al. (2014) where small/medium/large circles = M_b < 5.5, 5.5 − 6.5, and > 6.5; vertical black bars = depth to lithosphere-asthenosphere boundary in the Pacific Ocean based upon peak variations in azimuthal anisotropy (Burgos et al., 2014); dashed box = envelope of depths to lithosphere-asthenosphere boundary for plate ages > 100 Myr (Steinberger and Becker [308]); horizontal black dashed line = base of plate model.

A word about the time step. It is limited by the value h^2/κ where h is the mesh size and κ is the thermal diffusivity. Here, we build the mesh so that $h = 1000$ m and $\kappa = k/\rho C_p = 3.138/1171.52 \cdot 3330 \simeq 8 \cdot 10^{-7}$ (which is very close to the typical 10^{-6} value), so that the maximum timestep is about 1.24e12s, or about 39421 yr. We therefore set $\delta t = 10$kyr. Models are run for 170 Myr, i.e. 17000 time steps!

In essence the code starts by setting the initial temperature to the bottom value and then proceeds to solve the equation in time, thereby producing a profile that starts on the left of the figure above and ends on the right side of it.

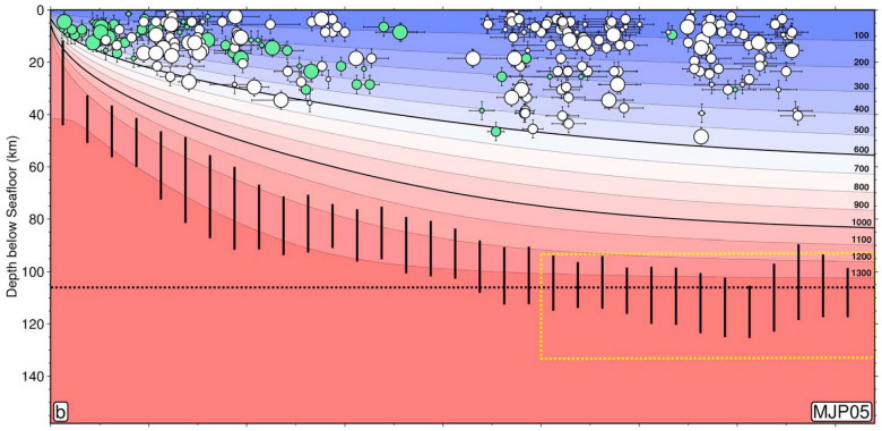


Left: Isocontours every 100°C between 0°C and 1300°C. Right: our results (green lines) super-imposed on those of Richards, Hoggard, Cowton, and White [262].

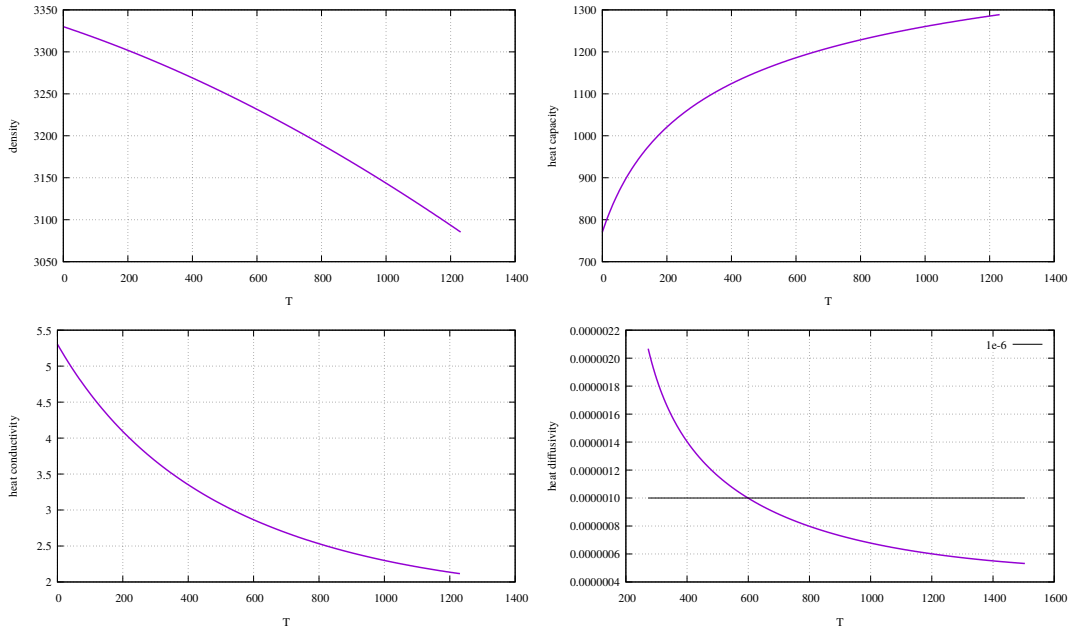


Temperature profile evolution as a function of time. The analytical solution is plotted at $t = 20$ Myr and it agrees with out numerical solution, which thereby benchmarks our code.

T-dependent parameters a la McKenzie et al (2015)



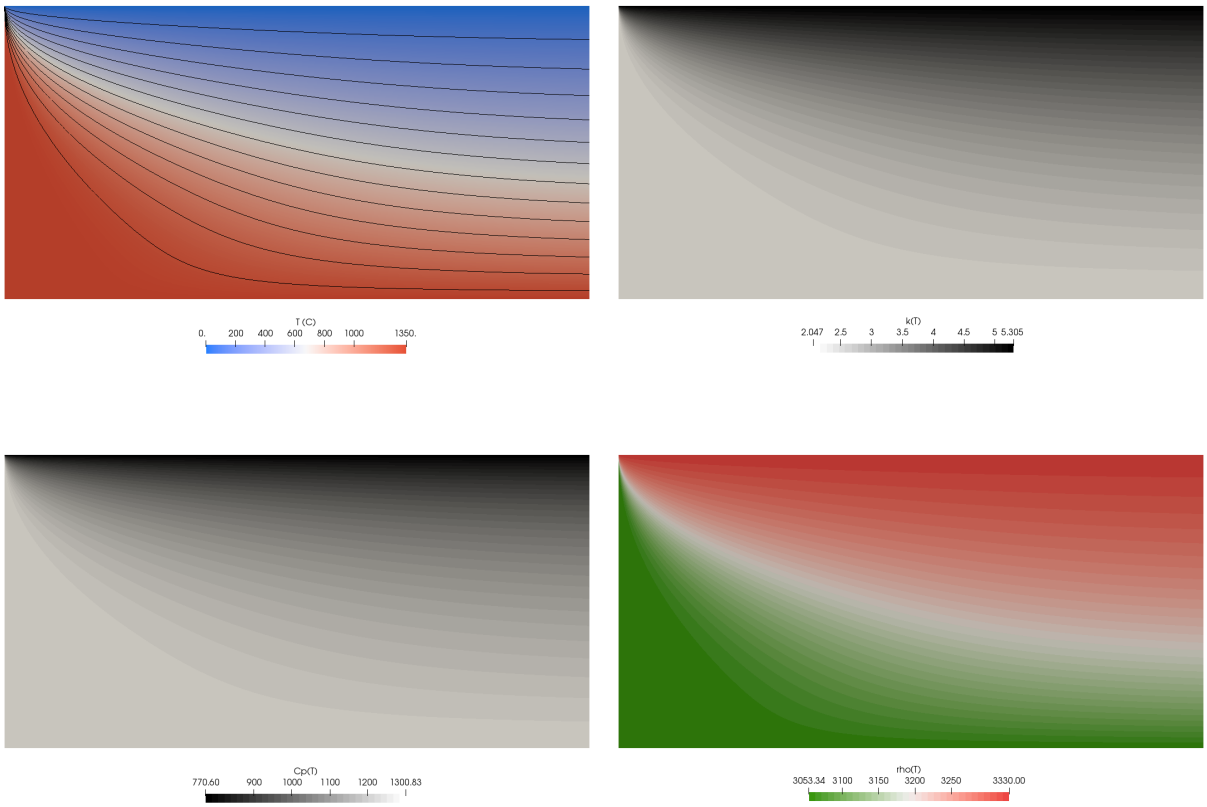
Same for the purely temperature-dependent plate model using parameter values from McKenzie, Jackson, and Priestley [232]. Note that the initial temperature on the left seems to have changed?



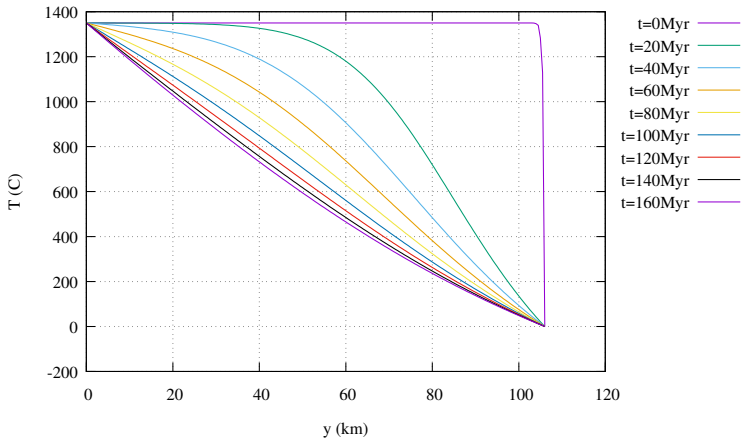
Density, heat capacity and heat conductivity as obtained from Eqs. (83.1), (83.2), and (83.3), and heat diffusivity κ.

Looking at the heat diffusivity values, we see that these are at most twice as large as above. As such

the value of $\delta t = 10$kyr is still warranted.



Temperature, heat conductivity, heat capacity and density profiles as a function of time.



Temperature profile evolution. In this case there is obviously no analytical solution.

Although there is no appropriate algorithm to deal with the nonlinear character of the equation, we can carry out a simple test by changing the timestep to 5,10,20kr and after 20Myr we find that temperatures do not differ by more than 0.2K. It does not replace a proper nonlinear Picard algorithm but this simple check allows us to assess that when using extremely small time steps (akin to in fact doing many nonlinear iterations per larger time step - cost wise), the solution does not change significantly.

Stone 84

computing gravity above objects



3D
box
analytical_benchmark
error_convergence
paraview_output
gravity

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_84`

This stone was developed in collaboration wth Sverre Hassing.

This stone deals with the forward calculations of gravity fields (potential, vector and tensor). It implements to different ways:

- the point mass approach: each cell of size (h_x, h_y, h_z) and of constant density ρ is contracted to a single point in its middle of equivalent mass $m = \rho h_x h_y h_z$.
- the prism approach, as explained in Appendix K.
- the Gauss quadrature approach using n_q^3 points per cell.

Three types of gravity measurements are implemented:

- on a plane, defined by z=z_plane, of size $Lx \times Ly$, counting nnx_plane X nny_plane points
- on a line, starting at point (x_begin,y_begin,z_begin) and ending at point (x_end,y_end,z_end)
- at a single point, atcoordinates xpt,ypt,zpt from the center of the object
- on a 3D spiral starting at the north pole and going to the south pole, parameterised only by the total number of points and the distance to the center/radius of all spiral points. This is called the Fibonacci spiral, or lattice on a sphere. The code is borrowed from the one in the gravity post-processor of ASPECT.

We will try to answer those questions: Is it worth using prisms instead of mass points ? If so, how much more accurate are the former than the latter? How much more expensive in terms of cpu time are prism-based calculations ? What about the Gauss quadrature approach ?

The domain is a cuboid of size $L_x \times L_y \times L_z$ cut into $nel_x \times nel_y \times nel_z = nel$ cells. Note that in all what follows a cell has a constant density.

The function `grav_calc` computes u, $\vec{g}$ and $\boldsymbol{T}$ using one of the three methods hilighted above.

Note that the ASPECT postprocessor relies on the Gauss-Legendre quadrature, see also von Frese, Hinze, Braile, and Luca [342] (1981).

Buried sphere

The domain is a cube of $L = 1$km size. All elements inside a sphere of radius $R = L/2$ are assigned $\rho_s = 100\text{kg}\,\text{m}^{-3}$. The mass of the sphere is

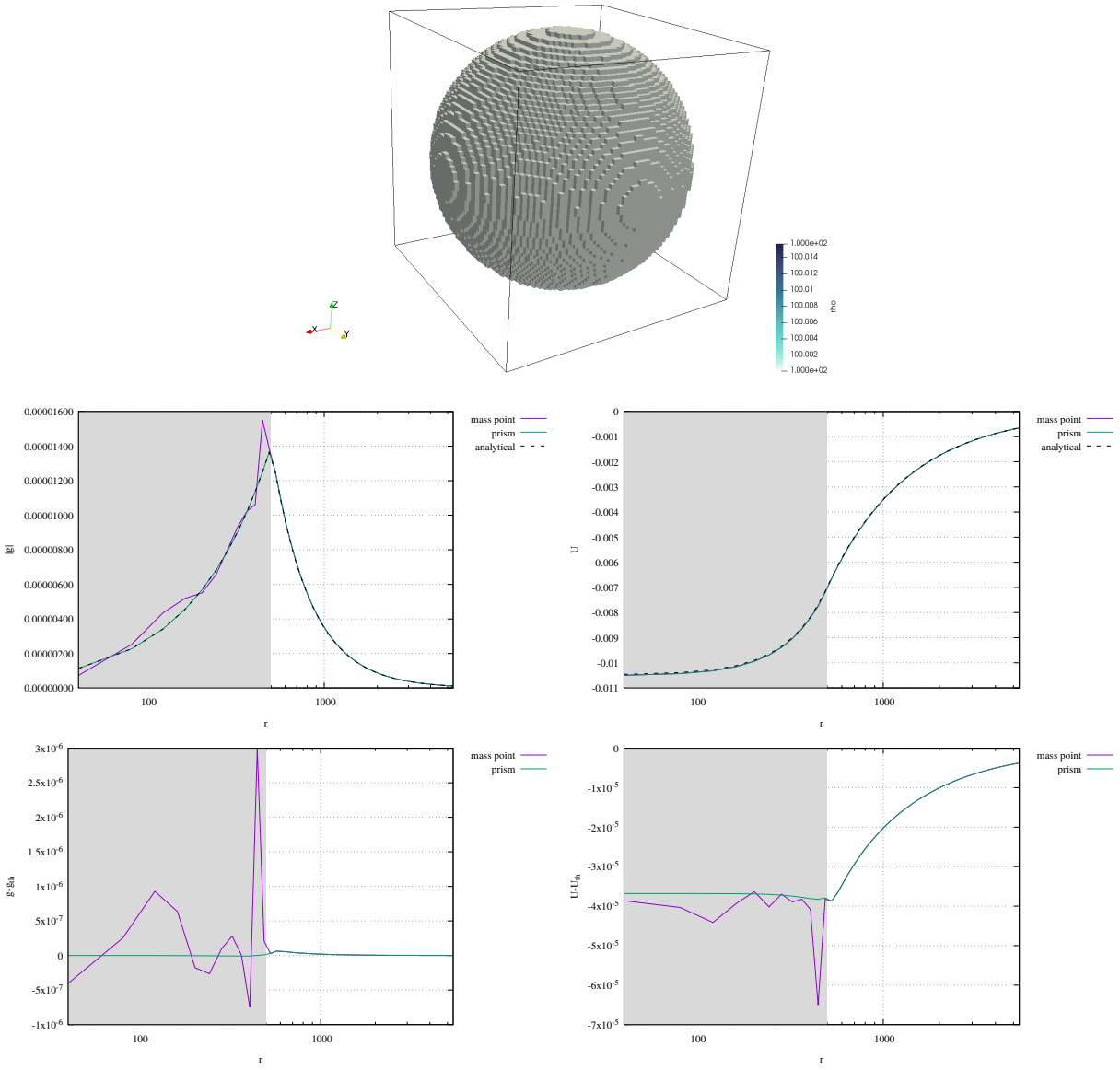
$$M_s = \frac{4}{3}\pi R^3 \rho \simeq 5.23598775598 \cdot 10^{10}\text{kg}$$

So that outside of the sphere the gravity field and potential are trivial to compute:

$$|\vec{g}(r)| = \mathcal{G}\frac{M_s}{r^2} \qquad U(r) = \mathcal{G}\frac{M_s}{r}$$

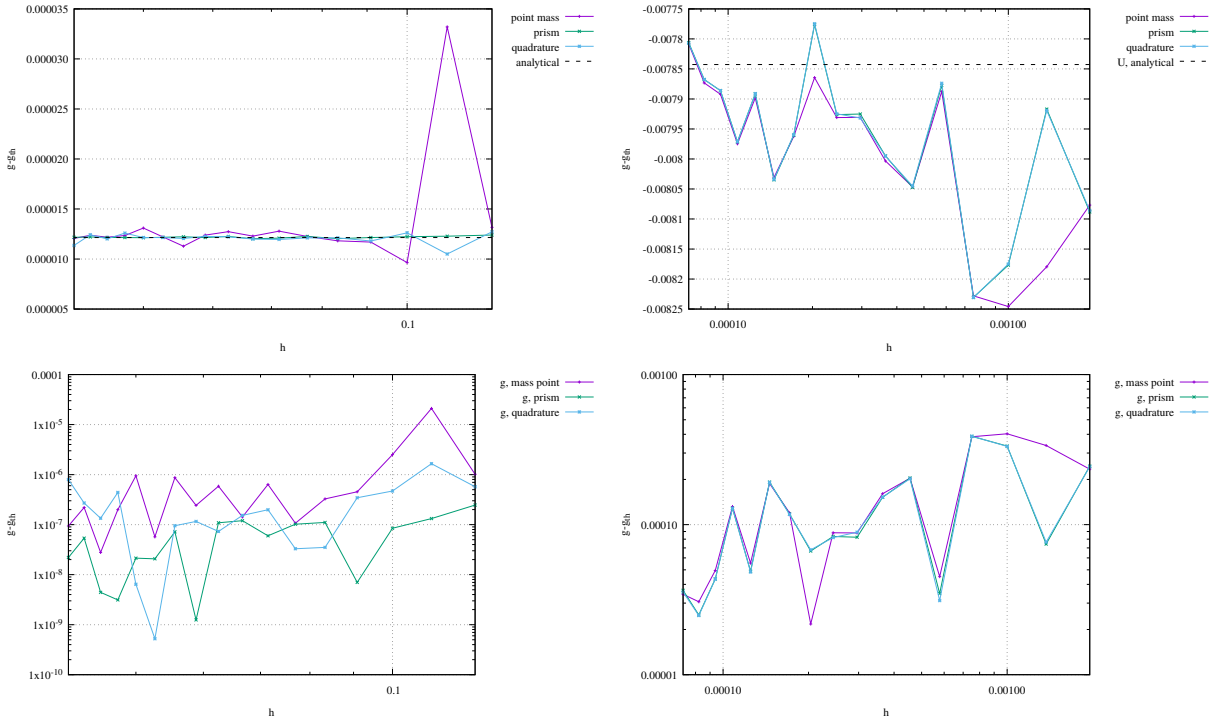
Gravity terms are computed on a line starting at $\vec{r} = (0,0,0)$ and ending at $\vec{r} = (1.11, 2.22, 5.55)$km (this is to avoid any cancelling due to symmetry). The analytical expressions for the gravity and gravity potential are

$$\begin{aligned} |\vec{g}(r)| = g(r) &= \mathcal{G}\rho\frac{4}{3}\pi r \qquad \text{inside} \\ &= \mathcal{G}\rho\frac{4}{3}\pi\frac{R^3}{r^2} \qquad \text{outside} \\ U(r) &= -2\pi\mathcal{G}\rho(R^2 - r^2/3) \qquad \text{inside} \\ &= -\mathcal{G}\rho\frac{4}{3}\pi\frac{R^3}{r} \qquad \text{outside} \end{aligned}$$



The grey area indicates the sphere. Mesh resolution?

The gravity was also measured at location $(123, 234, 345)$m counted from the center of the sphere for increasing resolutions.

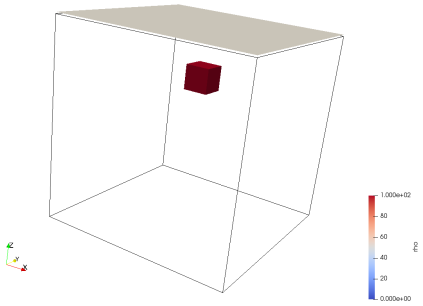


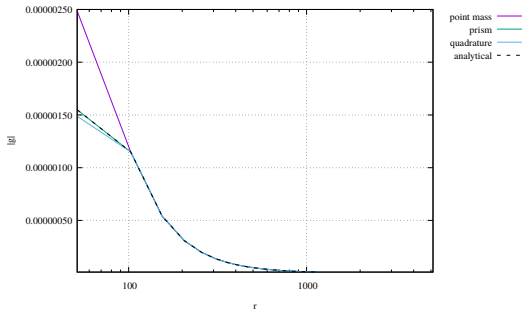
Buried cube

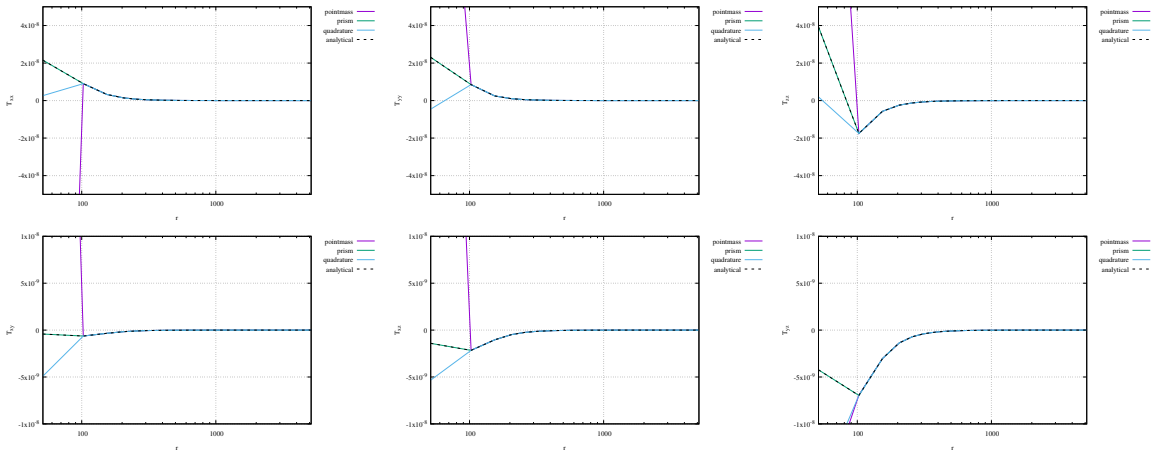
The domain is a cube of $L = 1\text{km}$ size. All elements inside a cube of size $d = L/8$ are assigned $\rho_s = 100\text{kg}\,\text{m}^{-3}$. The mass of the cube is then

$$M_c = \rho d^3 = 1.95312500 \cdot 10^8 \text{kg}$$

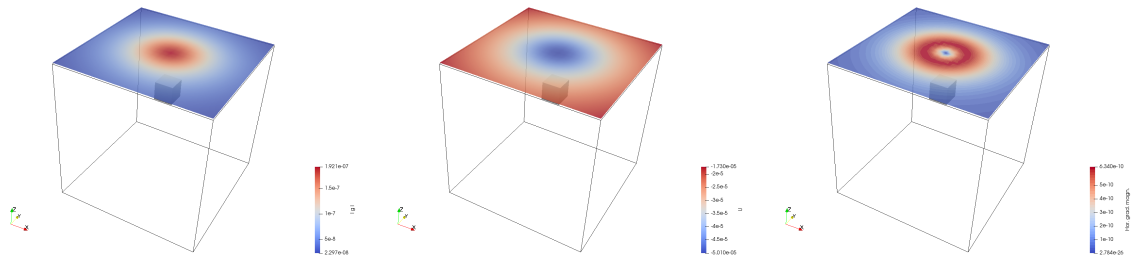
Far away from the cube $g(r) \to \mathcal{G}M_c\rho/r^2$ and $U(r) \to -\mathcal{G}M_c/r$.

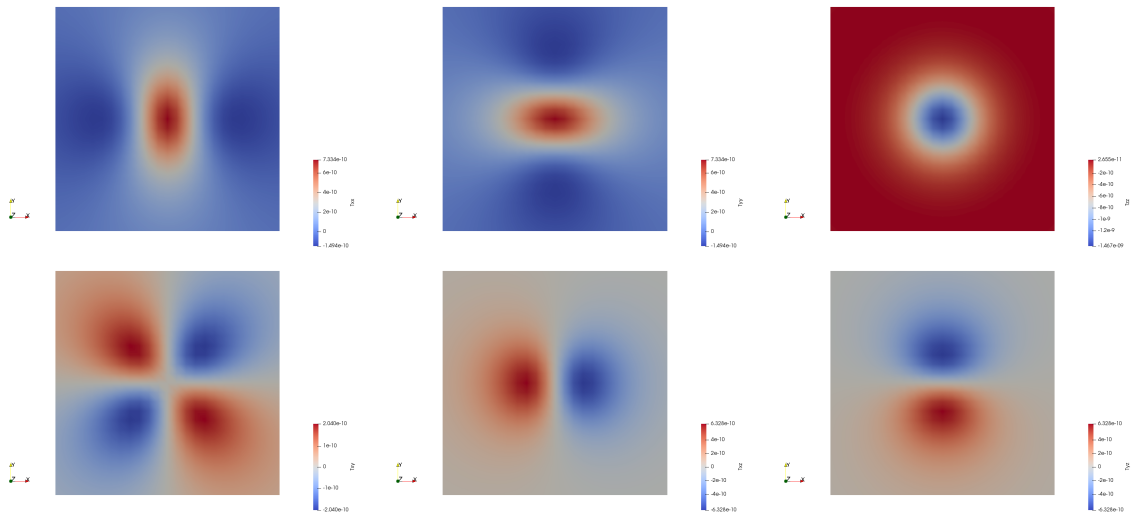






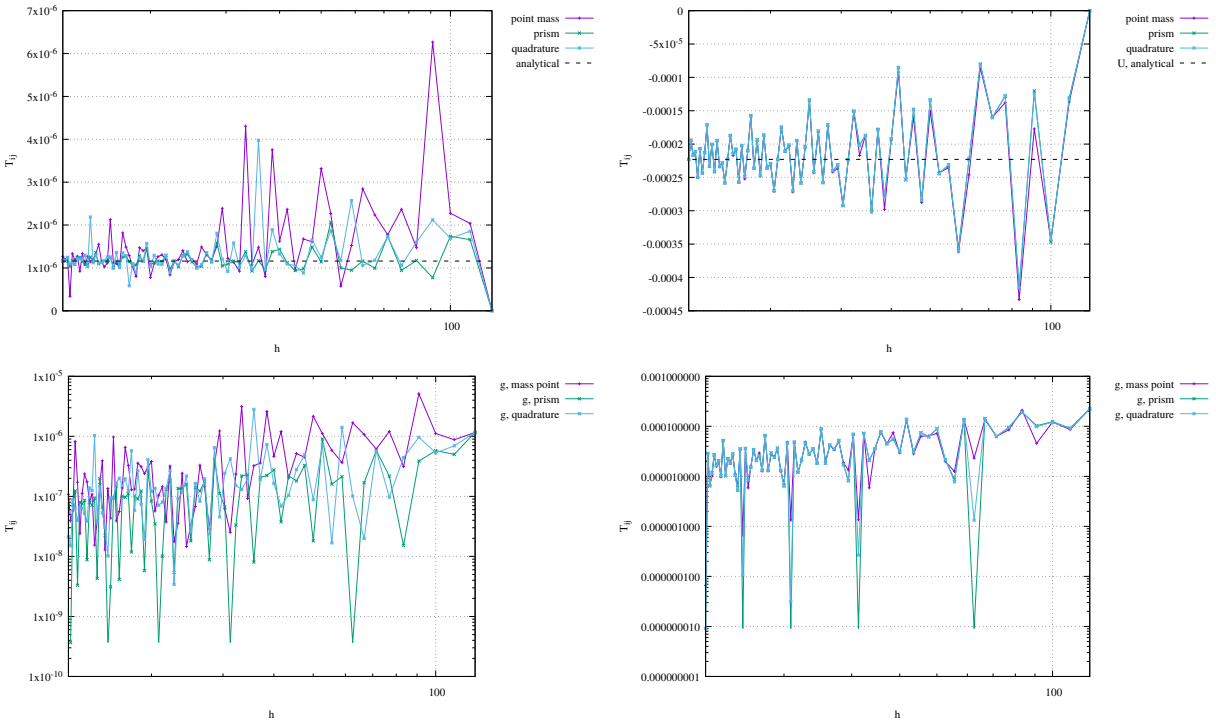
The grey area indicates the sphere. resolution $32 \times 32 \times 32$.





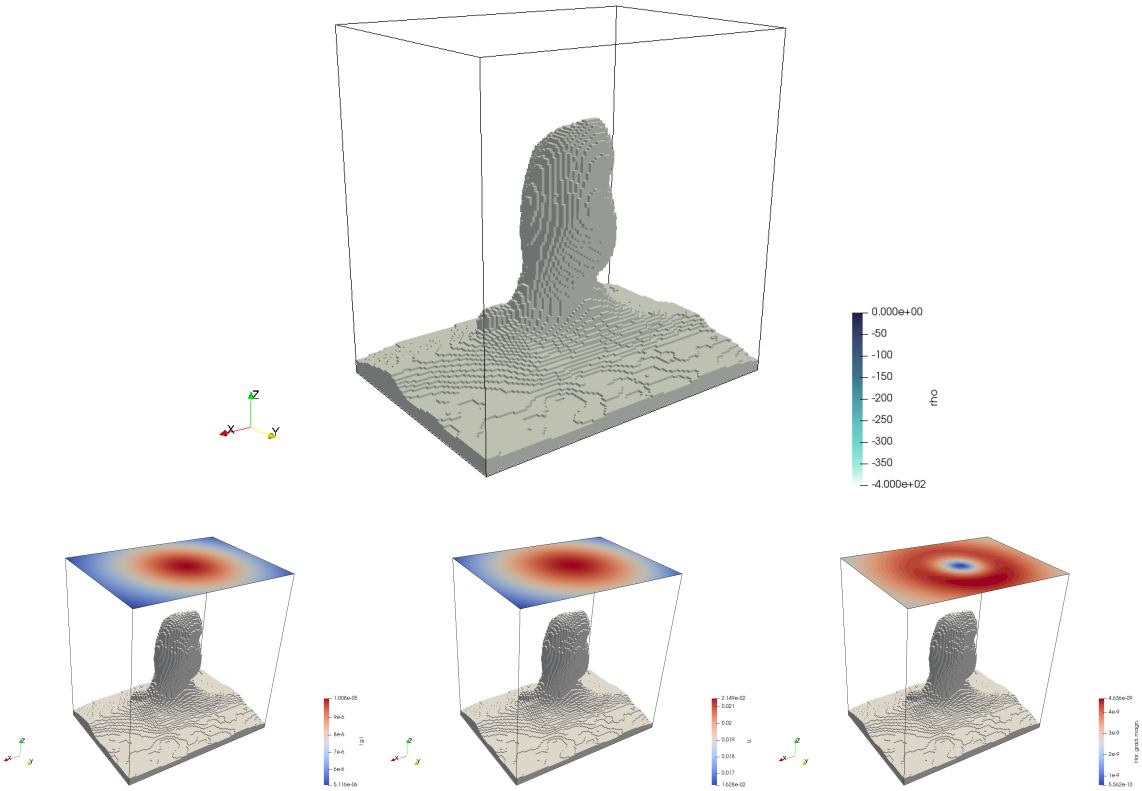
xxx

The gravity was also measured at location $(12, 23, 34)$m counted from the center of the cube for increasing resolutions.



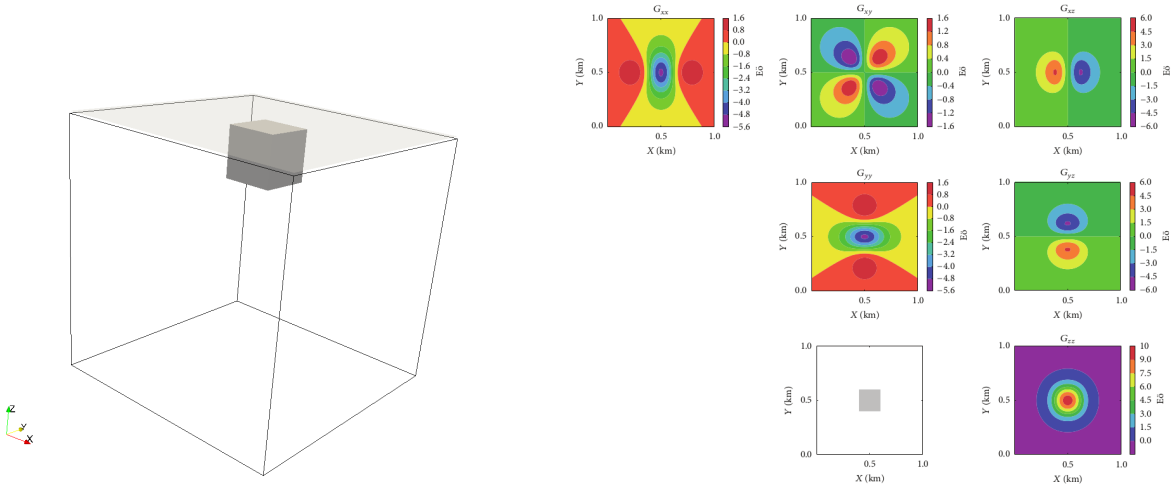
Salt diapir

The domain has dimensions $L_x = 2940$, $L_y = 2100$, $L_z = 3060$ and the data set makes that nelx=98, nely=70 and nelz=153. The data is synthetically generated using methods described in Clausolles, Collon, and Caumon [78] (2019) and the file containing the data is *salt_dome.data*. The salt has a relative density of $-400\text{kg}\,\text{m}^{-3}$.

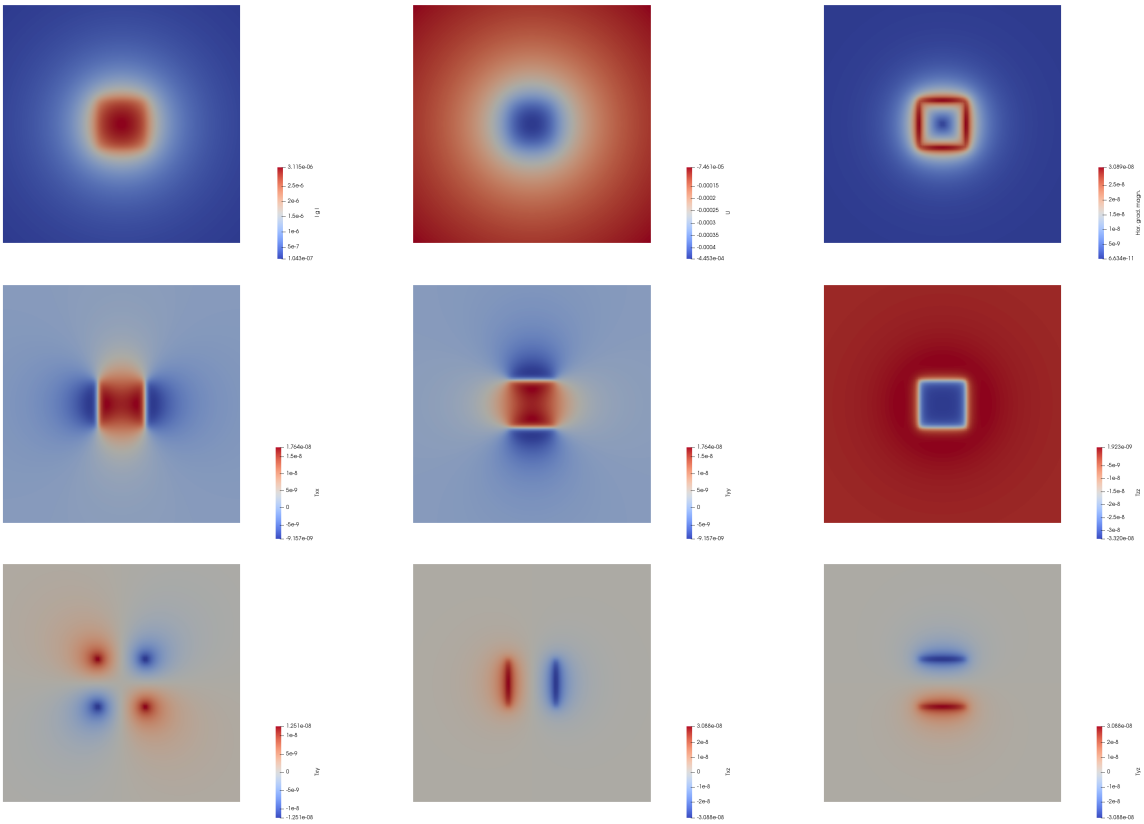


gravity vector norm, gravity potential, and horizontal gradient

Prism of Arroyo et al (2015)

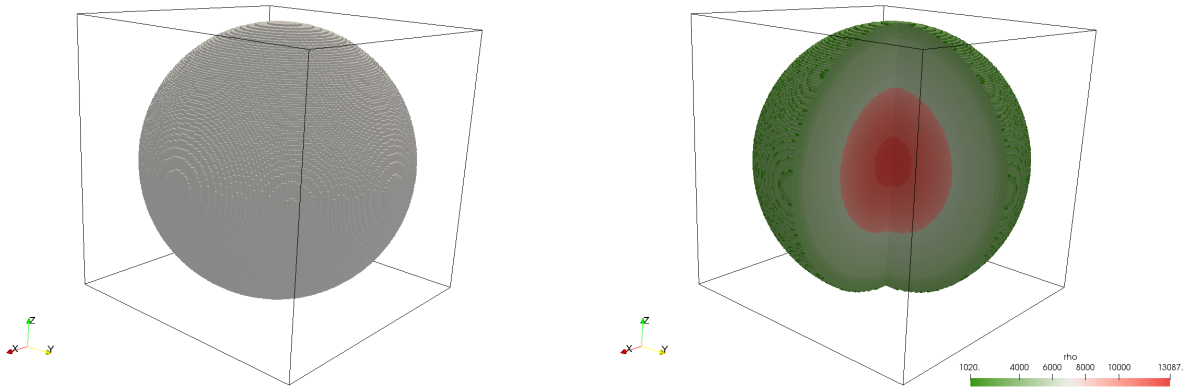


Left: setup. Right: taken from Arroyo, Couder-Castaneda, Trujillo-Alcantara, Herrera-Diaz, and Vera-Chavez [6] (2015).

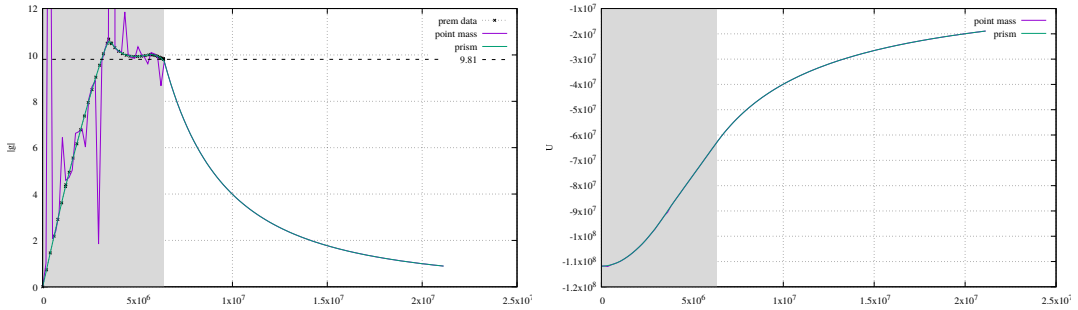


gravity vector norm, gravity potential, horizontal gradient, and 6 components of $\boldsymbol{T}$.

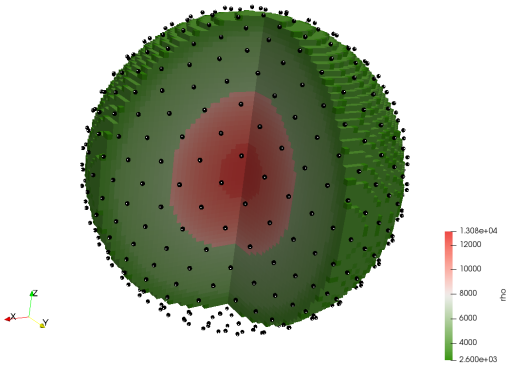
Whole Earth with PREM density



resolution 150 × 150 × 150. Density is obtained from the PREM model [101].



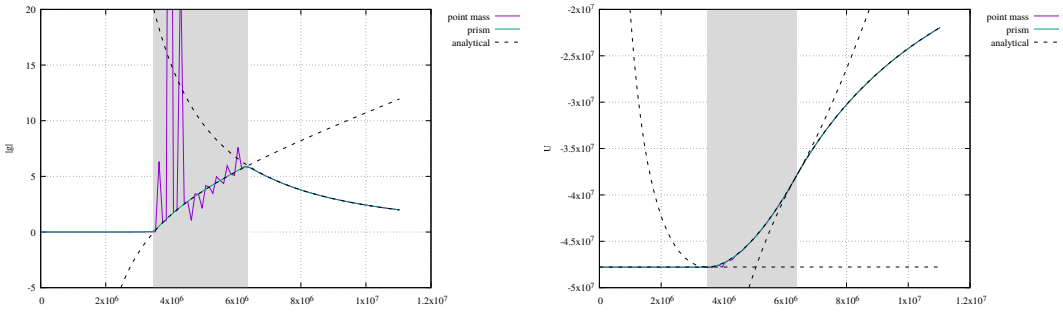
resolution 64 × 64 × 64.



Spiral of 500 points at height 250 km above the Earth surface. Earth is ased on 64 × 64 × 64 resolution.

Hollow Earth with constant density

The density is constant and set to $\rho = 4000\text{kg}\,\text{m}^{-3}$. Outer radius is 6371 km and inner radius is 3480 km.



resolution 64 × 64 × 64

Stone 85

Using the S20RTS and S40RTS datasets



Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_85`

This stone was developed with input from Rens Elbertsen, Bart Root, and Ross Maguire.

The S20RTS [265] and S40RTS [264] are commonly used shear-velocity models for the mantle. The dataset are widely available and used (in Specfem, in ASPECT , ...) and come in the form of two files *S20RTS.sph* and *S40RTS.sph* (supplemented by a file with the location of the knots of the 21 splines used for the radial interpolation). Each *.sph* file has a 1 line header, and the first number on this line indicates the maximum degree number l (20 or 40, then)[1]. The rest of these files consists of many rows and colums of numbers.

Let us recall that for a given value of the degree l the order m is such that $-l \leq m \leq +l$, i.e. there are $2l+1$ values of m for a single value of l. Concretely:

- $l=0$, $m=1$
- $l=1$, $m=-1,0,+1$
- $l=2$, $m=-2,-1,0,+1,+2$
- $l=3$, ...

The first few lines of *S20RTS.sph* are shown hereunder:

```
          20 11111111111111111111  24 000111111111111111111111
 0.1534E-01
 0.1590E-01 -0.1336E-01  0.3469E-02
-0.3480E-02  0.1165E-01  0.8376E-02  0.2158E-01 -0.9923E-02
-0.1301E-02 -0.5792E-02  0.1049E-02 -0.7702E-02  0.1281E-02  0.1419E-01  0.8916E-02
 0.1353E-02 -0.5517E-02 -0.1429E-02 -0.1105E-01 -0.1247E-02  0.4788E-02 -0.8670E-03 -0.2317E-03  0.3142E-01
-0.7365E-02 -0.1193E-01 -0.4838E-02 -0.1277E-01  0.1034E-03  0.6585E-02 -0.1351E-01  0.2126E-01 -0.1926E-01 -0.4675E-02 -0.7870E-02
 0.5379E-02  0.7332E-02  0.9048E-02 -0.7672E-02  0.1306E-01 -0.2900E-02 -0.1380E-01 -0.2366E-02  0.7911E-02 -0.9940E-02 -0.1281E-02  0.1321E-02 -0.1439E-02
```

We see that the first line (after the header) contains one coefficient, corresponding to $l=0, m=0$. The second line contains three coefficients which correspond to $l=1, m=-1,0,1$ (not per se in that order – see what follows). The third line contains 5 coefficients, the fourth 7, etc ... In the case of S20RTS this goes on until $l=20$ and the line therefore contains 41 coefficients. In the case of S40RTS this goes on until $l=40$ and the line contains 81 coefficients. Unfortunately only 11 numbers are stored on a single line so the 81 coeffs corresponding to $l=40$ (for instance) are spread across many lines, which makes read in the file a bit of nightmare. Both S20RTS and S40RTS models are composed of 21 concentric layers (from the moho down to the core-mantle boundary) so the above structure is repeated 21 times in the files.

[1] "The header lines in the sph file includes a 24 followed by 3 zeroes and 21 ones. This indicates that the first three splines, which apply to the crust, have been turned off." J. Ritsema, pers. comm.

The structure of the file is explained in the ASPECT manual[2] (but it remains a mystery as to where this information comes from since it is not explained in the papers) in a cookbook authored by Jacqueline Austermann:

> The first number in the first line denotes the maximum degree. This is followed in the next line by the spherical harmonic coefficients from the surface down to the CMB. The coefficients are arranged in the following way:
> a_{00}
> a_{10} a_{11} b_{11}
> a_{20} a_{21} b_{21} a_{22} b_{22}
> ...
> a_{lm} is the cosine coefficient of degree l and order m; b_{lm} is the sine coefficient of degree l and order m.

The "Initial temperature model" subsection of ASPECT describing the "S40RTS perturbation" is at `https://aspect-documentation.readthedocs.io/en/latest/parameters/Initial_20temperature_20model.html`.



In the file *aspect/source/initial_temperature/S40RTS_perturbation.cc* it is indicated that the implementation is based on Eq. (B.99) of Dahlen & Tromp [83], i.e. the velocity anomaly on any of the shells (i.e. at constant radius) at coordinate (θ, ϕ) can be computed as follows:

$$f = \sum_{l=0}^{\infty} \left[a_{l0} X_{l0}(\theta) + \sqrt{2} \sum_{m=1}^{l} X_{lm}(\theta) \left(a_{lm} \cos m\phi + b_{lm} \sin m\phi \right) \right]$$

Note however that the implementation of Ritsema does *not* include the $\sqrt{2}$ term (go figure...) and this was the nature of a later pull request in ASPECT [3] so that the used formula is simply:

$$\boxed{f = \sum_{l=0}^{\infty} \left[a_{l0} X_{l0}(\theta) + \sum_{m=1}^{l} X_{lm}(\theta) (a_{lm} \cos m\phi + b_{lm} \sin m\phi) \right]}$$

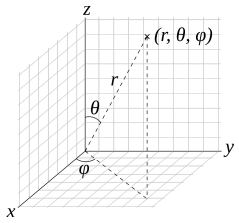
with $X_{lm}(\theta)$ given by Eq.(B.58) of [83]:

$$\boxed{X_{lm} = (-1)^m \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} P_l^m(\cos\theta)}$$

and note that in this equation P_l^m does not contain the $(-1)^m$ term.

[2] `https://aspect-documentation.readthedocs.io/en/latest/user/cookbooks/cookbooks/initial-condition-S20RTS/doc/initial-condition-S20RTS.html`

[3] `https://github.com/geodynamics/aspect/pull/966`



Spherical coordinates conventions

In these equations θ is the colatitude ($0 \leq \theta \leq \pi$ from North pole to South pole), ϕ is the longitude ($0 \leq \phi \leq 2\pi$) and P_l^m are associated Legendre polynomials.

This STONE uses the `scipy.special.lpmv` function for the product $(-1)^m P_l^m(x)$:

SciPy.org

SciPy.org | Docs | SciPy v1.5.2 Reference Guide | Special functions (scipy.special)

scipy.special.lpmv

scipy.special.lpmv(*m, v, x*) = <ufunc 'lpmv'>

Associated Legendre function of integer order and real degree.

Defined as

$$P_v^m = (-1)^m (1 - x^2)^{m/2} \frac{d^m}{dx^m} P_v(x)$$

where

$$P_v = \sum_{k=0}^{\infty} \frac{(-v)_k (v+1)_k}{(k!)^2} \left(\frac{1-x}{2}\right)^k$$

is the Legendre function of the first kind. Here $(\cdot)_k$ is the Pochhammer symbol; see poch.

Parameters: m : *array_like*
Order (int or float). If passed a float not equal to an integer the function returns NaN.

v : *array_like*
Degree (float).

x : *array_like*
Argument (float). Must have `|x| <= 1`.

Returns: pmv : *ndarray*
Value of the associated Legendre function.

In Ritsema, Deuss, Heijst, and Woodhouse [264] (2011), one reads:

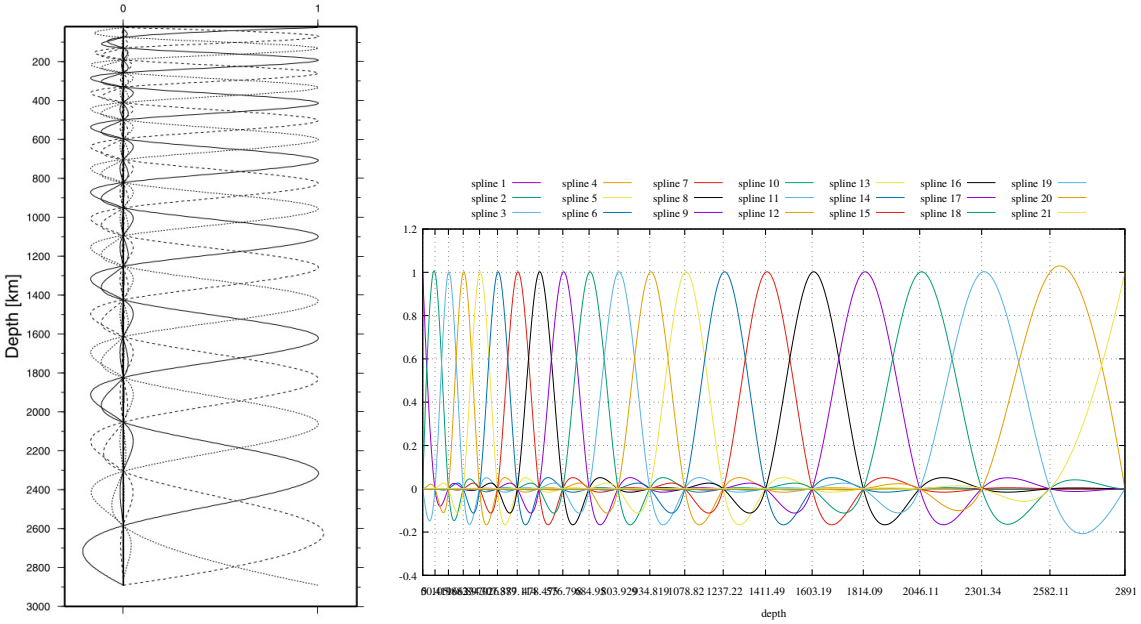
> "We use the same spline functions as in Ritsema, Van Heijst, and Woodhouse [266] (2004) to parametrize vertical variation of shear velocity. The separation of the spline functions increases with depth. Relatively dense spline distribution helps to accommodate strong vertical shear-velocity variations across the lithosphere–asthenosphere boundary and the phase transition in the transition zone albeit at the expense of the vertical resolution at the base of the mantle.

Unfortunately the 2004 paper does not reveal the values of the spline knots, but the values are available in the *splhsetup.f* file included in the tools available on J. Ritsema's website `https://jritsema.earth.lsa.umich.edu/Research.html`.

normalised	radius	depth	spline number
1.00000	6346619.	24381	0 (moho)
0.96512	6296625.16464	74374.83536	1
0.92675	6241629.079125	129370.920875	2
0.88454	6181129.08513	189870.91487	3
0.83810	6114566.19195	256433.80805	4
0.78701	6041338.409595	329661.590405	5
0.73081	5960786.415695	410213.584305	6
0.66899	5872179.222405	498820.777595	7
0.60097	5774685.510215	596314.489785	8
0.52615	5667445.293425	703554.706575	9
0.44384	5549469.58848	821530.41152	10
0.35329	5419683.413255	951316.586745	11
0.25367	5276897.120865	1094102.879135	12
0.14409	5119835.065855	1251164.934145	13
0.02353	4947035.272535	1423964.727465	14
-0.10909	4756949.766645	1614050.233355	15
-0.25499	4547829.910595	1823170.089405	16
-0.41550	4317769.40275	2053230.59725	17
-0.59207	4064689.944335	2306310.055665	18
-0.78631	3786283.907055	2584716.092945	19
-1.00000	3480000.	2891	20 (CMB)

Spline knots. These range from radius 3480km (cmb) to radius 6346.619km (moho) as defined in Ritsema's file getfdp.f and splhsetup.f

The values in the table above seem to correspond to the figure taken from Ritsema et al (2004) [266] shown hereunder:



Left: Taken from Ritsema et al (2004) [266]. Right: Data obtained from R. Maguire's code on github[4]. We see that the splines are zero at the depths/nodes indicated by the tics on the x axis, and that they are 1 on their respective node, i.e. $B_i(d_j) = \delta_{ij}$

The structure of the code is as follows: the user chooses the value of three parameters:

- degree: (maximum) degree, i.e. 20 or 40.
- use_degree: a value between 0 and degree
- depth: depth in m.

[4] `https://github.com/romaguir/sph_models`

Based on the value of degree, either *S20RTS.sph* or *S40RTS.sph* is read, and the coefficients are stored in the array

```
flm = np.empty((degree+1,2*degree+1,nspline),dtype=np.float64)
```

The first dimension is l, the second runs over m (i.e. $2l+1$ values), and this structure is necessary for all 21 splines (third dimension). A regular grid of 360×179 longitude-latitude points is then generated (as well as their connectivity, required for vtu output). This choice is to match the default output of the Ritsema code. Spline knots locations are then read, and converted to depths and radii between the CMB and the Moho. A double for loop then runs over all points of the longitude-latitude plane and computes dv/v as defined by the equations above for each spline depth, and the 21 values are then averaged/interpolated so as to return the value at the desired depth.

The core algorithm goes as follows: we wish to compute the seismic velocity anomaly at location r, θ, ϕ. We then compute $X_{lm}(\cos\theta)$, $\cos(m\phi)$, and $\sin(m\phi)$ for all the existing/relevant (l, m) combinations. These are then combined to store the terms multiplying the a_{lm} and b_{lm} coefficients in the arrays a_coeffs and b_coeffs (remember that the $(-1)^m$ term is inside the `lpmv` function).

```
a_coeffs   = np.zeros((use_degree+1,use_degree+1), dtype=np.float64)
b_coeffs   = np.zeros((use_degree+1,use_degree+1), dtype=np.float64)
for ilat in range(0,nlat):
    for ilon in range(0,nlon):
        for l in range(0,use_degree+1):
            m=0
            x=np.cos(theta)
            Xlm=np.sqrt( (2*l+1)/4./np.pi ) * scipy.special.lpmv(m,l,x)
            a_coeffs[l,0]=Xlm
            for m in range(1,l+1):
                A=math.factorial(l-m)
                B=math.factorial(l+m)
                x=np.cos(theta)
                Xlm=np.sqrt( (2*l+1)/4./np.pi*float(A)/float(B) ) * scipy.special.lpmv(m,l,x)
                a_coeffs[l,m]=Xlm*np.cos(m*phi)
                b_coeffs[l,m]=Xlm*np.sin(m*phi)
```

For each spline shell we use the corresponding a_{lm} and b_{lm} coefficients to compute the anomaly at (θ, ϕ). We end up with 21 values stored in the array `shell_values` which are then combined with the radial spline functions to obtain the anomaly at the desired depth:

```
for ilat in range(0,nlat):
    for ilon in range(0,nlon):
        [...code snippet above...]
        shell_values=np.zeros(nspline,dtype=np.float64)
        for ispline in range(0,nspline):
            for l in range(0,use_degree+1):
                shell_values[ispline]+=flm[l,0,ispline]*a_coeffs[l,0]
                for m in range(1,l+1):
                    alm=flm[l,2*m-1,ispline]
                    blm=flm[l,2*m,ispline]
                    shell_values[ispline]+=a_coeffs[l,m]*alm+b_coeffs[l,m]*blm
        tck=interpolate.splrep(spline_depths, shell_values)
        dv[counter]=100*interpolate.splev(depth,tck)
```

As per usual we see that the python code is ridiculously slow...

How to use the provided S20/S40RTS tools :

- Download *S20RTS_plotting.tar.gz* at `https://jritsema.earth.lsa.umich.edu/Research.html`
- Compile everything as described in the *readme* file
- Copy the desired *SX0RTS.sph* file to be used in the bin directory
- Move to the bin directory
- Create a file *dep_in* with on the first line the name of the .sph file (e.g. *S40RTS.sph*) and on the second line the desired depth (in km).

- Run the command: *./depmaphj_jr < dep_in > dep_out*
  This also creates a .raw file with the same name as the .sph file chosen in *dep_in*, containing the spherical harmonic coefficients belonging to the chosen depth

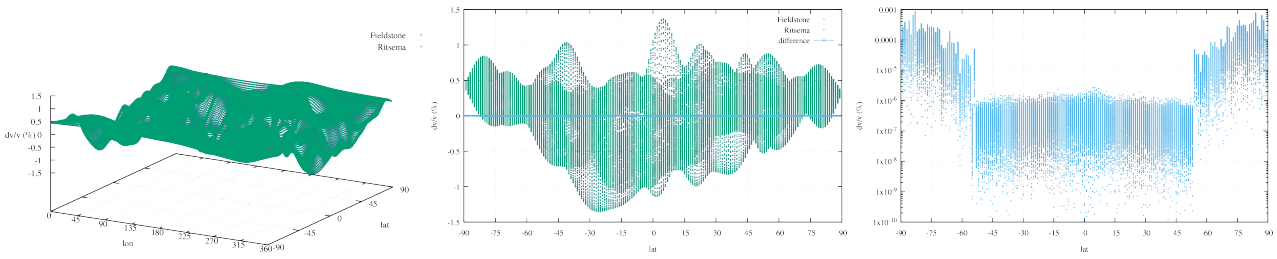
- Create a file *input* with the following lines (separate lines denoted by brackets):
  S40RTS.raw
  outputfile.xyz
  1
  1
  1 40
  -180
  where the first line is the .raw file created in the previous step, the second line is the name of the output file, the third line is the step size in degrees for lon and latitude centered around 0, the fourth line is a mystery to me, the fifth line are the degrees that should be used and the sixth line probably has to do with the starting or end point of the x-axis.

- Run *./raw2xyz_jr < input* this creates a *file.xyz* as chosen in the input file containing lon, lat and dlnvs in %

The Ritsema code is in the *S20RTS_plotting* folder. In the *bin* subfolder you will find my script *make_map* which generates .xyz file (i.e. all the values at the desired depth). Simply edit *dep_in* and *input* and adjust the parameters.

It is worth noticing that these tools are assembled from exisiting tools and libraries (e.g. Numerical Recipes) and that double precision floats are not used everywhere, which, given the thousands of summations and multiplications of terms like cosine/sine/square root/pi combined with polynomial splines is likely to result in a loss of accuracy. As such, given that the values we wish to compare are around unity one cannot hope to match the results from the python code with the fortran code with an accuracy better than $\sim 10^{-7}$, i.e. the accuracy of single precision floats (and even that value is highly optimistic).

One can for instance select a depth which corresponds to one of the knots of the spline, say $d = 2584716.092945$m. In this case we know that the spline interpolation will/should not take place since the spline associated to this depth is exactly 1 and the other ones exactly zero. The python and fortran codes were run at this depth and at the same resolution (nlat=179,nlon=360) and results are found to match nicely. The *compare.py* script subtracts the results of both and generates the *both.ascii* file as plotted hereafter:



Left: 3D surface plot of the signal as a function of longitude and latitude for both codes. Middle: same data, but projected on a 2D plane with the latitude as x-coordinate, with difference between both plotted in blue. Right: (absolute) difference as a function of latitude in log-scale.

When Ritsema's code is run at this depth a file *dep_out* is produced. Its structure is not totally clear but we see that the last column corresponds to the weights attributed to all 21 splines. These are all nearly zero except for the 20th which is nearly 1, as expected. This documents again the single precision character of the calculation since 0.999999881 is 1 at 10^{-7} accuracy. Rather worryingly, these nearby coefficients are about 10^{-7} and will therefore contribute to the synthetised signal.

```
model=S20RTS.sph
lm=           10
ip1+1=           1  lm-4=           6
mfl=S20RTS
n=            1  dep=    2584.71606
```

```
y1
y2
          20            4           24           21
          20            4           24           21
ii,dep(ii),ip,fp=             1   2584.71606                1   7.92752561E-17
ii,dep(ii),ip,fp=             1   2584.71606                2  -4.69479181E-16
ii,dep(ii),ip,fp=             1   2584.71606                3   1.70541370E-15
ii,dep(ii),ip,fp=             1   2584.71606                4  -5.56919674E-15
ii,dep(ii),ip,fp=             1   2584.71606                5   1.80295317E-14
ii,dep(ii),ip,fp=             1   2584.71606                6  -5.83941600E-14
ii,dep(ii),ip,fp=             1   2584.71606                7   1.89143623E-13
ii,dep(ii),ip,fp=             1   2584.71606                8  -6.12492045E-13
ii,dep(ii),ip,fp=             1   2584.71606                9   1.98377881E-12
ii,dep(ii),ip,fp=             1   2584.71606               10  -6.42507532E-12
ii,dep(ii),ip,fp=             1   2584.71606               11   2.08086742E-11
ii,dep(ii),ip,fp=             1   2584.71606               12  -6.73884906E-11
ii,dep(ii),ip,fp=             1   2584.71606               13   2.18269541E-10
ii,dep(ii),ip,fp=             1   2584.71606               14  -7.06871173E-10
ii,dep(ii),ip,fp=             1   2584.71606               15   2.28940222E-09
ii,dep(ii),ip,fp=             1   2584.71606               16  -7.41458184E-09
ii,dep(ii),ip,fp=             1   2584.71606               17   2.40124489E-08
ii,dep(ii),ip,fp=             1   2584.71606               18  -7.77709275E-08
ii,dep(ii),ip,fp=             1   2584.71606               19   2.92896800E-07
ii,dep(ii),ip,fp=             1   2584.71606               20  0.999999881
ii,dep(ii),ip,fp=             1   2584.71606               21  -1.05048358E-07
```

Based on these indications we can run the code and compare the obtained results with those obtained with Ritsema's and/or SubMachine [158]. Note that the data used in SubMachine are based on files (lat, lon, dvs) that Paula Koelemeijer provided to the lead author, and then interpolated between 25 km depths (pers. comm.).

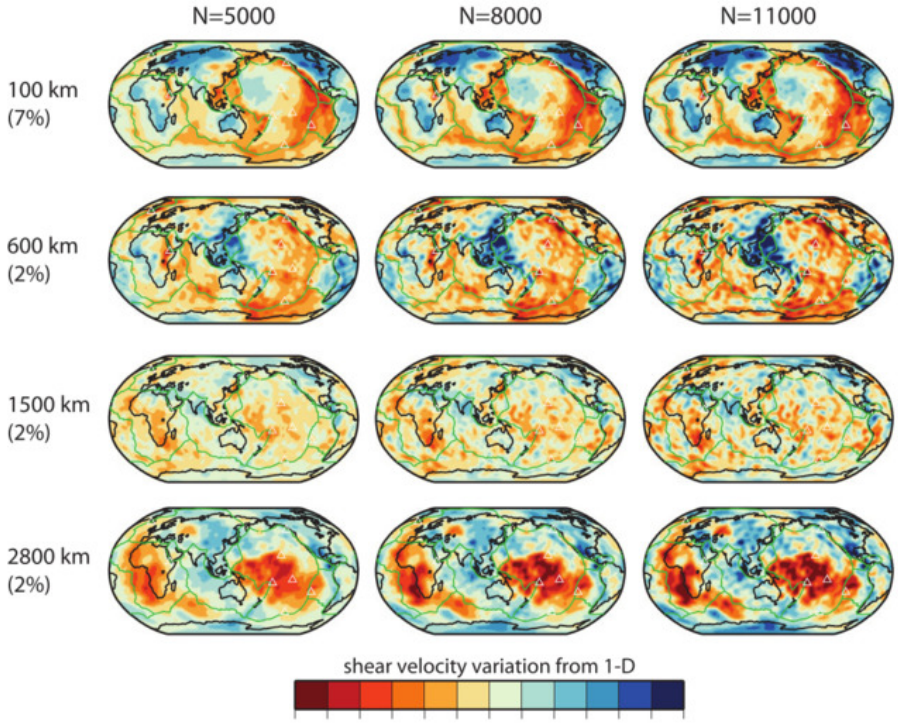
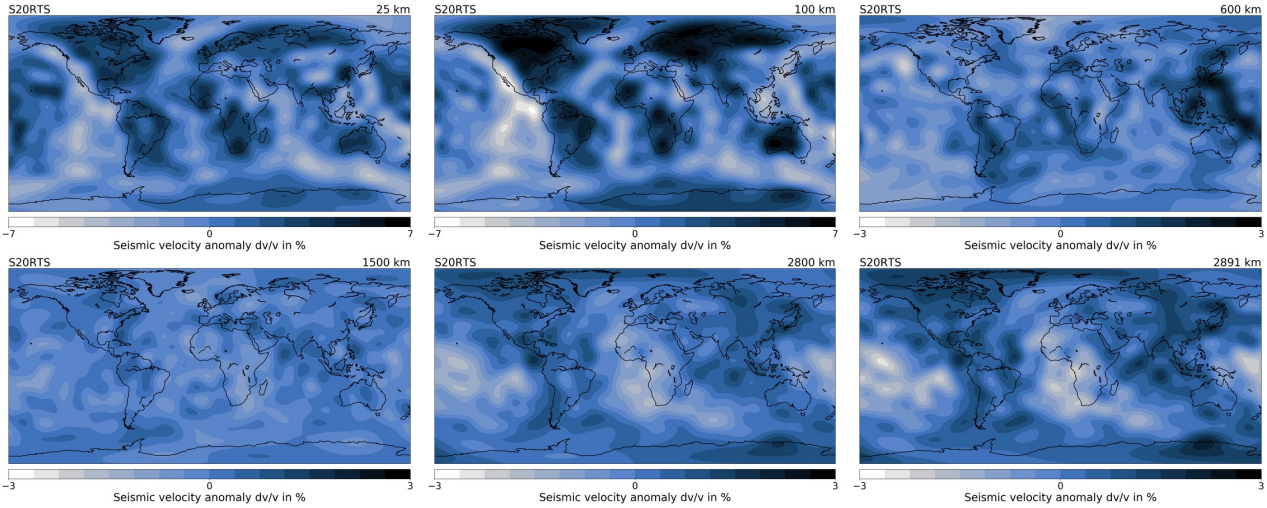


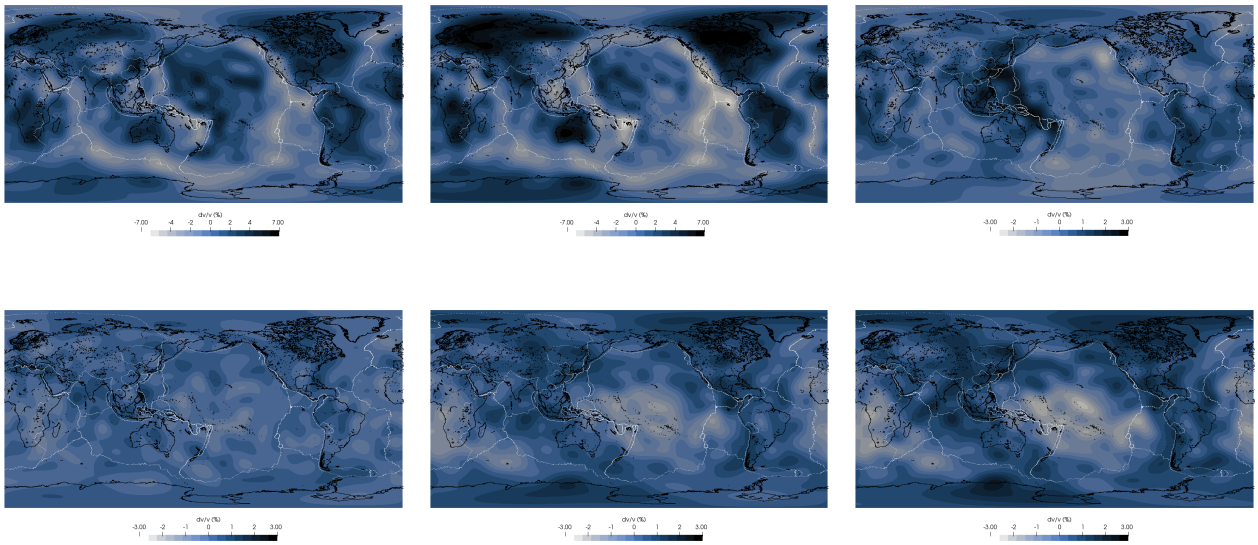
Figure 6. Maps of shear-velocity heterogeneity at, from top to bottom, 100 km, 600 km, 1500 km and 2800 km depth for, from left to right, a total number of unknowns equal to $N = 5000$, $N = 8000$ and $N = 11\,000$ (eq. 5). Model S40RTS has 8000 unknowns.

Taken from Ritsema et al (2011) [264]

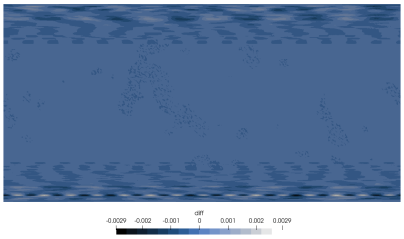
Comparison with SubMachine - S20RTS



Signal as obtained from SubMachine for various depths, using the Oslo colour scale, equidistant cylindrical map projection, 16 discrete colours.

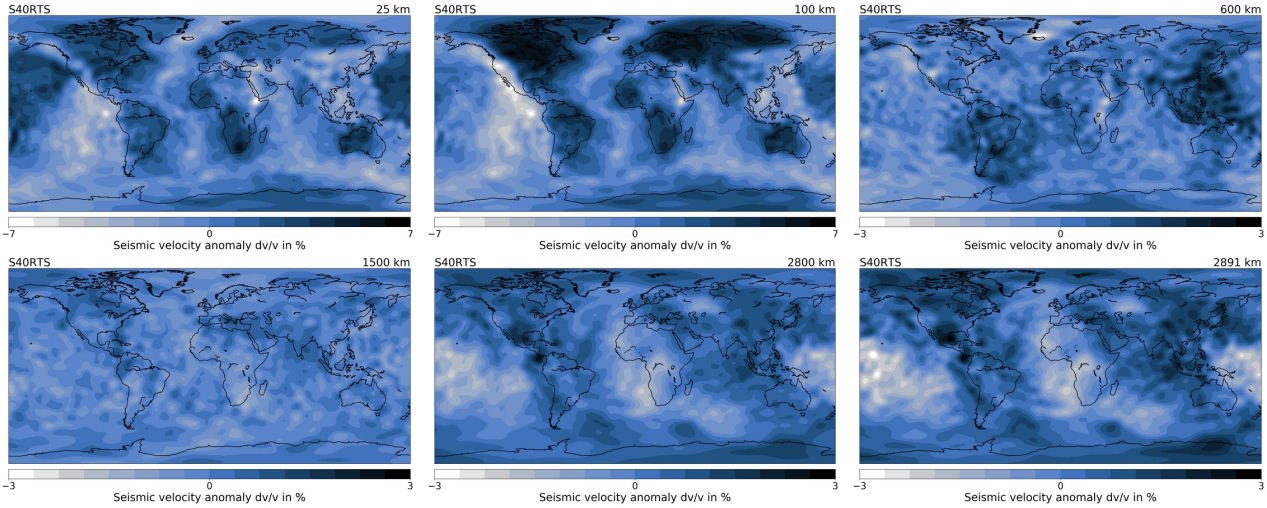


Results obtained with Fieldstone. Coastlines and plate boundaries are obtained from Stone 69.

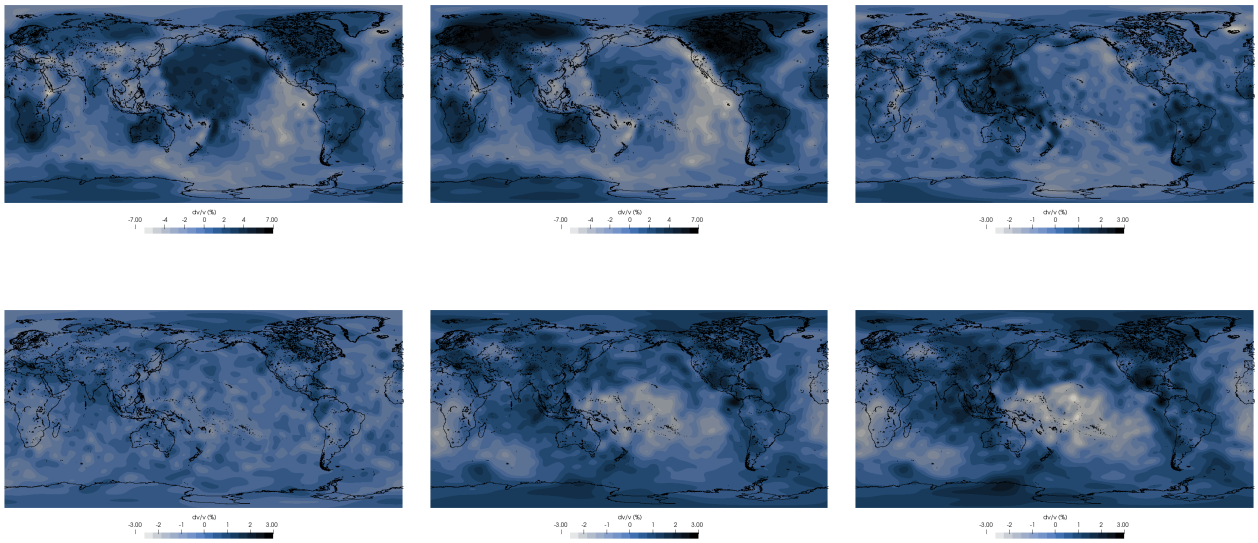


Difference between Ritsema's code results and fieldstone results at 600km depth.

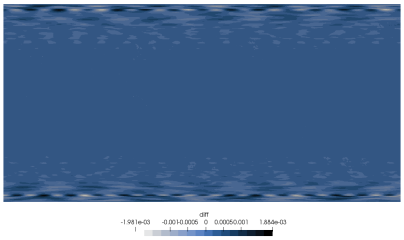
Comparison with SubMachine - S40RTS



Signal as obtained from SubMachine for various depths, using the Oslo colour scale, equidistant cylindrical map projection, 16 discrete colours.



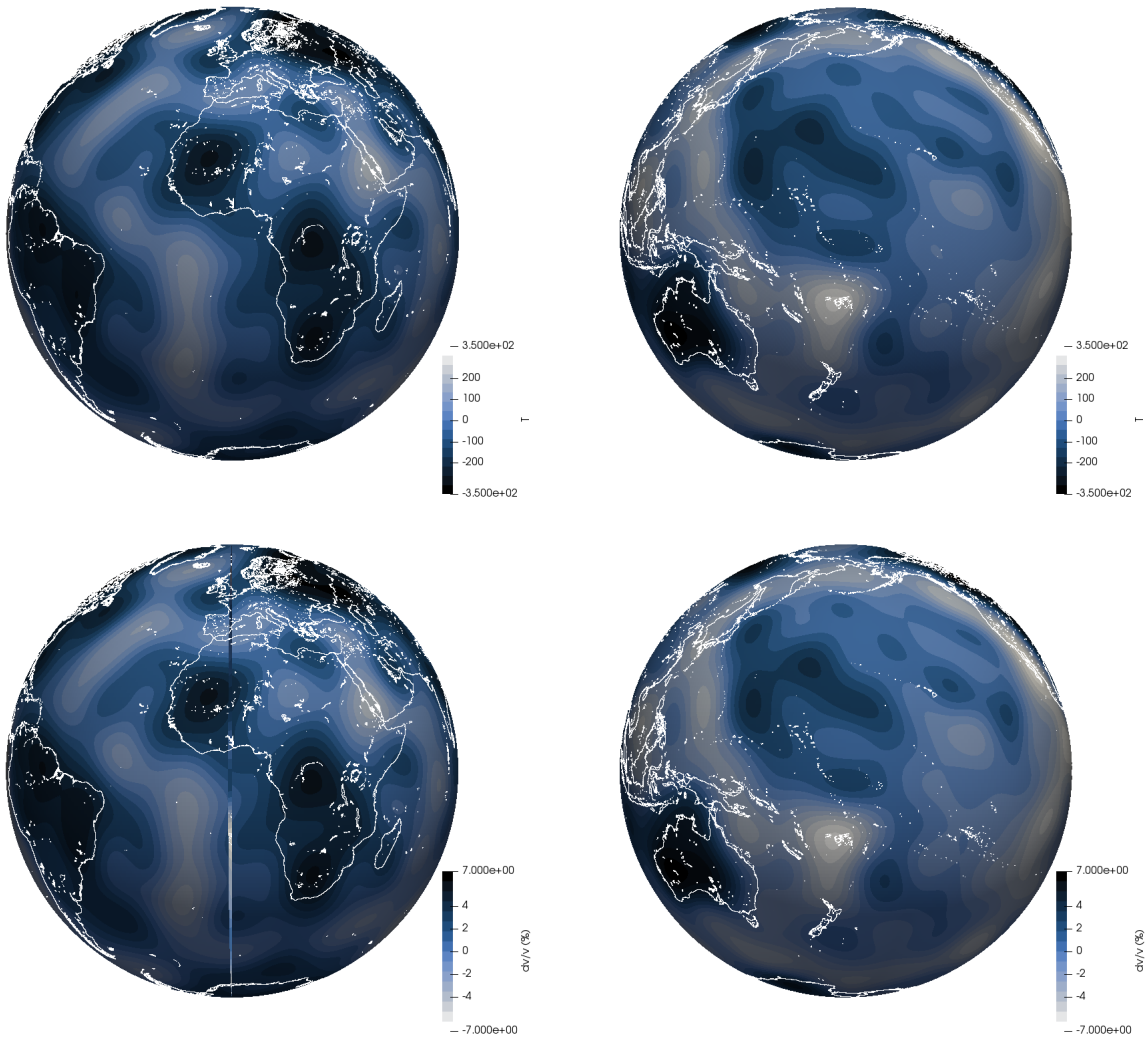
Results obtained with Fieldstone. Coastlines and plate boundaries are obtained from Stone 69.



Difference between Ritsema's code results and fieldstone results at 600km depth.

A look at S40RTS in Aspect As mentioned earlier ASPECT uses the same *SX0RTS.sph* files to arrive at dv/v but it goes and extra step and converts this quantity first to $d\rho/\rho$ and then to a temperature anomaly, which is then added to a background temperature. This is all happening in */source/initial_temperature/S40RTS_perturbation.cc*. In order to be able to visualise the temperature anomaly, I edited the `initial_temperature` function so that it returns only the `temperature_perturbation` quantity and not `background_temperature + temperature_perturbation`. I then proceeded to run the *S20RTS.prm* cookbook with `Initial global refinement` set to 4 and looked at the temperature field which is then the temperature anomaly (I have also switched the Stokes and advection solvers off).

In the input file `Vs to density scaling` is set to a constant value of 0.15 while the `Thermal expansion coefficient` is set to 3e-5, so that δT must be divided by 0.15/3e-5=5000 to obtain the original dv/v. This means that the range $[-7:7]$ for dv/v becomes $[-350:350]$ for T:



Top row is ASPECT , bottom row is fieldstone, depth of 100km.

There is a good (but not perfect) agreement between ASPECT and fieldstone...

Stone 86

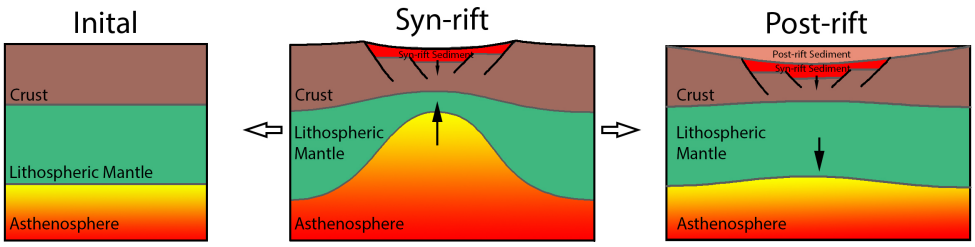
Thermal re-equilibrium of the Lith. Mantle during post-rift stage ($Q_1 \times P_0$)

FEM

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_86`

This stone was developed with input from D. Bonté.

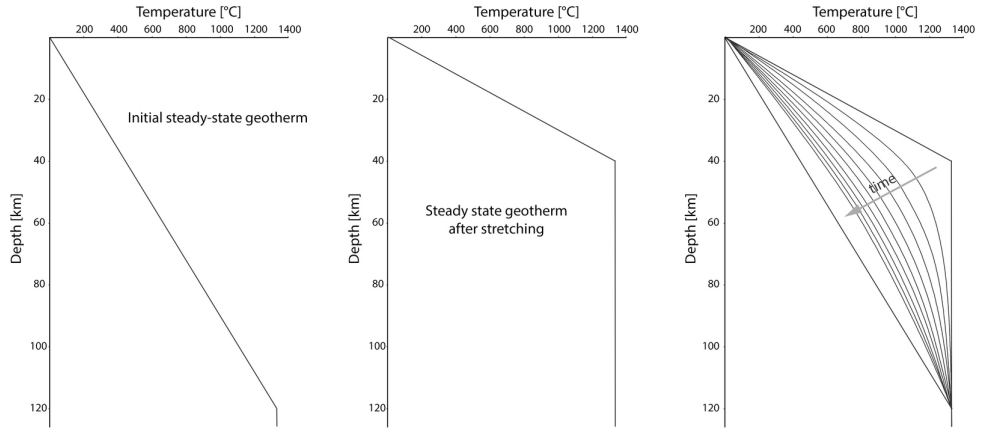
The rifting of the lithosphere is defined by 2 major steps and relates to the sedimentary records in the basins. These two steps are the syn-rift and post-rift phases:



Phases of a rift evolution (from MSc course of M. Seranne, 2013).

During the syn-rift phase, the extension is active and thinning down both the Crust and Lithospheric Mantle. Also, space is created at the surface during the deformation of the top of the crust, allowing sedimentation to occur. During the post-rift phase, the extension is no longer active and the evolution is on two fronts: (1) the thermal re-equilibration of the Lithospheric Mantle that creates a re-thickening of the Lithospheric Mantle and (2) a post-rift sedimentation in relation with the isostatic re-equilibration that follows the change in thickness of the Lithospheric Mantle.

In 1978, McKenzie presented a quantitative model [229] that allows the understanding of the lithospheric evolution related to the creation of a rifting. The model of McKenzie considered two steps. The first step (syn-rift) was considered as an instantaneous and uniform extension of the lithosphere and crust. The second step (post-rift) is the thermal re-equilibration of the thinned down lithospheric mantle, considering that (a) the temperature at the end of the pre-rift stage is steady state and (b) the composition of the lithospheric mantle and underlying Asthenosphere is equivalent. For the purpose of thermal modelling, the initial temperature is defined by a fixed temperature at the surface and at the base of the Lithosphere, and the radiogenic heat production is neglected.

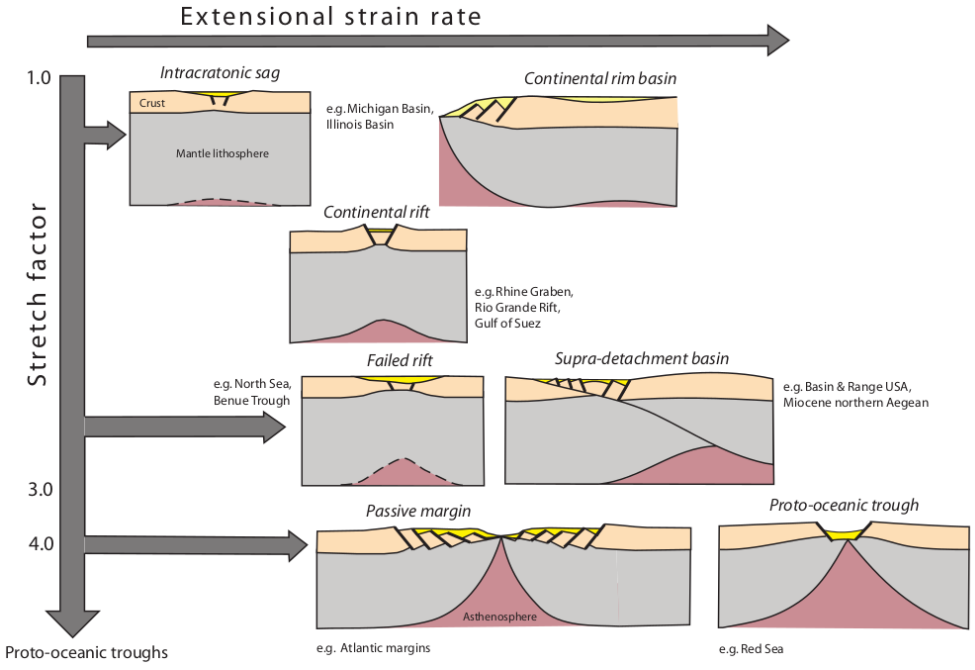


Thermal evolution during rifting. Figure by D. Bonté, after Fig. 1 of McKenzie (1987) [229].

Since this early work, the definition of the rifting has been refined with the addition of the sediments at the top of the system (that have a lower thermal conductivity than the crust), the addition of heat production (especially in the upper crust and the sediments), and the consideration of a non-uniform extension [270].

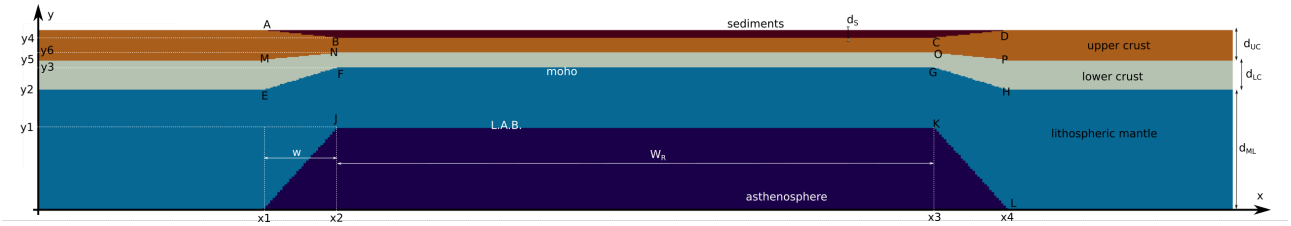
The aim of the modelling to be developed here is to observe and characterize the thermal evolution (recovery) during the post-rift sequence in two dimension as the lateral heat conduction is playing a role in the re-equilibration of the lithospheric Mantle. To obtain an even more accurate modelling outcome, two elements have been included in the modelling: the heat production and sediment. The heat production is commonly associated with ∼40% of the surface heat flow and is mostly located in the upper crust. The sediments, with a lower thermal conductivity than the crystalline basement, provide an insulating blanket to the system.

The evolution and timing of the post-rift thermal recovery of the Lithospheric Mantle is to be considered primarily against the intensity of the perturbation that occurs during the syn-rift phase. This perturbation of the lithosphere is commonly described by the β-factor (or stretch factor), which considers the vertical thinning during a syn-rifting phase. The β-factor defines the intensity of the rifting and is expressed as the initial thickness divided by the final crustal thickness. The higher the β-factor, the more developed the rifting is with the oceanisation occurring for a value of the β-factor beyond 4.



Basins of the rift–drift suite in terms of increasing stretch factor and extensional strain rate. [4].

The code solves the diffusion equation in a 2D Cartesian domain. There are five materials in the domain: the sediments (#1), the upper crust (UC, #2), the lower crust (LC, #3), the mantle lithosphere (ML, #4) and the asthenosphere (#5):



Points A..P are used in the code to build the geometry.

Each material is characterised by its density, its heat capacity, its heat conductivity and its radiogenic heat production coefficient.

material	$\rho(\mathrm{kg\,m^{-3}})$	C_p	k	$H(10^{-6}\mathrm{W\,m^{-3}})$
sediments	2100	790	2	0.4-1.8
upper crust	2900	1100	2.6	1.55
lower crust	2900	1100	2.6	0.5
lithospheric mantle	3400	1260	3	0
asthenosphere	3400	1260	3	0

These are the inputs of the code for the geometry:

- the domain size L_x and L_y
- the thickness of sediments d_S
- the thickness of the upper crust d_{UC}
- the thickness of the lower crust d_{LC}
- the width of the rift W_R
- the width of the taper on each side w
- the three β values:

$$\beta_{UC} = \frac{AM}{BN} = \frac{d_{UC}}{BN} \qquad \beta_{LC} = \frac{ME}{NF} = \frac{d_{LC}}{NF} \qquad \beta_{ML} = \frac{EI}{FJ} = \frac{d_{ML}}{FJ}$$

Note that the thickness of the lithospheric mantle d_{LM} is simply $d_{LM} = L_y - d_{UC} - d_{LC}$.

This translates as follows in the code:

```
Lx=800e3
Ly=120e3
WR=400e3
w=50e3

d_S=5e3
d_UC=20e3
d_LC=20e3
d_LM=Ly-d_UC-d_LC

beta_UC=2
beta_LC=2.2
beta_LM=2.75
```

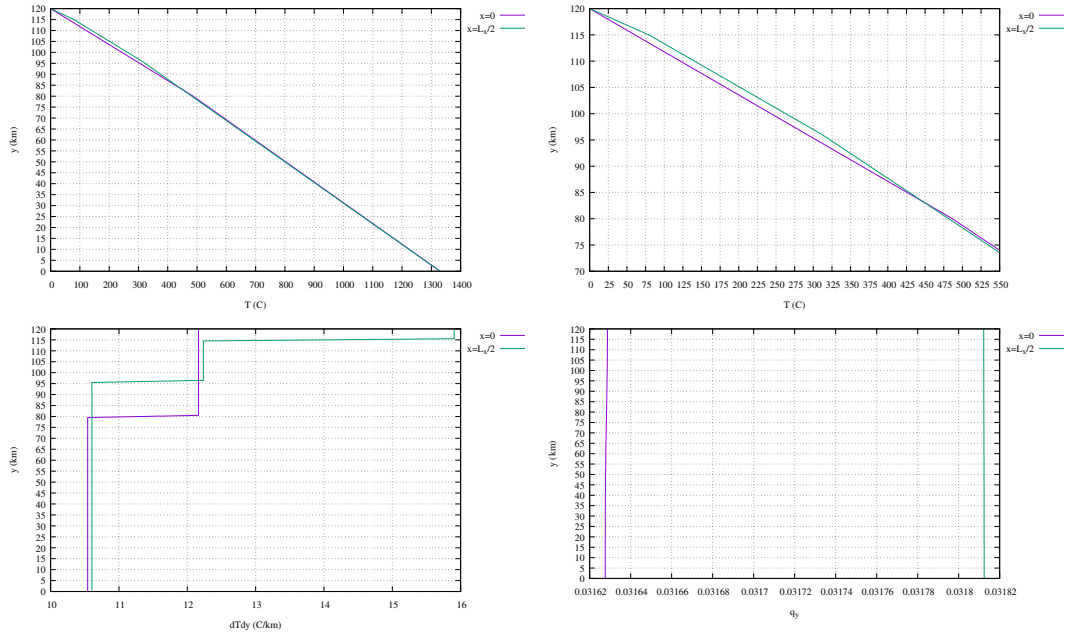
Concerning the temperature setup, we need to define 5 temperature values: the temperature at the surface, the temperature at the base of the upper crust, the temperature at the moho, and the temperature at the LAB.

```
T_surface = 0
T_sediments_base = 200
T_uppercrust_base = 310
T_moho = 550
T_lab = 1330
```

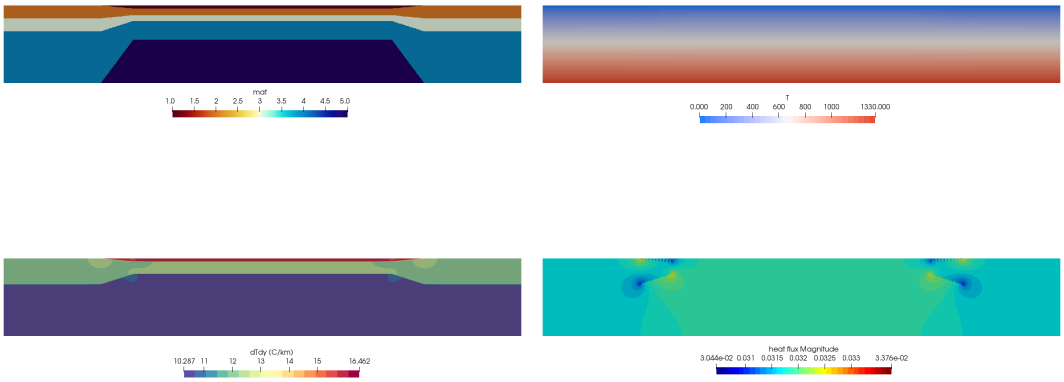
Steady state If we are only interested in the steady state solution, the equation to solve is then simply

$$\vec{\nabla} \cdot k\vec{\nabla}T + H = 0$$

In this case the initial temperature field is irrelevant and so are the heat capacity and density values.



top row: Temperature profile at steady state. Bottom row: vertical temperature gradient and vertical component of the heat flux.



ToDo: implement heat source !

Stone 87

Newton method for power-law rheologies ($Q_2 \times Q_1$)



2D
box
Q2Q1
sparse_matrix
analytical_benchmark
numerical_benchmark
error_convergence
stokes_equation
nonlinear
viscous_rheology
residual
paraview_output
pressure_scaling
pressure_normalisation
pressure_bc
defect_picard

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_87`

This stone was developed in collaboration with Riad Hassani.

The power law rheology

In what follows the viscosity is assumed to be of the power law type, i.e.

$$\eta(\dot{\boldsymbol{\varepsilon}}) = \beta \dot{\varepsilon}_e^{\frac{1}{n}-1}$$

where β is a scalar, n is a small number (but not necessarily an integer) and $\dot{\varepsilon}_e$ is the effective strain rate defined in Eq. (4.73), i.e.

$$\dot{\varepsilon}_e = \sqrt{\frac{1}{2}(\dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2) + \dot{\varepsilon}_{xy}^2}$$

We will also consider the generalised power-law rheology:

$$\eta_{\dot{\gamma}}(\dot{\boldsymbol{\varepsilon}}) = \beta(\dot{\varepsilon}_e^2 + \dot{\gamma}^2)^{\frac{1-n}{2n}}$$

This formulation has the advantage that viscosity does not become infinite when the strain rate becomes zero. Finally, to simplify notations we define $\alpha = \frac{1}{n} - 1$ so that

$$\boxed{\eta_{\dot{\gamma}}(\dot{\boldsymbol{\varepsilon}}) = \beta(\dot{\varepsilon}_e^2 + \dot{\gamma}^2)^{\alpha/2}} \qquad \text{or simply} \qquad \boxed{\eta(\dot{\boldsymbol{\varepsilon}}) = \beta \dot{\varepsilon}_e^{\alpha}}$$

Relevant Literature: Dynamics of strongly time-dependent convection with non-Newtonian temperature-dependent viscosity, Larsen, Yuen, Malevsky, and Smedsmo [206] (1996).

Newton-Raphson method for single-valued functions

Newton gave a version of the method in 1669. Raphson generalized and presented the method in 1690. Both mathematicians used the same concept, and both algorithms gave the same numerical results, which is why the method is often referred to as the Newton-Raphson method.

In numerical analysis, the Newton's method (also Newton-Raphson method) is an iterative root-finding algorithm. The most basic version for a function $f(x)$ is as follows[1]:

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$

where we assume that the derivative f' exists and we start the iterations with a guess x_0. If the function satisfies sufficient assumptions and the initial guess is close to the real solution then the method converges to the root. Note that if a stationary point of the function is encountered, i.e. the derivative is zero, then the method will terminate due to division by zero.

Newton-Raphson method for systems of equations

Let us consider the following system of N equations.

$$\boldsymbol{A} \cdot \vec{X} = \vec{b}$$

Solving this system is equivalent to finding the root of

$$\vec{R}(\vec{X}) = \boldsymbol{A} \cdot \vec{X} - \vec{b}$$

i.e. finding the zeroes of the continuously differentiable function $\vec{R} : \mathbb{R}^N \to \mathbb{R}^N$. In this case, the Newton algorithm is written as a function of the $N \times N$ Jacobian matrix $\boldsymbol{J}_R$:

$$\vec{X}^{k+1} = \vec{X}^k - \boldsymbol{J}_R^{-1}(\vec{X}^k) \cdot \vec{R}(\vec{X}^k) \tag{87.1}$$

or,

$$\boldsymbol{J}_R(\vec{X}^k) \cdot (\vec{X}^{k+1} - \vec{X}^k) = -\vec{R}(\vec{X}^k) \tag{87.2}$$

where the Jacobian matrix is defined as follows[2]:

$$\boldsymbol{J}_R = \begin{pmatrix} \frac{\partial R_1}{\partial X_1} & \cdots & \frac{\partial R_1}{\partial X_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial R_N}{\partial X_1} & \cdots & \frac{\partial R_N}{\partial X_N} \end{pmatrix}$$

The super simple / no questions asked approach

We have to solve

$$\boldsymbol{A}(\vec{X}) \cdot \vec{\mathcal{X}} = \vec{b}$$

where the matrix $\boldsymbol{A}(\vec{\mathcal{X}})$ comes from the discretization of the incompressible Stokes equations and the vector $\vec{b}$ corresponds to body forces, surface forces and boundary conditions. We assume here for simplicity that $\vec{b}$

[1] https://en.wikipedia.org/wiki/Newtons_method

[2] https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant

is independent of the vector of unknowns $\vec{X}$, which is made of $\vec{\mathcal{V}}$ and $\vec{\mathcal{P}}$. The dependence of $\boldsymbol{A}$ on $\vec{\mathcal{X}}$ comes from the dependence of the viscosity on strain rate (and therefore velocity) and pressure (although in this particular case of the power law rheology pressure does not enter the equations).

The matrix $\boldsymbol{A}(\vec{\mathcal{X}})$ has the following structure:

$$\boldsymbol{A}(\vec{\mathcal{X}}) = \begin{pmatrix} \mathbb{K}(\vec{\mathcal{X}}) & \mathbb{G} \\ \mathbb{G}^T & 0 \end{pmatrix} \tag{87.3}$$

and the discretised Stokes system is then

$$\begin{pmatrix} \mathbb{K}(\vec{\mathcal{V}}) & \mathbb{G} \\ \mathbb{G}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{\mathcal{V}} \\ \vec{\mathcal{P}} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ \vec{h} \end{pmatrix} \tag{87.4}$$

The discrete residual is defined by

$$\begin{aligned} \vec{\mathcal{R}}(\vec{\mathcal{X}}) &= \boldsymbol{A}(\vec{\mathcal{X}}) \cdot \vec{\mathcal{X}} - \vec{b} && (87.5) \\ &= \begin{pmatrix} \mathbb{K}(\vec{\mathcal{V}}) \cdot \vec{\mathcal{V}} + \mathbb{G} \cdot \vec{\mathcal{P}} - \vec{f} \\ \mathbb{G}^T \cdot \vec{\mathcal{V}} - \vec{h} \end{pmatrix} && (87.6) \\ &= \begin{pmatrix} \vec{R}_{\mathcal{V}} \\ \vec{R}_{\mathcal{P}} \end{pmatrix} && (87.7) \end{aligned}$$

- Standard Picard iterations are as follows:

$$\boxed{\boldsymbol{A}(\vec{\mathcal{X}}^k) \cdot \vec{\mathcal{X}}^{k+1} = \vec{b}} \tag{87.8}$$

- Defect correction Picard iterations. We can use Eq. (87.5) to write $\vec{b} = \boldsymbol{A}(\vec{\mathcal{X}}^k) \cdot \vec{\mathcal{X}}^k - \vec{R}(\vec{\mathcal{X}}^k)$ and then replace $\vec{b}$ in Eq. (87.8):

$$\boldsymbol{A}(\vec{\mathcal{X}}^k) \cdot \vec{\mathcal{X}}^{k+1} = \boldsymbol{A}(\vec{\mathcal{X}}^k) \cdot \vec{\mathcal{X}}^k - \vec{R}(\vec{\mathcal{X}}^k) \tag{87.9}$$

  and finally, defining $\delta\vec{\mathcal{X}}^k = \vec{\mathcal{X}}^{k+1} - \vec{\mathcal{X}}^k$, we can write

$$\boxed{\boldsymbol{A}(\vec{X}^k) \cdot \delta\vec{X}^k = -\vec{R}(\vec{X}^k)}$$

  This approach must be supplemented with

$$\vec{\mathcal{X}}^{k+1} = \vec{\mathcal{X}}^k + \delta\vec{\mathcal{X}}^k$$

  Remark. *As mentioned in the Petsc manual*[3]*: The defect correction form of the Picard iteration converges much more generally when inexact linear solvers are used then the direct Picard iteration* $A(x^n)x^{n+1} = b(x^n)$*. Note that when an exact solver is used this corresponds to the "classic" Picard* $A(x^n)x^{n+1} = b(x^n)$ *iteration.*

- Newton iterations. We start from

$$\vec{\mathcal{R}}(\vec{\mathcal{X}}) = \boldsymbol{A}(\vec{\mathcal{X}}) \cdot \vec{\mathcal{X}} - \vec{b}$$

  and apply the methodology of Eq. (87.2) and a Newton iteration then consists of solving for $\delta\vec{\mathcal{X}}^k$ the following linear system

$$\boxed{\boldsymbol{J}_R(\vec{\mathcal{X}}^k) \cdot \delta\vec{\mathcal{X}}^k = -\vec{R}(\vec{\mathcal{X}}^k)}$$

  and updating $\vec{\mathcal{X}}^k$:

$$\vec{\mathcal{X}}^{k+1} = \vec{\mathcal{X}}^k + \delta\vec{\mathcal{X}}^k \tag{87.10}$$

  Note that we recover the defect correction Picard when setting $\boldsymbol{J}_R \to \boldsymbol{A}$. Also, the update of Eq. (87.10) can be rewritten

$$\vec{\mathcal{X}}^{k+1} = \vec{\mathcal{X}}^k + \alpha^k\delta\vec{\mathcal{X}}^k$$

[3] `https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/SNES/SNESSetPicard.html`

where α is a step length parameter that can be determined, for example, using a line search.

Deriving the exact expression for $\boldsymbol{J}_R$ is actually where the difficulty really lies. From the structure of $\boldsymbol{A}$, we expect the Jacobian matrix to take the form

$$\boldsymbol{J}_R = \begin{pmatrix} \mathbb{J}_{vv} & \mathbb{J}_{vp} \\ \mathbb{J}_{pv} & 0 \end{pmatrix}$$

The term $\mathbb{J}_{vv}$ corresponds to the derivative of $\vec{R}_{\mathcal{V}} = \mathbb{K}(\vec{\mathcal{V}}) \cdot \vec{\mathcal{V}} + \mathbb{G} \cdot \vec{\mathcal{P}} - \vec{f}$ with respect to $\vec{\mathcal{V}}$ For a power law rheology, it can be written[4]

$$\mathbb{J}_{vv}(\vec{\mathcal{X}}^k) = \mathbb{K}_0(\vec{\mathcal{V}}^k) + \mathbb{K}_1(\vec{\mathcal{V}}^k)$$

where $\mathbb{K}_0$ is the standard matrix obtained from

$$\mathbb{K}_0 = \int_\Omega \eta(\dot{\boldsymbol{\varepsilon}}^k) \boldsymbol{B}^T \cdot \boldsymbol{C} \cdot \boldsymbol{B} dV \tag{87.11}$$

$$\mathbb{K}_1 = \tag{87.12}$$

The term $\mathbb{J}_{vp}$ corresponds to the derivative of $\vec{R}_{\mathcal{V}} = \mathbb{K}(\vec{\mathcal{V}}) \cdot \vec{\mathcal{V}} + \mathbb{G} \cdot \vec{\mathcal{P}} - \vec{f}$ with respect to $\vec{\mathcal{P}}$. Since the viscosity does not depend on pressure, then we have $\mathbb{J}_{vp} = \mathbb{G}$. Likewise $\mathbb{J}_{pv}$ corresponds to the derivative of $\vec{R}_{\mathcal{V}} = \mathbb{G}^T \cdot \vec{\mathcal{V}} - \vec{h}$ with respect to $\vec{\mathcal{V}}$ which yields $\mathbb{J}_{vp} = \mathbb{G}^T$. Finally

$$\boldsymbol{J}_R = \begin{pmatrix} \mathbb{K}_0(\vec{\mathcal{V}}) + \mathbb{K}_1(\vec{\mathcal{V}}) & \mathbb{G} \\ \mathbb{G}^T & 0 \end{pmatrix}$$

This justifies why we have chosen a power law rheology to start with the implementation of the Newton-Raphson method: the modifications to the FE matrix are small and limited to the viscous block and the rhs is simply the previous residual.

Implementation details

The code is a $Q_2 \times Q_1$ element code which solves a few nonlinear problems, some of which having analytical solutions.

It is established that the Newton method converges only if the initial guess is 'close enough' to the real solution. It is then customary of carrying out a few Picard iterations before switching over to the Newton method. This is why we define the parameter $\theta \in [0, 1]$ in the code such that

$$\mathbb{J}_{vv} = \mathbb{K}_0(\vec{\mathcal{V}}) + \theta \mathbb{K}_1(\vec{\mathcal{V}})$$

Indeed, if $\theta = 0$ we recover the defect correction Picard method and if $\theta = 1$ we recover the standard Newton method. Moreover, the parameter θ can be adapted according to the norm of the residual, for example by choosing

$$\theta = 1 - \frac{||\vec{R}^k||}{||\vec{R}^p||} \tag{87.13}$$

or

$$\theta = 1 - \sqrt{\frac{||\vec{R}^k||}{||\vec{R}^p||}} \tag{87.14}$$

where $\vec{R}^p$ is the residual at the iteration p before which Newton is switched on. As $||\vec{R}^k||$ becomes small, $\theta \to 1$. This then ensures a smooth transition between both methods. The version with the square root makes the transition even more smooth. In the code we then defined the `theta_method` parameter:

- `theta_method`=1: $\theta = 1$.
- `theta_method`=2: θ as given by Eq. (87.13).

[4] Skipping a lot of steps for now

- `theta_method`=3: θ as given by Eq. (87.14).

Concerning the defect Picard method, two main modifications are needed: build a different rhs, and adapt the boundary conditions. The (elemental) rhs is split across two arrays f_el and h_el. In a standard code f_el receives the contribution of the buoyancy forces at each quadrature point:

```
for i in range(0,mV):
    f_el[ndofV*i+0]+=NNNV[i]*jcob*weightq*gx(xq,yq)*rho
    f_el[ndofV*i+1]+=NNNV[i]*jcob*weightq*gy(xq,yq)*rho
```

This is in fact the $\vec{b}$ term of Eq.(**??**). Also the array h_el is zero before boundary conditions are applied so there is no direct contribution to it inside the loop over quadrature points.

For each element we store the velocity and pressure field in dedicated arrays V_el and p_el:

```
V_el=np.zeros((mV*ndofV),dtype=np.float64)
P_el=np.zeros((mP*ndofP),dtype=np.float64)

for i in range(0,mV):
    V_el[2*i+0]=solution[2*iconV[i,iel]+0]
    V_el[2*i+1]=solution[2*iconV[i,iel]+1]

for i in range(0,mP):
    P_el[i]=p[iconP[i,iel]]
```

and we then proceed to add the necessary terms to both f_el and h_el:

```
f_el-=K_el0.dot(V_el)+G_el.dot(P_el)
h_el-=G_el.T.dot(V_el)
```

The second modification concerns the boundary conditions. As per usual in our codes, there is an array bc_val which contains the prescribed value of the boundary condition. It is then necessary to transfer it to the global solution vector before iterations are carried out:

```
solution[0:NfemV]=bc_val[0:NfemV]
```

Since the unknowns of the system are successive corrections on the velocity and pressure, we therefore need to start with the known values in the solution vector. Further down, when we apply boundary conditions, we must then apply zero, so that the correction is zero where boundary conditions are applied in the domain.

We monitor three quantities during the nonlinear iterations:

- the 2-norm of nonlinear residual (i.e. the rhs)

  ```
  Rnorm=LA.norm(rhs,2)
  ```

- the 2-norm of the velocity (correction) vector

  ```
  LA.norm(dvel,2)
  ```

- the 2-norm of the pressure (correction) vector

  ```
  LA.norm(dp,2)
  ```

We then compute the three corresponding normalised quantities:

```
index_res=Rnorm/Rnorm0
index_vel=LA.norm(dvel,2)/LA.norm(vel,2)
index_p=LA.norm(dp,2)/LA.norm(p,2)
```

When all three are smaller than the nonlinear tolerance then the system is said to have converged.

```
converged=(index_res<tol_nl and index_vel<tol_nl and index_p<tol_nl)
```

Using only the normalised nonlinear residual proved problematic because it often occurred that it would drop more than six orders of magnitude over the first 2-3 nonlinear iterations.

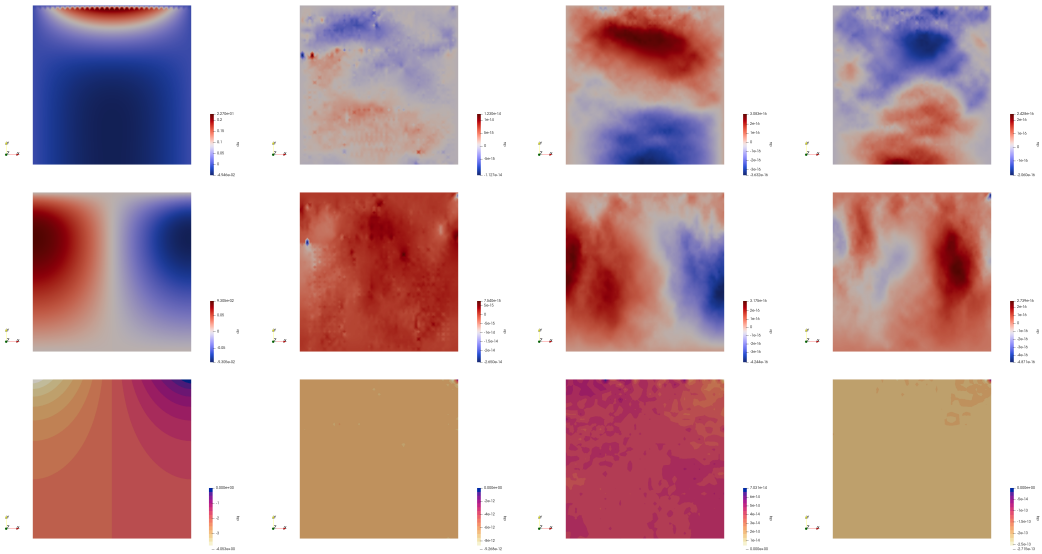
Experiment 1 - the (regularised) lid driven cavity

The domain is a unit square. Free slip boundary conditions are prescribed on the left, right and bottom boundaries, while $\vec{\mathbf{v}} = (x(1-x), 0$ is prescribed on the top. There are no buoyancy forces and the viscosity is set to $B = 1$. The pressure is normalised so as to have a zero volume average. This is a linear problem and a single Stokes solve returns the solution. Any further iteration should then not alter this solution.



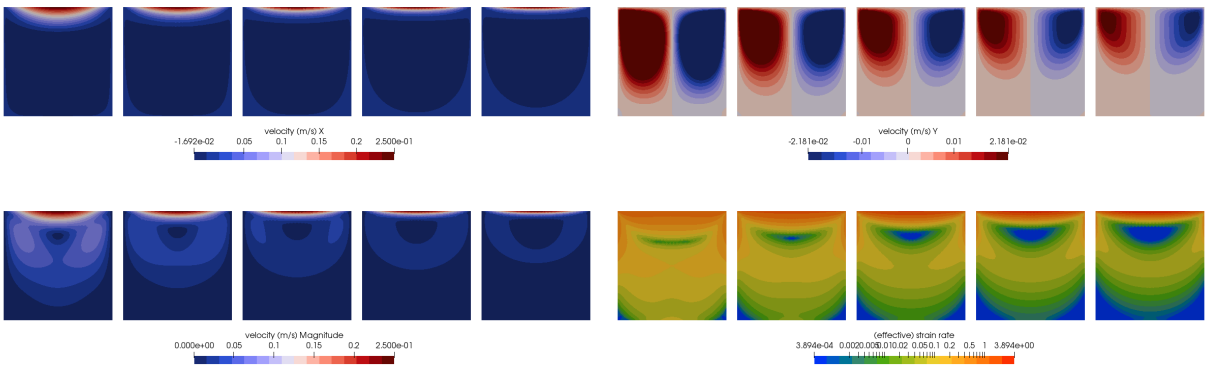
Solution as obtained on a 32×32 grid.

Four defect correction Picard iterations are carried out and the solution for each iteration is shown here under:



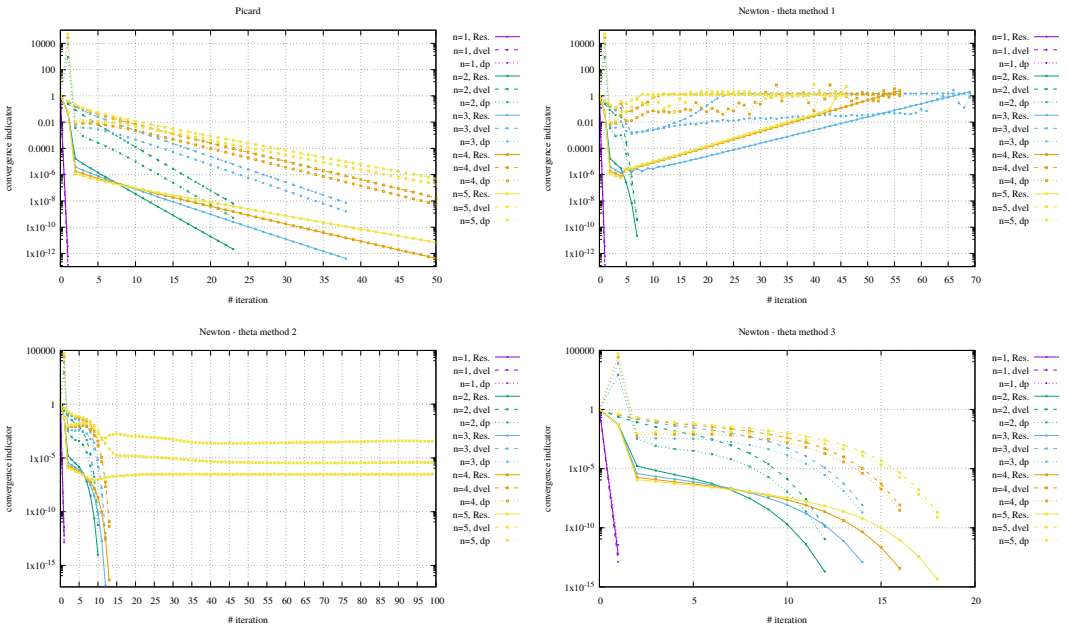
From left to right: iteration 0,1,2,3. Top to bottom: horizontali and vertical component of the velocity correction, and pressure correction, as obtained on a 32×32 grid. The fields obtained at iteration 0 are in fact the solution, and all other subsequently obtained fields are essentially zero, as expected.

This is now the same experiment as above but the viscosity is now of the power law type with $B = 1$ and $n > 1$. Also, no-slip boundary conditions are prescribed on the left, right and bottom sides. The regularisation parameter $\dot{e}$ is set to 10^{-6}.

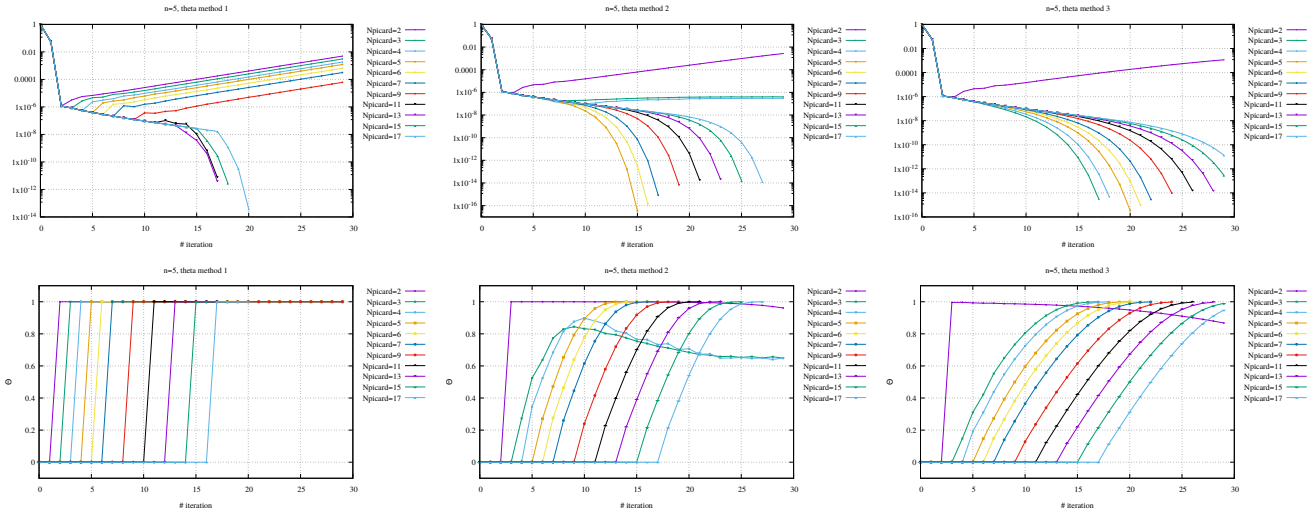




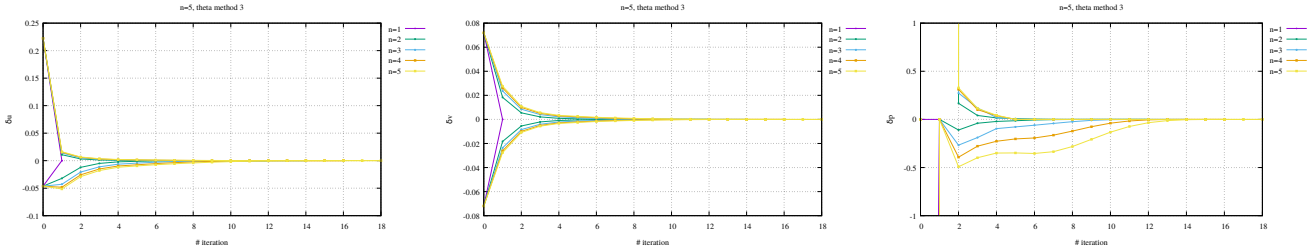
Velocity, pressure, strain rate and viscosity fields as a function of n (from left to right: 1,2,3,4,5).

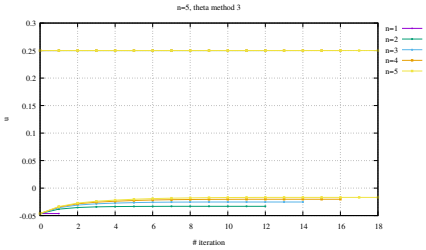


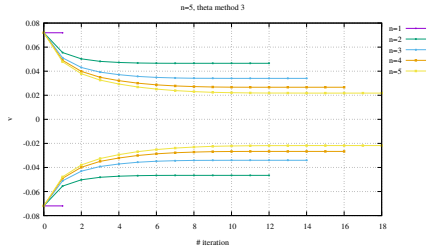
Npicard =4. $\dot{\epsilon} = 10^{-6}$

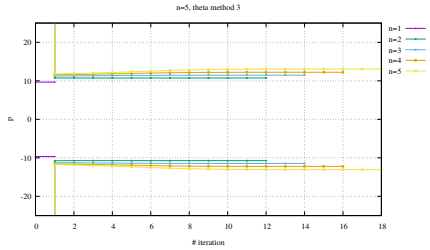


Influence of the number of Picard iterations before wtiching to Newton ones for $n = 5$.




n=5, theta method 3
u
# iteration
n=1
n=2
n=3
n=4
n=5


n=5, theta method 3
v
# iteration
n=1
n=2
n=3
n=4
n=5


n=5, theta method 3
p
# iteration
n=1
n=2
n=3
n=4
n=5

Experiment 2 - the brick

Following Christensen (1992) [71] and Ciskova et al (2002) [75] one can use the following relationship to include a form of plasticity through stress limiting:

$$\frac{\tau}{\tau_{lim}} = \left(\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_{lim}} \right)^{1/n}$$

where τ_{lim} is the yield stress and n is the power-law index defining the 'brittleness' of the material. In Ciskova et al. (2002), the authors use $n = 5$, $\dot{\varepsilon}_{lim} = 10^{-15}\text{s}^{-1}$, and $\tau_{lim} = 10^9\text{Pa}$.

In this experiment the domain is 40x10km and the resolution is 64×16 elements. The regularisation parameter is set to $\dot{e} = 10^{-20}\text{s}^{-1}$. Because we here consider a shallow upper-crustal layer, the material is characterised by a cohesion of $c = 40\text{MPa} = \tau_{lim}$. Extensional boundary conditions are applied so that the background strain rate is also 10^{-15}s^{-1}.

One can also look at the effective viscosity by setting $\tau = 2\eta_{eff}\dot{\varepsilon}_e$ so that

$$\frac{2\eta_{eff}\dot{\varepsilon}_e}{\tau_{lim}} = \left(\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_{lim}} \right)^{1/n}$$

which yields

$$\eta_{eff} = \left(\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_{lim}} \right)^{1/n} \frac{1}{2\dot{\varepsilon}_e} \tau_{lim} = \left(\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_{lim}} \right)^{1/n} \frac{\dot{\varepsilon}_{lim}}{\dot{\varepsilon}_e} \frac{\tau_{lim}}{2\dot{\varepsilon}_{lim}} = \left(\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_{lim}} \right)^{\frac{1}{n}-1} \frac{\tau_{lim}}{2\dot{\varepsilon}_{lim}}$$

Defining $\eta_{lim} = \tau_{lim}/2\dot{\varepsilon}_{lim}$, then

$$\eta_{eff} = \eta_{lim} \left(\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_{lim}} \right)^{\alpha}$$

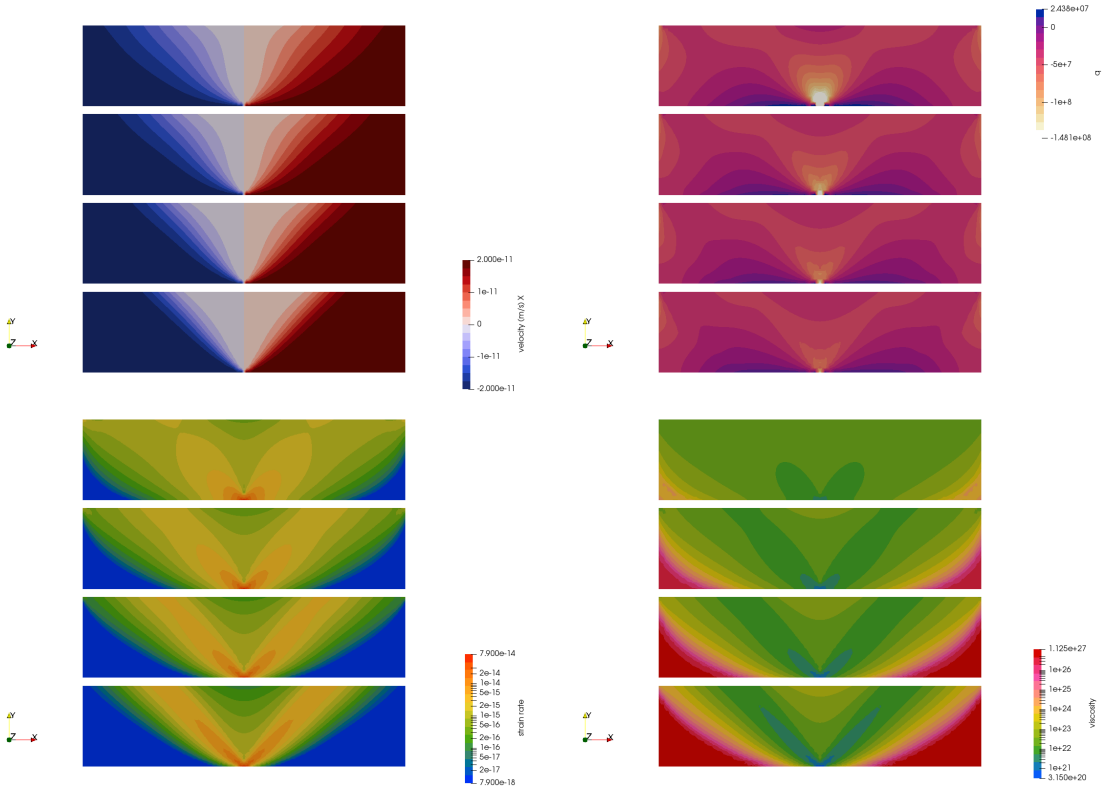
In our case, $\eta_{lim} = 4^7/2/10^{-15} = 2 \times 10^{22}$. In the present context, we then define

$$B = \frac{\eta_{lim}}{\dot{\varepsilon}_{lim}^{\alpha}}$$

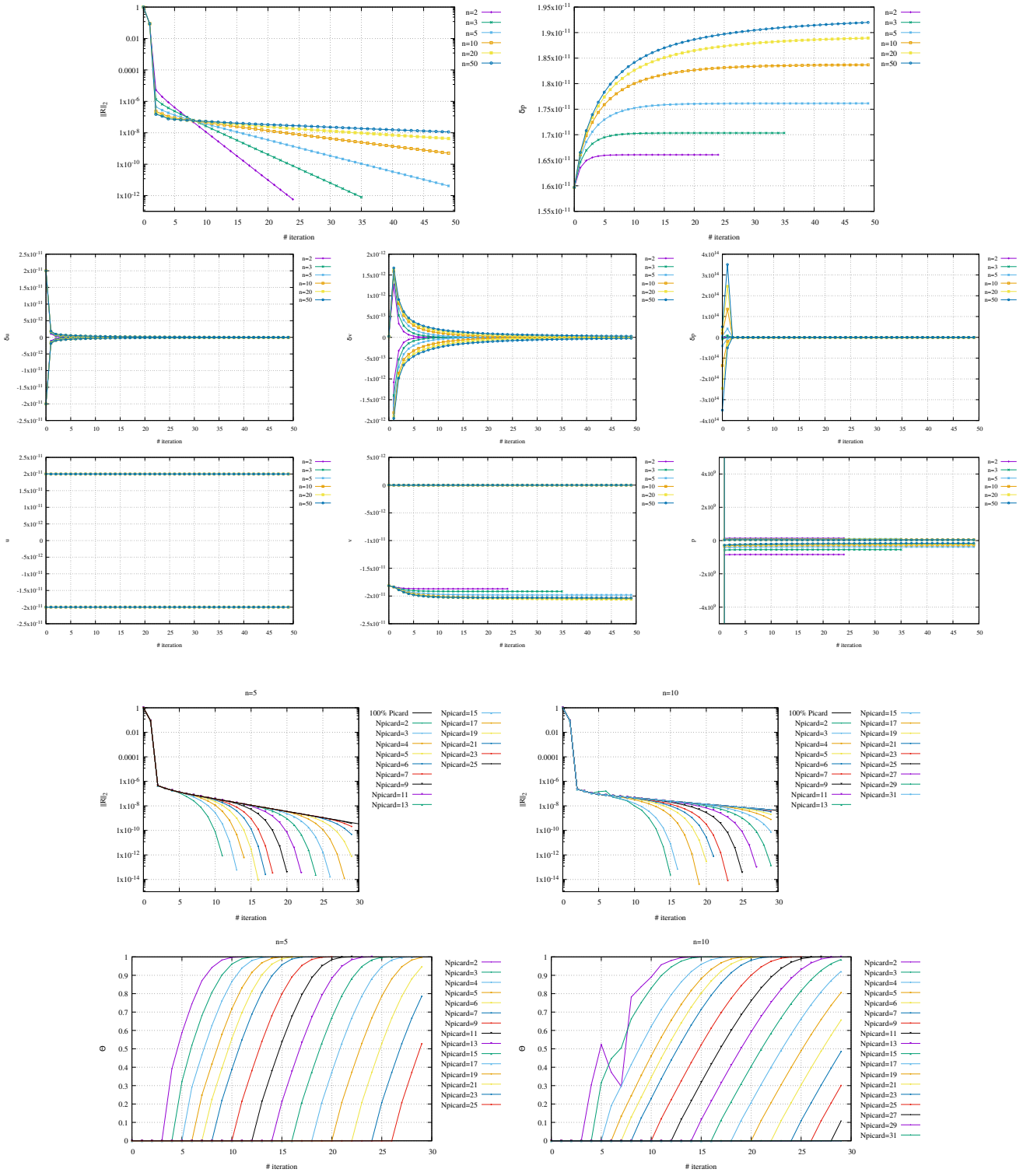
and finally

$$\eta_{eff} = B\dot{\varepsilon}_e^{\alpha} \rightarrow B(\dot{\varepsilon}_e^2 + \dot{e}^2)^{\alpha/2}$$

Boundary conditions are as follows: $\vec{\mathbf{v}} = (-u_{bc}, 0)$ is imposed on the left and the left half of the bottom side, and $\vec{\mathbf{v}} = (+u_{bc}, 0)$ is imposed on the right and the right half of the bottom side. The top is left free.



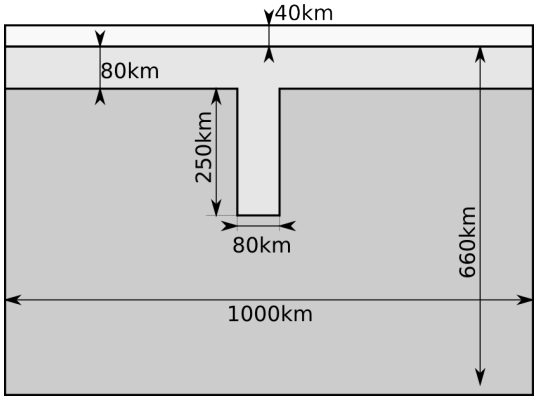
Velocity, pressure, strain rate and viscosity fields as a function of n (from top to bottom: 2,5,10,20).



Influence of the number of Picard iterations before wtiching to Newton ones for $n = 5$ (left) and $n = 10$ (right).

Experiments 3 & 4 - Slab detachment benchmark

Two materials are present in the domain: the lithosphere (mat.1) and the mantle (mat.2):



the top layer may or may not be there, depending on the chosen case, see below.

The overriding plate is 80km thick and is placed at the top of the domain. An already subducted slab (also mat 1) of 250km length hangs vertically under this plate. The mantle occupies the rest of the domain. Several experiments with increasing levels of complexity have been designed and the first two ones are carried out for now.

- case 1a: The mantle has a constant viscosity $\eta_0 = 10^{21}\mathrm{Pa\,s}$ and a density $\rho = 3150\mathrm{kg\,m^{-3}}$. The slab has a density $\rho = 3300\mathrm{kg\,m^{-3}}$ and is characterised by a power law rheology so that its effective viscosity depends on the effective strainrate $\dot{\varepsilon}_e$. We set $n_s = 4$ and $A = (2 \times 4.75 \times 10^{11})^{-n_s}$ so that

$$\eta_{eff} = \frac{1}{2} A^{-1/n_s} \dot{\varepsilon}_e^{1/n_s - 1} = \frac{1}{2}[(2 \times 4.75 \times 10^{11})^{-n_s}]^{-1/n_s} \dot{\varepsilon}_e^{1/n_s - 1} = 4.75 \times 10^{11} \dot{\varepsilon}_e^{1/n_s - 1} = B \dot{\varepsilon}_e^{\alpha} \quad (87.15)$$

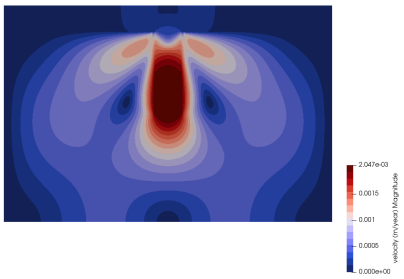
- case 1b: Both mantle and slab are have a power law rheology. The slab is the same as in case 1a, but the mantle rheology is now characterised by $A = (2 \times 4.54 \times 10^{10})^{-n_m}$ or $B = 4.54 \times 10^{10}$ and $n_m = 3$.

- case 2a: same as case 1a, but the system now has a free surface. Depending on the code you are using, a conforming mesh or a sticky air approach can be used. If the sticky air option is chosen, $40km$ of air are added at the top of the system and no boundary conditions are prescribed on the top of the domain. The choice of air viscosity is yours and you may wish to read [80]. Ideally it should be low enough so that decreasing it even more does not alter the results.

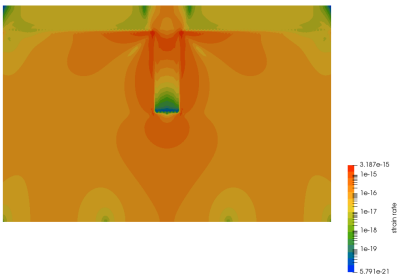
- case 2b: same as case 2a, but with nonlinear mantle of case 1b.

Boundary conditions are no-slip on the sides and free-slip on the top and bottom. Note that calculations are actually carried out with a density field to which the value 3150 has been subtracted, i.e. the lithosphere has a density of 150 while the mantle has zero density. The regularisation parameter is set to $\dot{\gamma} = 10^{-20}\mathrm{s}^{-1}$. Gravity is vertical and set to $-10\mathrm{m\,s^{-2}}$.

Case 1a (Experiment #3)

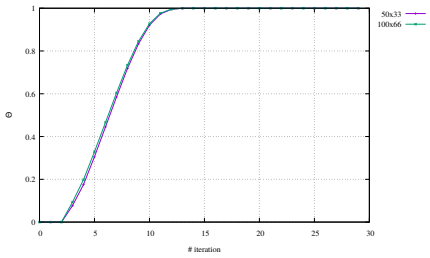


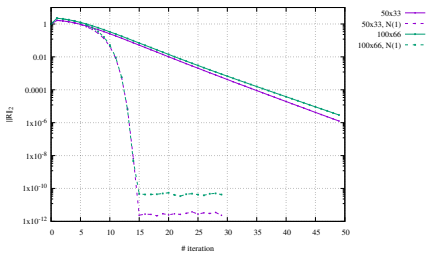


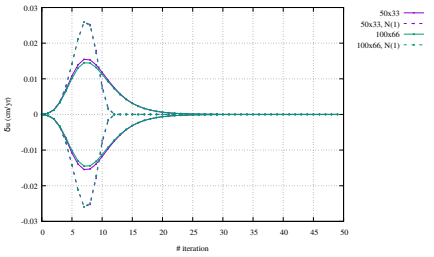


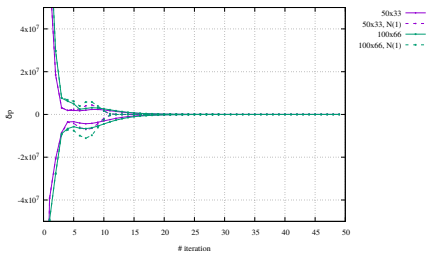


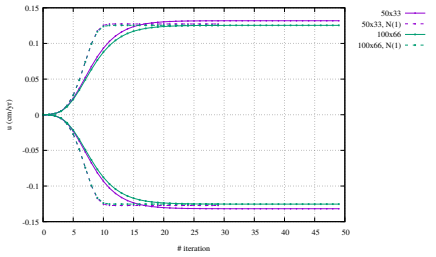


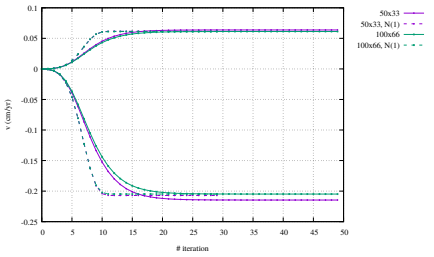


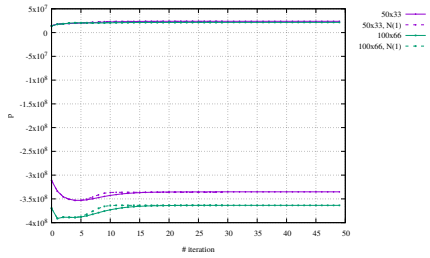


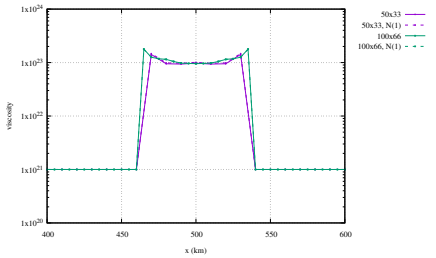


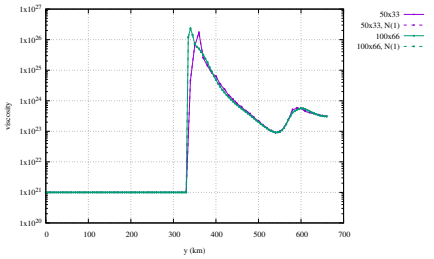


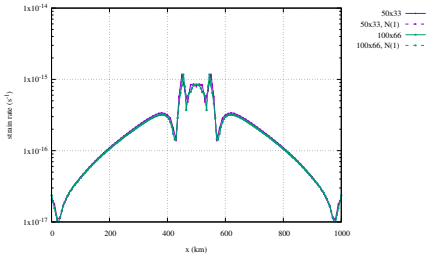


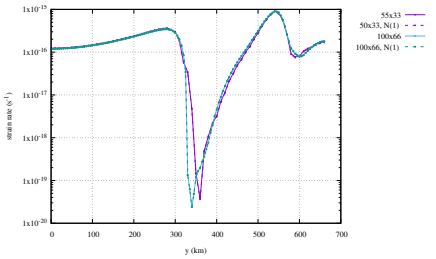


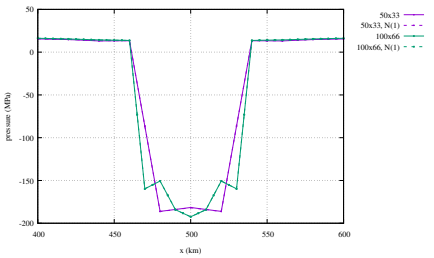


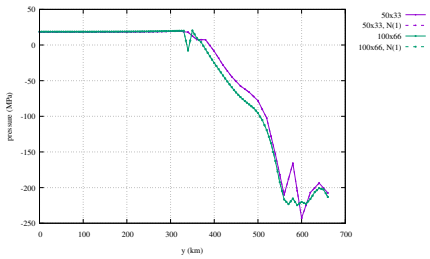




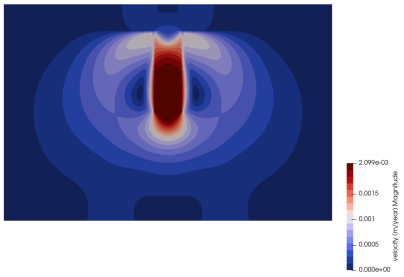




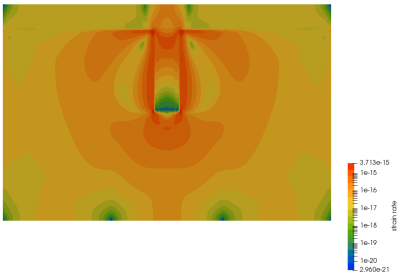


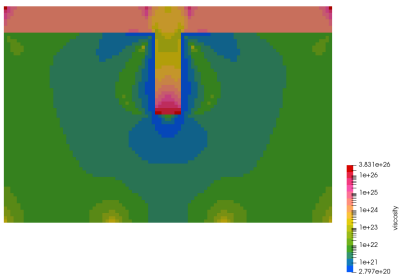


Case 1b (Experiment #4)

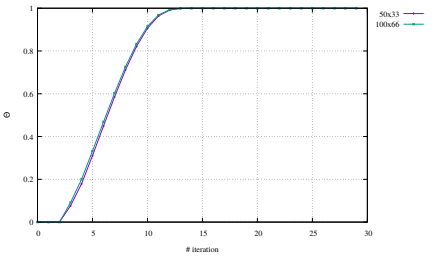


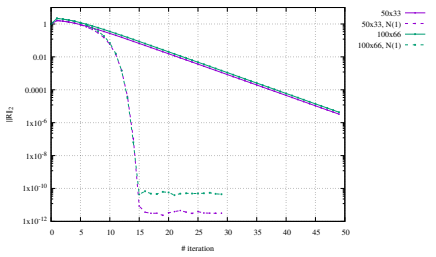


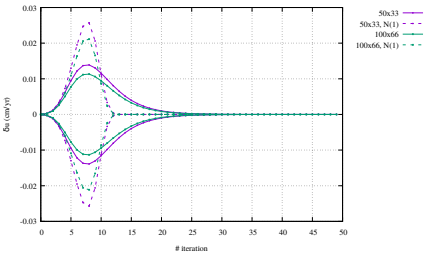


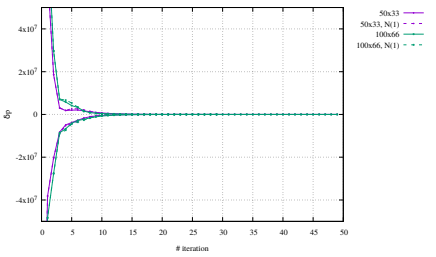


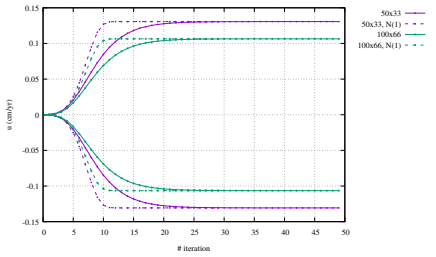


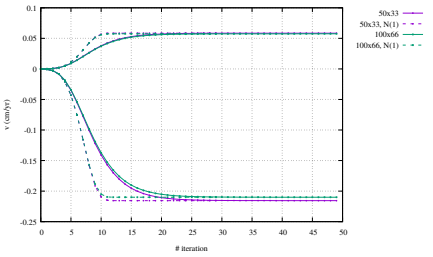


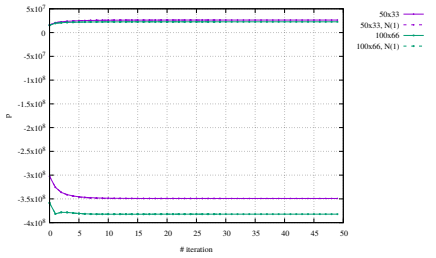


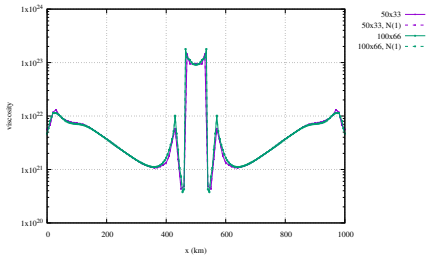


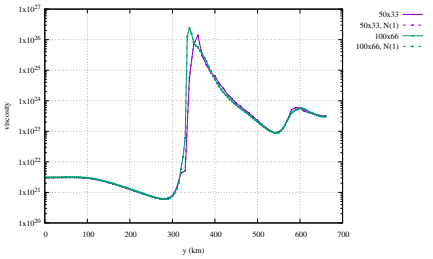


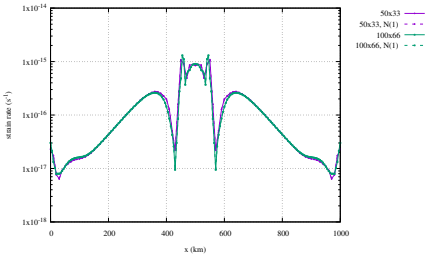


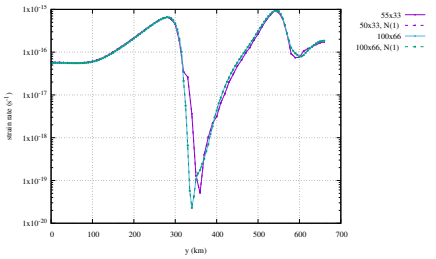


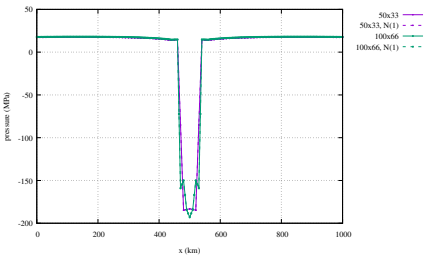


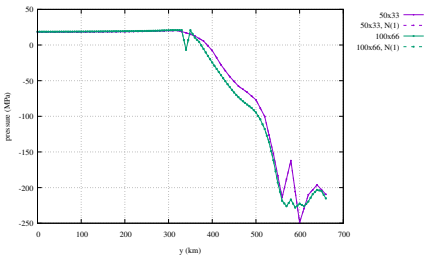












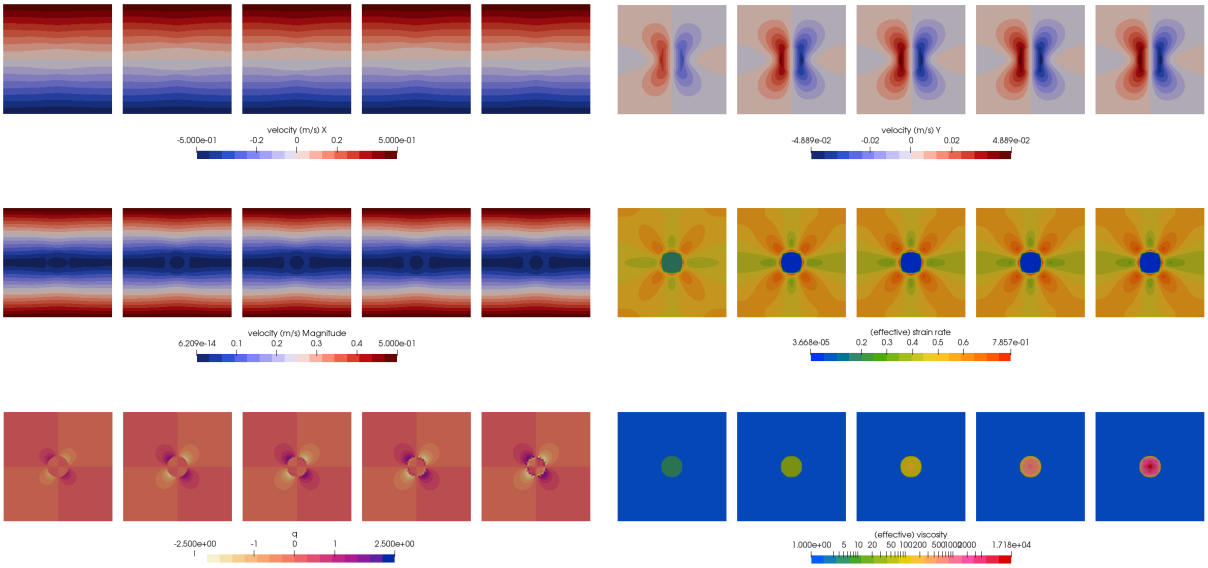
Experiment 5 - inclusion in linear matrix under simple shear

The setup originates in Tenczer et al (2001) [320]. The domain is a unit square. In its middle a circular inclusion of radius $R = 0.1$ is present. While the matrix is a Newtonian fluid, the inclusion is characterised by a power law rheology. Boundary conditions are $\vec{\mathbf{v}} = (0.5, 0)$ at the top, $\vec{\mathbf{v}} = (-0.5, 0)$ at the bottom, and $v = 0$ on the sides. The background strain rate $\dot{\varepsilon}_{xy}$ is then given by

$$\dot{\varepsilon}_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \simeq \frac{1}{2}\frac{\Delta u}{\Delta y} = \frac{1}{2}\frac{0.5 - (-0.5)}{1} = 0.5$$

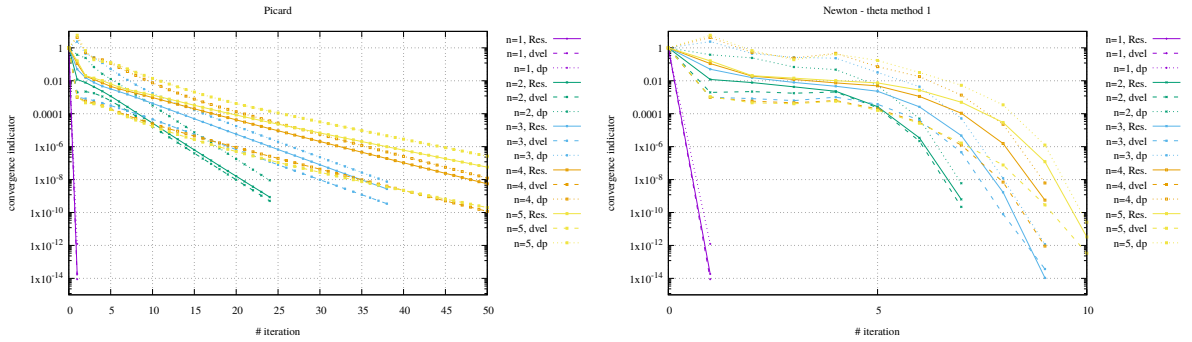
The authors use the BASIL finite element code (see Section **??**) with a mesh based on triangles. We here however keep relying on quadrilateral elements so that the edges of the elements cannot be aligned with the material discontinuity. The material model is called at every quadrature point.

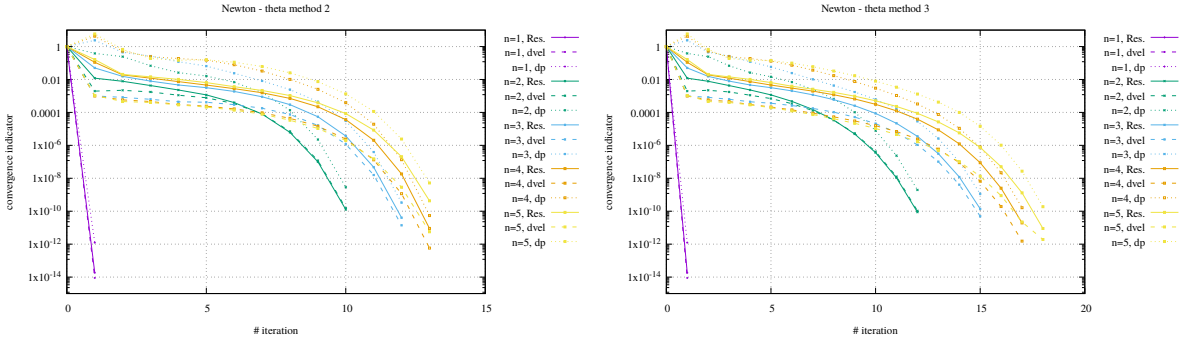
In what follows we use a 64×64 element mesh. We set $B = 1$ and $n = 1$ (i.e. $\alpha = 0$) for the matrix, so that its viscosity is simply 1. The inclusion is characterised by $B = 5$ and $n = \{1, 2, 3, 4, 5, 10\}$. Pressure is volume-average normalised. The regularisation parameter is set to $\dot{e} = 10^{-5}$.



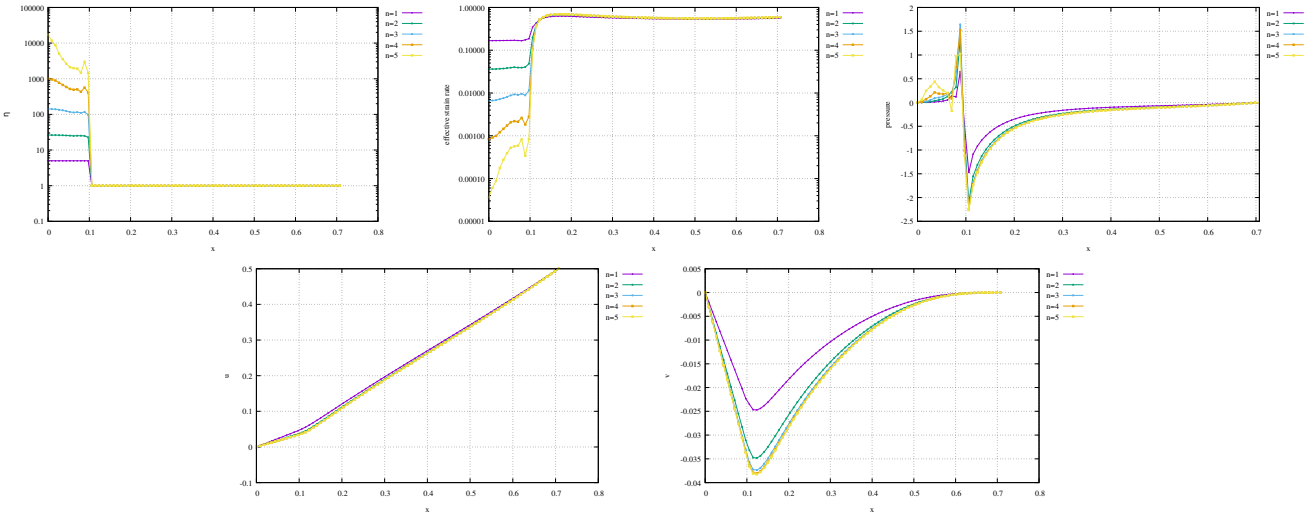
From left to right: $n = 1, 2, 3, 4, 5, 10$. resolution 64x64. Newton.

We see that when n increases so does the effective viscosity in the inclusion. When n becomes very large this experiment becomes very similar to the SolVi (see Section **??**) benchmark.

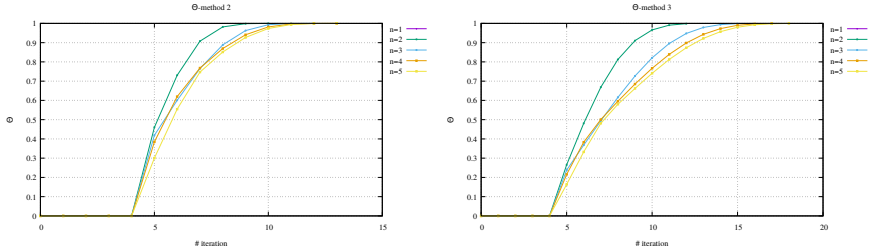




Npicard =4. $\dot{e} = 10^{-16}$



Effective viscosity, strain rate and pressure along the diagonal in the upper right quadrant (i.e. $x \geq 0.5$ and $y \geq 0.5$)



Experiment 6 - the Stokes sphere

This is a simple two-dimensional experiment in which a Stokes sphere is placed at coordinates $(L_x/2, L_y/2)$ and has a radius of 100km. The domain is 600×600km, gravity is set to $g_z = -9.81\,\mathrm{m\,s^{-2}}$. The temperature is a linear gradient between $T = 550°\mathrm{C}$ at the surface and $T = 1330°\mathrm{C}$ at the bottom, except in the sphere where it is set to a constant $T = 550°\mathrm{C}$. The density of the surrounding mantle is $\rho_0 = 3000\mathrm{kg\,m^{-3}}$ and the sphere reference density is $\rho_0 = 3300\mathrm{kg\,m^{-3}}$. The thermal expansion coefficient is set to $\alpha = 3 \cdot 10^{-5}\mathrm{K^{-1}}$ and the reference temperature to $T_0 = 0°\mathrm{C}$. The density is given by:

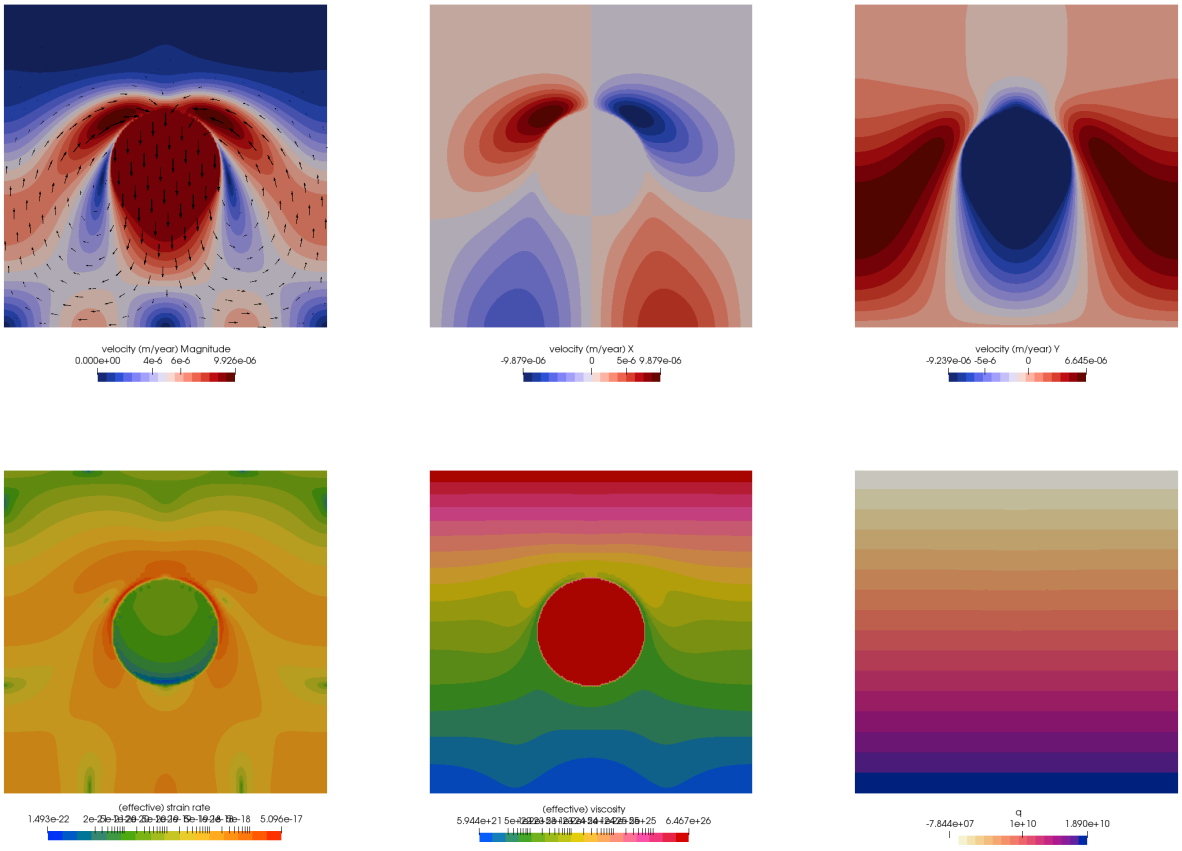
$$\rho(T) = \rho_0(1 - \alpha(T - T_0))$$

Boundary conditions are free slip at the bottom and the sides, open at the top (free surface).

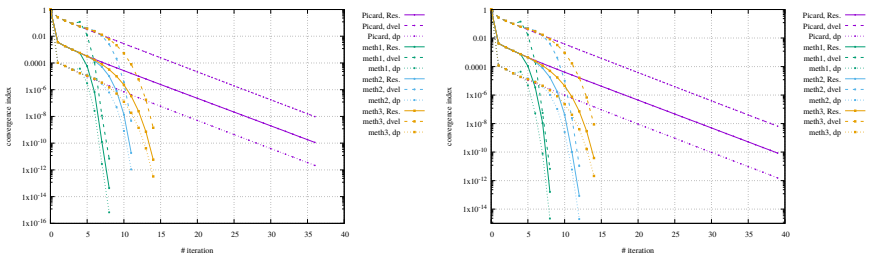
The rheology is of the dislocation creep type and the effective viscosity is then computed as follows:

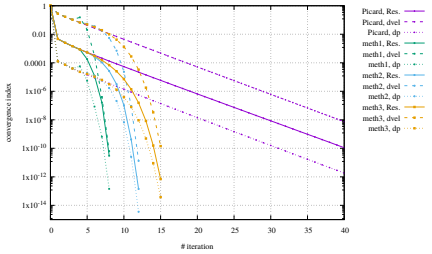
$$\eta = \frac{1}{2}A^{-\frac{1}{n}}\dot{\varepsilon}_e^{\frac{1}{n}-1}\exp\frac{Q}{nRT} = \underbrace{\frac{1}{2}A^{-\frac{1}{n}}\exp\frac{Q}{nRT}}_{B}\dot{\varepsilon}_e^{\alpha}$$

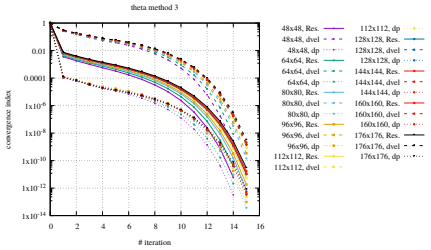
Note that we still use a regularisation parameter set to $\dot{e} = 10^{-18}\mathrm{s^{-1}}$, and that no viscosity cutoff is needed here.

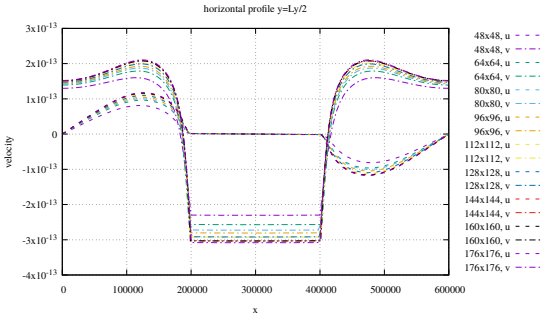


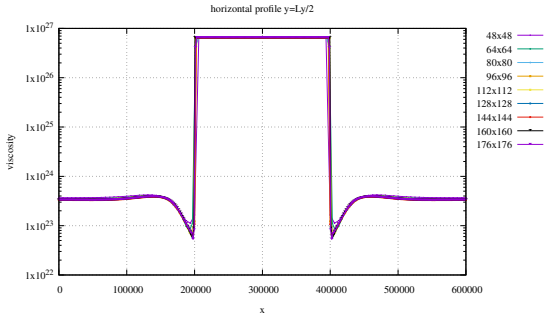
Velocity, strain rate, viscosity and pressure fields. 128x128 mesh.

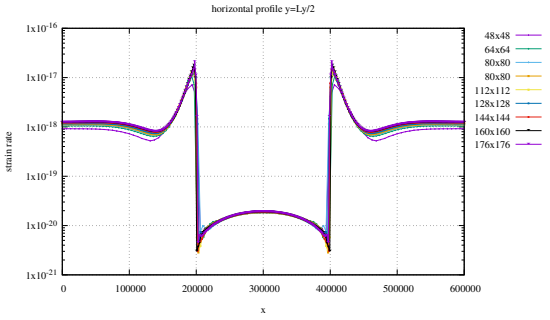


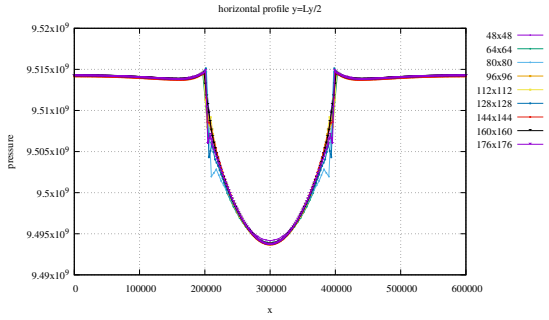

convergence index
iteration
Picard, Res.
Picard, dvel
Picard, dp
meth1, Res.
meth1, dvel
meth1, dp
meth2, Res.
meth2, dvel
meth2, dp
meth3, Res.
meth3, dvel
meth3, dp

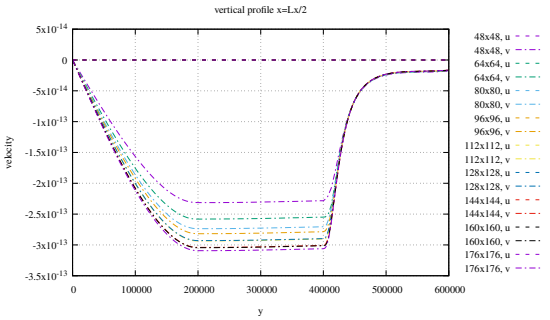

theta method 3
convergence index
iteration
48x48, Res.
48x48, dvel
48x48, dp
64x64, Res.
64x64, dvel
64x64, dp
80x80, Res.
80x80, dvel
80x80, dp
96x96, Res.
96x96, dvel
96x96, dp
112x112, Res.
112x112, dvel
112x112, dp
128x128, Res.
128x128, dvel
128x128, dp
144x144, Res.
144x144, dvel
144x144, dp
160x160, Res.
160x160, dvel
160x160, dp
176x176, Res.
176x176, dvel
176x176, dp

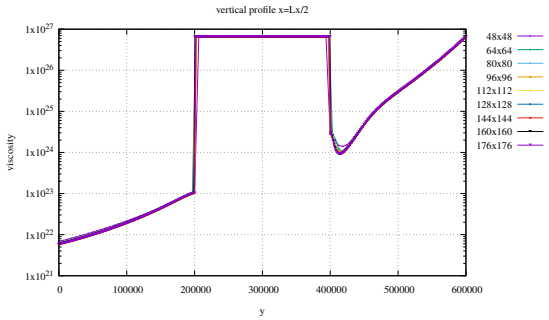

horizontal profile y=Ly/2
velocity
x
48x48, u
48x48, v
64x64, u
64x64, v
80x80, u
80x80, v
96x96, u
96x96, v
112x112, u
112x112, v
128x128, u
128x128, v
144x144, u
144x144, v
160x160, u
160x160, v
176x176, u
176x176, v

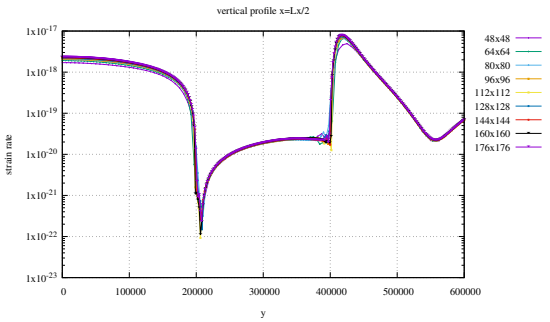

horizontal profile y=Ly/2
viscosity
x
48x48
64x64
80x80
96x96
112x112
128x128
144x144
160x160
176x176

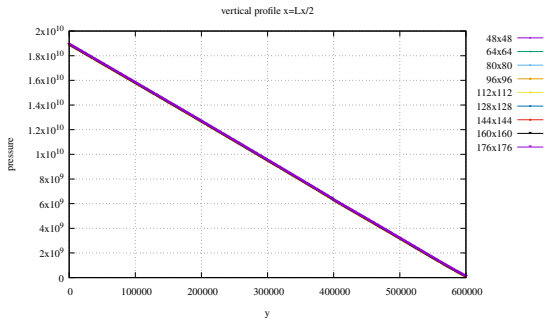

horizontal profile y=Ly/2
strain rate
x
48x48
64x64
80x80
80x80
112x112
128x128
144x144
160x160
176x176


horizontal profile y=Ly/2
pressure
x
48x48
64x64
80x80
96x96
112x112
128x128
144x144
160x160
176x176


vertical profile x=Lx/2
velocity
y
48x48, u
48x48, v
64x64, u
64x64, v
80x80, u
80x80, v
96x96, u
96x96, v
112x112, u
112x112, v
128x128, u
128x128, v
144x144, u
144x144, v
160x160, u
160x160, v
176x176, u
176x176, v

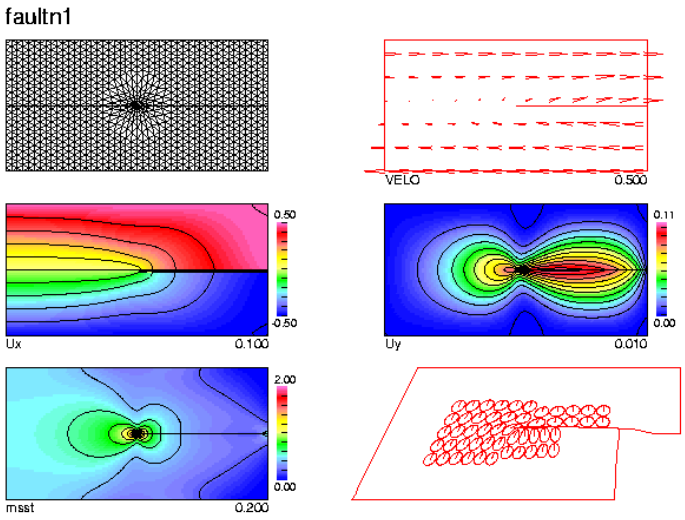

vertical profile x=Lx/2
viscosity
y
48x48
64x64
80x80
96x96
112x112
128x128
144x144
160x160
176x176


vertical profile x=Lx/2
strain rate
y
48x48
64x64
80x80
96x96
112x112
128x128
144x144
160x160
176x176


vertical profile x=Lx/2
pressure
y
48x48
64x64
80x80
96x96
112x112
128x128
144x144
160x160
176x176

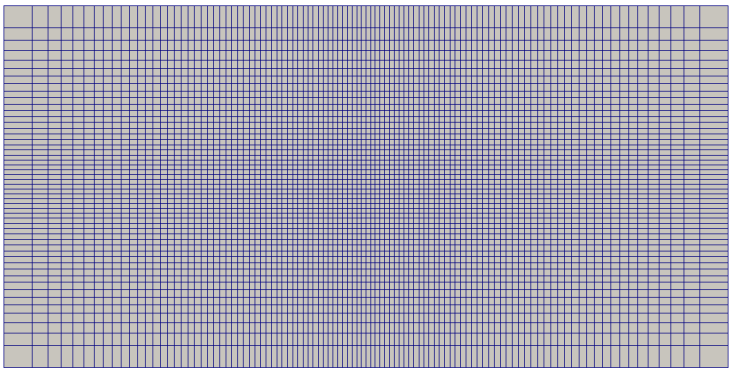
Experiment 7 - Deformation around a Fault that cuts a Block of Viscous Material

The idea for this experiment comes from Barr & houseman [10, 9].



Taken from `http://homepages.see.leeds.ac.uk/~eargah/basil/exa_fault.html`: Here we show the calculated deformation field around a fault that cuts the right hand boundary of the block as far as the centre of the block. The deformation field includes a singularity in the stress components at the fault tip. For that reason, we concentrate mesh points around the fault-tip as shown in the upper left diagram, but the accuracy with which the deformation field is computed can be checked against an analytic solution for the crack tip. The fault permits a displacement discontinuity parallel to its direction, but requires continuity in the perpendicular directions, as for a buried geological fault which separates distinct sliding blocks of rock. If the rock is hot enough (deep in the crust) it creeps like a viscous fluid. The two diagrams in the middle row show the variation of the two components of displacement rate: Ux and Uy. In Ux, we see the transition between a distributed shear flow on the left to the discontinuous displacement on the right. To balance the horizontal flow, there is also a minor component of vertical flow as shown with Uy. With elapsed time finite deformation of the block is shown on the lower right diagram. The variation of displacement and deformation along the fault is diagnostic of the stress vs strain-rate exponent of the viscous material.

We do not necessarily wish to reproduce the results in these papers, but since the authors use a power-law rheology their setup is interesting to us here. Their fault is defined by the following boundary conditions: (1) the normal stress across the fault is continuous; (2) the velocity normal to the fault is continuous but otherwise unconstrained; (3) the shear stress on the fault is equal to some constant, which is set to be zero. This would require a bespoke mesh where the points on the fault would need to be represented by 2 sets of velocity dofs. In order not to implement such a feature, we resort to a very thin sliver of weak Newtonian material at the fault, with a thickness δ_F. Given the geometry, we also implement a form of mesh adaptivity which consists in stretching the elements in both x and y directions so as to yield smaller elements towards the tip of the fault:



Stretched mesh of 120×51 elements. This unstretched mesh would have square elements of size 196m. After stretching, elements in the middle of the domain have a dimension of about 127m, while near the boundaries the elements have a dimension of about 610m.

For $n = 1$ the matrix around the fault is Newtonian and the authors specify its viscosity to be $\eta_0 = 10^{20}$Pa s. For $n > 1$ the authors do not specify the value of B but rather resort to an indirect and unpractical

way. We therefore postulate

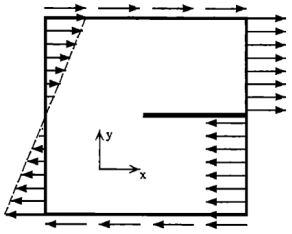
$$B = \eta_0 \dot{\varepsilon}_0^{-\alpha}$$

so that the viscosity is then

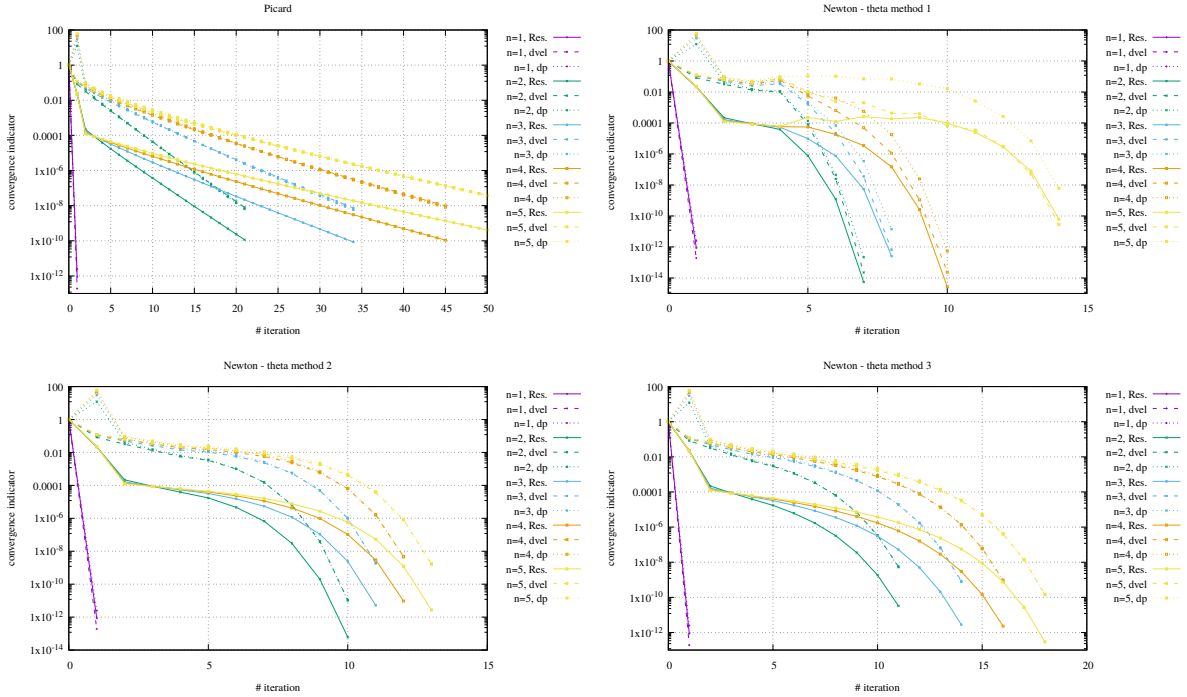
$$\eta = \eta_0 \left(\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_0} \right)^{\alpha}$$

where the reference strain rate $\dot{\varepsilon}_0$ is defined by the slip rate of the fault (i.e. $u_0 = 10\text{mm}\,\text{yr}^{-1}$) divided by L_y. Ideally the fault should have no viscosity but due to machine precision considerations we set it to $\eta_F = 10^{16}\text{Pa}\,\text{s}$.

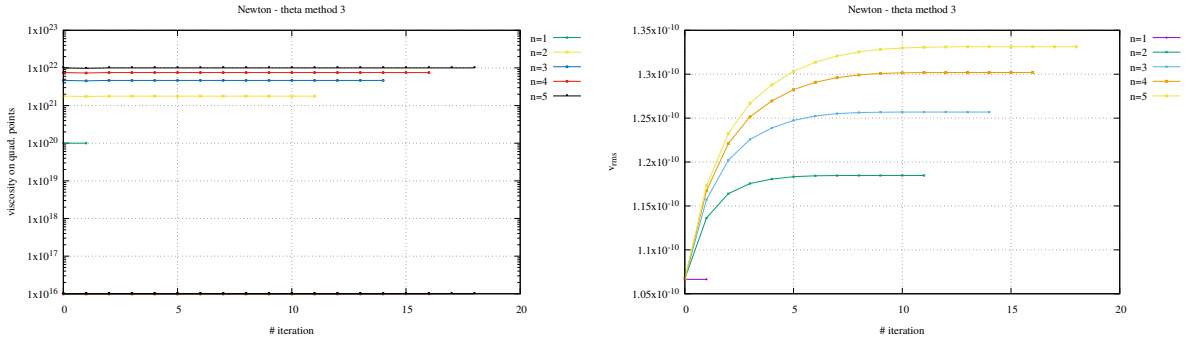
The fault has a length $R_0 = 10\text{km}$ and the domain is then $2R_0 \times R_0$. Boundary conditions are as follows:

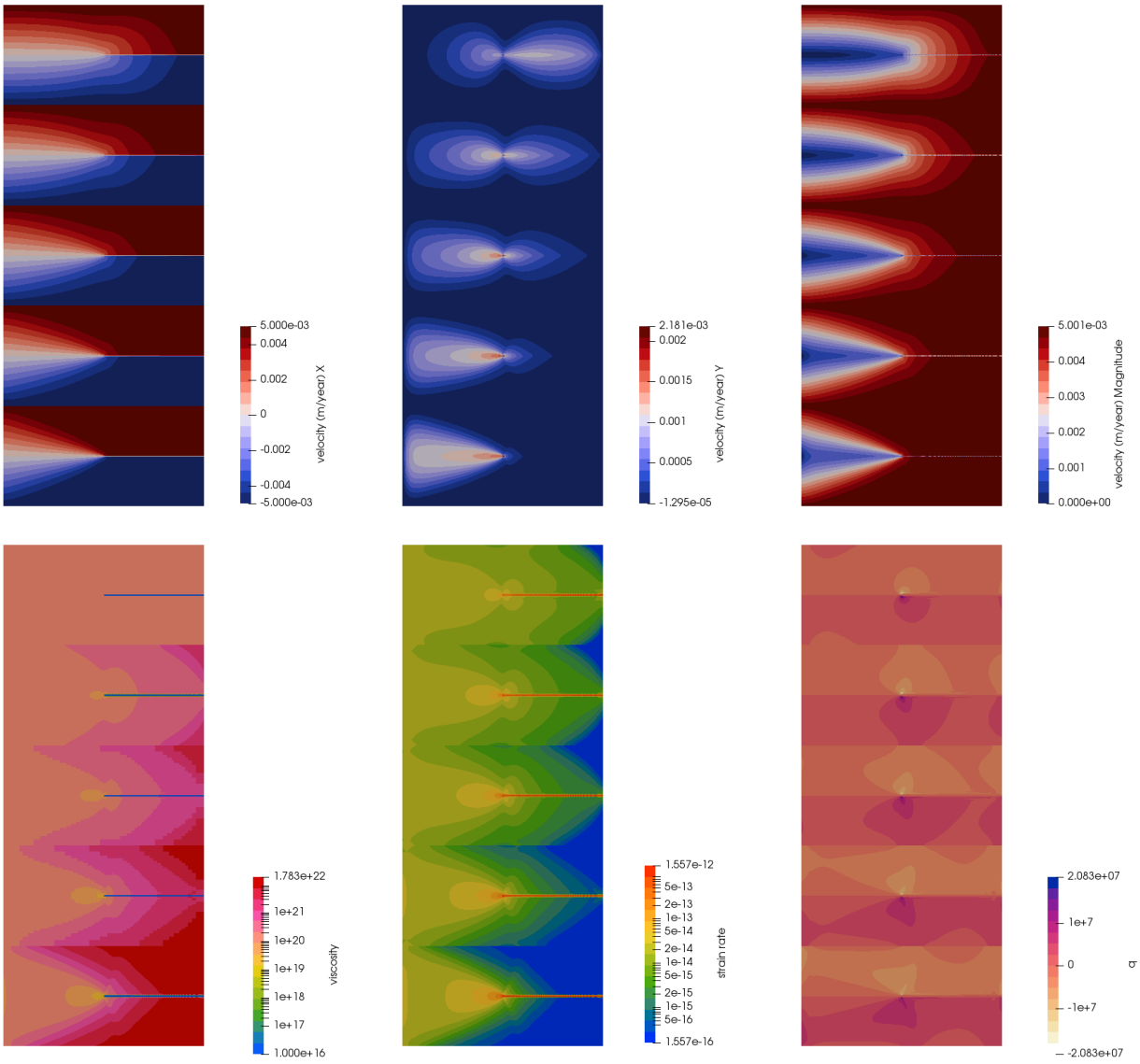


Boundary conditions. Taken from [10], in which the domain is square. $+u_0/2$ is prescribed at the top, and $-u_0/2$ is prescribed at the bottom.



Npicard =4. $\dot{e} = 10^{-16}$





From top to bottom: n=1,2,3,5,10. Mesh 102x51

Experiment 8 - Poiseuille flow

Boundary conditions are no slip at the top and bottom, and no vertical velocity on the sides. Neumann (pressure) boundary conditions are applied on the left and right boundaries: $p_L = 1$, $p_R = -1$ respectively. We set $B = 1$, $L_x = 2$, $L_y = 1$. Resolution is 80x40. The regularisation parameter is set to $\dot{e} = 10^{-8}$.

There is an analytical solution to this problem. In Gerya [127], the viscosity of the non-Newtonian flow is defined by the following rheological equation formulated in term of second stress and strain rate invariants

$$2\dot{\varepsilon}_e = C_1 \sigma_e^n$$

where C_1 is a constant. The effective viscosity is formulated as a function of second strain rate invariant

$$\eta_{eff} = \frac{\sigma_e}{2\dot{\varepsilon}_e} = C_1^{-\frac{1}{n}} (2\dot{\varepsilon}_e)^{\frac{1}{n}-1} = \underbrace{C_1^{-\frac{1}{n}} 2^{\alpha}}_{B} \dot{\varepsilon}_e^{\alpha}$$

In this example we will set $C_1 = 2^{n-1}$. This has the advantage of setting $\eta = 1$ at the top and bottom walls. The vertical component of the velocity is zero, while the horizontal component is given by

$$u(x,n) = \frac{C_1}{n+1} \left(\frac{p_R - p_L}{L_x} \right)^n \left(\left(\frac{L_y}{2} \right)^{n+1} - \left| y - \frac{L_y}{2} \right|^{n+1} \right)$$

the viscosity by

$$\eta(y,n) = \frac{1}{C_1} \left(\frac{p_R - p_L}{L_x} y \right)^{1-n}$$

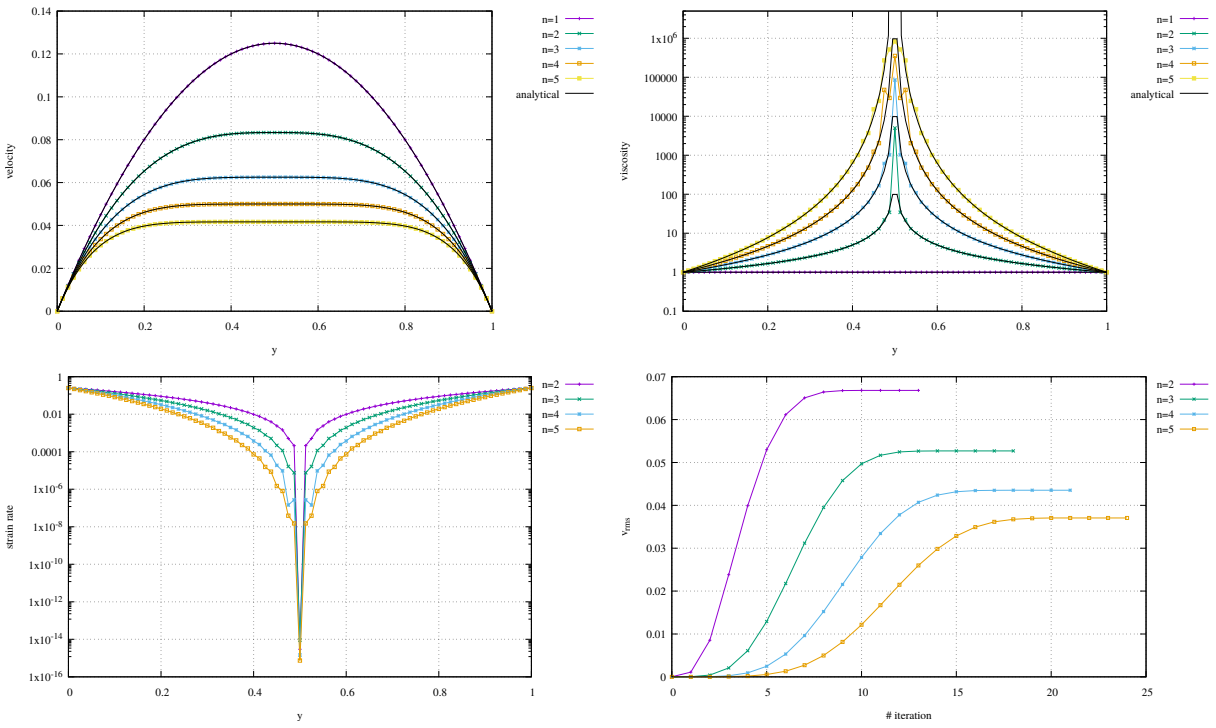
and the pressure by

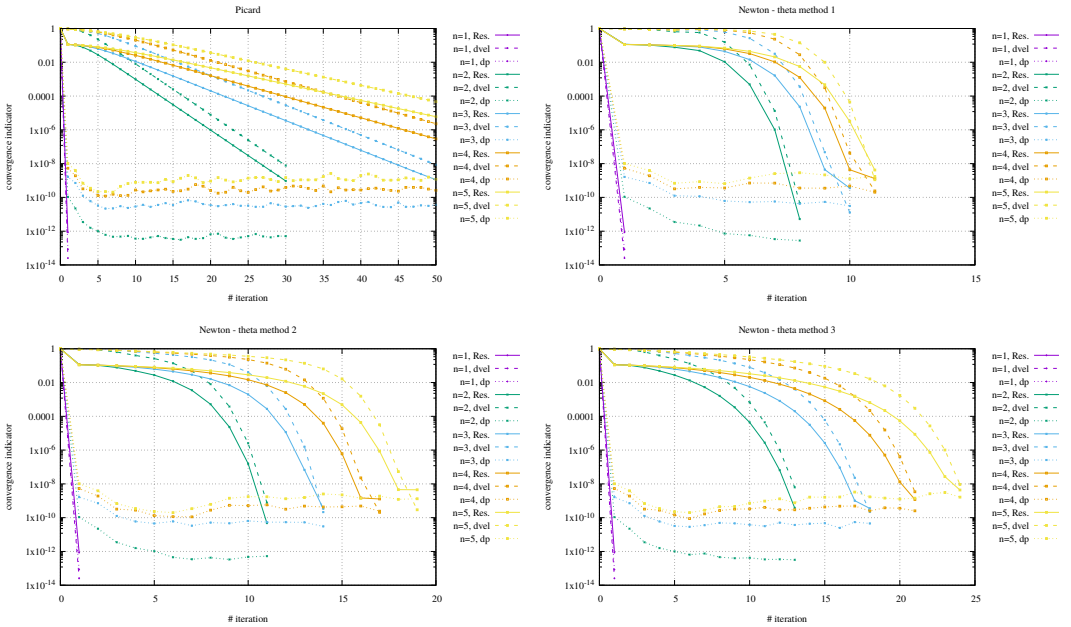
$$p(x) = \frac{p_R - p_L}{L_x} x + p_L$$

In the Newtonian case of constant viscosity, then

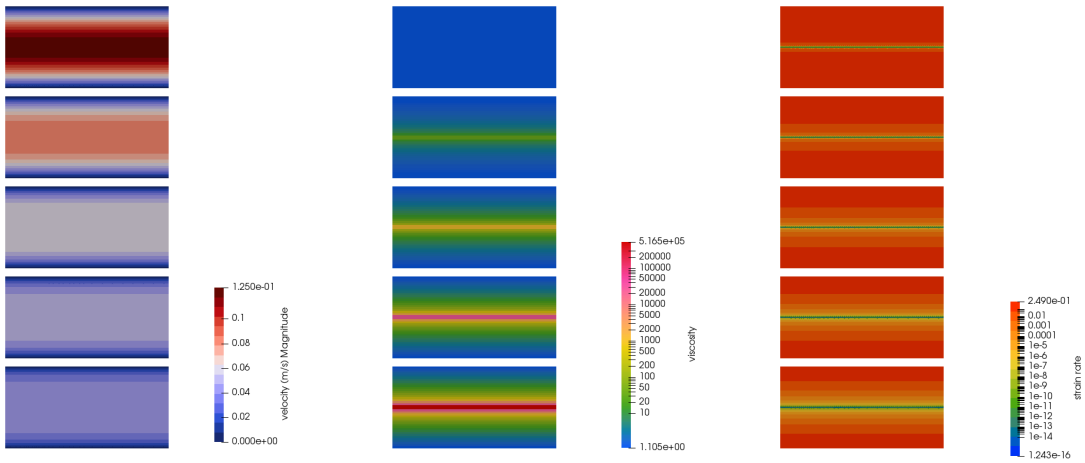
$$u(y) = \frac{1}{2} \frac{p_R - p_L}{L_x} \frac{1}{\eta_0} (y^2 - yH)$$

When the viscosity is equal to 1 ($\alpha = 0$) we recover the standard parabolic velocity profile and a linear pressure field. When $n > 1$ we see that the low strain rate in the middle of the flow yields high viscosities and these quickly grow to untractable values. We therefore limit ourselves to $n = 2, 3, 4, 5$:





Npicard =4



From top to bottom: $n = 1, 2, 3, 4, 5$.

Experiment 9 - the punch problem

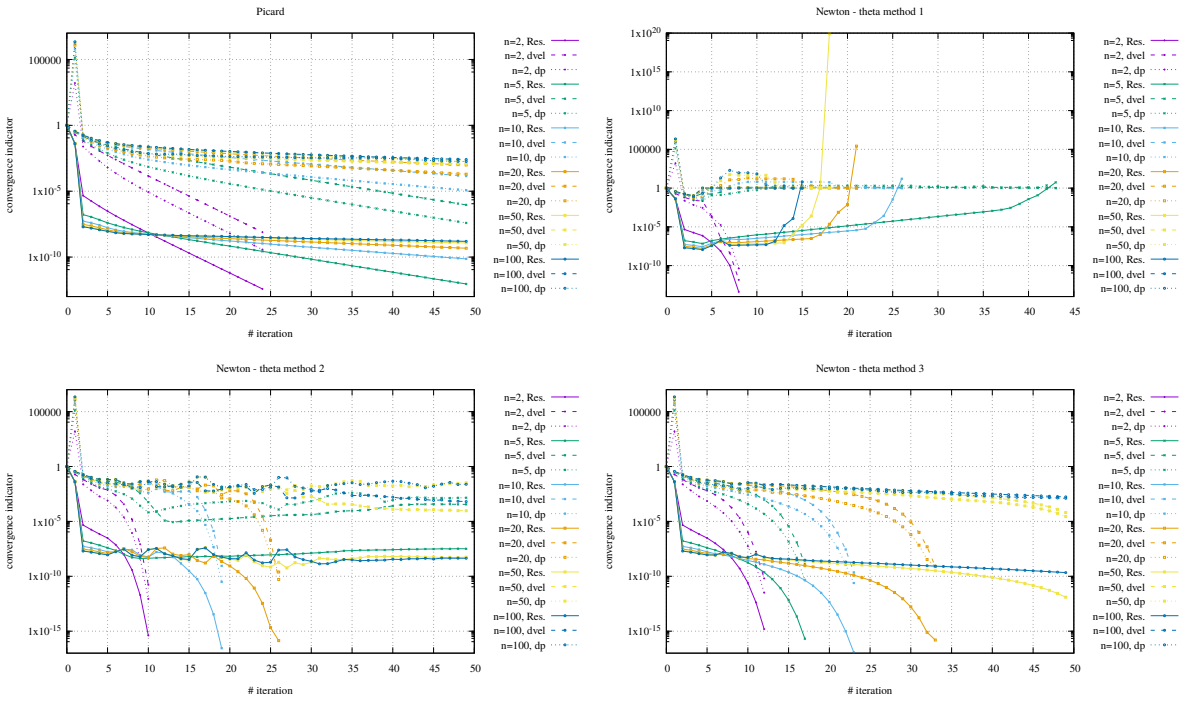
The domain is 1×0.5. No-slip boundary conditions are prescribed on the left, right, and bottom boundaries. On the top boundary $\mathbf{v} = (0, -1)$ is prescribed for $|x - L_x/2| < 0.123456$. We set $\dot{e} = 10^{-6}$ and $B = 1$. When $n >> 1$ then $\alpha \to -1$ and then the effective viscosity tends to

$$\eta_{eff} \to \frac{B}{\dot{\varepsilon}_e} = \frac{Y}{2\dot{\varepsilon}_e}$$

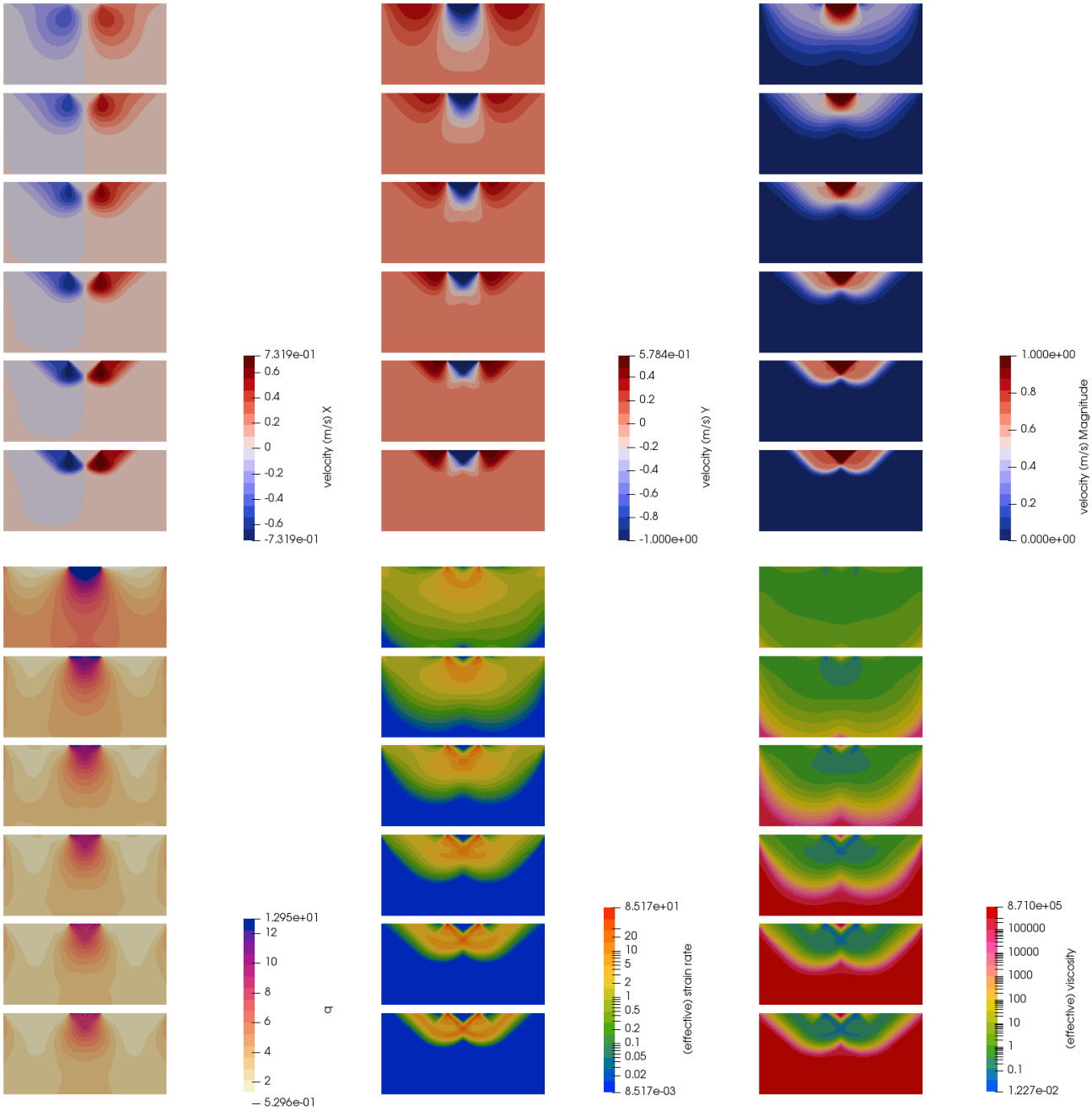
with $Y = 2B$. This is the standard form for the viscosity rescaling method in (simple) plasticity. If the regularisation parameter is present, then

$$\eta_{eff} \to \frac{Y}{2(\dot{\varepsilon}_e + \dot{e})}$$

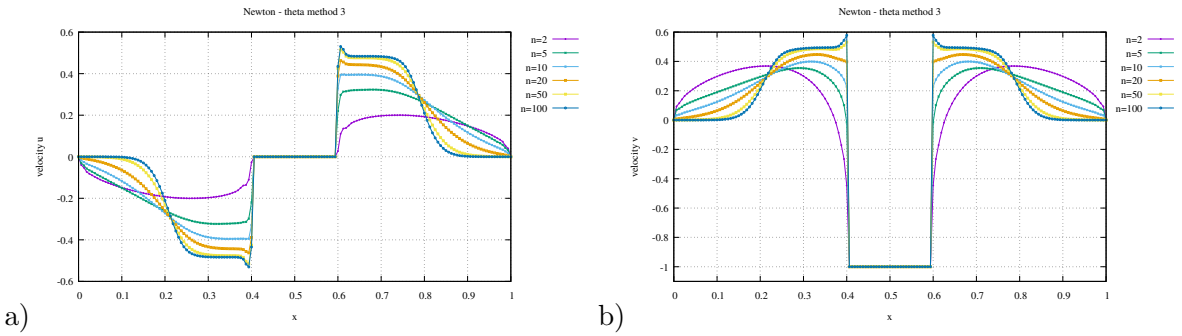
with $Y = 2B$. This is the standard form for the viscosity rescaling method in (simple) plasticity. Interestingly, when $\dot{e} \sim \dot{\varepsilon}_e$ then the viscosity tends to $\sim B/\dot{e}$ which acts as a maximum viscosity (soft) limiter. In this case, $B/\dot{e} = 10^6$ and we see that the viscosity in the rigid regions tends towards this value.



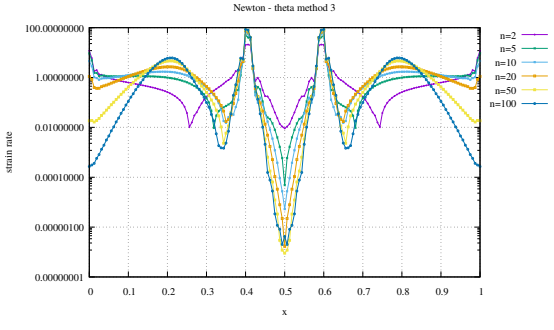
Npicard =4



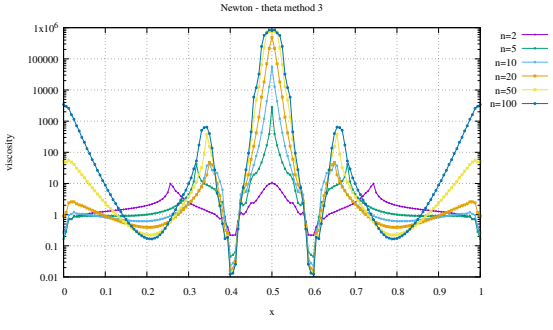
Resolution 80x40. $n = 2, 5, 10, 20, 50, 100$. Npicard=4. θ method 3. When $n >> 1$ we indeed recover the classical solution of the rigid-plastic punch, see Section **??**



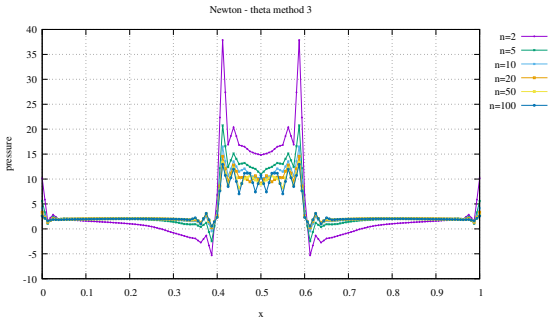
a) b)



c)

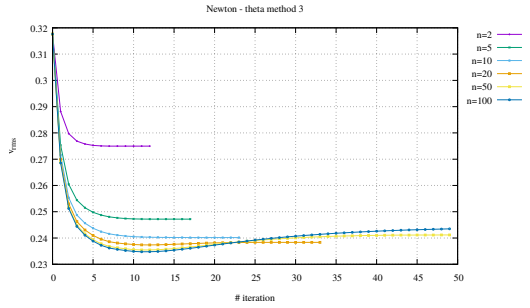


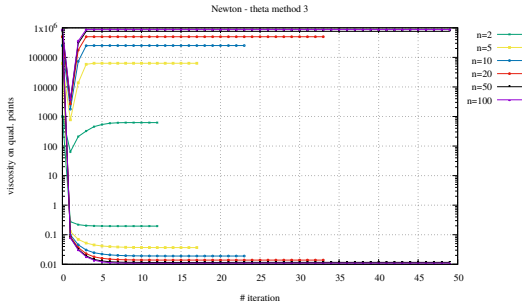
d)

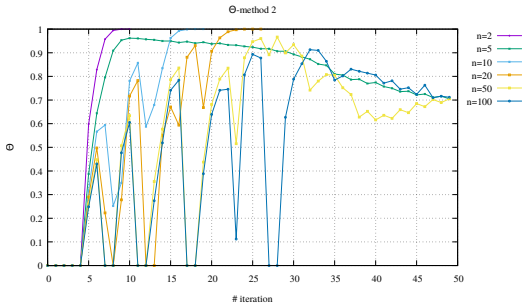


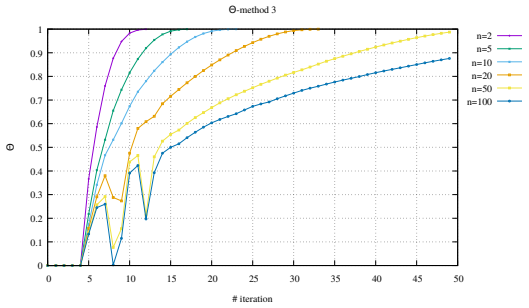
e)

a,b,c,d,e) Velocity, strain rate, viscosity and pressure measured at the top of the domain, Npicard=4, 80x40.









Npicard =4, θ-method

Experiment 10 - Spiegelman et al (2016) brick

This experiment is presented in Spiegelman (2016) [302] and is also carried out in Fraters et al (2019) [116]. The domain is 120×30km. Since the rheology does not depend on pressure we set $\vec{g} = \vec{0}$. Boundary conditions and material parameters are shown hereunder:

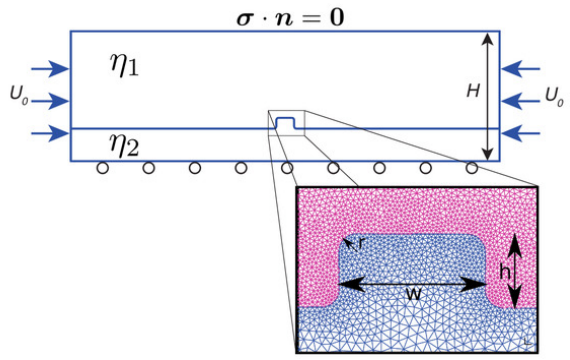


Table 1. Parameters Used in the Compressional Notch Test

Name	Description	Values
η_1	background viscosity of upper viscoplastic layer	10^{23}–5×10^{24} Pa s
η_2	Viscosity of the lower layer and notch (isoviscous)	10^{21} Pa s
η_0	Reference viscosity for scaling	10^{22} Pa s
H	Domain depth (and scale length)	30 km
H_1	Upper layer depth	$0.75H$
H_2	Lower layer depth	$0.25H$
L	Domain width	$4H$
w	Notch width	$1/6H$
h	Notch height	$1/12H$
r	Radius of curvature of notch corners	$0.02H$
U_0	Horizontal velocity (compression)	0.0025–0.0125 m/yr
C	Cohesion	10^8 Pa
φ	Friction angle	0–30 °
ρ	Density (both layers)	2700 kg m^{-3}
g	Acceleration due to gravity	9.81 m s^{-2}

The background strain rate is then

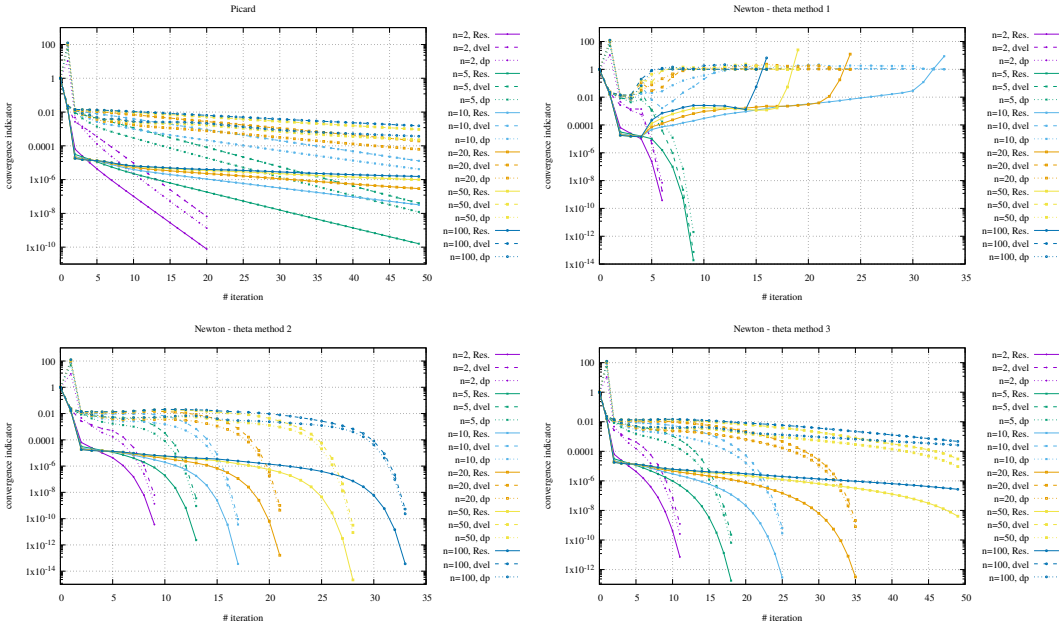
$$e = \frac{2U_0}{L_x} = \frac{0.005\text{m yr}^{-1}}{120\text{km}} \simeq 1.32 \cdot 10^{-15}\text{s}^{-1}$$

Although the rounded edges of the weak seed are implemented they are not 'seen' because of the low resolution: $r = 0.02H = 600$m while the resolution we use here is ~ 1200m

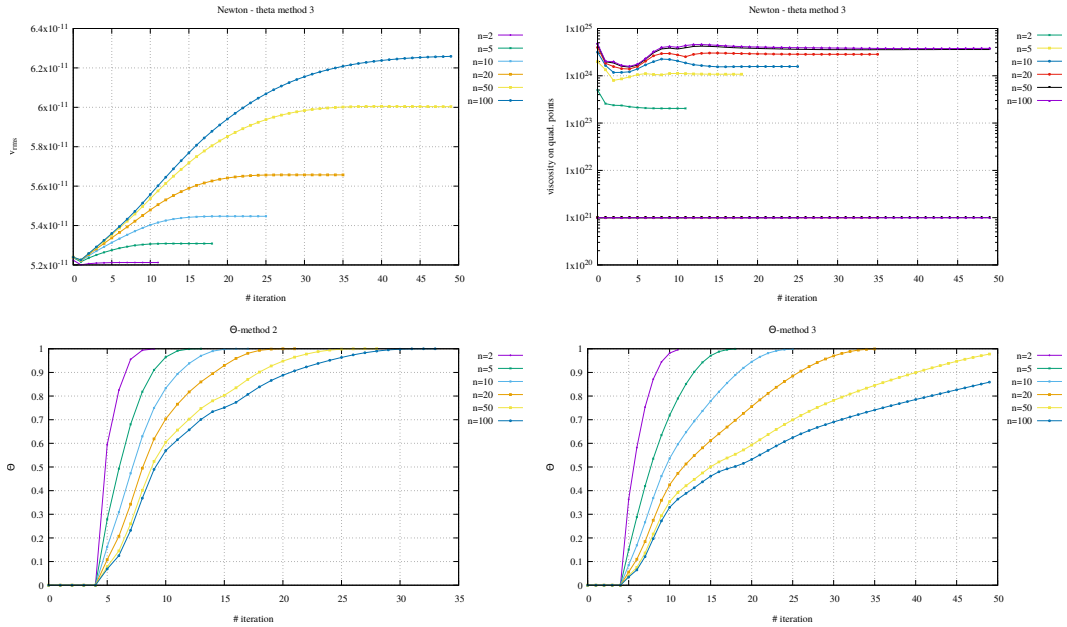
In the von Mises rheology case (no pressure dependence), the viscosity of the top layer is given by

$$\eta_p = \frac{C}{2\dot{\epsilon}_e}$$

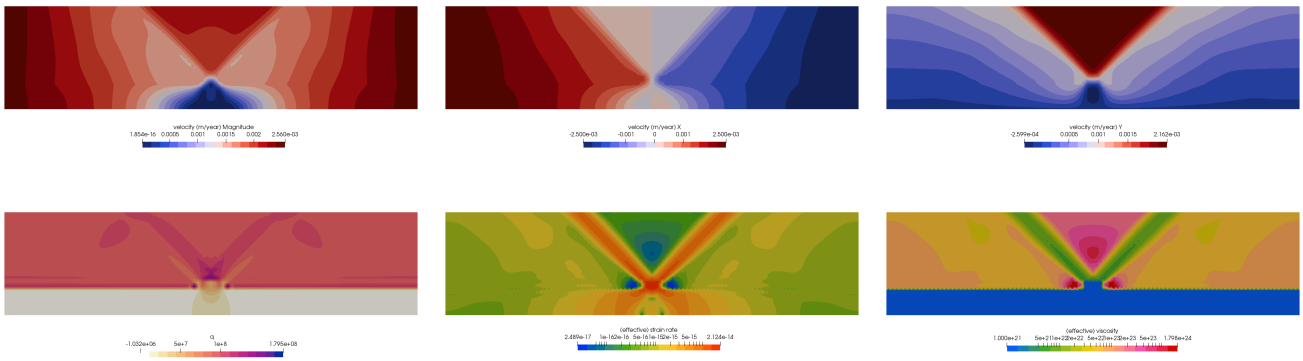
which we can replace by the following power law formulation: $\eta = B\dot{\varepsilon}_e^{\alpha}$ where $B = \frac{\eta_{lim}}{e^{\alpha}}$ and $e = 10^{-15}s^{-1}$ and $\eta_{lim} = C/2e$.



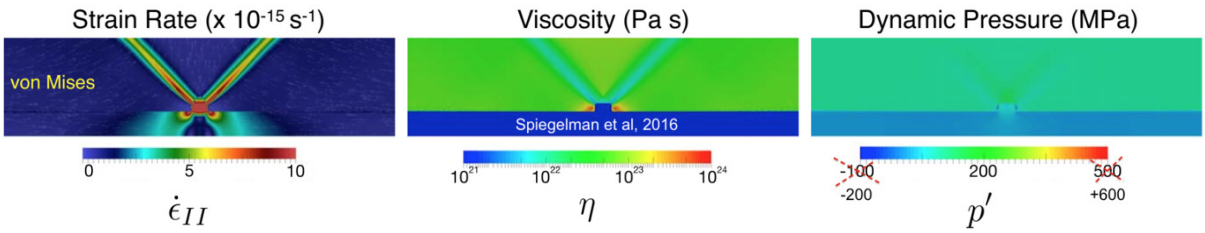
Npicard =4



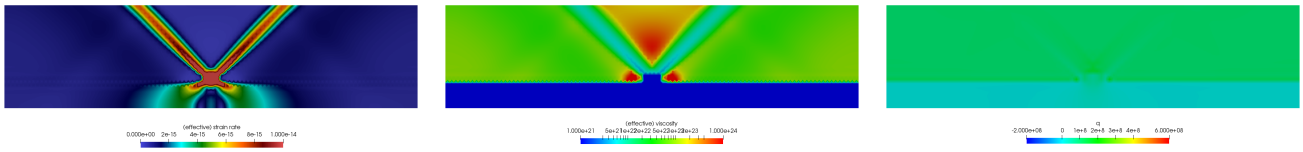
Npicard =4, θ-method



Npicard=4, θ-method=3, resolution 100x25, $n = 100$.



Taken from Spiegelman et al. Note that I am 99% sure that the pressure range in the paper is wrong.

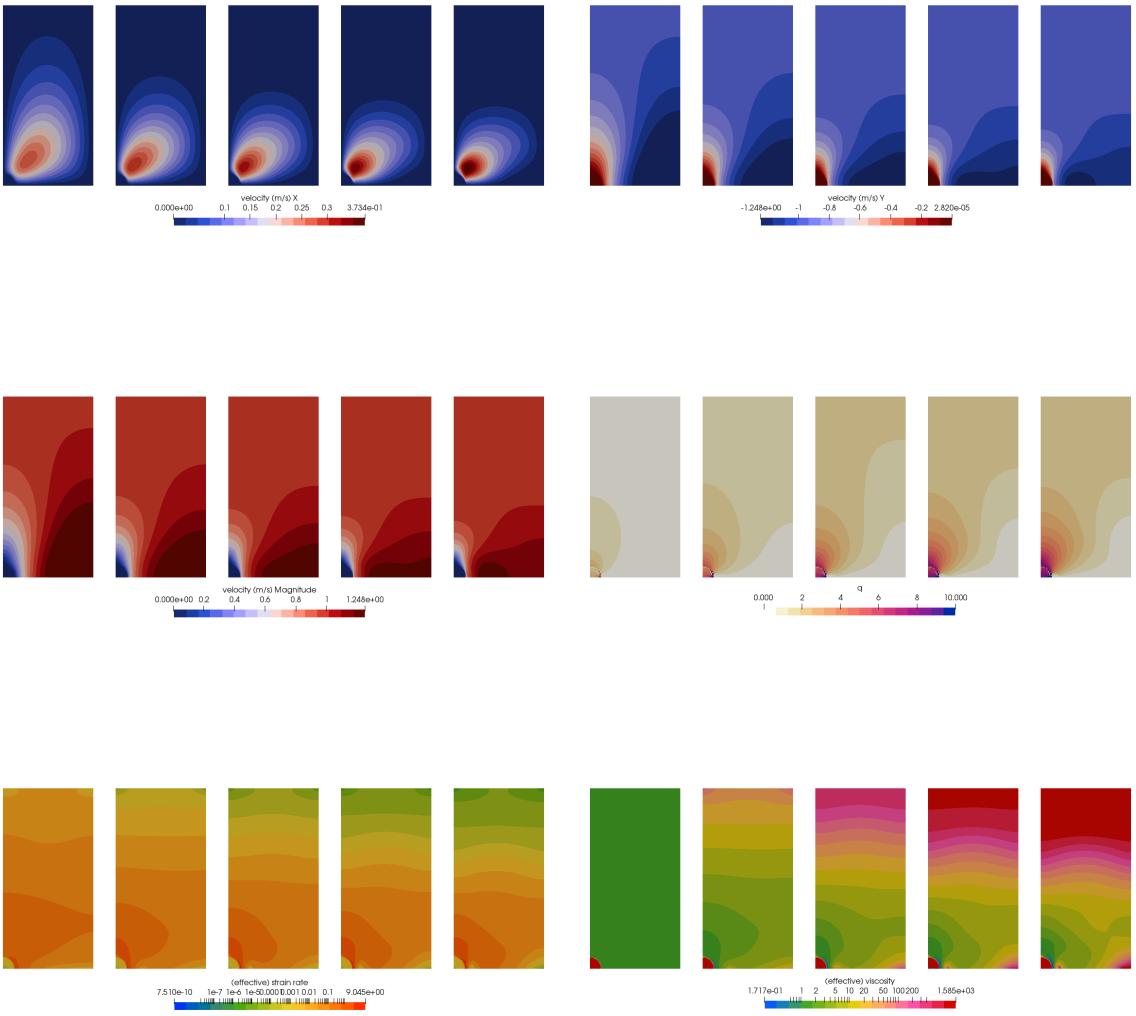


Same as above but with coloscales of Spiegelman et al

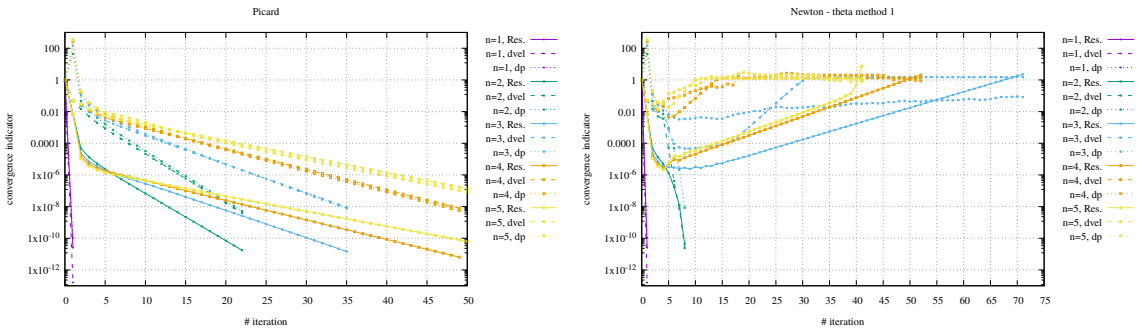
Flow around a cylinder

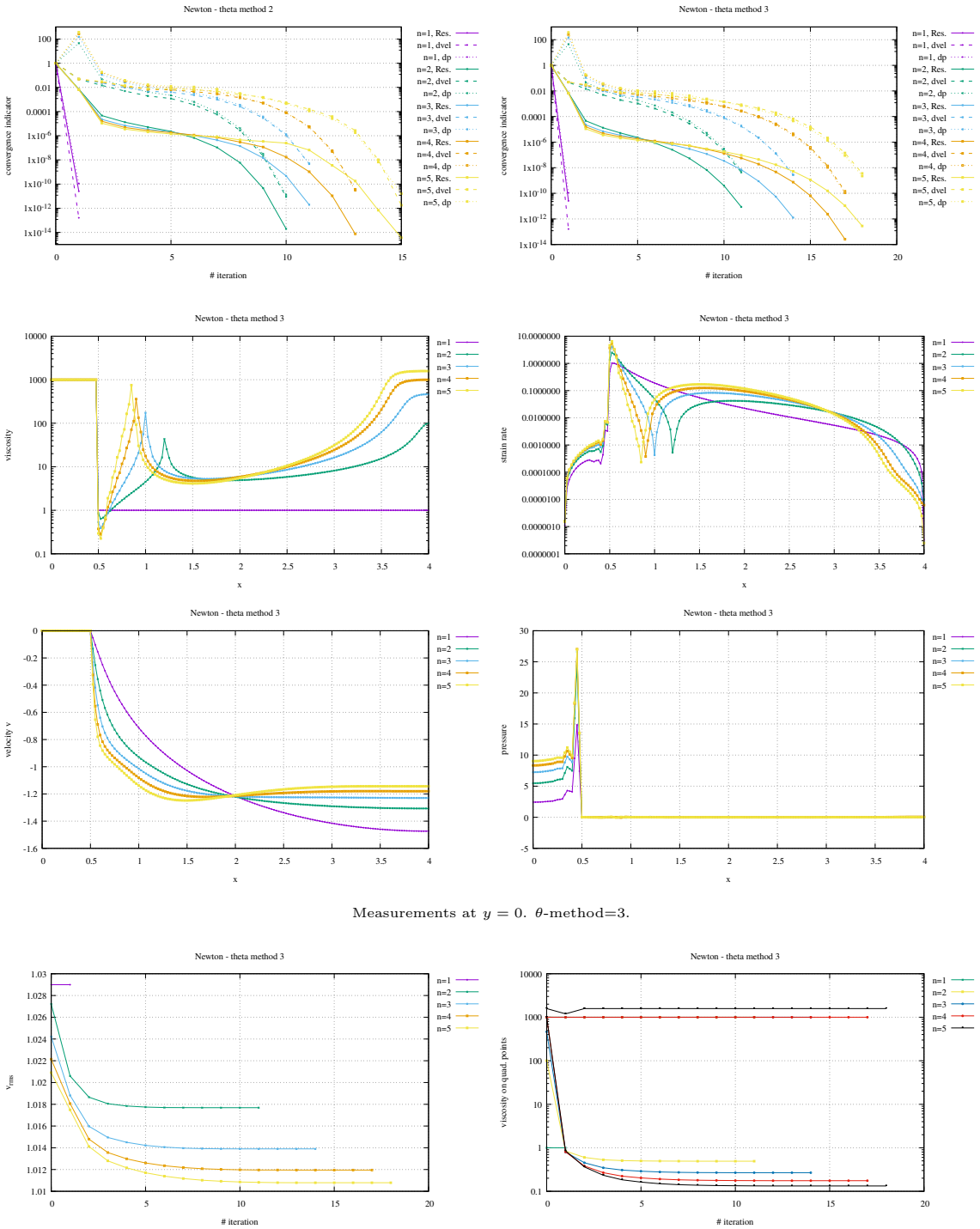
Because of symmetries we model only one fourth of the system. The domain is 4x8 and the cylinder is centered on the origin with a radius of 1/2. Free slip boundary conditions are prescribed on the sides, and $\vec{\mathbf{v}} = (0, -1)$ is prescribed at the top. There is no gravity so density is irrelevant.

We start with a power-law rheology, with $B = 1$ and $n = 5$, $\dot{e} = 10^{-4}$:



Resolution 40x80. Npicard=4. θ method 3.





Measurements at $y = 0$. θ-method=3.

TODO: The Herschel-Bulkley viscosity is given by

$$\eta = K\dot{\varepsilon}_e^{\tilde{n}-1} + \frac{\tau_0}{\dot{\varepsilon}_e}$$

and it is often regularised for small strain rates as follows:

$$\eta = K\dot{\varepsilon}_e^{\tilde{n}-1} + \frac{\tau_0}{\dot{\varepsilon}_e}\left(1 - \exp\frac{\dot{\varepsilon}_e}{\dot{\varepsilon}_0}\right)$$

The Oldroyd number (Od) is used to quantify the relation between the plastic and viscous effects. [18]

$$Od = \frac{\tau_0}{K(U/2r)^n}$$

By choosing $K = 1$, $U = 1$, $r = 1/2$ then $Od = \tau_0$. There remains to choose $\dot{\varepsilon}_0$. When $\dot{\varepsilon}_e \to 0$, then $\eta \to\to K\dot{\varepsilon}_e^{\tilde{n}-1} + \tau_0/\dot{\varepsilon}_0$.

Vary Od from 0.1 to 100

Relevant Literature: Rudi, Shih, and Stadler [271] Spiegelman, May, and Wilson [302]
Remaining questions
- how to choose reg parameter
- how to measure convergence ?
- line search - ask Menno
- todo : - carreau example - harmonic avrg example - vrm example
- higher order newton
- regularisation parameter or max viscosity ?
- positive definite

Stone 88

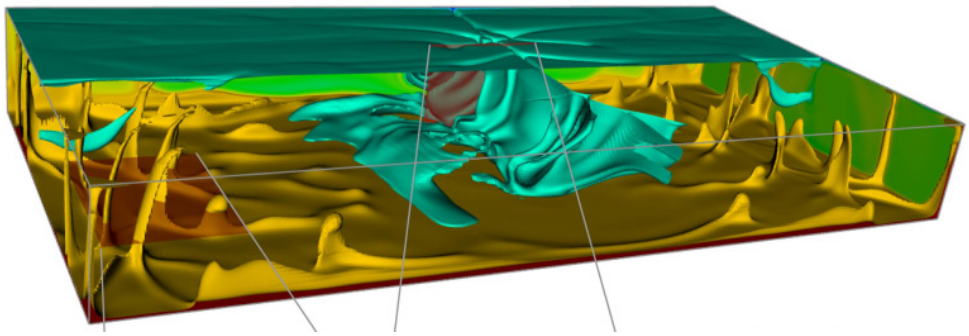
convection in a stratified mantle ($Q_1 \times P_0$, $Q_2 \times Q_1$, $Q_3 \times Q_2$)

FEM

2D
box
Q1P0
Q2Q1
Q3Q2
Q4Q3
sparse_matrix
stokes_equation
energy_equation
thermo_mechanical
time_stepping
viscous_rheology
buoyancy_driven
vrms
rayleigh_number
paraview_output
pressure_scaling
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_88`

The motivation for this stone is based on Burstedde et al. [58] (2008):



Taken from [58]. Dimensionless dimensions are $8 \times 4 \times 1$.

This stone solves the dimensionless thermo-mechanical equations:

$$\vec{\nabla} \cdot \vec{\mathbf{v}} = 0 \tag{88.1}$$

$$-\vec{\nabla} p + \vec{\nabla} \cdot [2\eta(T, \dot{\varepsilon}_e, y)\dot{\boldsymbol{\varepsilon}}] = Ra\, T \vec{e}_y \tag{88.2}$$

$$\frac{\partial T}{\partial t} + \vec{\mathbf{v}} \cdot \vec{\nabla} T = \Delta T \tag{88.3}$$

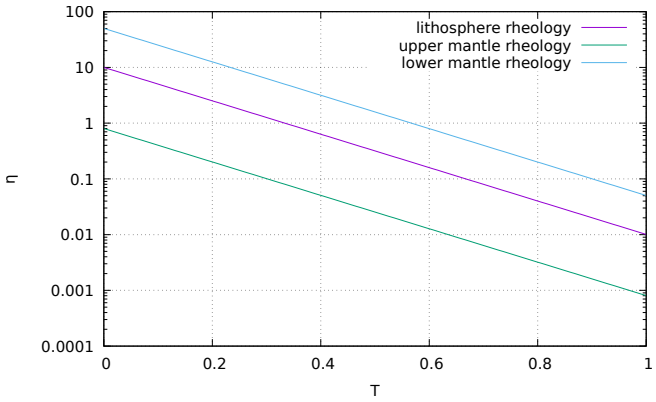
with $\dot{\varepsilon}_e$ is the effective strain rate and the viscosity η is a simple piecewise function of temperature, depth and strain rate.

This approach is not unique and we therefore look into three other similar implementations. If you are aware of another one, please let me know and I will add it!

viscosity_model=1

The viscosity in Burstedde et al. [58] (2008) is given by

$$\eta(T,\dot{\varepsilon}_e,y) = \begin{cases} \min\left(\frac{\sigma_y}{2\dot{\varepsilon}_e}, 10\exp(-6.9T)\right) & y \leq 0.9 \\ 0.8\exp(-6.9T) & 0.77 \leq y \leq 0.9 \\ 50\exp(-6.9T) & y \leq 0.77 \end{cases}$$



Dimensionless viscosity values as a function of the dimensionless temperature.

In the paper the value of σ_y is not specified, nor is the Rayleigh number (although they mention it is between 10^6 and 10^9). Their box is $8 \times 4 \times 1$ and temperatures range from 0 at the top to 1 at the bottom. In this case 3D modeling is out of the question here so we restrict ourselves to 2D with a 4×1 domain. We set $\sigma_y = 1.5 \cdot 10^5$ and $Ra = 10^6$.

The initial temperature is based on the steady state conduction profile $T_(y) = 1 - y$, to which a perturbation $\delta T(x, y)$ is added with

$$\delta T(x,y) = -0.03\left[\cos\left(2.132\pi\frac{x}{L_x}\right) + \cos\left(3.333\pi\frac{x}{L_x}\right) + \cos\left(7.123\pi\frac{x}{L_x}\right)\right]\sin\left(\pi\frac{y}{L_y}\right)$$

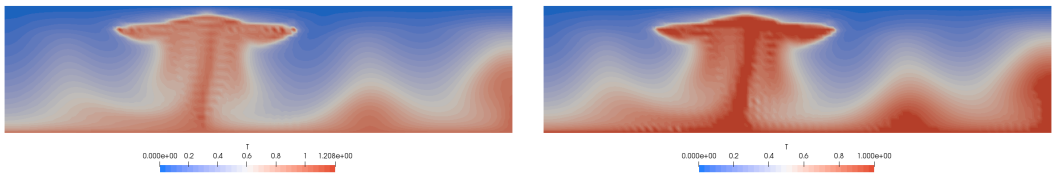
The exact form of this perturbation is not critical, but it serves as a predictable and repeatable way of initiating the convection.

To start with, we implement the advection-diffusion equation without stabilisation but it is clear that large under- an overshoots occur due to the advection dominated heat transport in the large upwelling generated by the perturbation.

We then apply the Lenardic and Kaula [209] (1993) filter, as explained in Section 14.4.

1. Compute the initial sum S_0 of all values of C.
2. Find the minimal value C_{min} below 0.
3. Find the maximal value C_{max} above 1.
4. Set $C_i = 0$ if $C_i \leq |C_{min}|$.
5. Set $C_i = 1$ if $C_i \geq 2 - C_{max}$.
6. Compute the sum S_1 of all values of C
7. Compute the number num of $0 < C_i < 1$.
8. Add $(S_1 - S_0)/num$ to all $0 < C_i < 1$

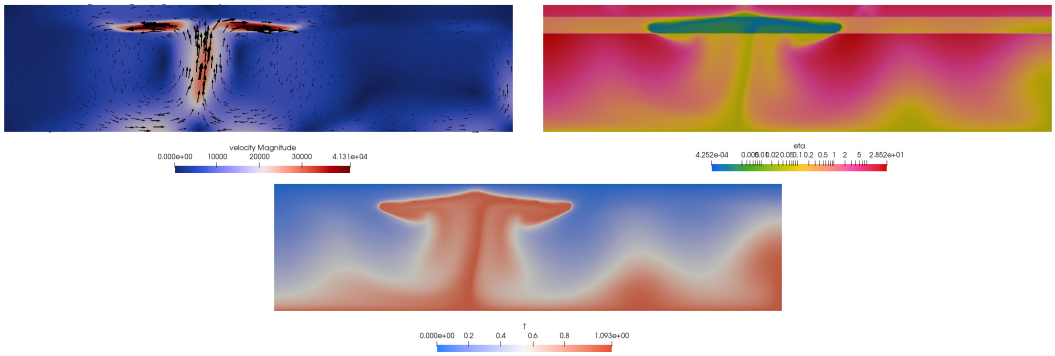
The filter is designed to prevent under- and overshoot but it cannot remove the oscillations between 0 and 1. It therefore cannot 'cure' alone the poor quality of the solution within these bounds.



Top: no filter, bottom: with filter. 64x16, CFL=0.5

Notice the different scales.

We now switch the filter off implement the SUPG method (see Section 9.6). The resolution is set to 200×50:



Velocity, viscosity and temperature fields at time $t \sim 0.00015$, CFL=0.5

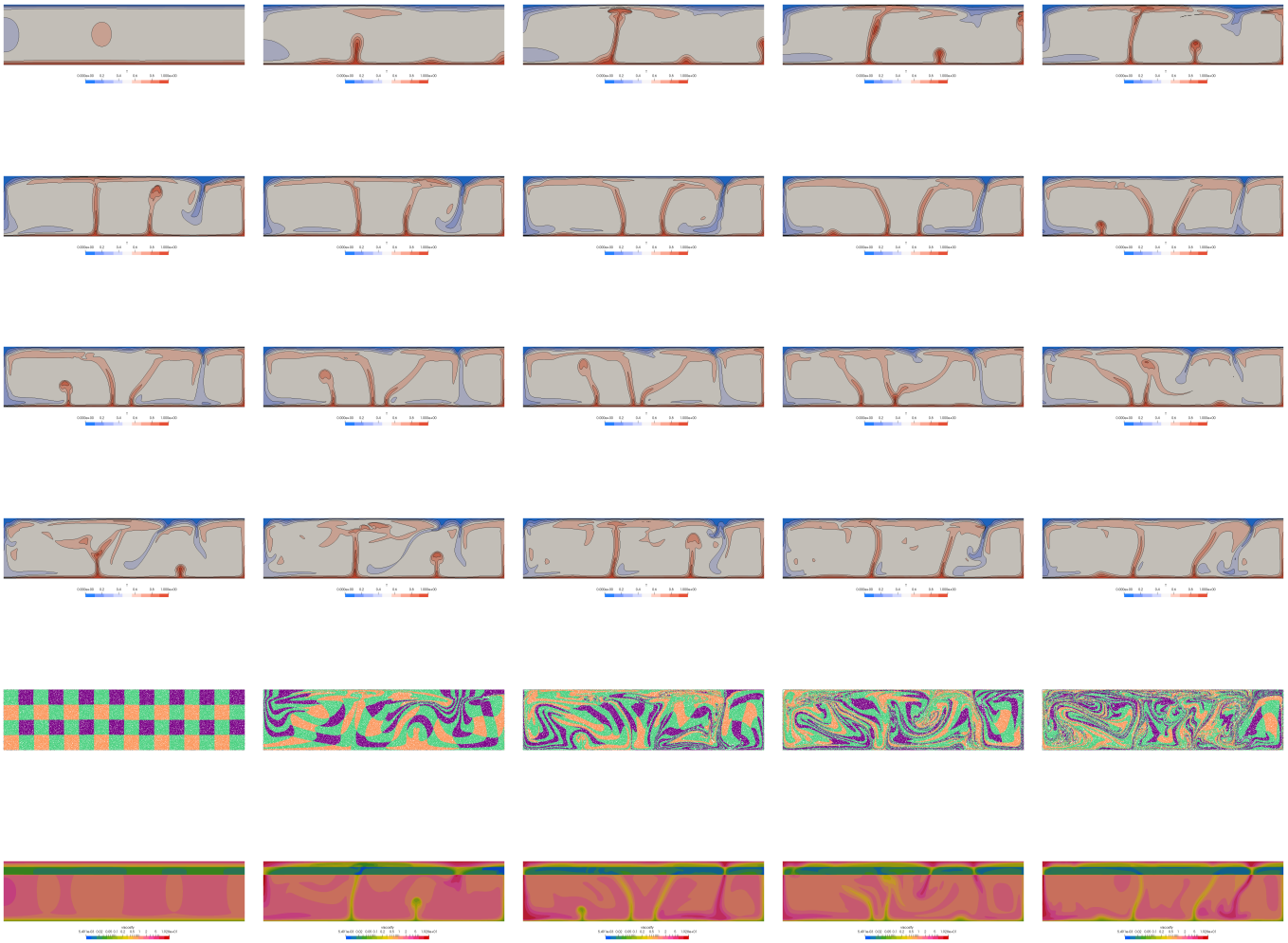
We see that a combination of high-resolution and SUPG stabilisation keeps the temperature field within reasonable bounds.

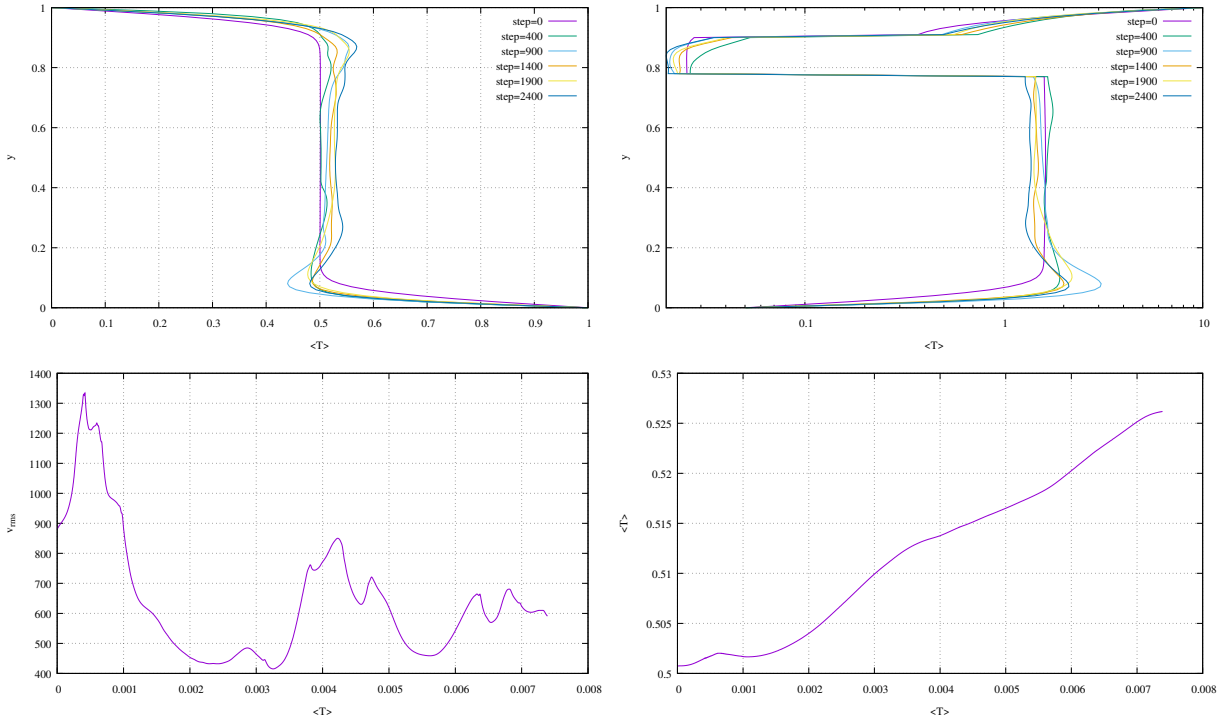
It is well known that in a well-mixed mantle, the average temperature profile follows the so-called adiabatic profile in its middle and that there are two boundary layers at the top and bottom. This adiabatic profile temperature gradient is much smaller than the conductive profile we start with. We therefore revisit the topic of the initial temperature field and now start from a constant background temperature set to 0.5.

The diffusion boundary layers are prescribed by means of a half-space cooling equation. A rather artificial 'age' is set to $age = 10^{-3}$ (the diffusion coefficient is 1, see dimensionless equations above). The initial temperature is then obtained as follows: 1) set temperature everywhere to 0.5; 2) add half-space cooling contribution to generate a field that smoothly transitions from 0.5 to 0 at the top and 0.5 to 1 at the bottom; 3) add perturbation.

```
for i in range(0,NV):
    T[i]=0.5
    T[i]-=0.5*special.erfc((Ly-yV[i])/2/np.sqrt(1*0.001))
    T[i]+=0.5*special.erfc(yV[i]/2/np.sqrt(1*0.001))
    T[i]-=0.03*(np.cos(2.132*np.pi*xV[i]/Lx)+\
                np.cos(3.333*np.pi*xV[i]/Lx)+\
                np.cos(7.123*np.pi*xV[i]/Lx)) *np.sin(np.pi*yV[i]/Ly)
```

In what follows we set CFL=0.5 and SUPG is used. We see that the behaviour of the system is very different. The perturbation generates much smaller buoyancy forces. The huge upwelling is gone and rather plumes form. Also, in order to better visusalise the flow, we add passive markers which are advected with the computed velocity at each time step (see STONE 37). These are painted with a checkerboard pattern.





viscosity model 1. Temperature, markers and velocity fields. Resolution 200 × 50.

viscosity_model=2

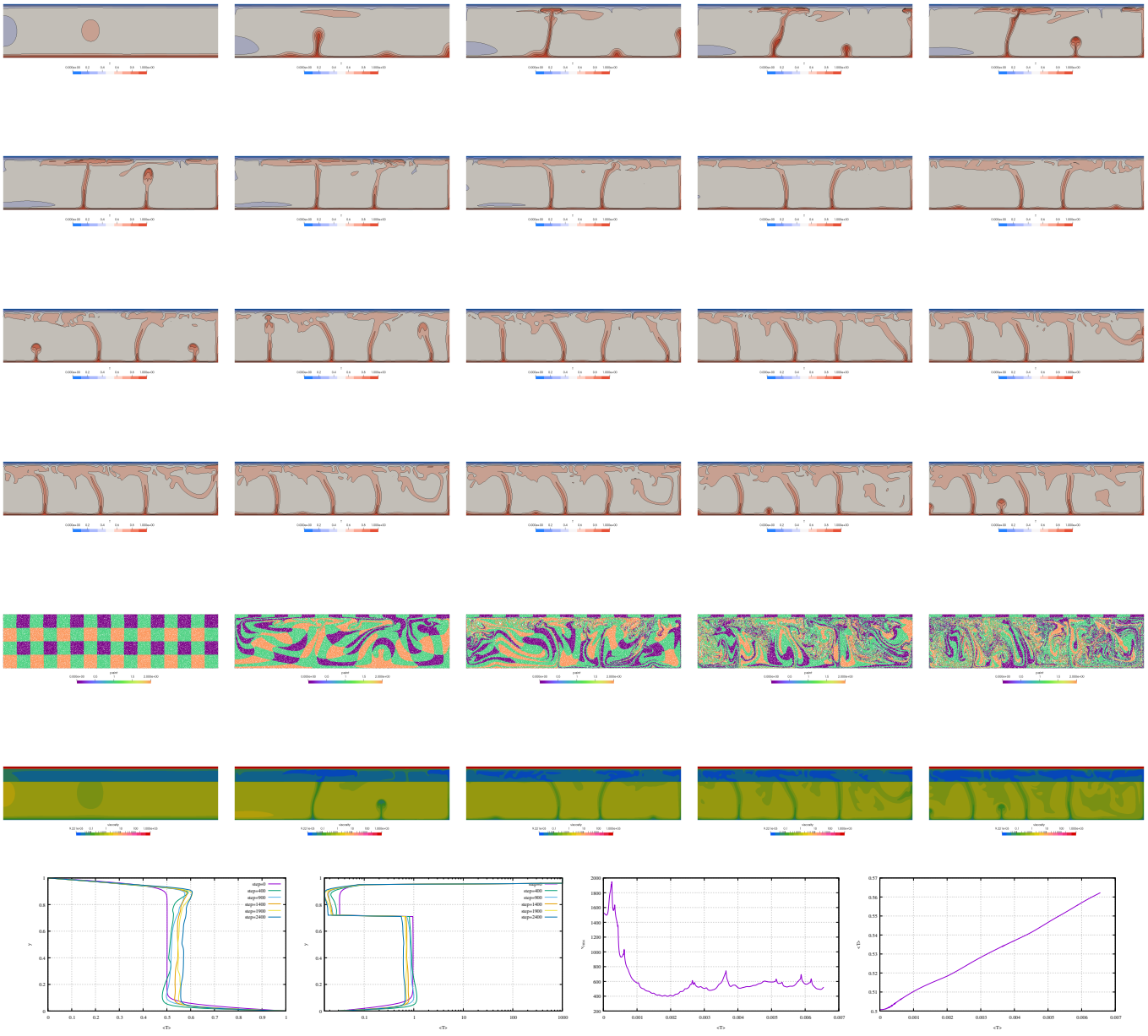
We can also explore a different rheology, as given in Brandenburg, Hauri, Keken, and Ballentine [34] (2008):

$$\eta(y,T) = \eta_0(y)\exp[-b(cT)]$$

with

$$\begin{cases} \eta_0(y) = 1000, c = 0 & y > 0.9551 \\ \eta_0(y) = 1, c = 1, & 0.7187 < y < 0.9551 \\ \eta_0(y) = 30, c = 1 & y \leq 0.0.7187 \end{cases}$$

and $b = \ln 1000$.



viscosity model 1. Temperature, markers and velocity fields. Resolution 200x50.

viscosity_model=3

A similar yet different rheology is found in Zhang, Zhong, Leng, and Li [366] (2010):

$$\eta(y,T) = \eta_0(y)\exp[A(0.5-T)]$$

with $A = 9.2103$. The expression for $\eta_0(r)$ is given in the text (section 3.1): "The depth-dependent prefactors $\eta_0(r)$ in equation (9) for the lithosphere and the upper mantle are 1 and 1/30, respectively. For the lower mantle, $\eta_0(r)$ increases linearly from 2.0 at the 670 km depth to 6.8 at the CMB. This leads to a mantle viscosity structure in which the lower mantle viscosity on average is ~ 2 orders of magnitude higher than that in the upper mantle." Note that this corresponds to case FS1 in the paper.

The base of the lithosphere is set to 120 km, so its dimensionless vertical coordinate is

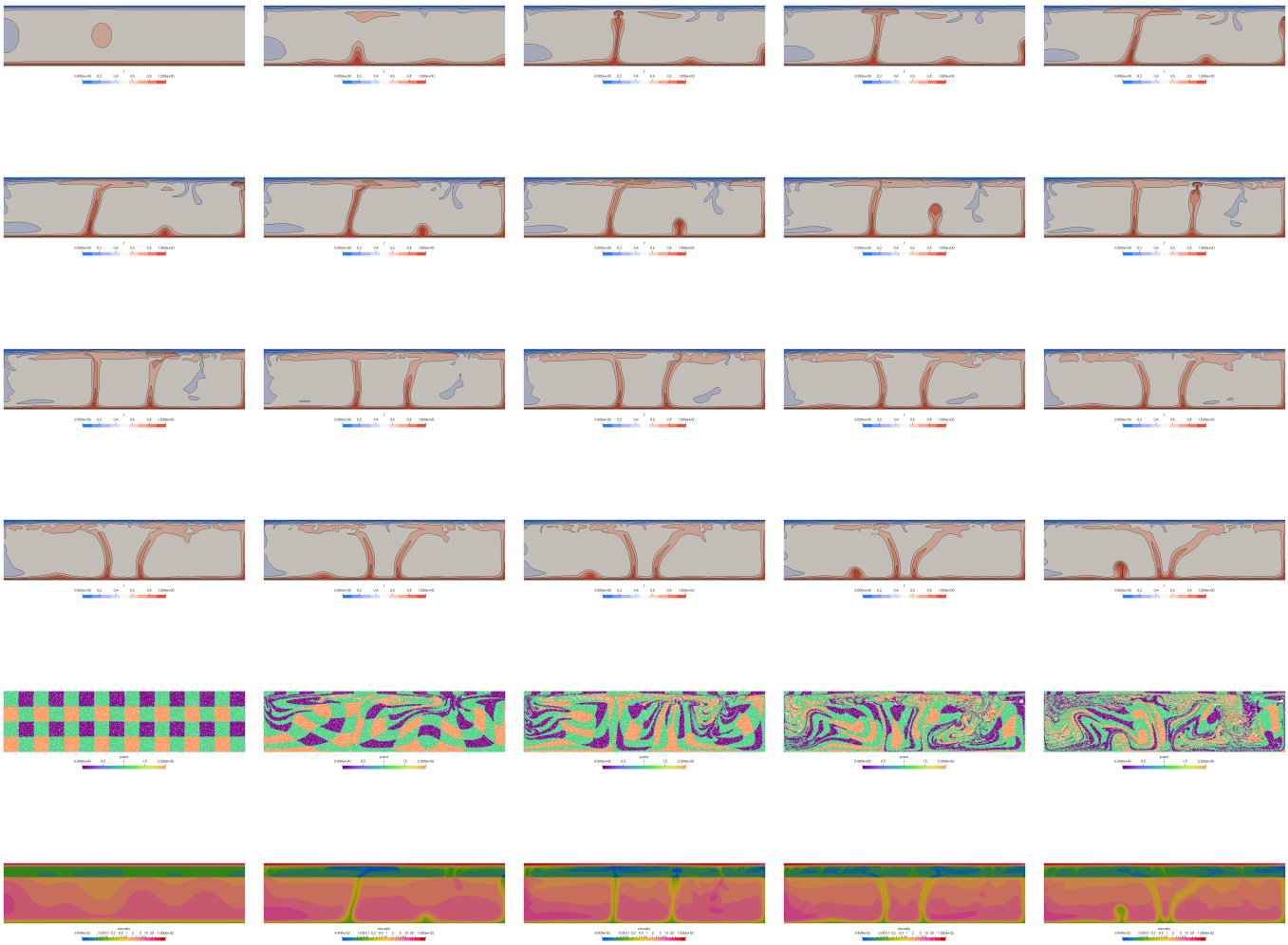
$$y_l = \frac{2891-120}{2891} \simeq 0.9585$$

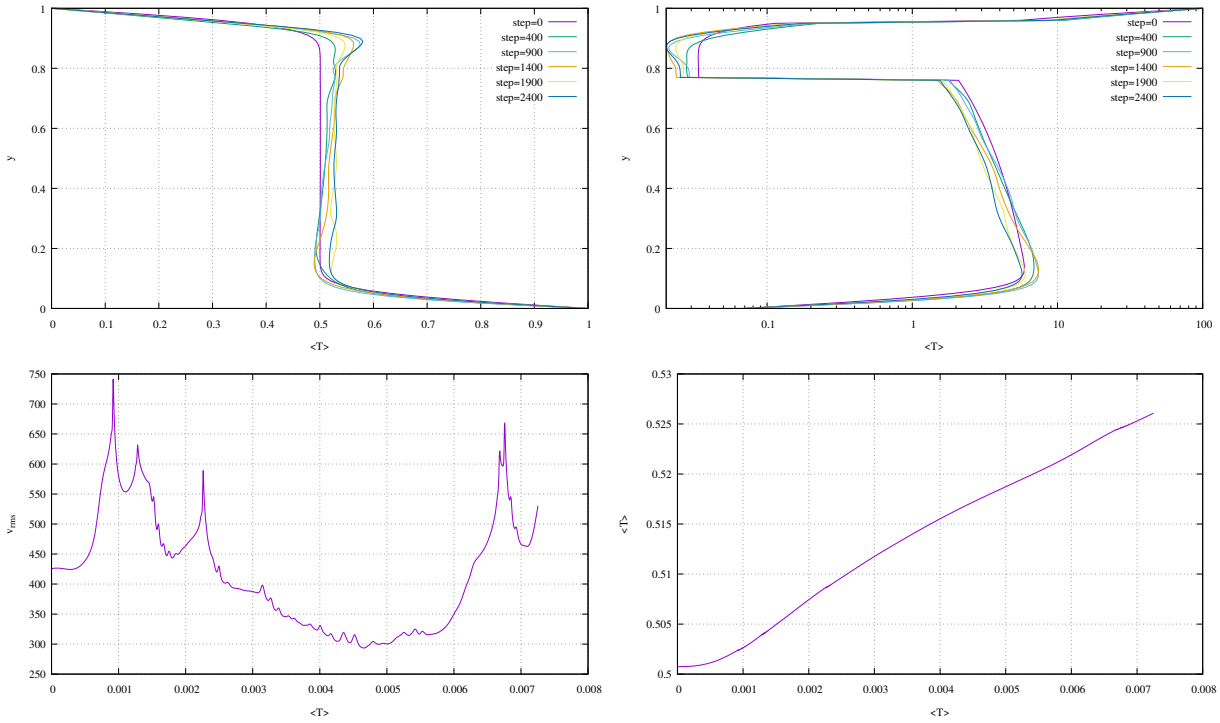
The base of the upper mantle is set to 670 km, so its dimensionless vertical coordinate is

$$y_{um} = \frac{2891-670}{2891} \simeq 0.7682$$

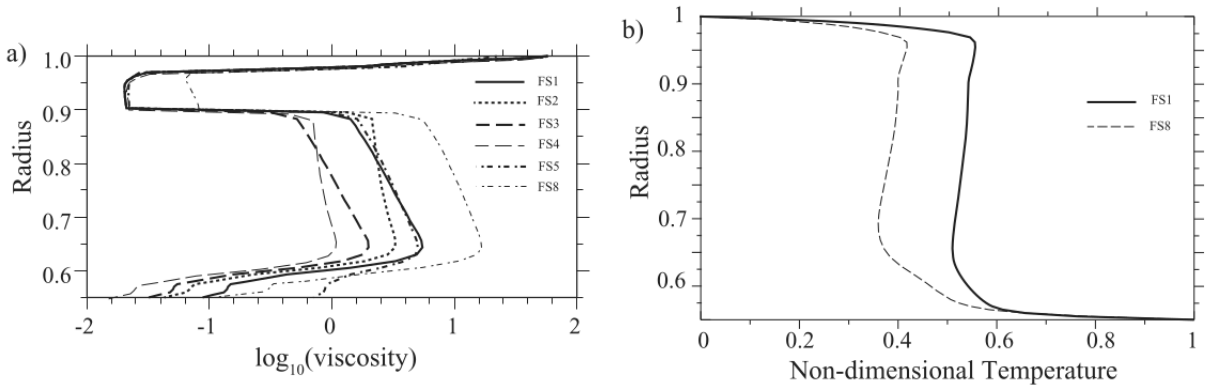
Finally the expression for $\eta_0(r)$ in the lower mantle is

$$\eta_0(y) = -\frac{6.8-2}{y_{um}}y + 6.8 = -6.24837y + 6.8$$





viscosity model 3. Temperature, markers and velocity fields. Resolution 200x50.



Taken from Zhang, Zhong, Leng, and Li [366] (2010). Depth dependences of horizontally averaged (a) mantle viscosities for cases FS1, FS2, FS3, FS4, FS5, and FS8 and (b) mantle temperatures for cases FS1 and FS8.

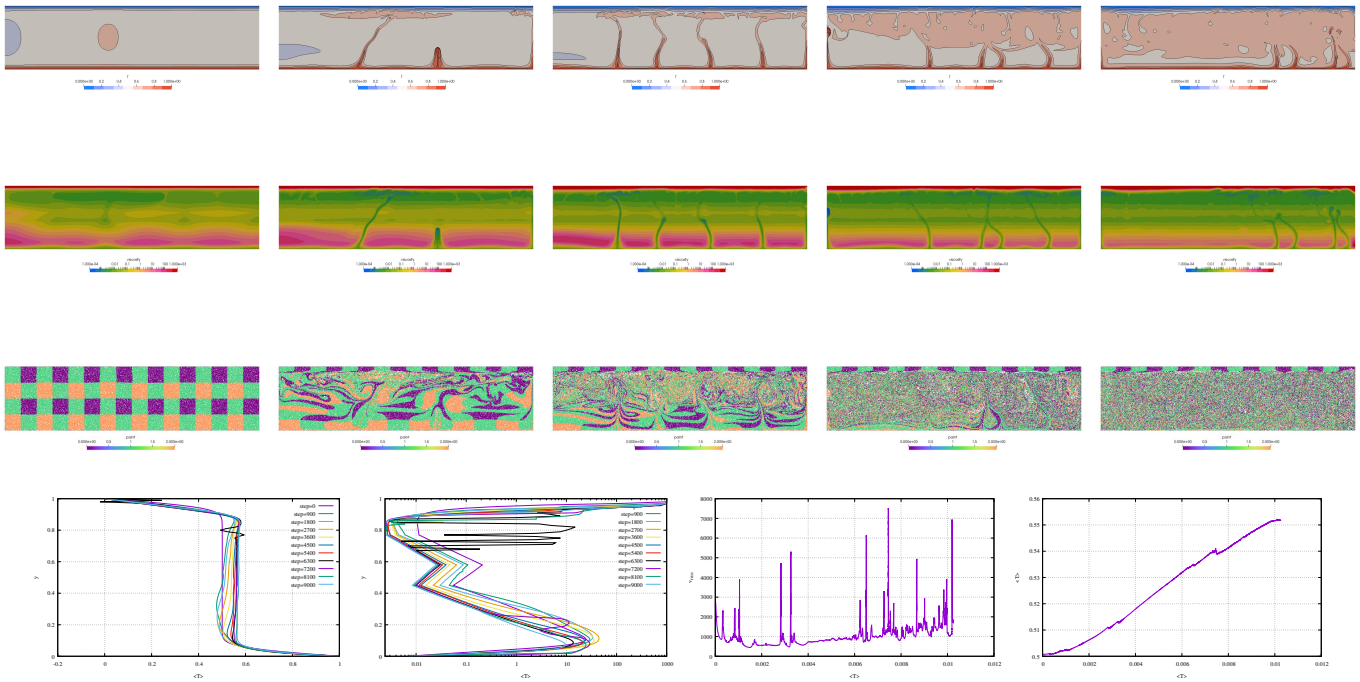
viscosity_model=4

From Matyska, Yuen, Wentzcovitch, and Čížková [225] (2011) we get:

$$\eta = \eta_{ref}\left[\exp 12.66\left(\frac{G(z)}{0.15+1.7T}-1\right)\right]$$

where T is the dimensionless temperature equal to 0 at the surface $z = 0$ and equal to 1 at the CMB $z = 1$. The non-monotonic depth-dependence of the activation energy is approximated by a simple piece-wise linear function

$$\begin{aligned}
G(z) &= 1 && \text{for } 0.00 \leq && z \leq 0.23\\
G(z) &= 1.0 + 0.2(z-0.23)/0.19 && \text{for } 0.23 \leq && z \leq 0.42\\
G(z) &= 1.2 - 0.1(z-0.42)/0.13 && \text{for } 0.42 \leq && z \leq 0.55\\
G(z) &= 1.1 + 0.7(z-0.55)/0.45 && \text{for } 0.55 \leq && z \leq 1.00
\end{aligned}$$



viscosity model 4. Temperature, markers and velocity fields. Resolution 200x50.

Problems & ToDos:
pressure nullspace
the code can do 4 diff FE pairs, use them!
nonlinear iterations missing
run plenty of models with 4 elements, diff resolutions and see when/how they differ? implement vrms as a fct of depth a la [194]

Stone 89

on the topic of integrated deformation

2D
box
Q1
Q2
analytical_benchmark
time_stepping
paraview_output

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_89`

This stone was developed in collaboration with Taco Broerse.

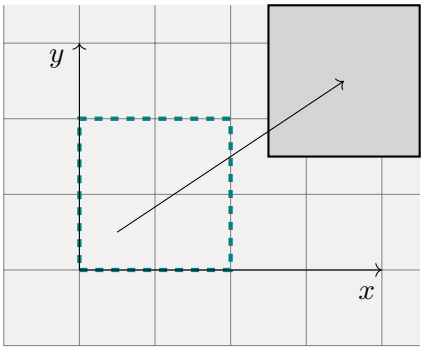
What follows (and more particularly the concrete examples of deformations) is borrowed from the excellent site `https://www.continuummechanics.org/`.

Let us start by defining the deformed vector $\vec{x}$ and the reference vector $\vec{X}$. We assume that deformation occurs and that it transforms $\vec{X}$ into $\vec{x}$.

Some simple examples

Rigid body displacement For instance:

$$x = X + 2.5 \tag{89.1}$$
$$y = Y + 1.5 \tag{89.2}$$



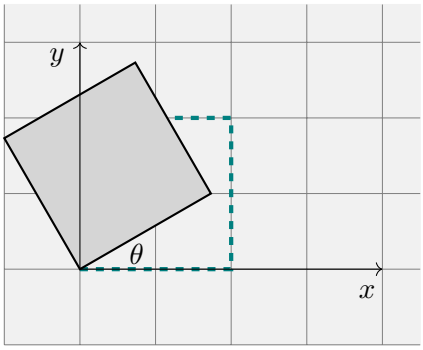
There is no deformation to speak of since the body remains the same but translated. The equation above also writes

$$\begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}_{\boldsymbol{F}} \cdot \begin{pmatrix} X \\ Y \end{pmatrix} + \begin{pmatrix} 2.5 \\ 1.5 \end{pmatrix}$$

We will see the significance of the matrix $\boldsymbol{F}$ in what follows.

Rigid body rotation For instance:

$$x = X\cos\theta - Y\sin\theta \tag{89.3}$$
$$y = X\sin\theta + Y\cos\theta \tag{89.4}$$



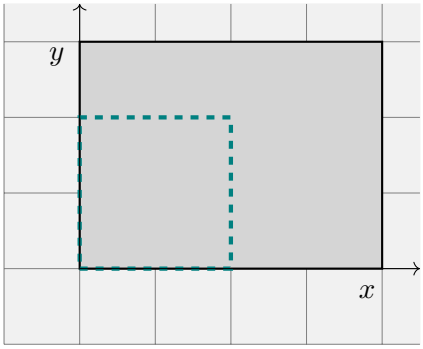
This can be rewritten as $\vec{x} = \boldsymbol{F}\cdot\vec{X}$, where $\boldsymbol{F}$ is a rotation matrix counter-clockwise about the z axis (perpendicular to the page):

$$\boldsymbol{F} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Actually the body is not deformed, but simply rotated. There is no strain associated to this transformation.

Stretching Let us now turn to a 'real' deformation with shape change: stretching. For instance:

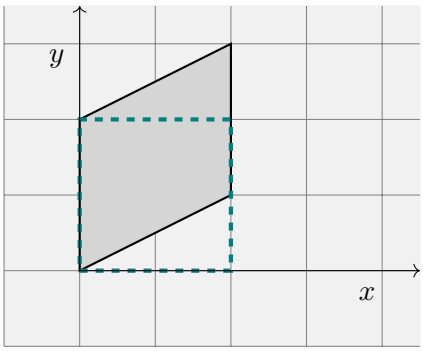
$$\begin{cases} x = 2X \\ y = 1.5Y \end{cases} \quad\Leftrightarrow\quad \boldsymbol{F} = \begin{pmatrix} 2 & 0 \\ 0 & 1.5 \end{pmatrix} \tag{89.5}$$



This deformation has increased the area of the body by a factor 3.

Shearing Another possible deformation (and a very relevant one in structural geology) is shearing:

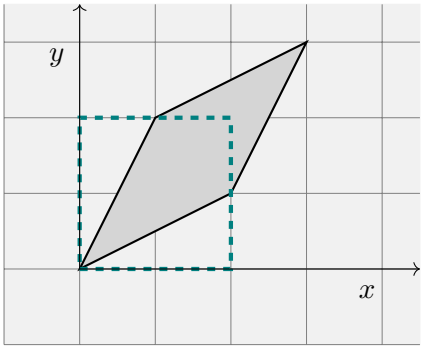
$$\begin{cases} x = X \\ y = \frac{1}{2}X + Y \end{cases} \quad\Leftrightarrow\quad \boldsymbol{F} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix} \tag{89.6}$$



This is a case of simple shear: a deformation in which parallel planes in a material remain parallel and maintain a constant distance, while translating relative to each other. We see that the area of the body is conserved but not its shape.

Let us now consider this shear example:

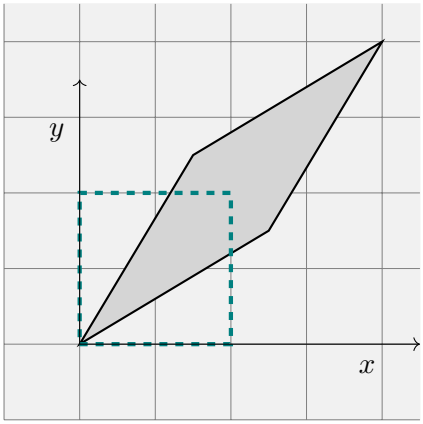
$$\begin{cases} x &= X + \frac{1}{2}Y \\ y &= \frac{1}{2}X + Y \end{cases} \quad \Leftrightarrow \quad \boldsymbol{F} = \begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix} \tag{89.7}$$



Here too the shape changes and we obtain a parallelogram. However, the area is not conserved since the area of the undeformed square is 4, while the area of the parallelogram is 3 (Split it in two triangles. The area of the triangle inside the square is 4-1-1-0.5).

So while the expression for $\boldsymbol{F}$ above may superficially resemble pure shear (irrotational, area preserving shear), it is not. To describe pure shear with $\boldsymbol{F}$, the diagonal terms are no longer 1. The following shear example does describe pure shear:

$$\begin{cases} x &= 1.25X + 0.75Y \\ y &= 0.75X + 1.25Y \end{cases} \quad \Leftrightarrow \quad \boldsymbol{F} = \begin{pmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix} \tag{89.8}$$



The area of the resulting parallelogram is still 1, so this deformation conserves volume.

Polar decomposition of $\boldsymbol{F}$

A polar decomposition separates $\boldsymbol{F}$ into a rotation and a stretching of the space along a set of orthogonal axes[1].

For instance, let us now consider the following transformation:

$$\begin{cases} x &= 1.3X - 0.375Y \\ y &= 0.75X + 0.65Y \end{cases} \quad \Leftrightarrow \quad \boldsymbol{F} = \begin{pmatrix} 1.3 & -0.375 \\ 0.75 & 0.650 \end{pmatrix} \tag{89.9}$$

[1] `https://en.wikipedia.org/wiki/Polar_decomposition`

It can be decomposed as

$$\boldsymbol{F} = \boldsymbol{R} \cdot \boldsymbol{U} = \begin{pmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{pmatrix} \cdot \begin{pmatrix} 1.5 & 0 \\ 0 & 0.75 \end{pmatrix}$$

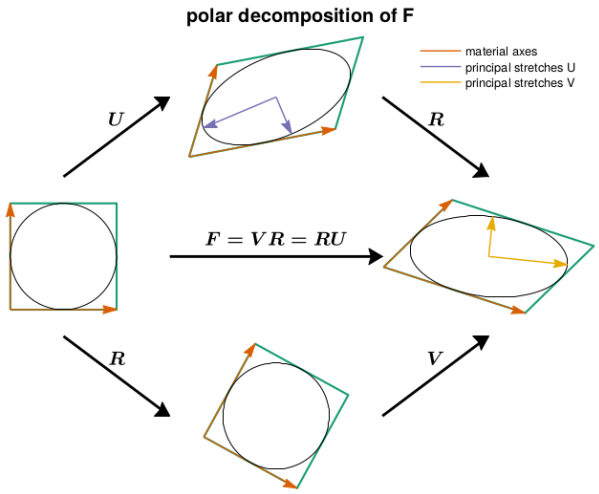
where $\boldsymbol{R}$ is the aforementioned rotation, $\boldsymbol{U}$ is the right stretch tensor, or as

$$\boldsymbol{F} = \boldsymbol{V} \cdot \boldsymbol{R} = \begin{pmatrix} 1.313 & 0.325 \\ 0.325 & 0.938 \end{pmatrix} \cdot \begin{pmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{pmatrix}$$

where $\boldsymbol{V}$ is the left stretch tensor. We see that $\boldsymbol{U} \neq \boldsymbol{V}$ but it can easily be proven that

$$\boldsymbol{V} = \boldsymbol{R} \cdot \boldsymbol{U} \cdot \boldsymbol{R}^T \qquad \text{and} \qquad \boldsymbol{U} = \boldsymbol{R}^T \cdot \boldsymbol{V} \cdot \boldsymbol{R}$$

Note that $\boldsymbol{V}$ and $\boldsymbol{U}$ are symmetric, and have positive eigenvalues, and the eigenvalues correspond to stretches in $[0, \infty]$. These stretches then form the axes of a strain ellipse. U and V have furthermore the same eigenvalues, but their eigenvectors are rotated with regards to each other since $\boldsymbol{U} = \boldsymbol{R}^T \cdot \boldsymbol{V} \cdot \boldsymbol{R}$ (which is basically a coordinate rotation).



Taken from Broerse et al, subm. Sketch of the polar decomposition of the deformation gradient $\boldsymbol{F}$ into an orthogonal rotation R and symmetric stretch tensors V or U. Here we apply the deformation of F to an initially square material element, with side vectors $d\vec{X}$. The deformed configuration (right) leads to an element with side vectors $dx = FdX$. The decomposition can be written as first a stretch U followed by a rotation R, or first a rotation R followed by a stretch V. Principal stretches $\vec{\lambda}$ of V provide the stretch in the final, deformed, configuration, whereas stretches $\vec{\lambda}$ of U provide the stretch in the original configuration. Material axes provide reference for the reader. We also show strain ellipses in black, that can be constructed using circle coordinates as dX. Because V and U are symmetric tensors, the rotation angle of R is non-zero for simple shear contributions.

- The actual decomposition algorithm is not trivial, even for 2×2 matrices. This is nicely documented online[2]. For instance, coming back to the simple shear example of Eq. (89.6), the tensor $\boldsymbol{F}$ can be written

$$\boldsymbol{F} = \boldsymbol{R} \cdot \boldsymbol{U} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} U_{xx} & U_{xy} \\ U_{yx} & U_{yy} \end{pmatrix}$$

  We must then determine U_{ij} and θ which is, once again, not trivial. We then use the method of Hoger & Carlson(1984) [157] (see note in a couple of pages further). We start by computing

$$\boldsymbol{C} = \boldsymbol{F}^T \cdot \boldsymbol{F} = \begin{pmatrix} 1 & 1/2 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 1/2 & 1 \end{pmatrix} = \begin{pmatrix} 5/4 & 1/2 \\ 1/2 & 1 \end{pmatrix}$$

  We then proceed by computing the 1st and 2nd invariants of $\boldsymbol{C}$:

$$\begin{aligned} I_C &= C_{xx} + C_{yy} = 5/4 + 1 = 9/4 \\ II_C &= C_{xx}C_{yy} - C_{xy}C_{yx} = 5/4 - 1/4 = 1 \end{aligned}$$

[2] `http://www.continuummechanics.org/polardecomposition.html`

and of $\boldsymbol{U}$

$$\begin{aligned} I_U &= \sqrt{I_C + 2\sqrt{II_C}} = \sqrt{9/4+2} = \frac{1}{2}\sqrt{17} \\ II_U &= \sqrt{II_C} = 1 \end{aligned}$$

Then the right stretch tensor is:

$$\boldsymbol{U} = \frac{1}{I_U}(\boldsymbol{C} + II_U\boldsymbol{I}) = \frac{2}{\sqrt{17}}\left[\begin{pmatrix} 5/4 & 1/2 \\ 1/2 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right] = \frac{2}{\sqrt{17}}\begin{pmatrix} 9/4 & 1/2 \\ 1/2 & 2 \end{pmatrix}$$

The inverse is simply:

$$\boldsymbol{U}^{-1} = \frac{1}{U_{xx}U_{yy} - U_{xy}U_{yx}}\begin{pmatrix} U_{yy} & -U_{xy} \\ -U_{yx} & U_{xx} \end{pmatrix} = \frac{2}{\sqrt{17}}\begin{pmatrix} 2 & -1/2 \\ -1/2 & 9/4 \end{pmatrix}$$

Finally

$$\boldsymbol{R} = \boldsymbol{F}\cdot\boldsymbol{U}^{-1} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix}\cdot\frac{2}{\sqrt{17}}\begin{pmatrix} 2 & -1/2 \\ -1/2 & 9/4 \end{pmatrix} = \frac{2}{\sqrt{17}}\begin{pmatrix} 2 & -1/2 \\ 1/2 & 2 \end{pmatrix}$$

Finally

$$\theta = \arctan R_{yx}/R_{xx} = \arctan\frac{1}{4} \simeq 14^o$$

This is very important since it means that simple shear deformation generates rotation!

It is also worth noting that since $\theta = 14^o$, then the rotation matrix is

$$\boldsymbol{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \simeq \begin{pmatrix} 0.9701 & -0.2425 \\ 0.2425 & 0.9701 \end{pmatrix} \simeq \boldsymbol{I} + \begin{pmatrix} 0 & -0.2425 \\ 0.2425 & 0 \end{pmatrix}$$

- Looking now at Eq. (89.8), then

$$\boldsymbol{C} = \boldsymbol{F}^T\cdot\boldsymbol{F} = \frac{1}{8}\begin{pmatrix} 17 & 15 \\ 15 & 17 \end{pmatrix}$$

We then proceed by computing the 1st and 2nd invariants of $\boldsymbol{C}$:

$$I_C = C_{xx} + C_{yy} = 17/4 \tag{89.10}$$

$$II_C = C_{xx}C_{yy} - C_{xy}C_{yx} = (17^2 - 15^2)/64 = 1 \tag{89.11}$$

and of $\boldsymbol{U}$

$$I_U = \sqrt{I_C + 2\sqrt{II_C}} = \sqrt{17/4 + \sqrt{1}} = \frac{5}{2} \tag{89.12}$$

$$II_U = \sqrt{II_C} = 1 \tag{89.13}$$

Then the right stretch tensor is:

$$\boldsymbol{U} = \frac{1}{I_U}(\boldsymbol{C} + II_U\boldsymbol{I}) = \frac{2}{5}\frac{1}{8}\begin{pmatrix} 17+8 & 15 \\ 15 & 17+8 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix}$$

Its inverse is

$$\boldsymbol{U}^{-1} = \frac{1}{4}\begin{pmatrix} 5 & -3 \\ -3 & 5 \end{pmatrix}$$

and finally

$$\boldsymbol{R} = \boldsymbol{F}\cdot\boldsymbol{U}^{-1} = \frac{1}{4}\begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix}\cdot\frac{1}{4}\begin{pmatrix} 5 & -3 \\ -3 & 5 \end{pmatrix} = \boldsymbol{I}$$

This corresponds to an angle $\theta = 0^o$, i.e. it is an irrotational deformation, and since we have shown that it conserves volume, it is pure shear.

Pure shear is an example of irrotational strain in which body is elongated in one direction while being shortened perpendicularly. Indeed, in this case we have extended the body along the (1,1) direction while squeezing it in the orthogonal (1,-1) direction. Pure shear is differentiated from simple shear in that pure shear involves no rigid body rotation.

The deformation gradient

There remains to formally define $\boldsymbol{F}$ as the deformation gradient. It is defined as :

$$\boxed{\boldsymbol{F} = (\vec{\nabla}_X \vec{x})^T = \left(\frac{\partial \vec{x}}{\partial \vec{X}}\right)^T = \begin{pmatrix} \frac{\partial x}{\partial X} & \frac{\partial y}{\partial X} \\ \frac{\partial x}{\partial Y} & \frac{\partial x}{\partial Y} \end{pmatrix}^T = \begin{pmatrix} \frac{\partial x}{\partial X} & \frac{\partial x}{\partial Y} \\ \frac{\partial y}{\partial X} & \frac{\partial x}{\partial Y} \end{pmatrix}}$$

The transpose is necessary and best explained through example. Looking at the first shearing example of Eq. (89.7) again, we have

$$\begin{cases} x &= X \\ y &= \frac{1}{2}X + Y \end{cases} \quad \Leftrightarrow \quad \boldsymbol{F} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix} \tag{89.14}$$

and

$$\vec{\nabla}_X \vec{x} = \begin{pmatrix} \frac{\partial x}{\partial X} & \frac{\partial y}{\partial X} \\ \frac{\partial x}{\partial Y} & \frac{\partial x}{\partial Y} \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{pmatrix}$$

which is indeed the transpose of $\boldsymbol{F}$.

We can write the displacement $\vec{u}$ as the difference between current and reference coordinates:

$$\vec{u} = \vec{x} - \vec{X}$$

and the deformation gradient can be reformulated as a function of the displacements:

$$\boldsymbol{F} = \frac{\partial \vec{x}}{\partial \vec{X}} = \frac{\partial}{\partial \vec{X}}(\vec{X} + \vec{u}) = \boldsymbol{I} + \frac{\partial \vec{u}}{\partial \vec{X}} = \boldsymbol{I} + \vec{\nabla}\vec{u}$$

From this it follows that the infinitesimal strain tensor can be written

$$\boxed{\boldsymbol{\varepsilon}(\vec{u}) = \frac{1}{2}(\vec{\nabla}\vec{u} + \vec{\nabla}\vec{u}^T) = \frac{1}{2}(\boldsymbol{F} + \boldsymbol{F}^T) - \boldsymbol{I}}$$

and the rotation tensor is then

$$\boldsymbol{\omega} = \frac{1}{2}(\vec{\nabla}\vec{u} - \vec{\nabla}\vec{u}^T) = \frac{1}{2}(\boldsymbol{F} - \boldsymbol{F}^T)$$

Purpose of this stone

This stone has two purposes:

- put the theory above into practice
- show that the 'standard' way of integrating strain on Lagrangian markers in geodynamical codes is merely an approximation and that this approximation no longer holds for large deformations/rotations. In other words, integrated strain rates lose their physical meaning when the deformation becomes large!

In all what follows we denote by 'old' the method by which strain is accumulated onto markers by time integration of the strain rate components. In practice here we interpolate $\vec{\nabla}\vec{\mathrm{v}}$ through Q_1 shape functions in the middle of the Lagrangian grid cells, we compute $\dot{\varepsilon}_{ij}$, we update the strain as $\varepsilon_{ij} += \dot{\varepsilon}_{ij}\delta t$. As we will show, this is fine only for (very) small total strains.

The 'new' method computes the marker strain by means of the deformation tensor. For each cell centroid, we compute the displacement gradient tensor $\boldsymbol{F}$, then carry out polar decomposition to write $\boldsymbol{F}$ as product of stretch and rotation, i.e. $\boldsymbol{F} = \boldsymbol{V} \cdot \boldsymbol{R}$. From $\boldsymbol{V}$ we compute the principal stretches $\lambda_{1,2}$ (eigenvalues of $\boldsymbol{V}$) and the eigenvectors as well since these represent the axes of the strain ellipse, i.e. the directions of maximum stretch. We then substract 1 to principal stretches to get principal strains, $\varepsilon_{1,2} = \lambda_{1,2} - 1$.

The 'new' method (which is not that new since it is to be found in Malvern (1969) [223] is as accurate as the measurement of $\boldsymbol{F}$ itself, and the difference tells us how bad the 'old' method actually can be (spoiler alert: up to few tens of %).

Polar decomposition a la Hoger & Carlson (1984)

The polar decomposition of $\boldsymbol{F}$ is not trivial, but an application of the Cayley-Hamilton theorem allows for this as explained in Hoger & Carlson (1984) [157].

The deformation gradient tensor $\boldsymbol{F}$ is computed in the middle of each cell at a given time t as follows:

$$\boldsymbol{F}(t) = \begin{pmatrix} F_{xx} & F_{xy} \\ F_{yx} & F_{yy} \end{pmatrix} = \boldsymbol{I} + \begin{pmatrix} \frac{\partial(x(t)-x_0)}{\partial x} & \frac{\partial(x(t)-x_0)}{\partial y} \\ \frac{\partial(y(t)-y_0)}{\partial x} & \frac{\partial(y(t)-y_0)}{\partial y} \end{pmatrix}$$

In what follows we omit the time dependence '(t)'. Each cell is assumed to be a Q_1 quadrilateral with a marker at each corner. We therefore store the initial position of markers, and make use of bilinear Q_1 shape functions to compute the derivatives above in the center of the cell.

1. The right Cauchy-Green deformation tensor is defined as

$$\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F} = (\boldsymbol{R}\cdot\boldsymbol{U})^T\cdot(\boldsymbol{R}\cdot\boldsymbol{U}) = \boldsymbol{U}^T\cdot\boldsymbol{R}^T\cdot\boldsymbol{R}\cdot\boldsymbol{U} = \boldsymbol{U}^T\cdot\boldsymbol{U}$$

2. We then find the eigenvalues of $\boldsymbol{C}$:

$$\mu_{1,2} = \frac{1}{2}(C_{xx}+C_{yy}) \pm \sqrt{\frac{1}{4}(C_{xx}-C_{yy})^2 + C_{xy}^2}$$

3. We compute the two invariants:

$$I_C = \mu_1 + \mu_2 \qquad II_C = \mu_1\mu_2$$

4. Compute invariants of right stretch tensor U

$$I_U = \sqrt{I_C + 2\sqrt{II_C}} \qquad II_U = \sqrt{II_C}$$

5. compute right stretch tensor $\boldsymbol{U}$

$$\boldsymbol{U} = \frac{\boldsymbol{C} + II_U\boldsymbol{I}}{I_U}$$

6. compute inverse of $\boldsymbol{U}$

$$\boldsymbol{U}^{-1} = -I_U\frac{\boldsymbol{C} - (II_U + I_C)\boldsymbol{I}}{II_U(II_U + I_C) + II_C}$$

7. compute rotation matrix $\boldsymbol{R} = \boldsymbol{F}\boldsymbol{U}^{-1}$

8. compute left stretch tensor $\boldsymbol{V} = \boldsymbol{F}\boldsymbol{R}^T$

9. Compute eigenvalues of $\boldsymbol{V}$, which are the square roots of those of $\boldsymbol{C}$

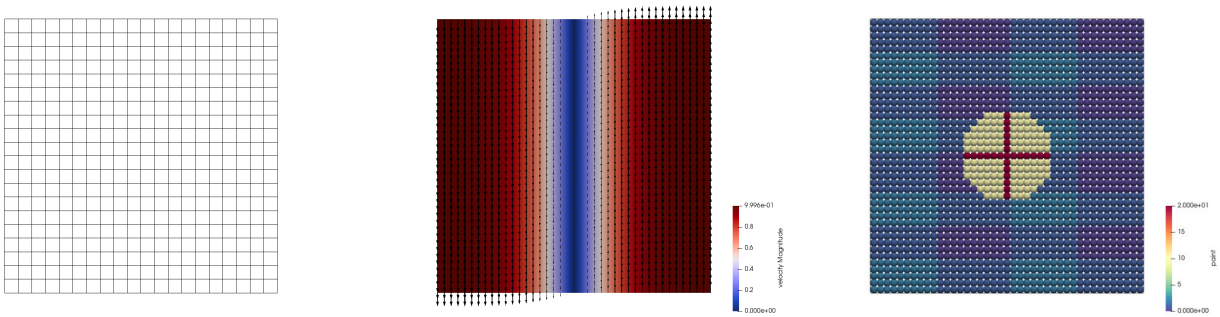
$$\lambda_1 = \sqrt{\mu_1} \qquad \lambda_2 = \sqrt{\mu_2}$$

The shear band (experiment=1)

The domain is a rectangle of size $L_x \times L_y$ discretised by nelx $\times$ nely Q_2 elements. The following velocity field is precribed in the domain:

$$u(x,y) = 0 \tag{89.15}$$
$$v(x,y) = \text{erf}[A(x - L_x/2)] \tag{89.16}$$

A swarm of 41×41 markers is then placed in the domain, and they are then painted as shown here:



Domain is unit square, resolution 20×20 elements. $A = 5$. Left: Eulerian grid; Middle: velocity field; Right: marker initial position.

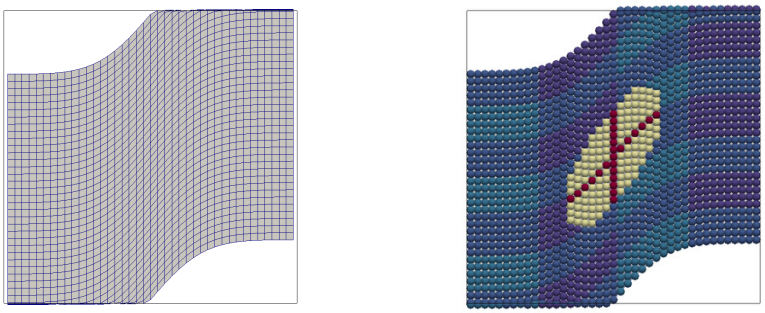
A time loop is implemented. At each time step the velocity field of the grid is interpolated onto each marker that is in turn advected with a simple euler step.

$$\vec{x}(t + \delta t) = \vec{x}(t) + \vec{\mathbf{v}}\ \delta t$$

Note that the timestep value δt is controlled by means of a CFL condition with $\mathcal{C} = 0.1$. Likewise, the components of the strain rate tensor are computed on each marker and used to update the strain on each marker:

$$\varepsilon_{ij}(t + \delta t) = \varepsilon_{ij}(t) + \dot{\varepsilon}_{ij}(t)\ \delta t$$

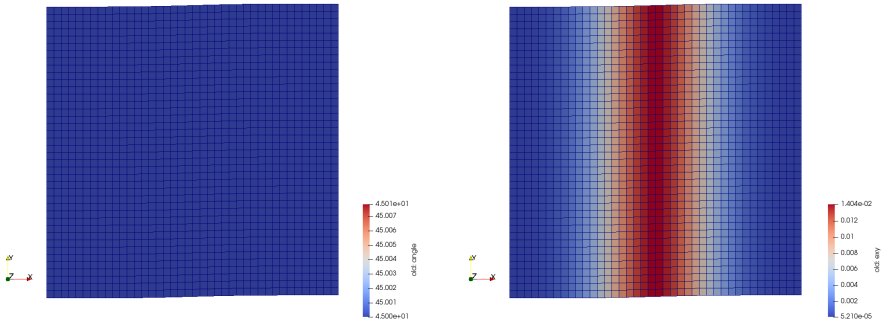
These markers form a Lagrangian mesh which deforms over time:

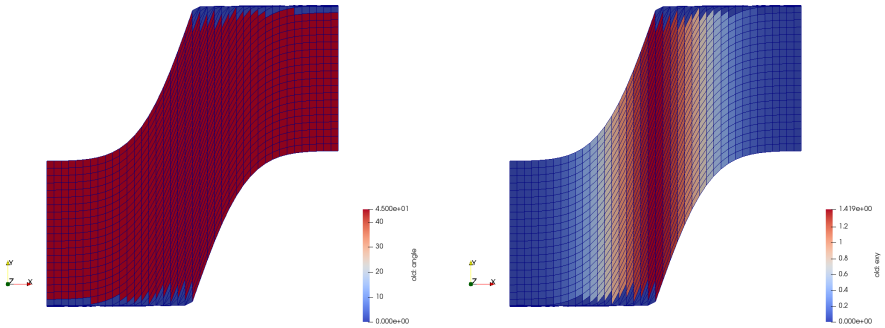


Lagrangian mesh after 50 time steps.

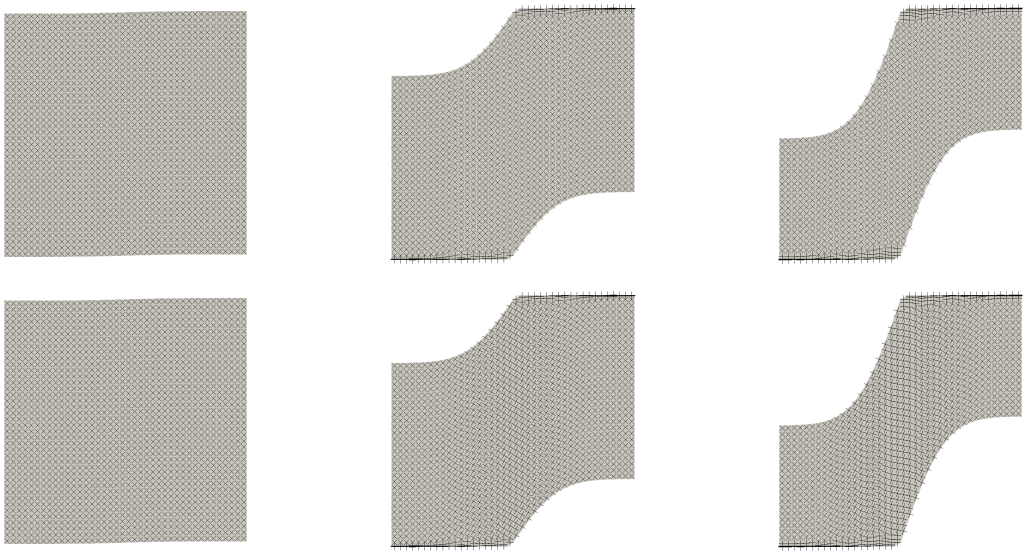
Markers which are advected outside of the domain are simply flagged inactive and these are no longer advected. If one marker of a cell is flagged inactive so is the cell itself.

At each time step the cumulative strain ε_{ij} components are computed at the center of the marker cells, as well as the principal strains ε_1 and ε_2 and the direction θ_ε of the principal strain:



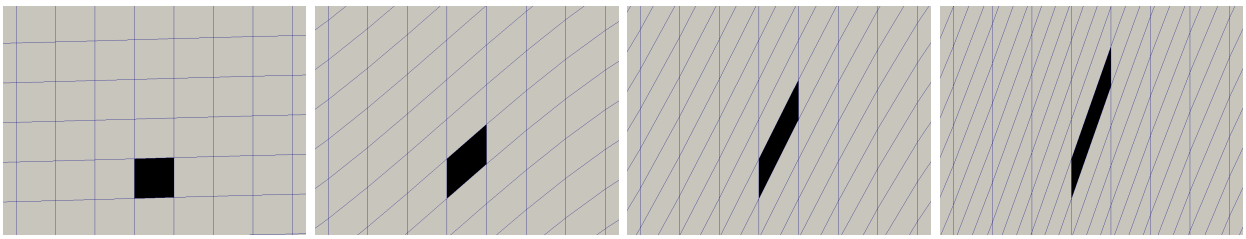


Left: principal strain directions, Right: ε_{xy}. Top row: first time step; Bottom row: 100th time step.

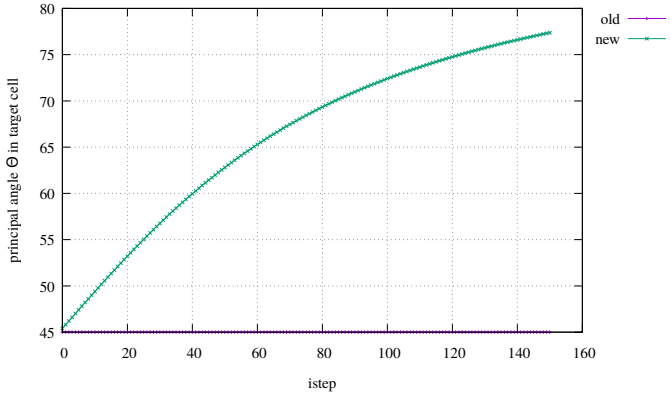


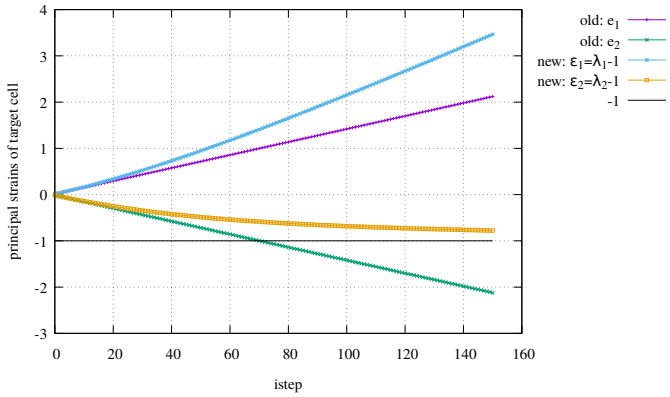
Directions of the principal strains. Top is old (strain rate integration), bottom is new (finite strain). We see that the old method does not record any rotation and the angles remain at 45°. The new method shows principal strains which align more and more with the direction of shearing towards the center.

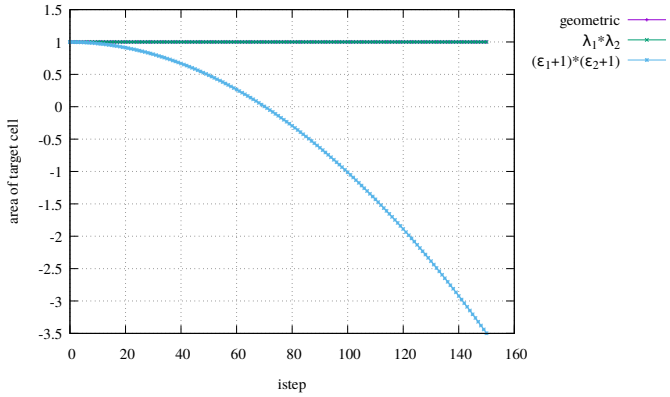
A single cell is chosen in the middle of the domain and its strain values recorded over time in a file:

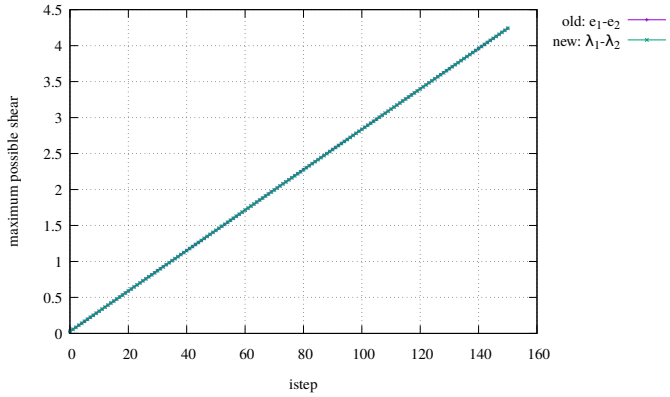


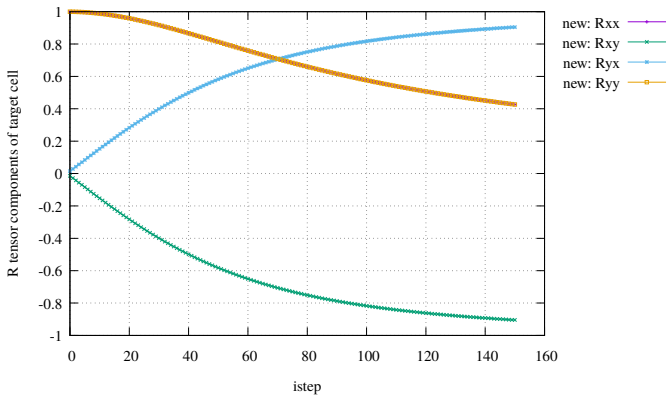
Cell undergoing deformation.

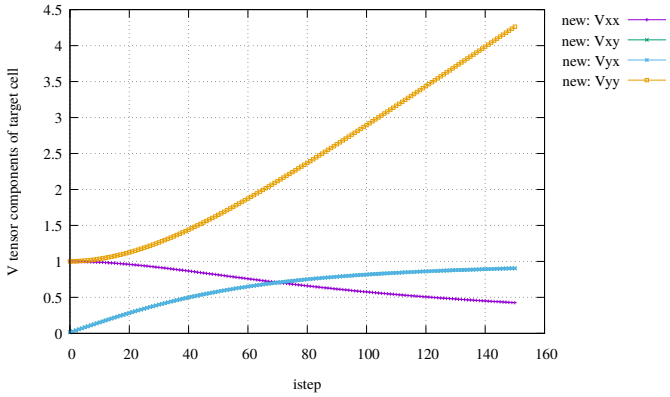



principal strains of target cell
istep
old: e1
old: e2
new: ε1=λ1-1
new: ε2=λ2-1
-1


area of target cell
istep
geometric
λ1*λ2
(ε1+1)*(ε2+1)


maximum possible shear
istep
old: e1-e2
new: λ1-λ2


R tensor components of target cell
istep
new: Rxx
new: Rxy
new: Ryx
new: Ryy



Time evolution of the principal angle θ_ε, the strain principal values, the relative area change of the target cell, the components of $\boldsymbol{R}$, the components of $\boldsymbol{V}$ and the maximum possible shear.

The vertical extension (experiment=2)

In this case the velocity field is prescribed to be:

$$u(x,y) = 0 \tag{89.17}$$
$$v(x,y) = y \tag{89.18}$$

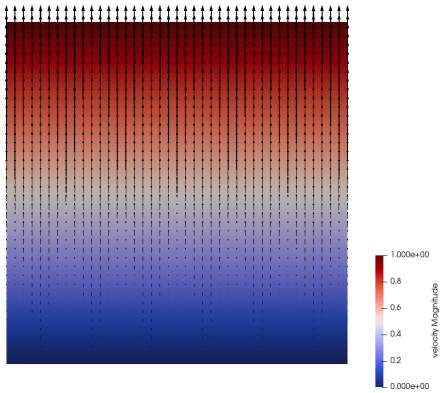
Since the divergence of this field is not zero, volume (area) is not conserved.

The components of the strain rate tensor are

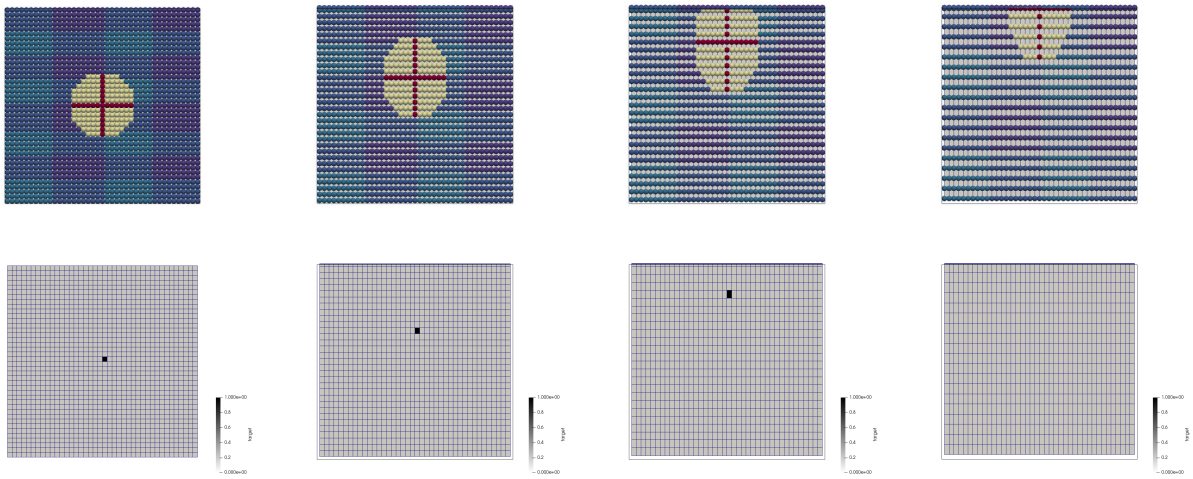
$$\dot{\varepsilon}_{xx} = 0 \qquad \dot{\varepsilon}_{yy} = 1 \qquad \dot{\varepsilon}_{xy} = 0$$

So that the components of the strain for the old method are given by:

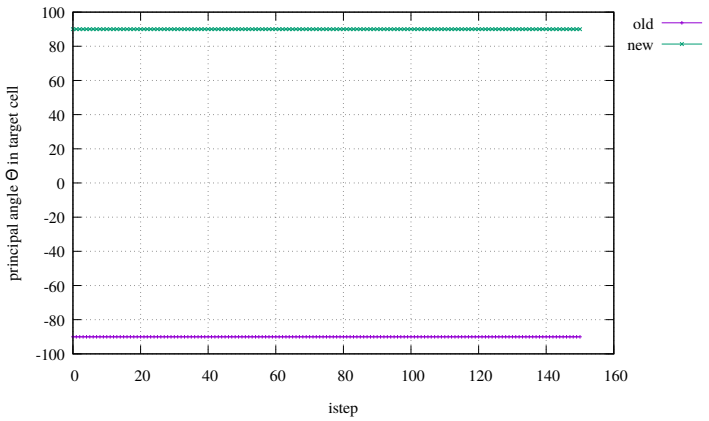
$$\varepsilon_{xx} = 0 \qquad \varepsilon_{yy} = nstep\ \delta t \qquad \varepsilon_{xy} = 0$$

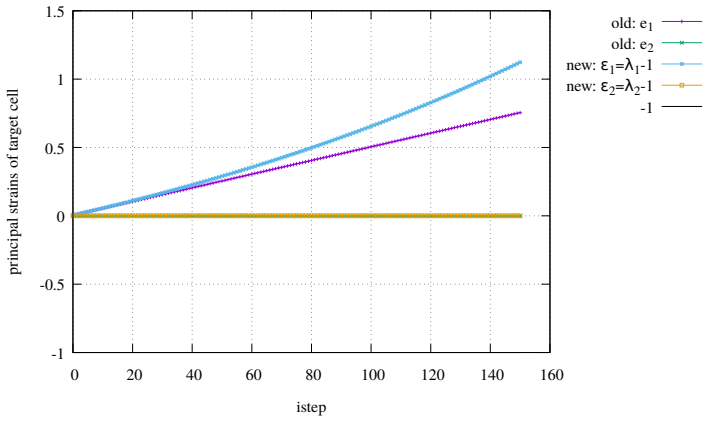


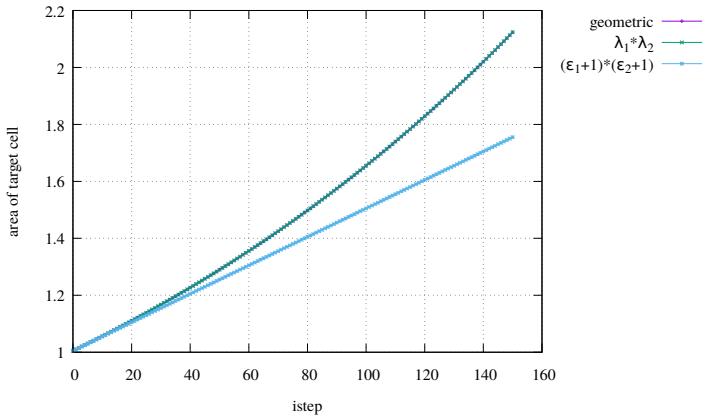
Velocity field on Eulerian mesh

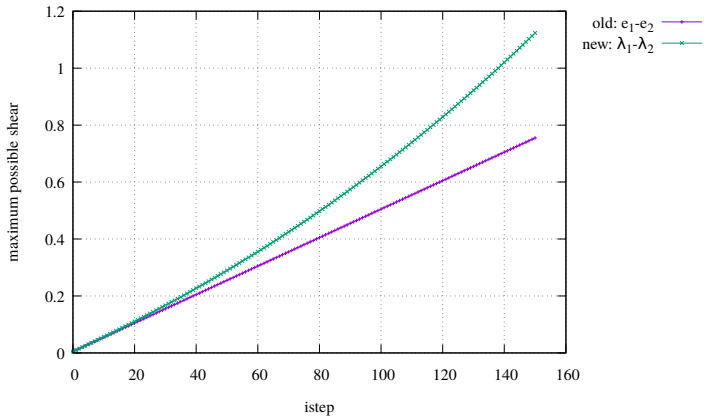


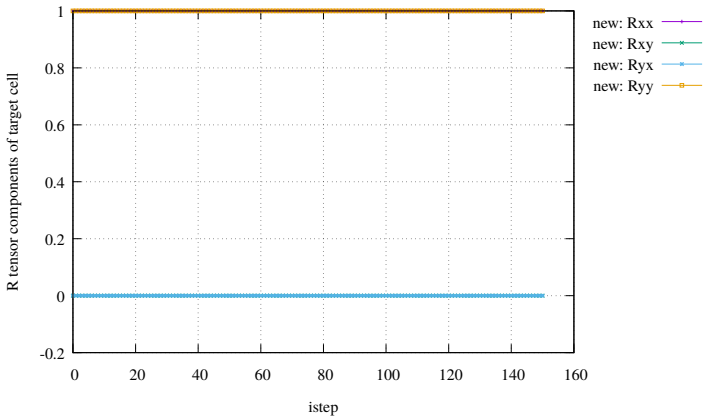
REDO

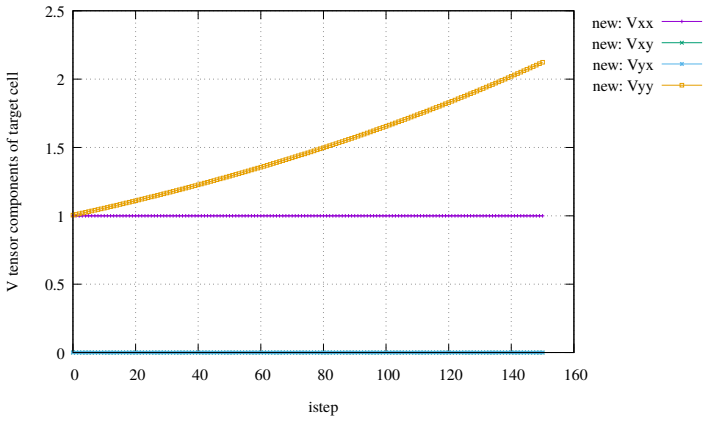



principal strains of target cell
istep
old: e1
old: e2
new: ε1=λ1-1
new: ε2=λ2-1
-1


area of target cell
istep
geometric
λ1*λ2
(ε1+1)*(ε2+1)


maximum possible shear
istep
old: e1-e2
new: λ1-λ2


R tensor components of target cell
istep
new: Rxx
new: Rxy
new: Ryx
new: Ryy



Time evolution of the principal angle θ_ε, the strain principal values, the relative area change of the target cell, the components of $\boldsymbol{R}$, the components of $\boldsymbol{V}$ and the maximum possible shear.

Pure shear (experiment=3)

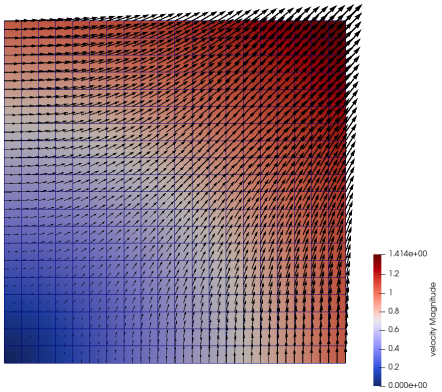
The velocity field is given by

$$\begin{aligned} u(x,y) &= x \\ v(x,y) &= y \end{aligned}$$

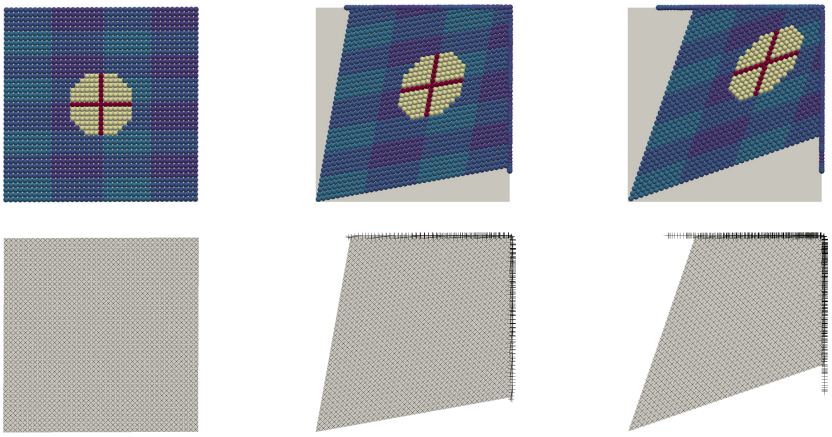
The components of the strain rate tensor are

$$\dot{\varepsilon}_{xx} = 1 \qquad \dot{\varepsilon}_{yy} = 1 \qquad \dot{\varepsilon}_{xy} = 0$$

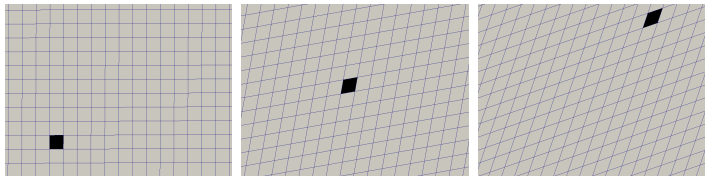
So that the components of the strain for the old method are given by:

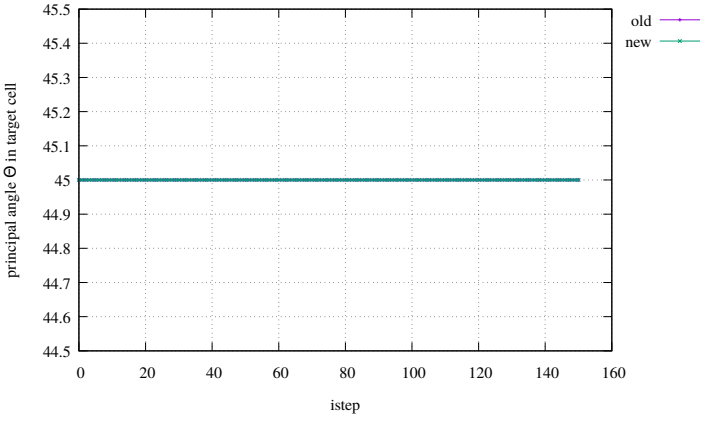


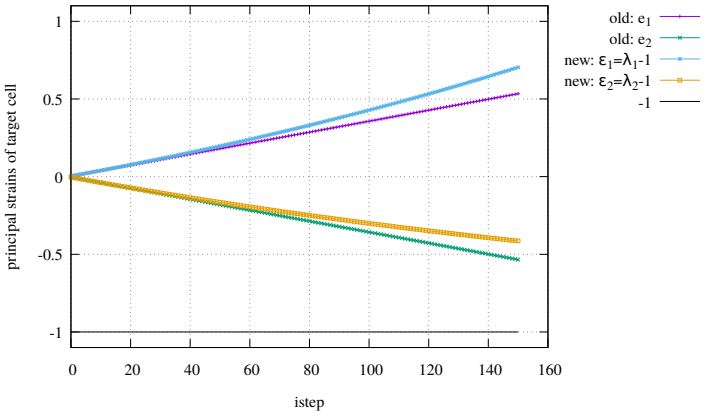
Velocity field on Eulerian mesh

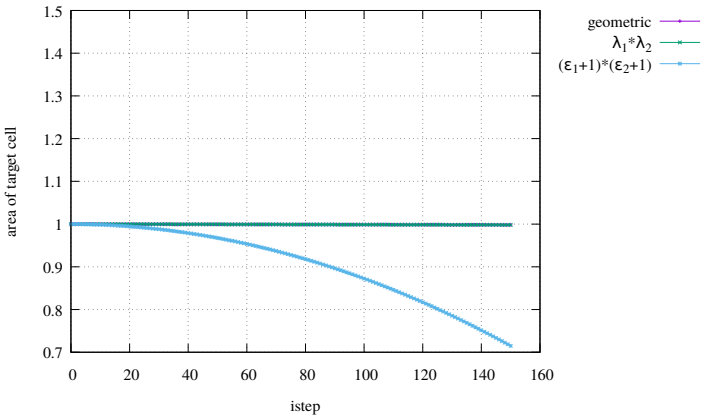


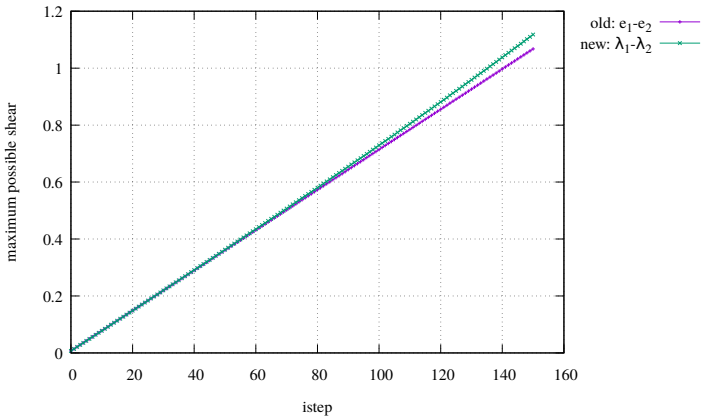
old and new are identical

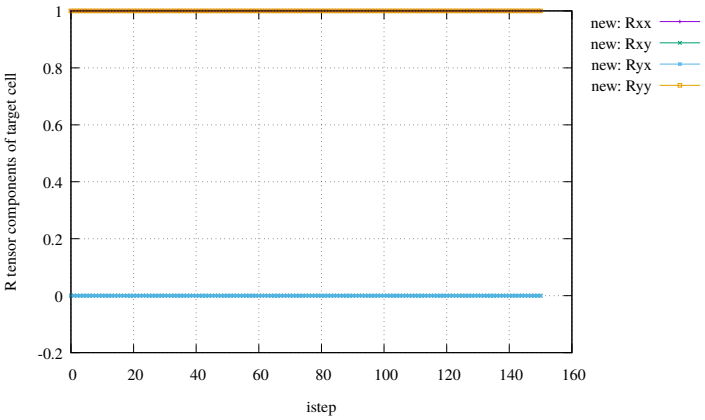


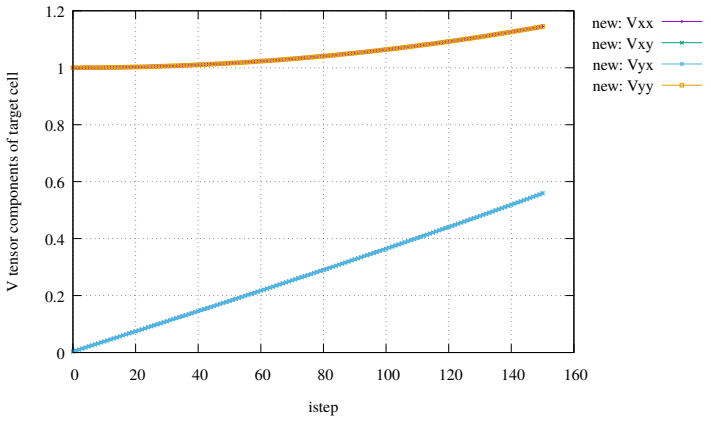



principal strains of target cell
istep
old: e1
old: e2
new: ε1=λ1-1
new: ε2=λ2-1
-1


area of target cell
istep
geometric
λ1*λ2
(ε1+1)*(ε2+1)


maximum possible shear
istep
old: e1-e2
new: λ1-λ2


R tensor components of target cell
istep
new: Rxx
new: Rxy
new: Ryx
new: Ryy



Time evolution of the principal angle θ_ε, the strain principal values, the relative area change of the target cell, the components of $\boldsymbol{R}$, the components of $\boldsymbol{V}$ and the maximum possible shear.

Bi-axial stretch (experiment=4)

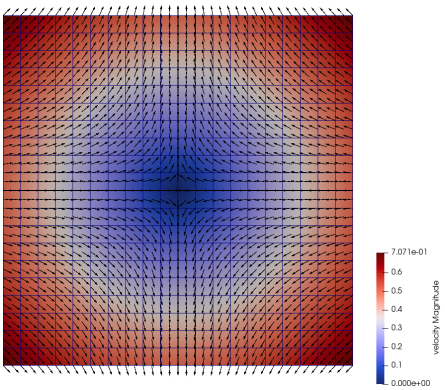
The velocity is given by

$$u(x,y) = -x + L_x/2 \tag{89.19}$$
$$v(x,y) = y - L_y/2 \tag{89.20}$$

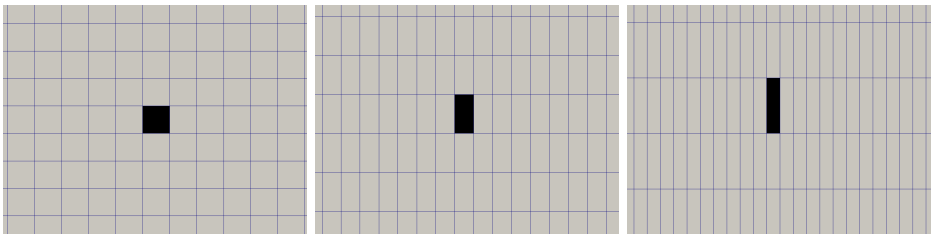
The components of the strain rate tensor are

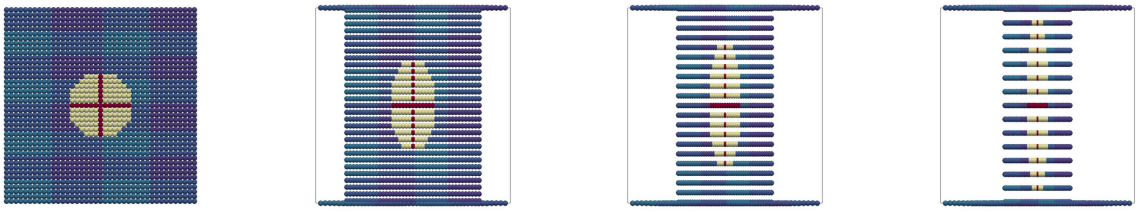
$$\dot{\varepsilon}_{xx} = -1 \qquad \dot{\varepsilon}_{yy} = 1 \qquad \dot{\varepsilon}_{xy} = 0$$

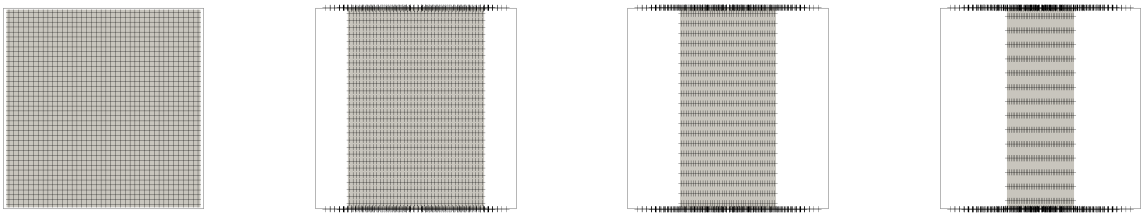
The velocity divergence is zero. The components of the strain for the old method are given by:

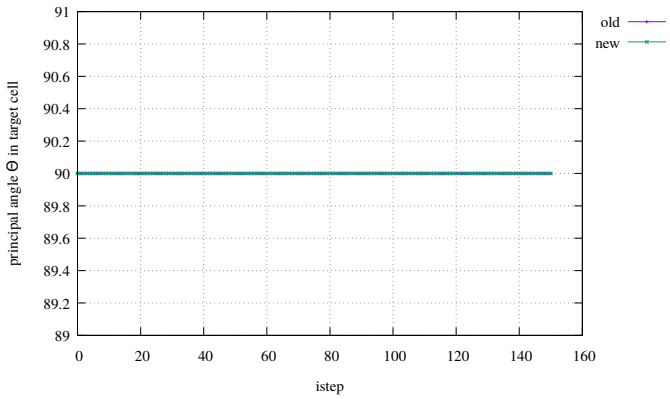


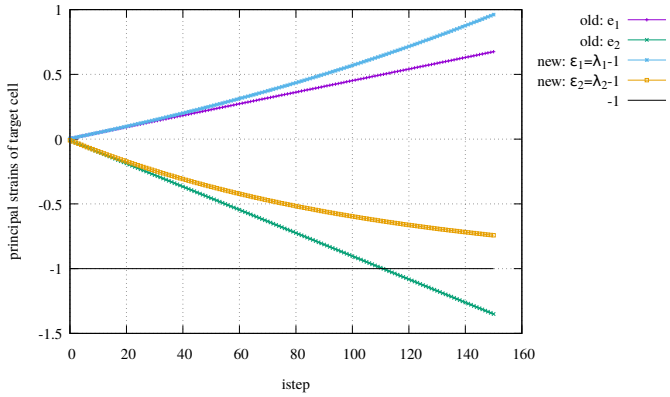
Velocity field on Eulerian mesh

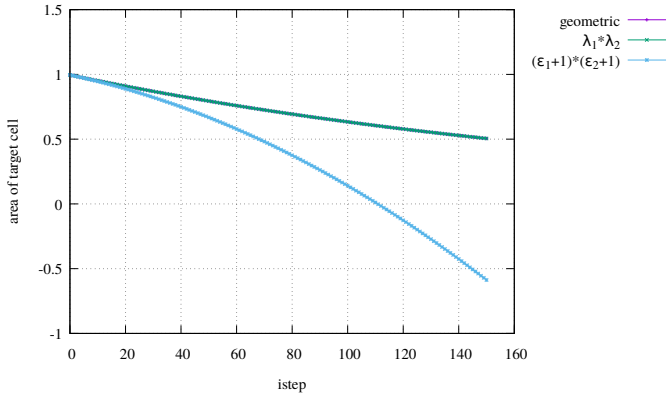


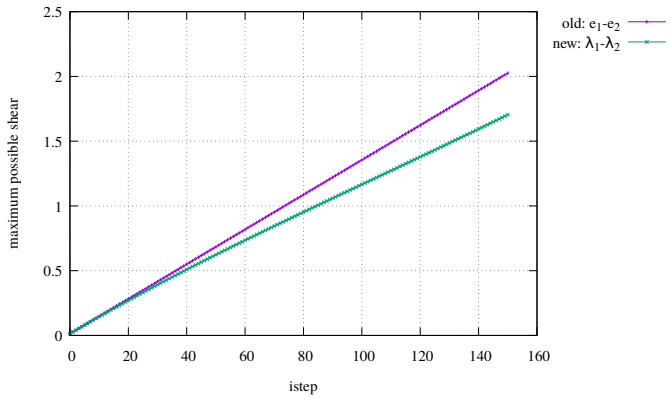


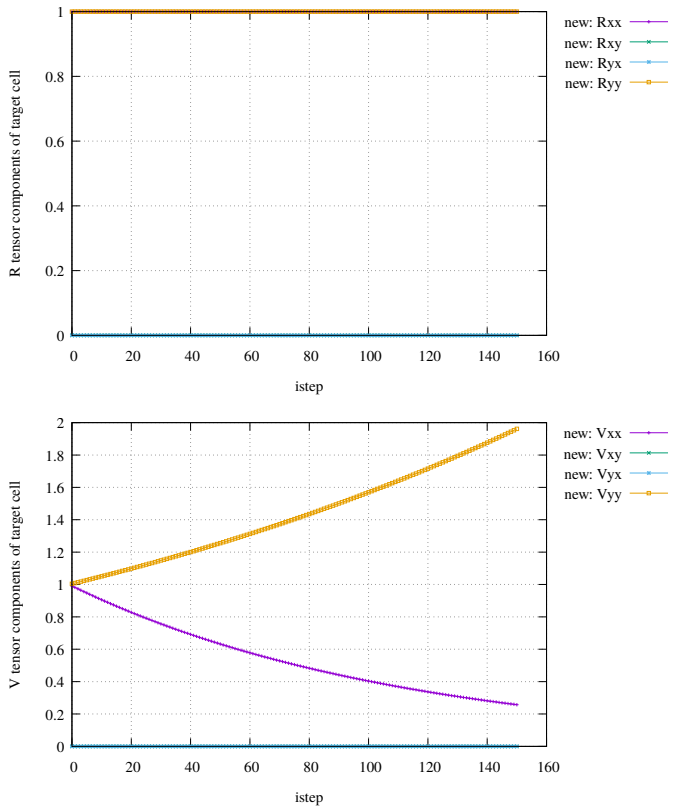



principal angle Θ in target cell
istep
old
new


principal strains of target cell
istep
old: e1
old: e2
new: ε1=λ1-1
new: ε2=λ2-1
-1


area of target cell
istep
geometric
λ1*λ2
(ε1+1)*(ε2+1)


maximum possible shear
istep
old: e1-e2
new: λ1-λ2



Time evolution of the principal angle θ_ε, the strain principal values and the area of the target cell.

PB: e2 cannot become less than -1

Discussion 'old' vs. 'new' cumulative strain

The first observation is that all quantities from the 'old' (integrated strain rates) and 'new' (strain computed from cumulative displacements) start at the same values. The 'old' cumulative strains are thus an acceptable approximation for small deformations. For increasing deformation a number of problems arise with the 'old' strains:

- principal axes do not rotate for simple shear (The shear band, experiment 1) while they should continuously change
- principal strain values deviate from the exact values (the 'new' principal strains) (all experiments)
- negative principal strains for the 'old' definition can decrease below the theoretical limit of -1 (The shear band, experiment 1; bi-axial stretch, experiment 4)
- related to the previous point, the area computed from these principal strains may incorrectly become negative (The shear band, experiment 1; bi-axial stretch, experiment 4)
- the maximum shear deviates from the correct value (all experiments except for experiment 1)

The problems in the integration of strain rates is due to two causes. Firstly, strain rates are determined relative to the current state (already deformed), which means that the state with respect the deformation is defined changes during integration. Rather, the 'new' strains based on the cumulative displacements are all references with respect to the original, undeformed, configuration. Secondly, because incremental principal axes change during general deformation that contains simple shear or rigid body rotation (which does not occur for the special cases of experiment 2, 3 and 4), incremental strains cannot be summed [223]. In the special cases where there is no rotation, strain rates may be integrated, and the result is a strain measure called natural strain or Hencky strain [223].

Relevant Literature: McKenzie [230] (1979), McKenzie and Jackson [231] (1983), Fossen and Tikoff [112] (1993), Tikoff and Fossen [331] (1993)

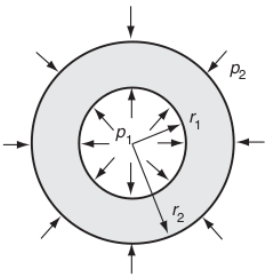
Stone 90

Thick-walled elastic cylinder under uniform boundary pressure (axisymmetric)

2D
box
Q1
compressible
analytical_benchmark
elastic_rheology
paraview_output
displacement_code
axisymmetric

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_90`

This axisymmetric elastic probem is common in the literature, see for instance Sadd [272]. A hollow thick-walled cylinder is placed under the action of uniform internal and external pressure loadings, as shown here:



Taken from Sadd [272]

Analytical solution

One can easily show that the stress components are such that

$$\sigma_{rr} = \frac{A}{r^2} + B \tag{90.1}$$

$$\sigma_{\theta\theta} = -\frac{A}{r^2} + B \tag{90.2}$$

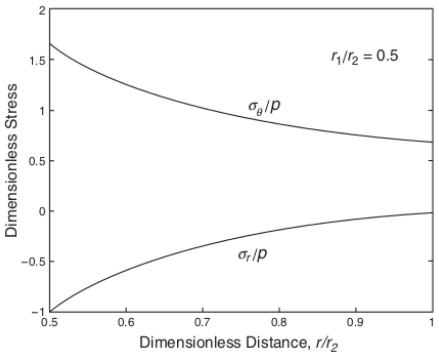
Applying the boundary conditions $\sigma_r(r_1) = -p_1$ and $\sigma_r = -p_2$ creates two equations for the two unknown constants A and B. Solving for these constants gives the result

$$A = \frac{r_1^2 r_2^2 (p_2 - p_1)}{r_2^2 - r_1^2} \tag{90.3}$$

$$B = \frac{r_1^2 p_1 - r_2^2 p_2)}{r_2^2 - r_1^2} \tag{90.4}$$

Substituting these values back into relations above gives the final result for the stress field.

Taking $r_1 = 0.5$, $r_2 = 1$, $p_2 = 0$ and $p_1 = p$, we can plot these:



Taken from Sadd [272]

Using the strain-displacement relations and Hooke's law, the radial displacement is easily determined as

$$u_r = \frac{1+\nu}{E} r \left[(1-2\nu)B - \frac{A}{r^2} \right]$$

Looking at the solution, we see that the stress field does not depend on the elastic constants, although the resulting displacements do depend on both E and ν.

$$\varepsilon_{rr} = \frac{\partial u_r}{\partial r} = \frac{1+\nu}{E} \left[(1-2\nu)B + \frac{A}{r^2} \right] \tag{90.5}$$

$$\varepsilon_{\theta\theta} = \frac{1}{r}\frac{\partial u_\theta}{\partial \theta} + \frac{u_r}{r} = \frac{u_r}{r} = \frac{1+\nu}{E} \left[(1-2\nu)B - \frac{A}{r^2} \right] \tag{90.6}$$

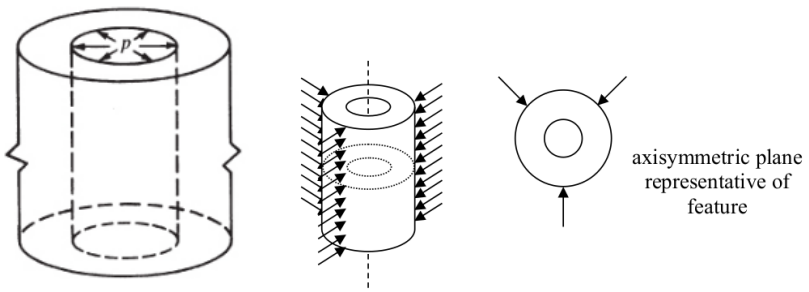
$$\varepsilon_{zz} = \frac{\partial u_z}{\partial z} = 0 \tag{90.7}$$

It then follows that

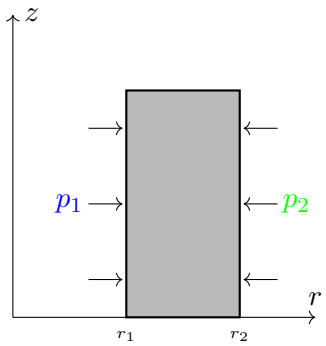
$$\sigma_{zz} = \lambda div(\vec{u}) + 2\mu\varepsilon_{zz} = \lambda(\varepsilon_{rr} + \varepsilon_{\theta\theta}) \qquad \text{and} \qquad \sigma_{rz} = 2\mu\varepsilon_{rz}$$

Implementation

We approach this problem by assuming that the cylinder is infinite in the direction perpendicular to the page. We have an obvious axisymmetry:

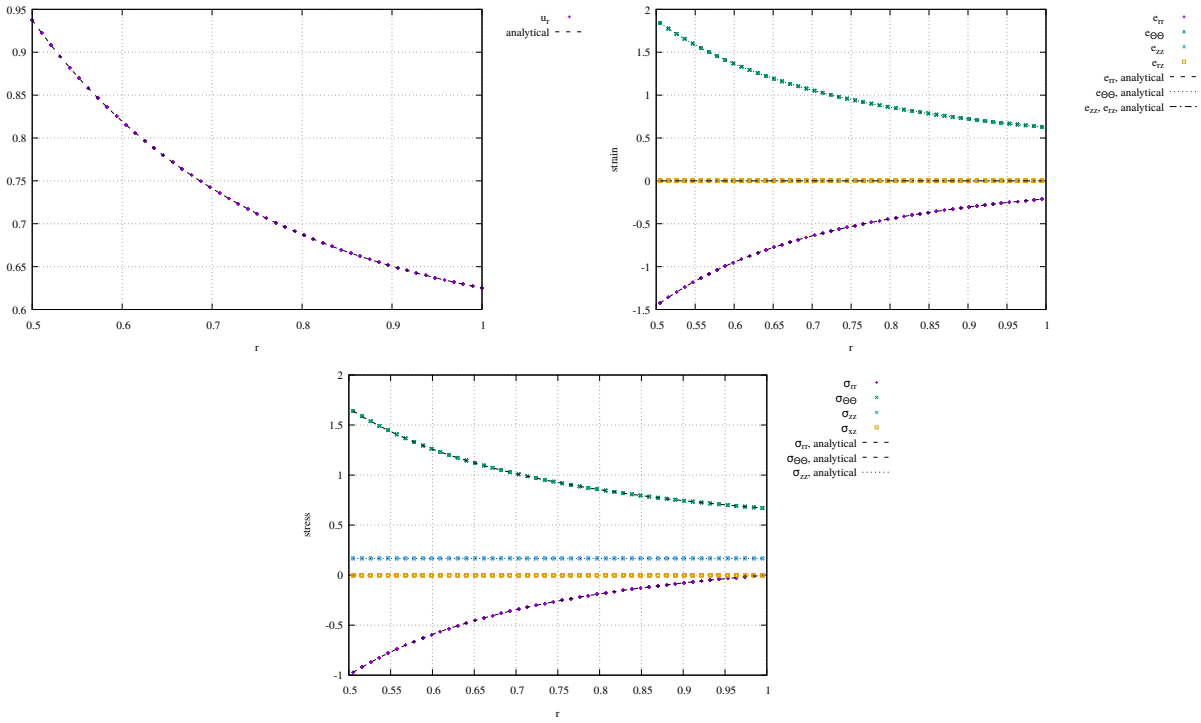


The setup is then as follows:



Because we have the analytical displacement field, we will be using it to impose Dirichlet boundary conditions on the sides, rather than pressure, and we will then recover strain and stress components.

Results for displacement, strain and stress as a function of r are shown hereunder:



The match is perfect and shows that the axisymmetric formulation presented in Section 10.9 is correct.

ToDo: error calculations are not finished !

Stone 91

Stokes sphere in a cylinder, axisymmetric formulation ($Q_2 \times Q_1$)



```
axisymmetric
box
Q2Q1
sparse_matrix
analytical_benchmark
stokes_equation
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_91`

This STONE carries out the Stokes sphere benchmark of Section 19.23 although it uses an axisymmetric formulation, so that the sphere is actually falling inside a vertical cylinder. As a consequence, the boundary conditions are: free slip on the left (the axis of symmetry), top and bottom, and no slip on the right (the wall of the cylinder). The default domain size is $L_r = 0.5$ and $L_z = 1$.

Having established the weak form of the Stokes equations and discretised them in Section 10.8.6, transforming a 2D plane-strain code into an axisymmetric one is rather straightforward. In a nutshell,

- x stands for r
- v_r is u, v_z is w
- the matrix $\boldsymbol{B}$ goes from $3 \times m_v \cdot ndofV$ to $4 \times m_v \cdot ndofV$.
- the matrix $\boldsymbol{N}$ goes from $3 \times m_p \cdot ndofP$ to $4 \times m_p \cdot ndofP$.
- the matrix $\boldsymbol{C}$ goes from 3×3 to 4×4.
- integrals must be multiplied by 2π and also contain an additional r term which translates into x_q in the code.

The Stokes sphere velocity is given by

$$\mathrm{v}_{Stokes} = \frac{2}{9} \frac{\delta\rho}{\eta_f} R_s^2 g$$

which yields $\nu_S \simeq 3.387 \cdot 10^{-5}$. However, this is for a sphere in an infinite domain. When placed in a cylinder, the influence of the walls will slow the sphere down. Two corrections have been proposed be Habermann and by Faxen.

We start by writing the drag force

$$F_H = 6\pi\eta_f R\nu_{sphere}\,\gamma\left(\frac{r}{R}\right)$$

In the case of the Stokes sphere in an infinite medium then $\gamma = 1$ and equating F_H with the buoyancy force $4/3\pi R^3\delta\rho g$ yields the above velocity. Habermann gives a coefficient γ such that

$$\gamma\left(\frac{r}{R}\right) = \frac{1 - 0.75857\left(\frac{r}{R}\right)^5}{1 + f_H\left(\frac{r}{R}\right)}$$

with

$$f_H\left(\frac{r}{R}\right) = -2.1050\left(\frac{r}{R}\right) + 2.0865\left(\frac{r}{R}\right)^3 - 1.7068\left(\frac{r}{R}\right)^5 + 0.72603\left(\frac{r}{R}\right)^6$$

Faxen gives a coefficient γ such that

$$\gamma\left(\frac{r}{R}\right) = \frac{1}{1 + f_F\left(\frac{r}{R}\right)}$$

with

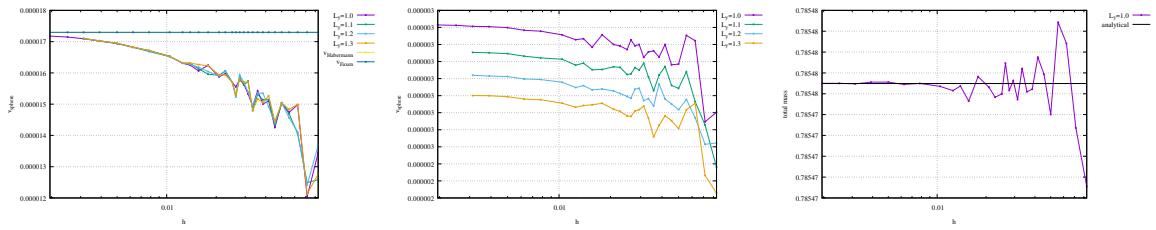
$$f_F\left(\frac{r}{R}\right) = -2.10444\left(\frac{r}{R}\right) + 2.08877\left(\frac{r}{R}\right)^3 - 0.94813\left(\frac{r}{R}\right)^5 - 1.372\left(\frac{r}{R}\right)^6 + 3.87\left(\frac{r}{R}\right)^8 - 4.19\left(\frac{r}{R}\right)^{10}$$

In all three cases we have

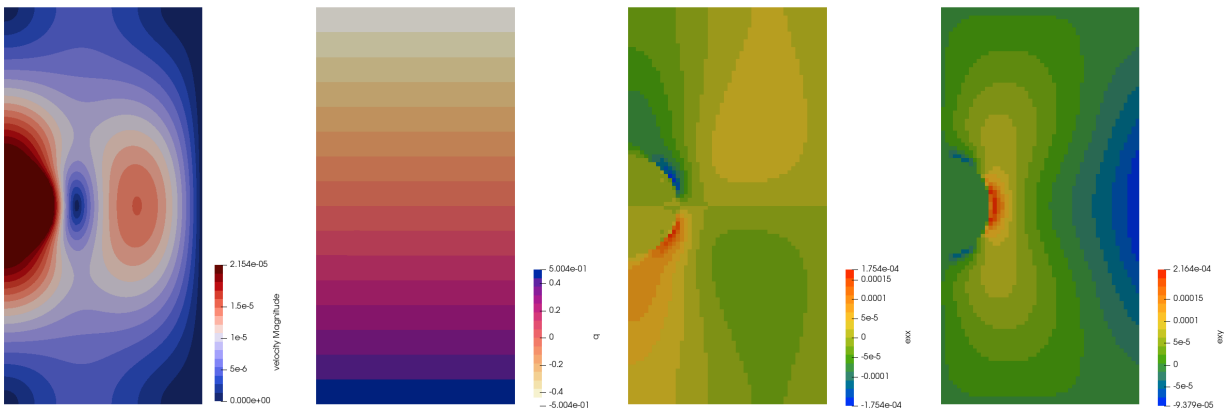
$$\nu_{sphere} = \frac{2}{9}\frac{\delta\rho}{\eta_f}R_s^2 g\frac{1}{\gamma(\frac{r}{R})}$$

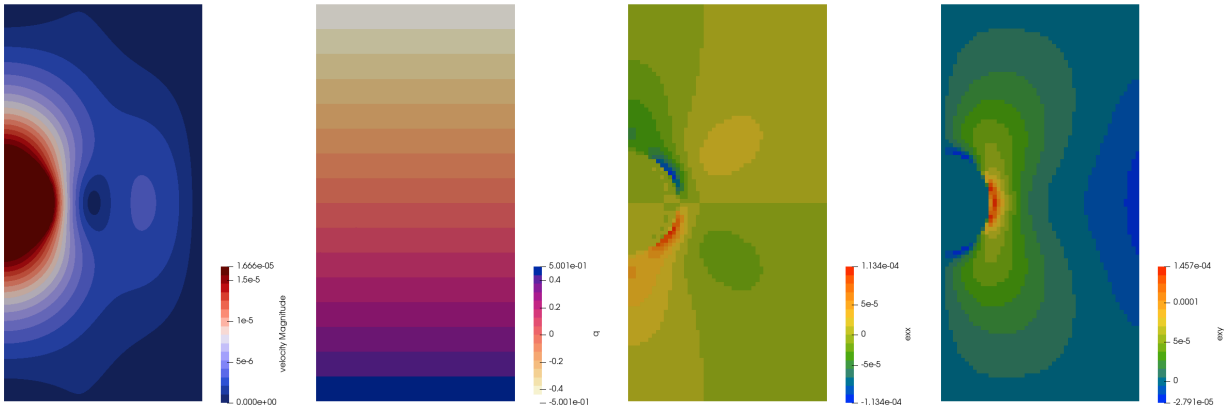
Note that the Habermann & Faxen expressions are copy-pasted from the GALE manual and are extremely comparable in value. I did not verify these yet. Please check Lindgren (1999) [217] to do so. They are about half the value of the Stokes velocity in this case. The three velocity values are computed directly in the code and exported alongside the measured velocity.

We see that increasing the height of the cylinder does not substantially alter the results, and that the velocity in the middle of the sphere seems to converge to the expected ones:



up to 256×512 elements.





Top row: plane strain; Bottom row: axisymmetric. Resolution 50×100.

Stone 92

Stokes sphere in a cylinder, axisymmetric formulation, $P_2^+ \times P_{-1}$

```
axisymmetric
box
P2pPm1
sparse_matrix
analytical_benchmark
stokes_equation
use_mesher
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_scaling
pressure_normalisation
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_92`

This stone solves the axisymmetric problem of the Stokes sphere in a cylinder, similarly to Stone 91. However, it relies on an unstructured grid with Crouzeix-Raviart elements (see Section **??**).

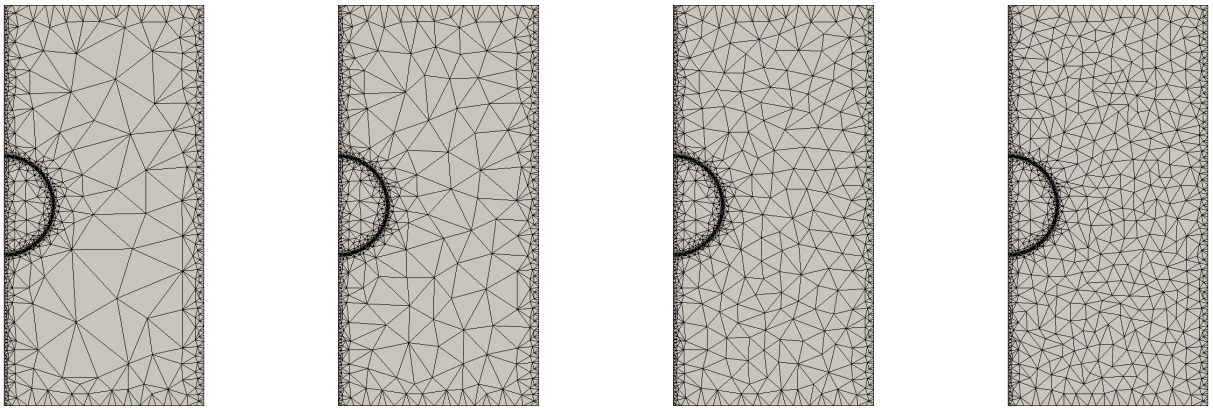
The workflow is as follows: the main parameters are defined in *parameters.py*. The file *generate_nodes.py* uses these parameters to generate points on the boundary of the rectangular domain and on the half circle. A call to *triangle* is then made and a very important additional parameter is passed as argument to triangle: the maximum area of a triangle. *triangle* returns the (constrained) Delaunay triangulation of these points, and this information is then read in by *stone.py* which solves the Stokes equation in axisymmetric formulation (see Section **??**). Boundary conditions are free slip everywhere except no slip on the right.

All these actions are carried out automatically when the following script is used[1]:

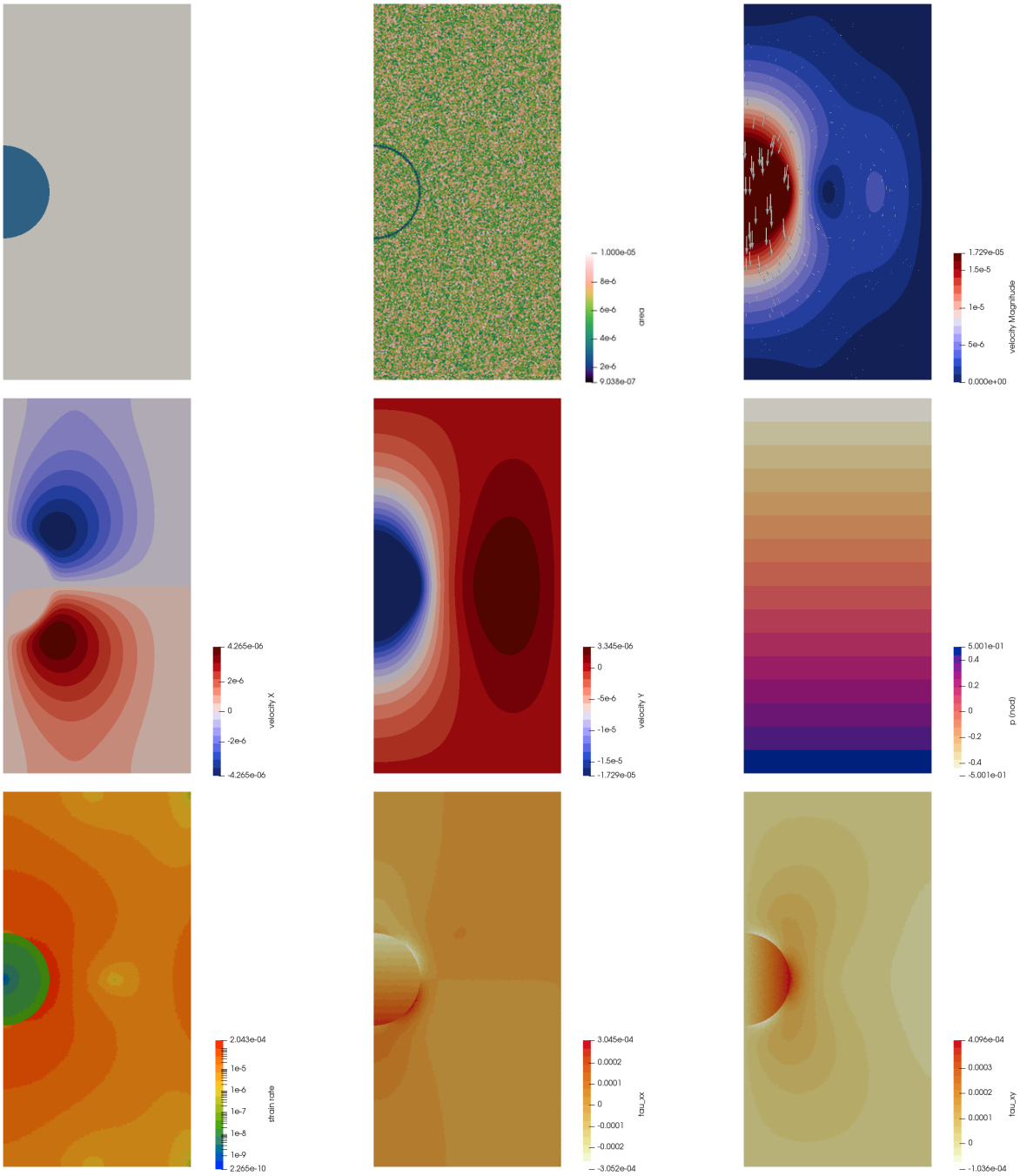
```
python3 generate_nodes.py
cat mypoints mysegments > mesh.poly
../../../../triangle/triangle -j -q -a0.00100 -o2 -pc mesh.poly
echo "nel="
head -1 mesh.1.ele
head -1 mesh.1.ele > temp
echo "NV0="
head -1 mesh.1.node
head -1 mesh.1.node >> temp
python3 stone.py
```

Two remarks: the code stems from another code which implemented time stepping, which explains there is now a single fake time step loop in it. Pressure normalisation is not done the right way: the pressure nullspace is not removed from the matrix, but the solution is later normalised so that its average over the domain is zero.

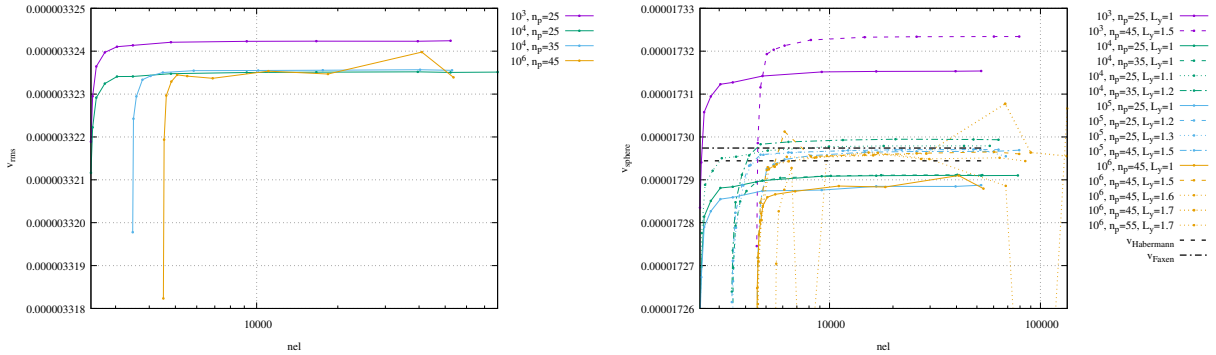
[1] It assumes that the triangle program has been compiled and is located outside of the fieldstone tree.



Left to right: nel=2432,2468,2588,2866



Because the triangle size varies a lot within a mesh, defining a mesh size h makes littel sense, so we plot the following results as a function of the number of elements:

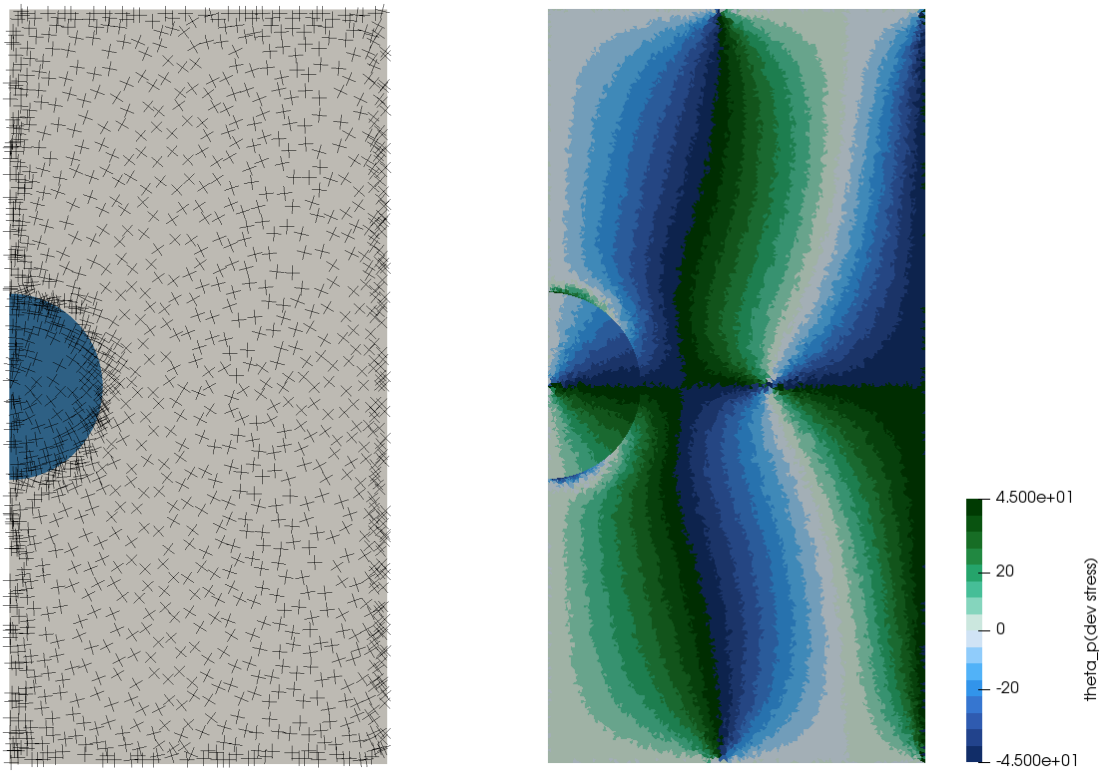


Left: vrms over the whole domain for $L_y = 1$; Right: measured velocity in the middle of the sphere. Some weird effect is taking place at very high resolution...

The parameter n_p is user defined and is an attempt to parameterise the resolution of the domain sides and the half circle with a single parameter. In *parameters.py* we have:

```
np_top=n_p
np_bottom=n_p
np_left=10*n_p
np_right=5*n_p
np_sphere=10*n_p
```

We see that the sphere velocity in its middle matches the Habermann and Faxen values remarkably well, especially when the domain is made taller (top and bottom boundaries are further away) and the viscosity of the sphere is brought up to 10^5.



Deviatoric stress principal directions (left) and angle (right).

Stone 93

Buoyancy-driven flow with $(P_2^+ \times P_{-1})$

```
2D
box
P2pPm1
sparse_matrix
numerical_benchmark
stokes_equation
time_stepping
use_mesher
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_normalisation
free_surface
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_93`

This stone solves the problem of buoyancy-driven flows generated by objects sinking in a fluid, potentially under a deformable free surface. It relies on an unstructured grid with Crouzeix-Raviart elements (see Section 10.5.9).

The workflow is as follows: the main parameters are defined in *parameters.py*. The file *generate_nodes.py* uses these parameters to generate points on the boundary of the rectangular domain, on the surface and on the circle and its center. A call to *triangle* is then made and a very important additional parameter is passed as argument to triangle: the maximum area of a triangle. *triangle* returns the (constrained) Delaunay triangulation of these points, and this information is then read in by *stone.py* which solves the Stokes equation. Boundary conditions are free slip everywhere. This benchmark is described in Section 19.25.

All these actions are carried out automatically when the following script is used[1]:

```
python3 generate_nodes.py
cat mypoints mysegments > mesh.poly
../../../../triangle/triangle -j -q -a0.00100 -o2 -pc mesh.poly
echo "nel="
head -1 mesh.1.ele
head -1 mesh.1.ele > temp
echo "NV0="
head -1 mesh.1.node
head -1 mesh.1.node >> temp
python3 stone.py
```

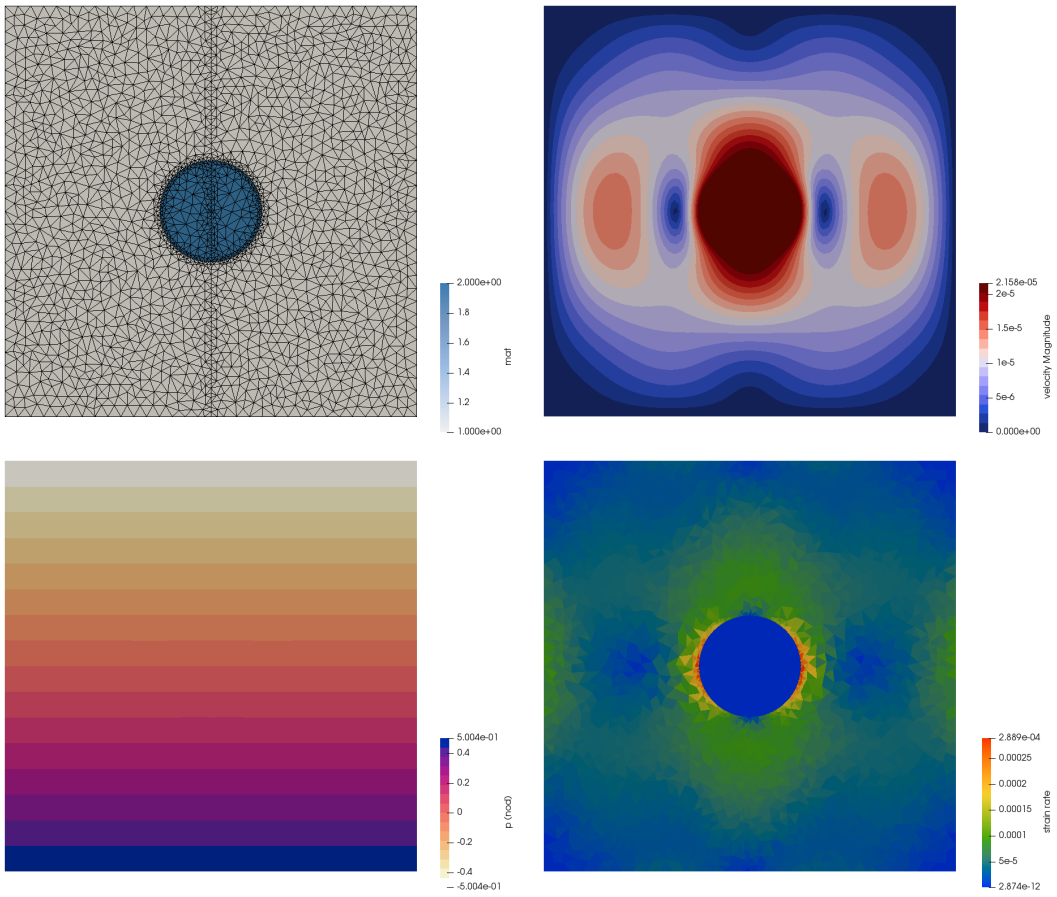
Note that ressure normalisation is not done the right way: the pressure nullspace is not removed from the matrix, but the solution is later normalised at the top of the domain or over the whole domain.

Time stepping is carried out and the time step δt is computed with a CFL criterion. Nodes are advected with their velocity but no remeshing is implemented so that when the deformation becomes to large around

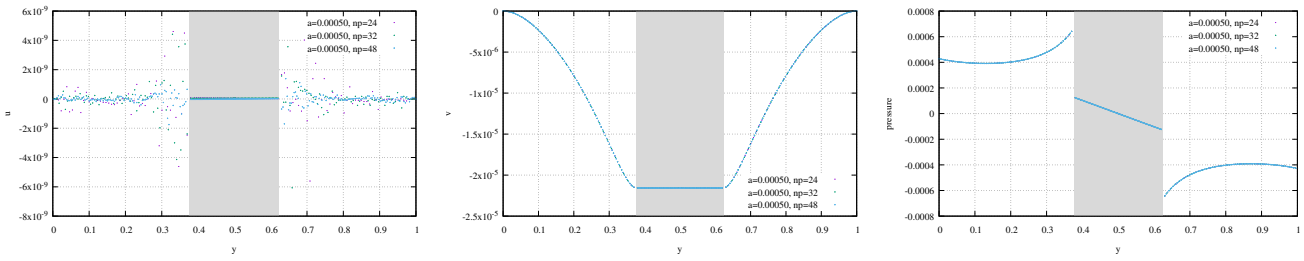
[1] It assumes that the triangle program has been compiled and is located outside of the fieldstone tree.

the sphere the triangles become really deformed and the simulation must be stopped. Given the parameters of this benchmark this corresponds to $t \simeq 30$, while the simulation should ideally have been run up to $t > 100$.

The Stokes sphere in a unit square The setup and all results are described in Section 19.24. Nodes are added on a vertical line to facilitate pressure and velocity profile measurements.

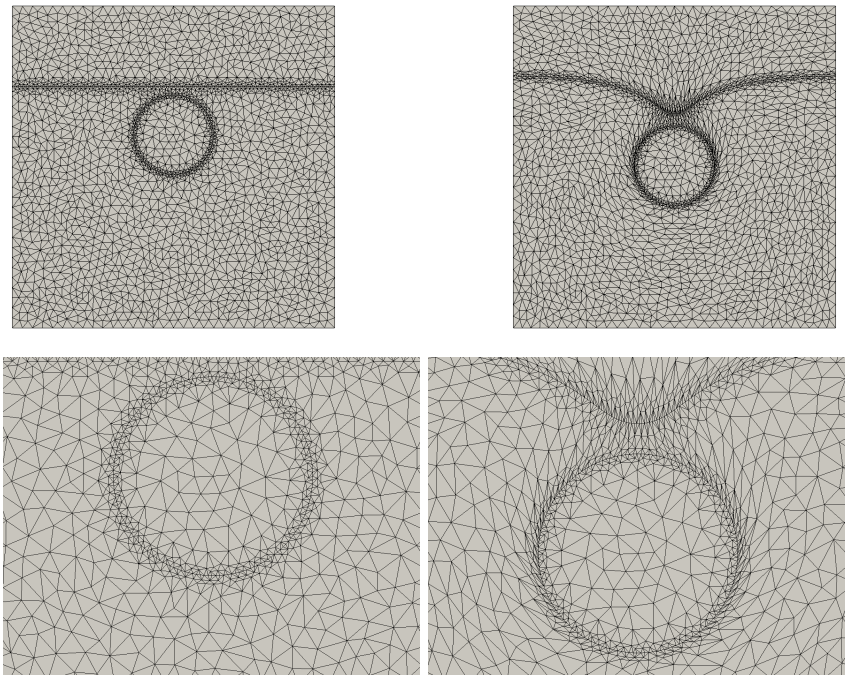


Mesh obtained with -a0.00050 in script, np=32. No-slip boundary conditions.

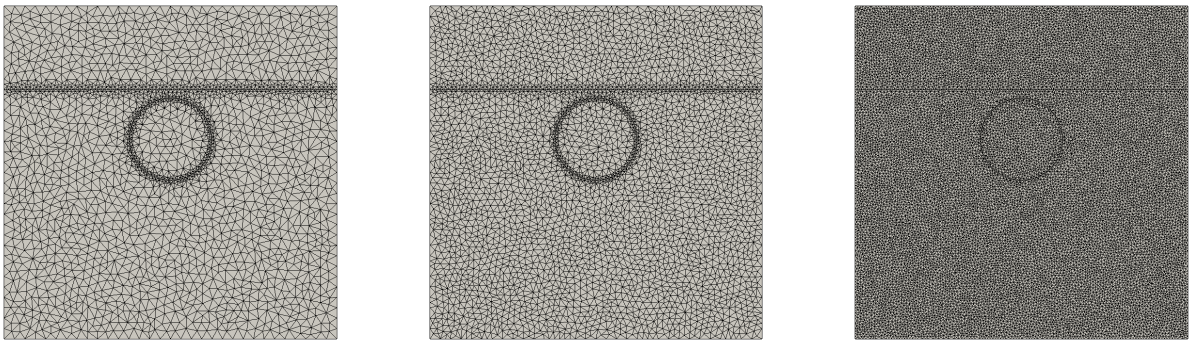


Mesh obtained with -a0.00050 in script, np=24,32,48. No-slip boundary conditions.

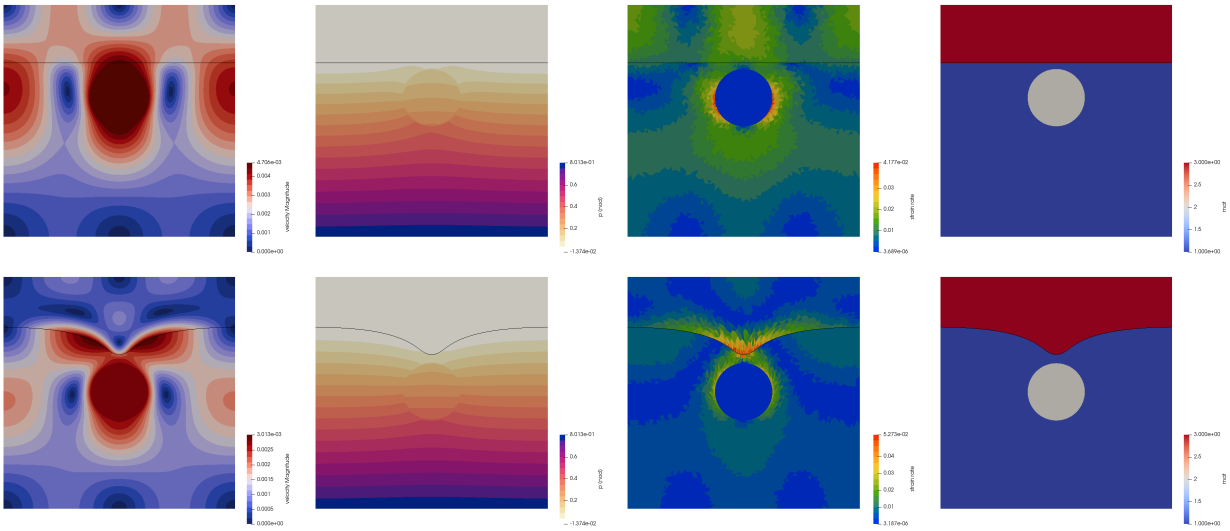
The Stokes sphere under a deformable surface Nodes on the deformable surface are flagged at the beginning of the calculations and exported to an ascii file for later postprocessing. The multiple measurements for this benchmark are thoroughly described and plotted in Section 19.25.



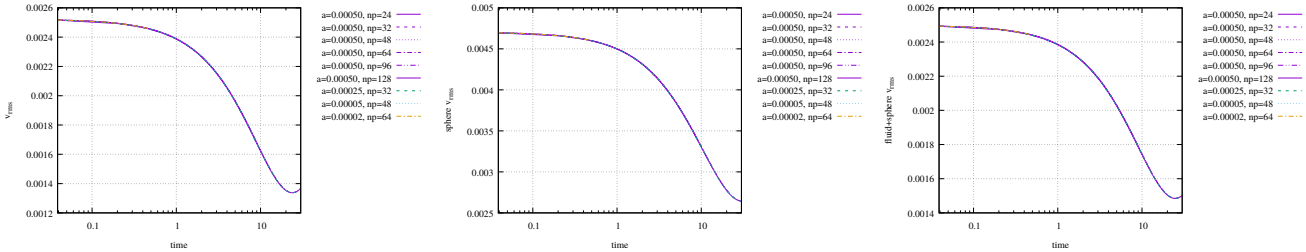
Mesh obtained with -a0.00050 in script



Mesh obtained with -a0.00050, -a0.00025 and -a0.00005 in script

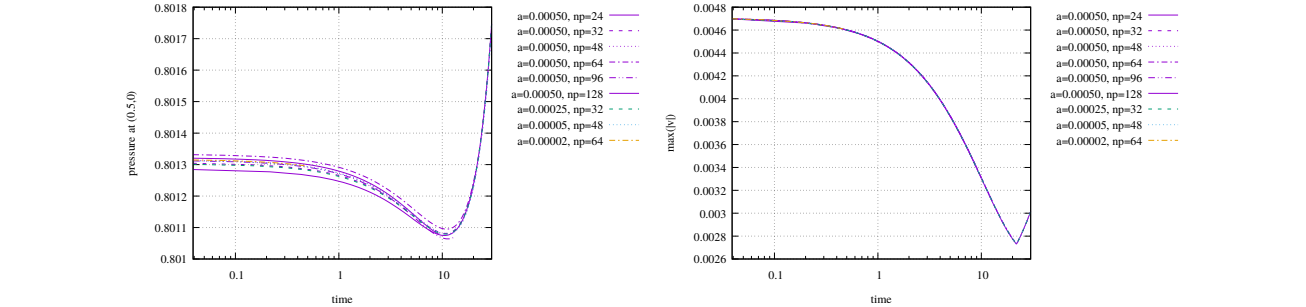


Mesh obtained with -a0.00025 in script. Top row $t = 0$, bottom row $t \simeq 30$.

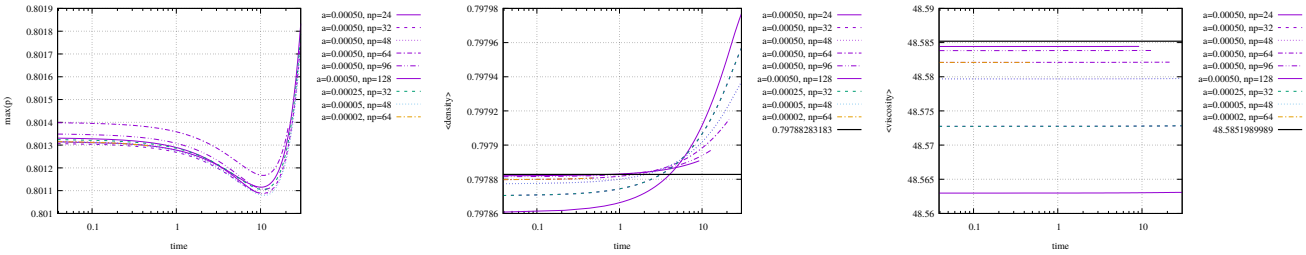

v_{rms}
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

sphere v_{rms}
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

fluid+sphere v_{rms}
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

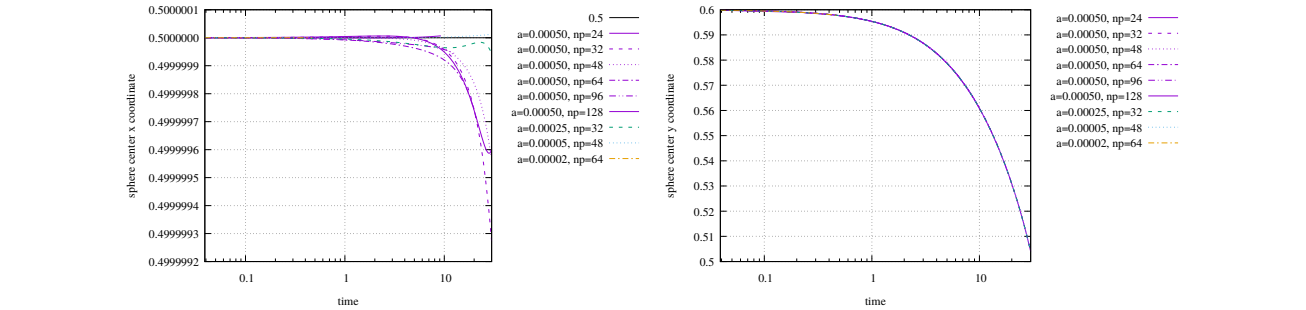

pressure at (0.5,0)
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

max(|v|)
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

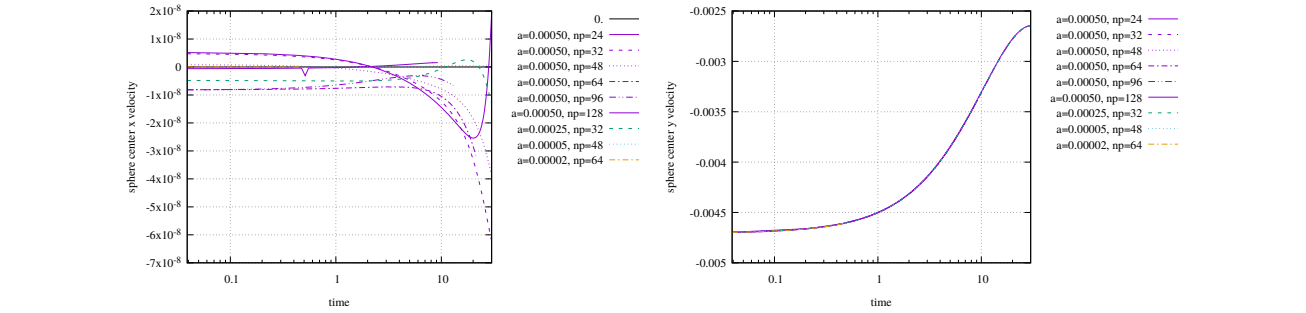

max(p)
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

<density>
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64
0.797882831830

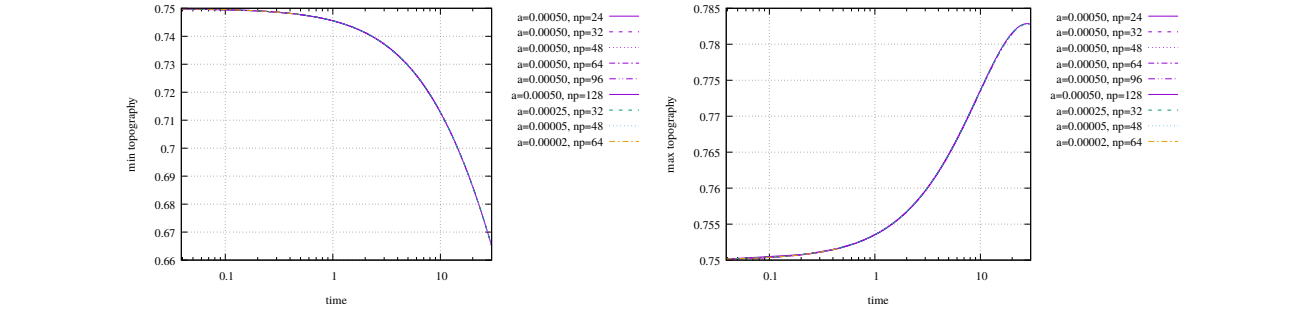
<viscosity>
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64
48.5851989899


sphere center x coordinate
time
0.5
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

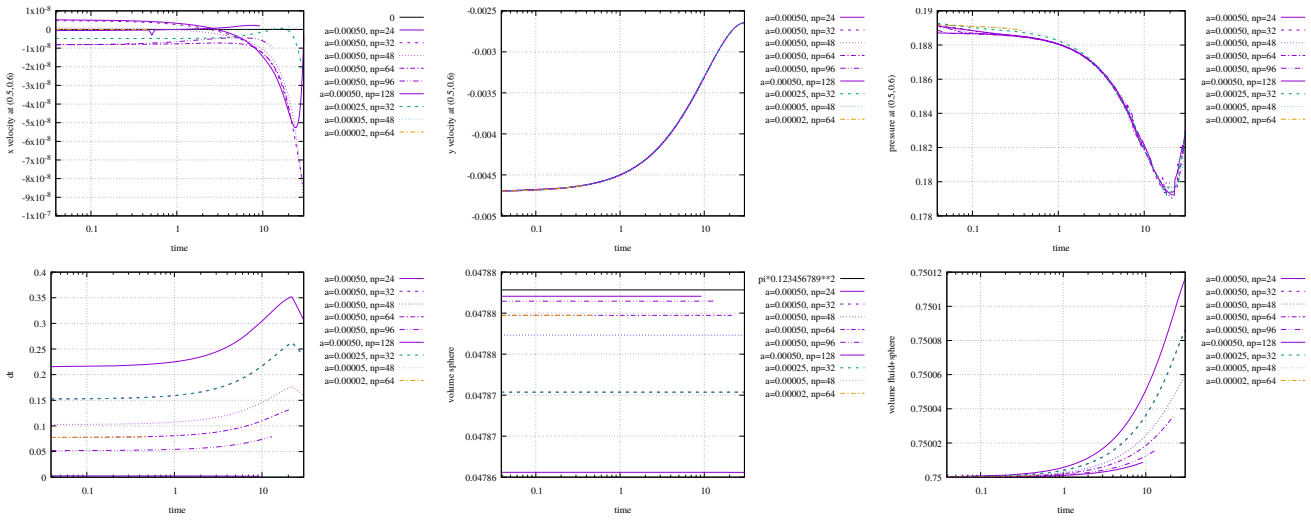
sphere center y coordinate
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64


sphere center x velocity
time
0.
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

sphere center y velocity
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

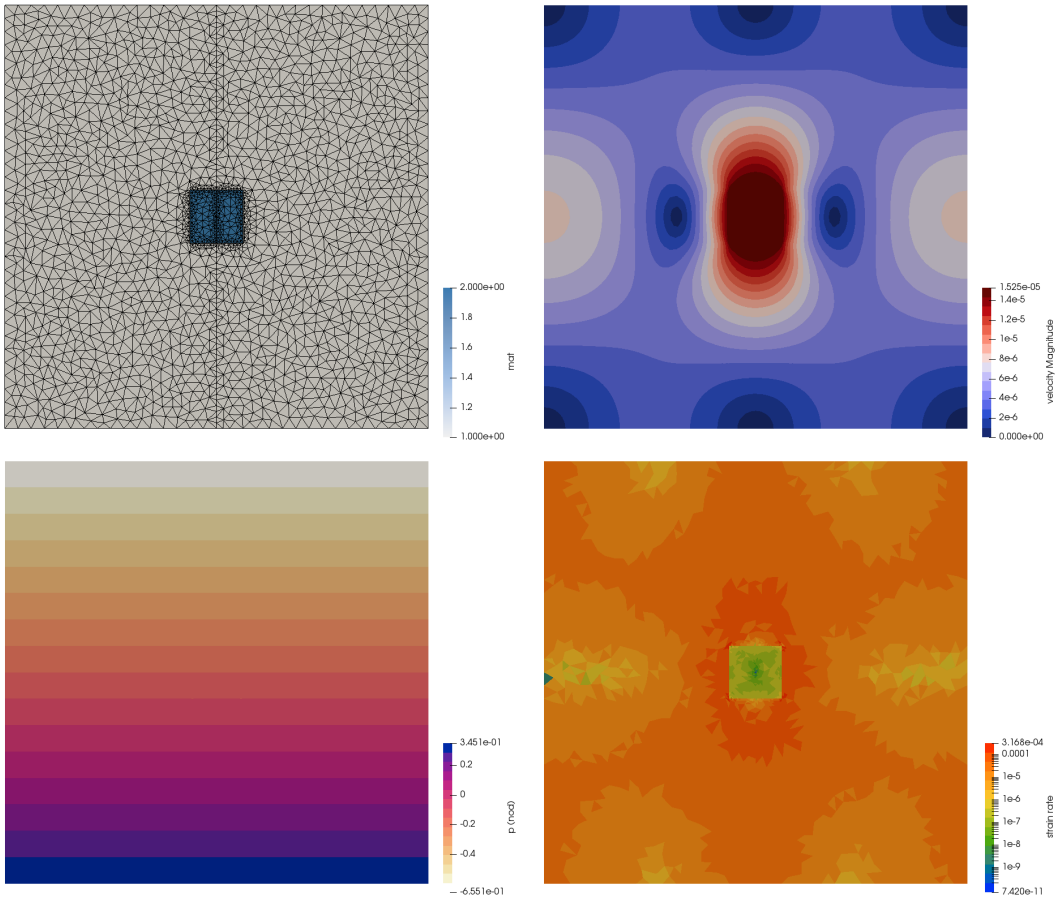

min topography
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

max topography
time
a=0.00050, np=24
a=0.00050, np=32
a=0.00050, np=48
a=0.00050, np=64
a=0.00050, np=96
a=0.00050, np=128
a=0.00025, np=32
a=0.00005, np=48
a=0.00002, np=64

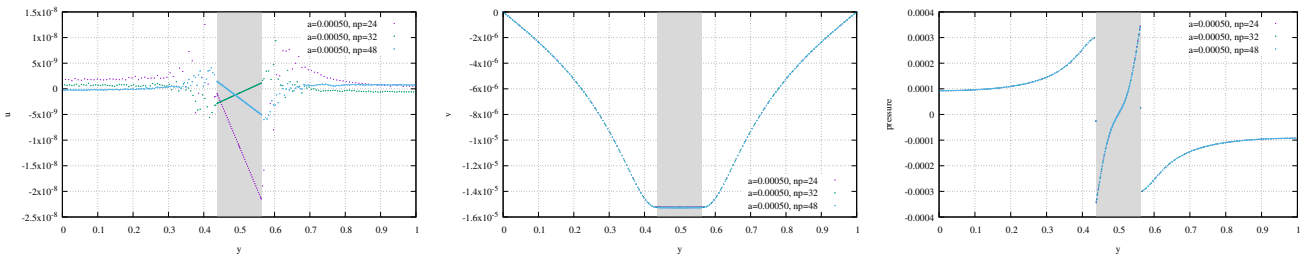


Results are remarkably similar for all resolutions. Note that because the pressure is discontinuous across element edges, the recorded value at (0.5,0.6) shows small jumps.

The square block in the middle of a unit square

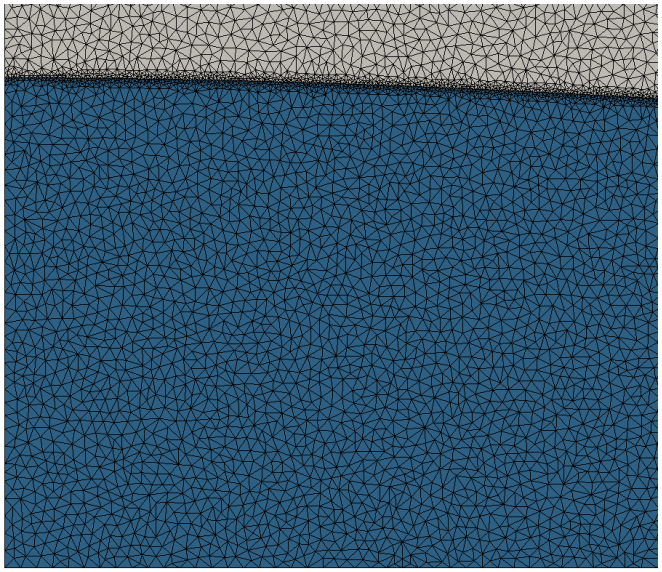


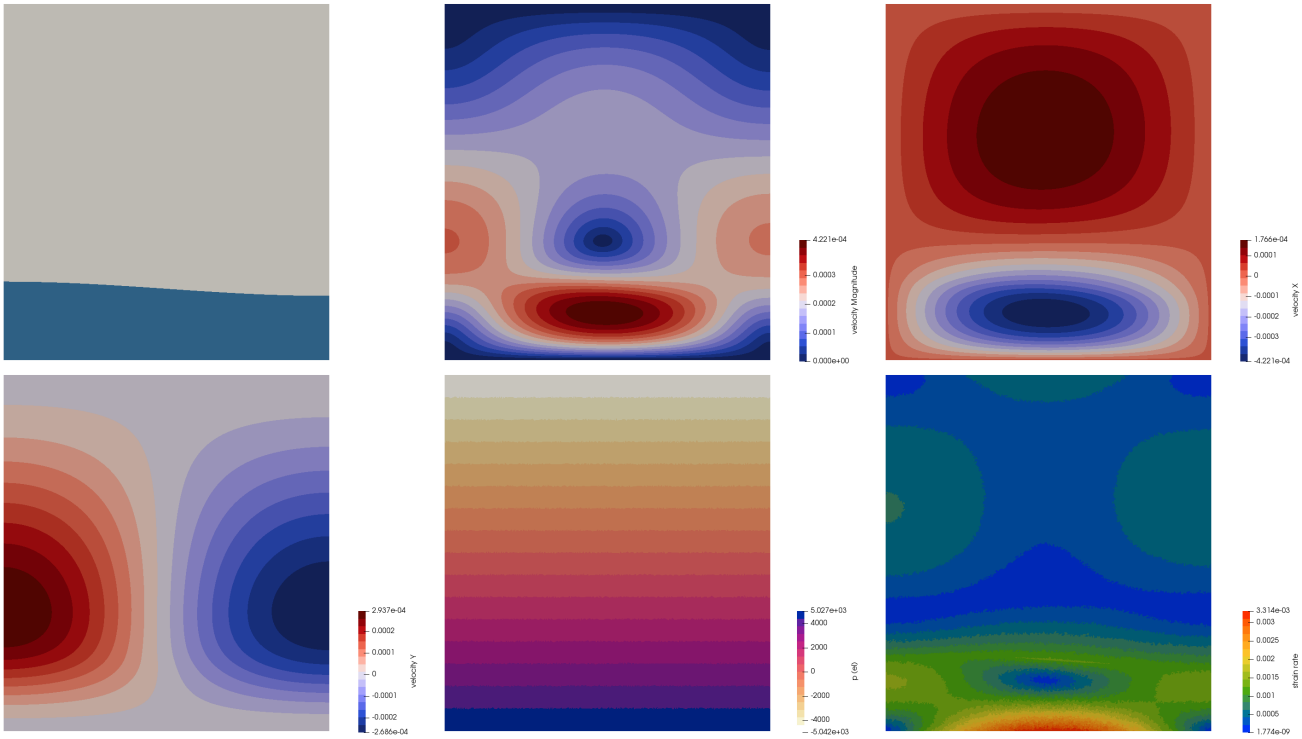
Mesh obtained with -a0.00050 in script, np=32. Free-slip boundary conditions.



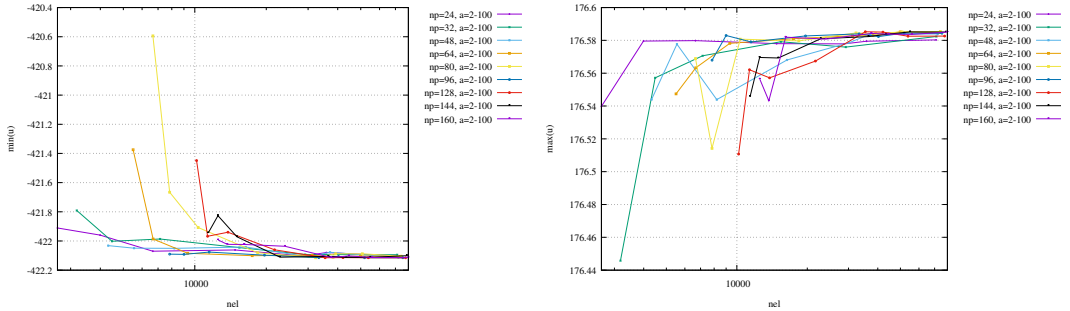
Mesh obtained with -a0.00050 in script, np=24,32,48. Free-slip boundary conditions.

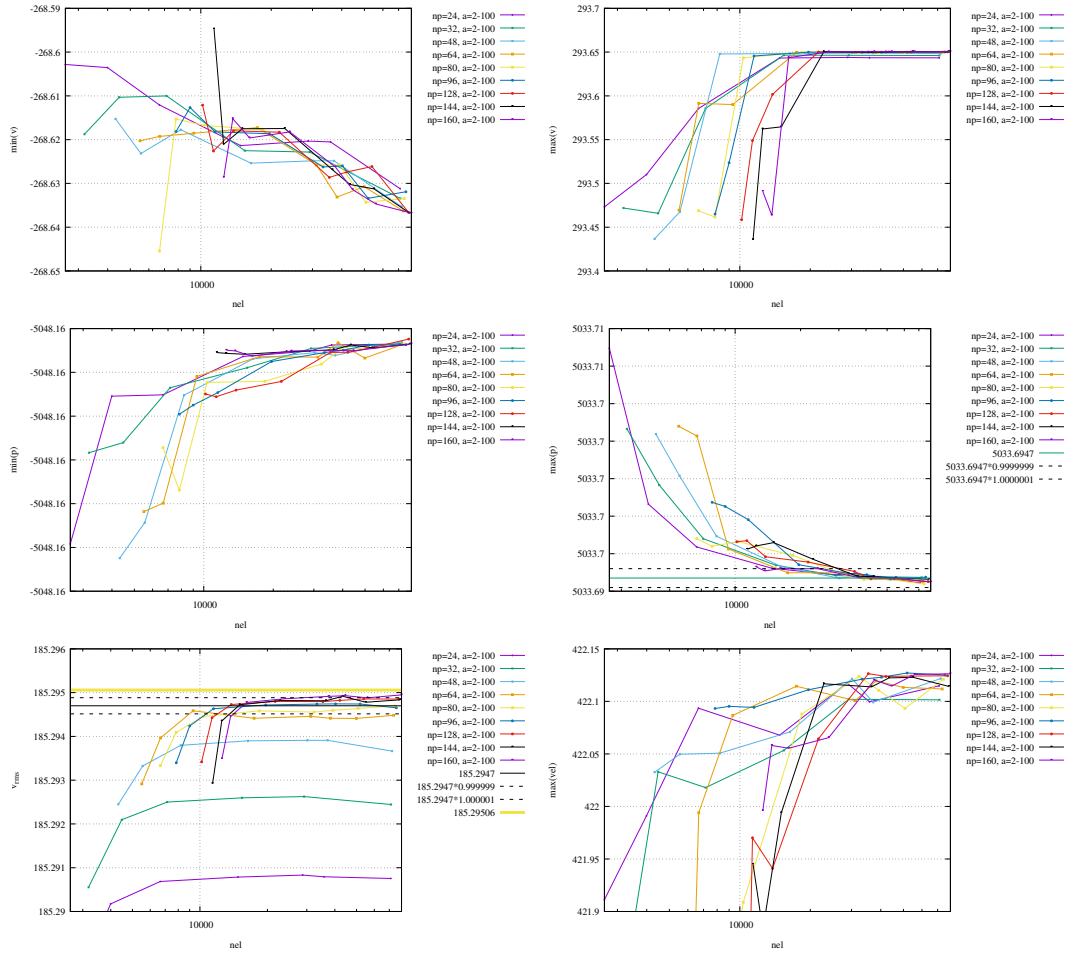
The Rayleigh-Taylor experiment of van Keken et al (1997) [338]

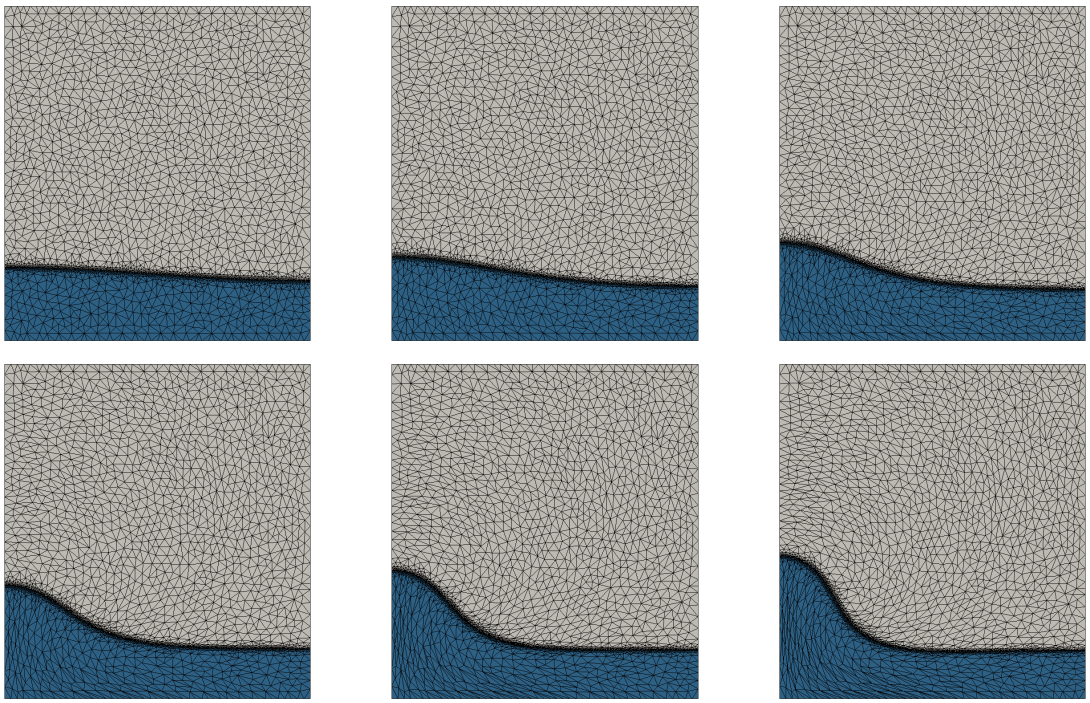




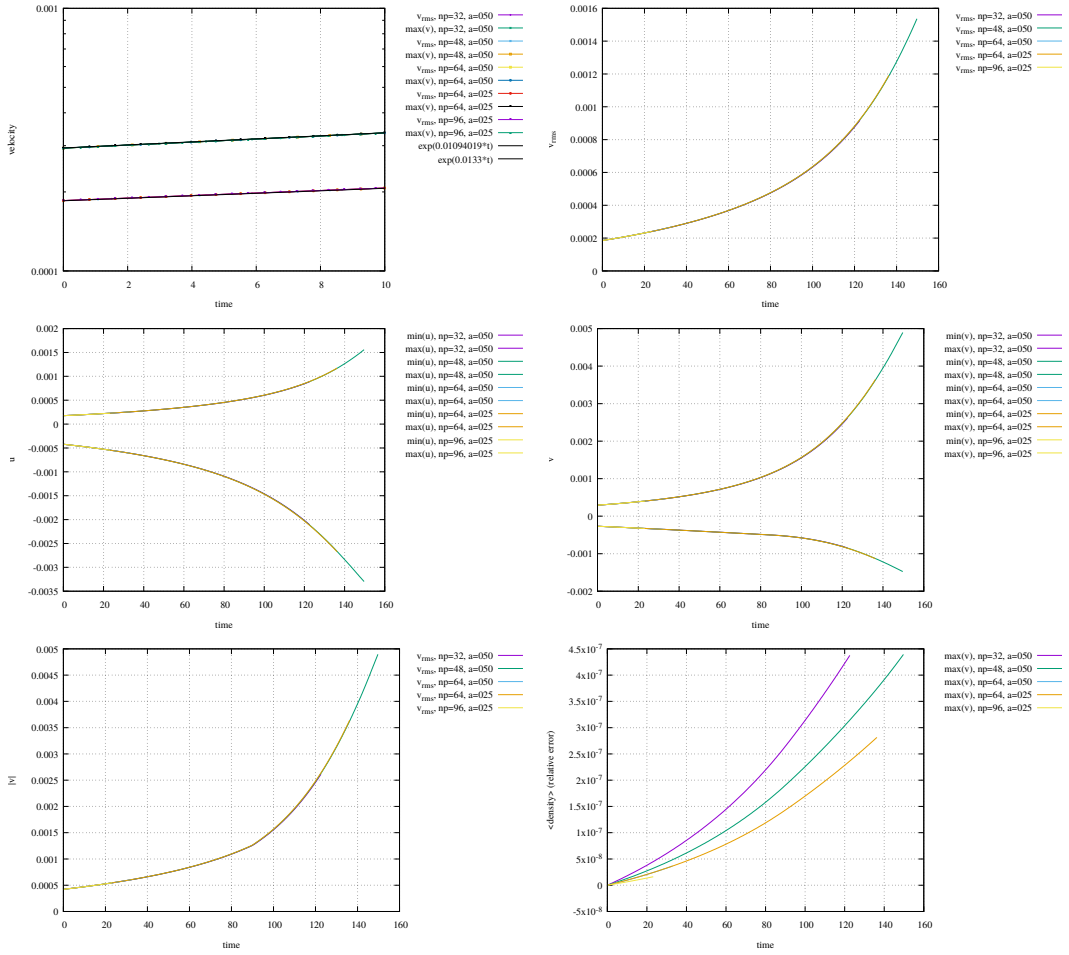
$np = 128,\ a = 2 \cdot 10^{-6}$







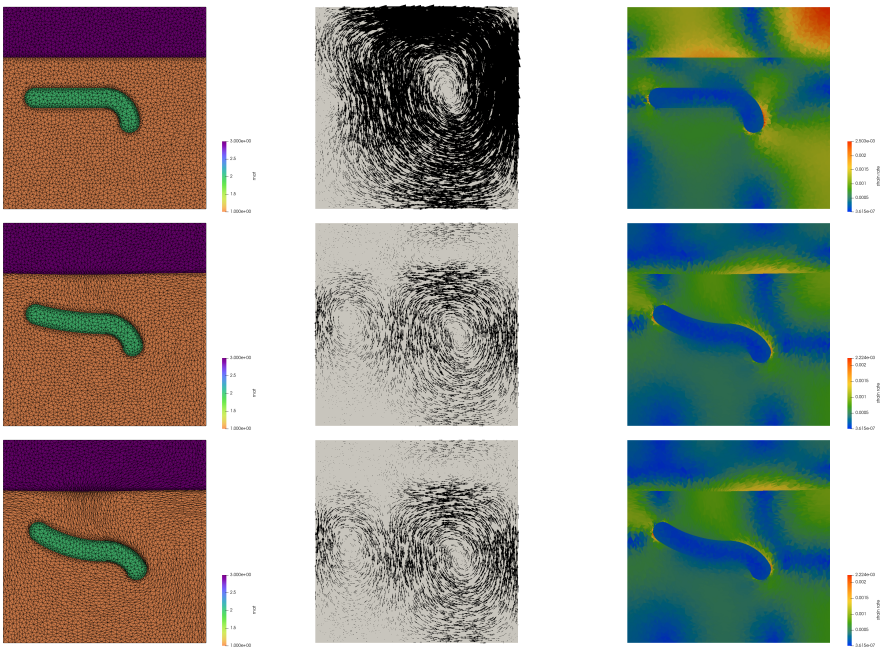
Evolution of the mesh as a function of time.



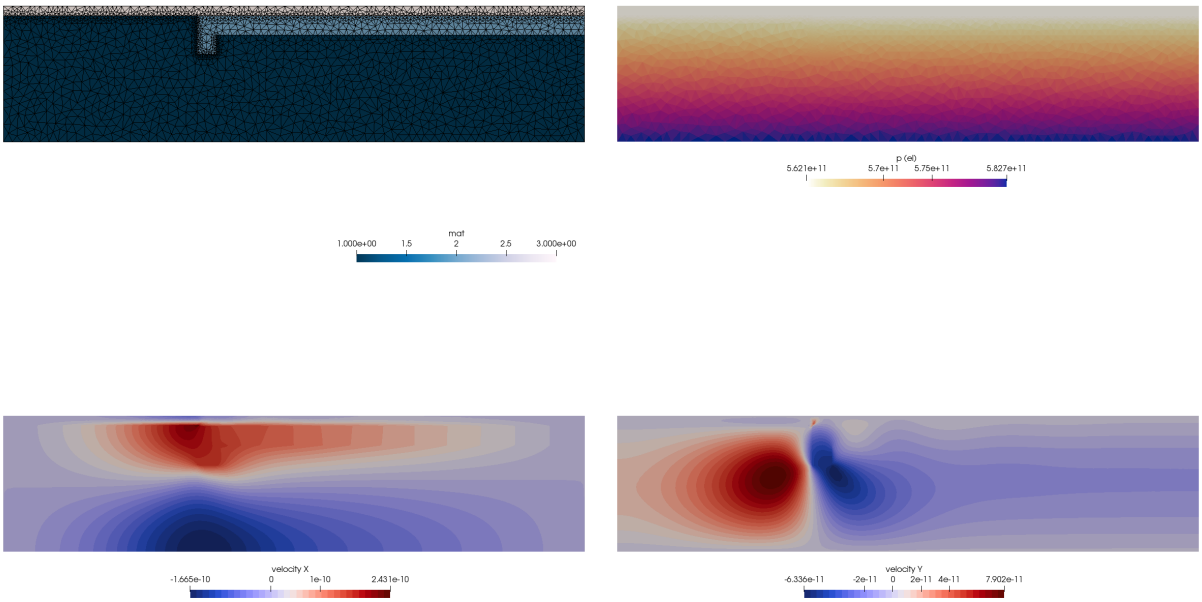
Evolution of ν_{rms}, velocity and average density/mass relative error as a function of time.

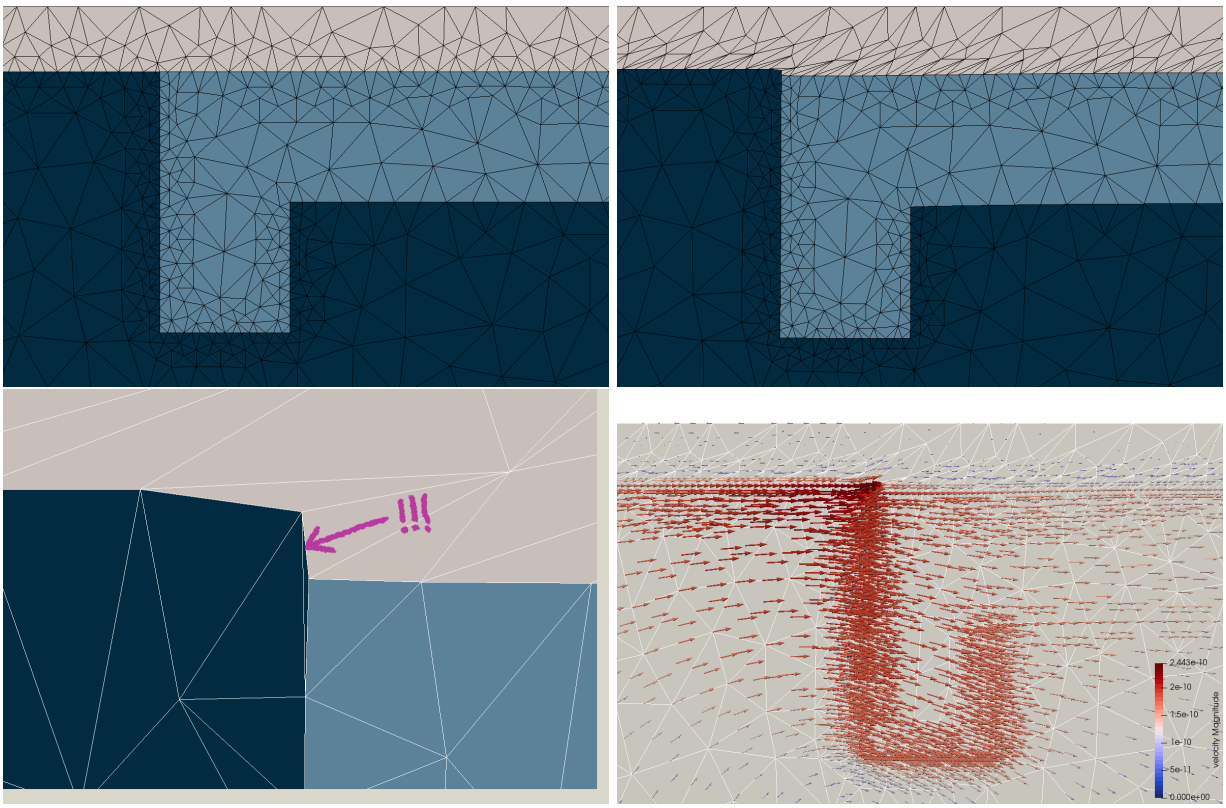
FINISH with other viscosities !

The sinking slabby blob An example of an asymmetric sinking object. The air-fluid interface deforms with both vertical and horizontal velocity. It is just a try out, should be refined later.



The 'subduction' experiment of Schmeling et al (2008) The setup is presented in Schmeling *et al.* (2008) [282]. This is not a benchmark and will never me one. This is in any case here a hopeless endeavour because remeshing is really needed.





mesh after 100 time steps. The triangle at the triple junction completely collapses and the simulation should be stopped.

Stone 94

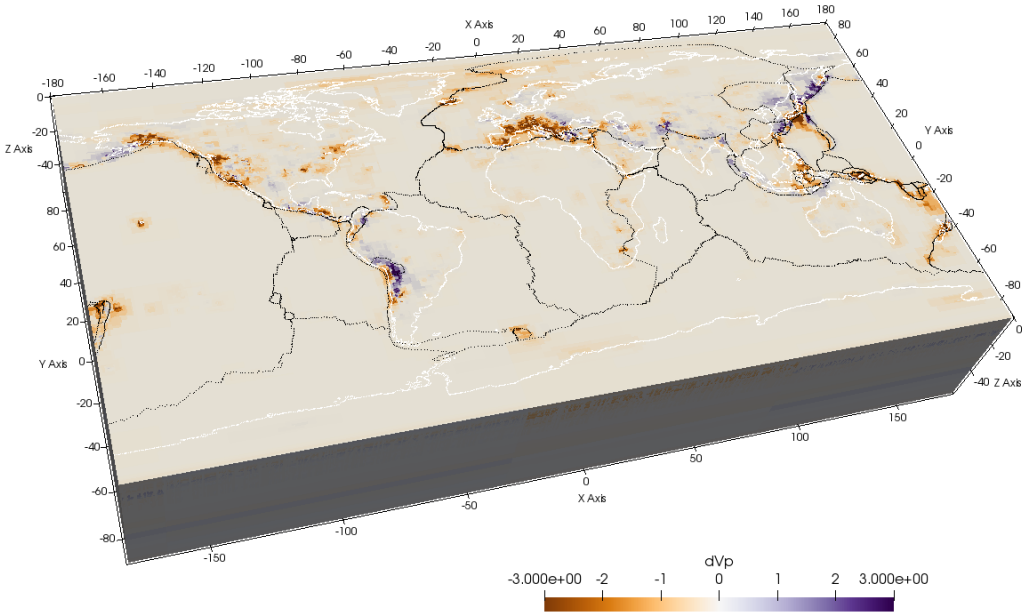
visualising UUP07 in Paraview

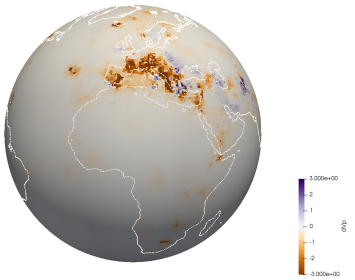
3D
box
hollow_sphere
paraview_output

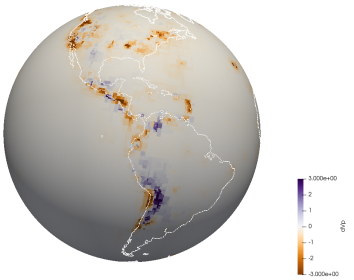
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_94`

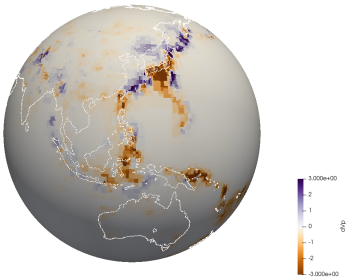
This stone was developed in collaboration with Lex Verbrugh.

This stone is a simple tool to visualise the UUP07 V_p tomography model [146] in paraview. The data is available online at `https://www.atlas-of-the-underworld.org/downloads/`. The file is 230Mb so it is not archived here. Simply download it and place in the same folder as the python file. The output can be either in a Cartesian domain or a hollow sphere of radius 6371km. Plates and continent lines obtained from Stone 69.




3.000e+00
2
1
0
-1
-2
-3.000e+00
dVp


3.000e+00
2
1
0
-1
-2
-3.000e+00
dVp


3.000e+00
2
1
0
-1
-2
-3.000e+00
dVp

Stone 95

van Keken *et al.* R-T instability with mesh adaptivity ($P_2^+ \times P_{-1}$)

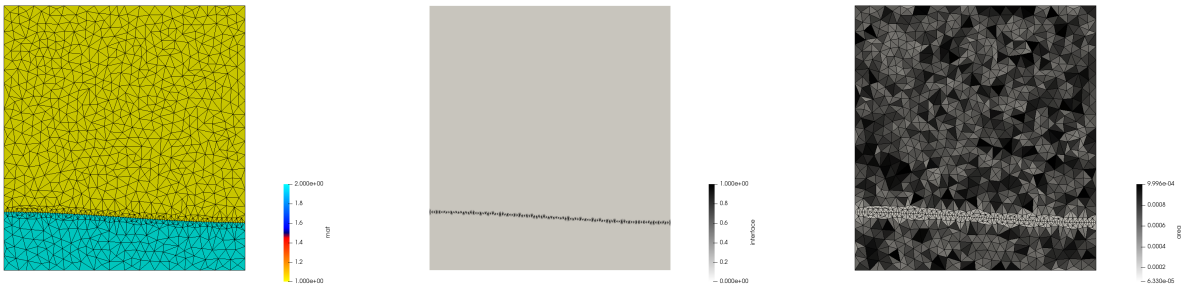


2D
box
MINI
P2pPm1
P2P1
sparse_matrix
numerical_benchmark
stokes_equation
time_stepping
use_mesher
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_scaling
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_95`

This stone tackles the Rayleigh-Taylor instability problem of van Keken *et al.* [338]. It builds on Stone 93 as it relies on triangular elements (the MINI element(see Section 10.5.7), the Taylor-Hood $P_2 \times P_1$ element (see Section 10.5.8), and the Crouzeix-Raviart element (see Section 10.5.9)) and Delaunay meshes generated by the Triangle code [293].

However, this code does implements mesh adaptation and the python code then calls the Triangle mesher code at every timestep (the remeshing cost is negligible). I unfortunately cannot prevent Triangle from adding nodes in between the interface nodes, thereby ruining the numbering and the flagging of these nodes. I have therefore implemented a refinement process which add nodes on the interface when two consecutive nodes are getting too far apart (which would have been necessary because of the shear stretching of the interface).



Interface counts 100 points, Triangle argument a=0.001,

The number of parameters one can vary is somewhat limited:

- the CFL number - default 0.25 (note that the time step is limited to 0.5);
- the maximum area a of triangles (passed as argument to Triangle) - default 0.001 for second order elements, 0.0005 of first order element;
- the initial number of triangle vertices on the interface np_{surf} - default 200;
- the stretch factor controlling the addition of points γ - default 1.5;
- the type of element (element 1: MINI, element 2: Crouzeix-Raviart, element 3: Taylor-Hood)

I also somewhat arbitrarily set the timestep to be limited to $\delta t_{\max} = 0.25$. This allows for accurate measurements of the growth rate across all models. In the end we wish to plot the time evolution of the root mean square velocity and establish the time and amplitude of the first two peaks. We also wish to monitor volume/mass conservation.

Isoviscous results - instantaneous results

.

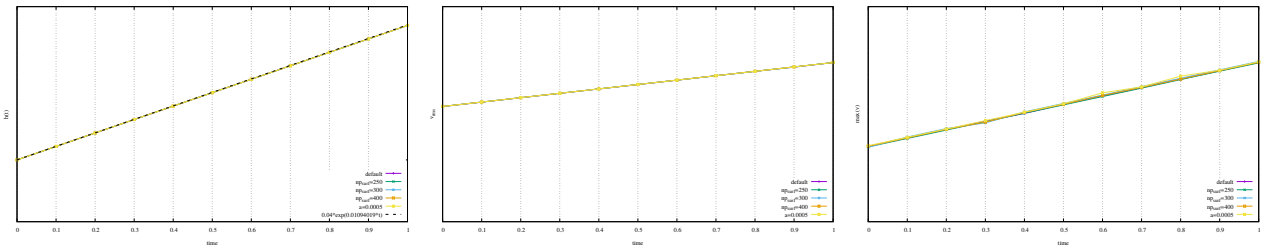
	$a = 0.002$	$a = 0.001$	a=0.0008	$a = 0.0005$	a=0.0004	$a = 0.0003$	$a = 0.000$
np_{surf}=150				1.852875e-04	1.852877e-04		
np_{surf}=200	1.852812e-04	1.852888e-04	1.852900e-04	1.852904e-04	1.852907e-04		
np_{surf}=250				1.852919e-04	1.852922e-04		
np_{surf}=300				1.852926e-04	1.852929e-04	1.852930e-04	
np_{surf}=350	1.852838e-04	1.852916e-04	1.852924e-04	1.852932e-04	1.852933e-04	1.852936e-04	
np_{surf}=400	1.852816e-04	1.852922e-04	1.852928e-04	1.852934e-04	1.852936e-04	1.852938e-04	1.852940
np_{surf}=450	1.852851e-04	1.852920e-04	1.852928e-04	1.852937e-04	1.852939e-04	1.852940e-04	1.852942
np_{surf}=500	1.852852e-04	1.852928e-04	1.852935e-04	1.852938e-04	1.852940e-04	1.852942e-04	1.852943
np_{surf}=550				1.852939e-04	1.852942e-04	1.852943e-04	1.852944

Root mean square velocity v_{rms} value for $t = 0$. Crouzeix-Raviart element.

Isoviscous results - short term

We now turn to the growth rate(s) which is measured in gnuplot as follows:

```
f(x)=a*exp(b*x)
fit f(x) 'benchmark_default.ascii' u 3:(($23-$21)) via a,b
```



model	$\gamma(h)$	$\gamma(v_{rms})$	$\gamma(\max(v))$
default	0.0108964	0.0110335	0.0132341
def. w/ np_{surf}=250	0.0108964	0.0110438	0.0132409
def. w/ np_{surf}=300	0.0108964	0.0110449	0.0134468
def. w/ np_{surf}=400	0.0108964	0.0110454	0.0132651
def. w/ $a = 0.0005$	0.0108964	0.0110449	0.0133781

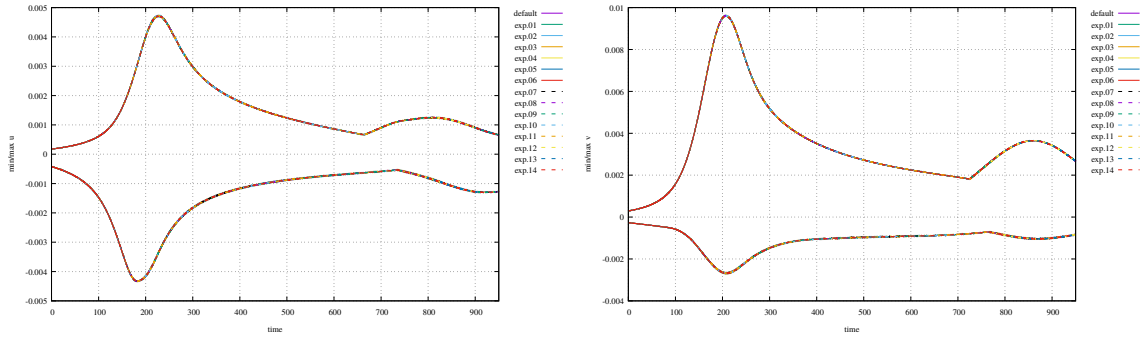
Isoviscous results - Long term evolution

The script `script_run_al` is designed to run for a long time (several days) and will lauch all 1+14 runs at the same time:

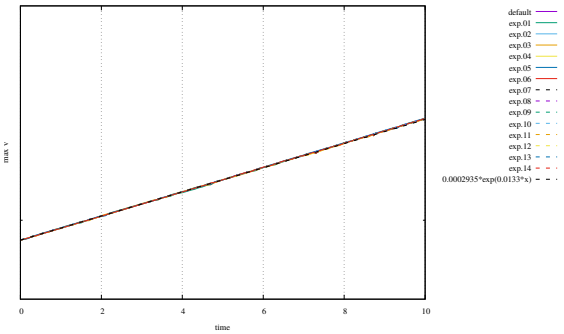
experiment	CFL	a	np_{surf}	γ	elt type
default	0.25	0.001	200	1.5	2
#1	’	’	250	’	’
#2	’	’	300	’	’
#3	’	’	350	’	’
#4	’	’	400	’	’
#5	’	’	500	’	’
#6	’	’	’	1.25	’
#7	’	0.0008	’	’	’
#8	’	0.0005	’	’	’
#9	’	’	’	’	3
#10	’	0.0005	400	’	’
#11	0.1	’	’	’	’
#12	0.5	’	’	’	’
#13	0.75	’	’	’	’
#14	1	’	’	’	’

Results are shown hereunder:

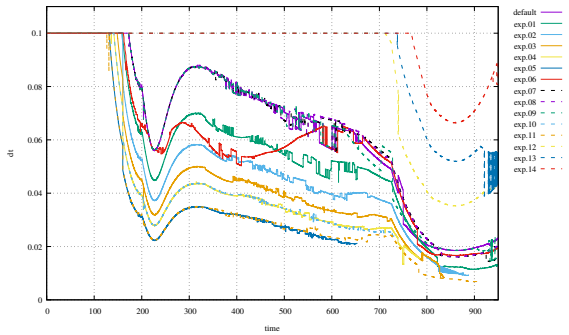
- minimum and maximum value of the velocity components u and v:



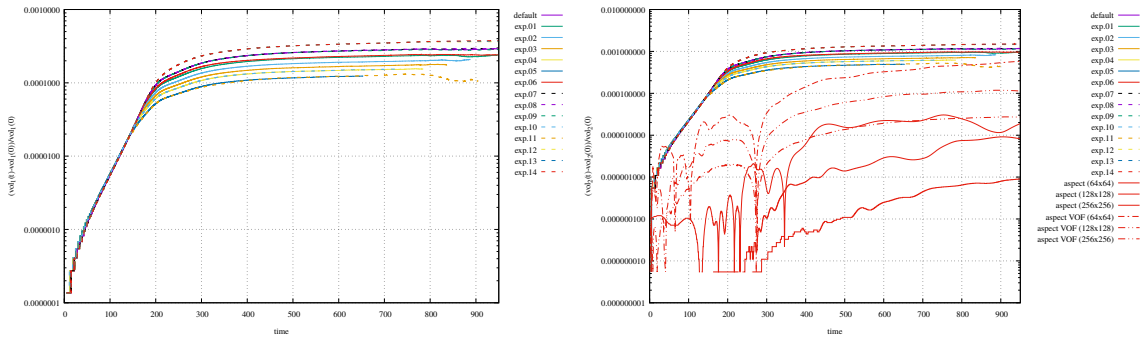
- max(v) for the first 10 seconds:



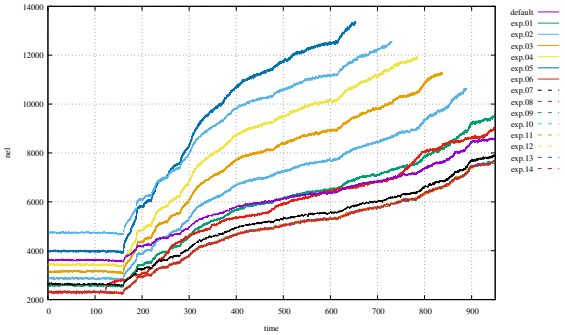
- time step δt value:



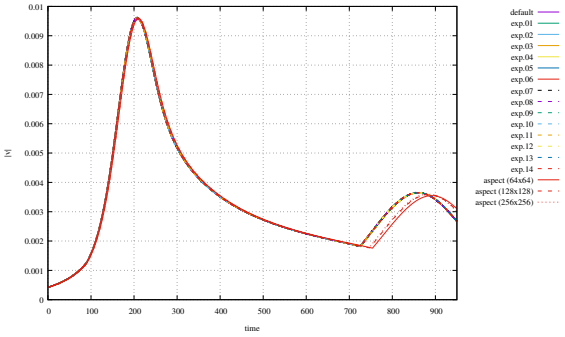
- volume of fluid 1 and 2 (relative change):



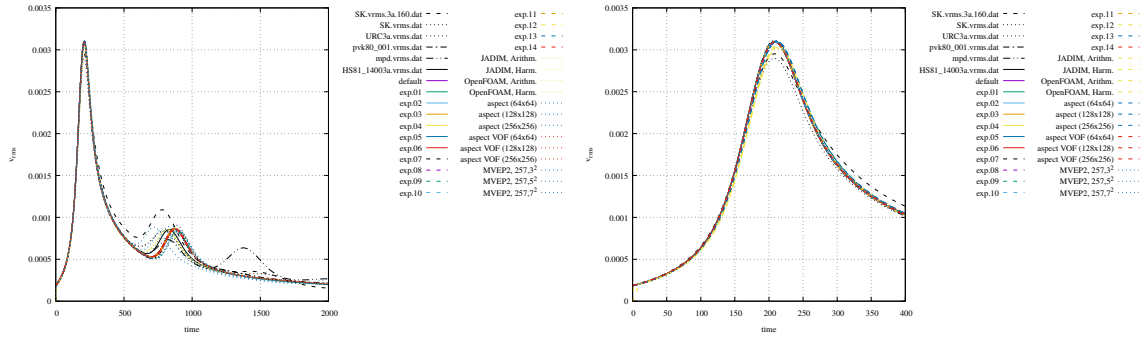
- number of elements:

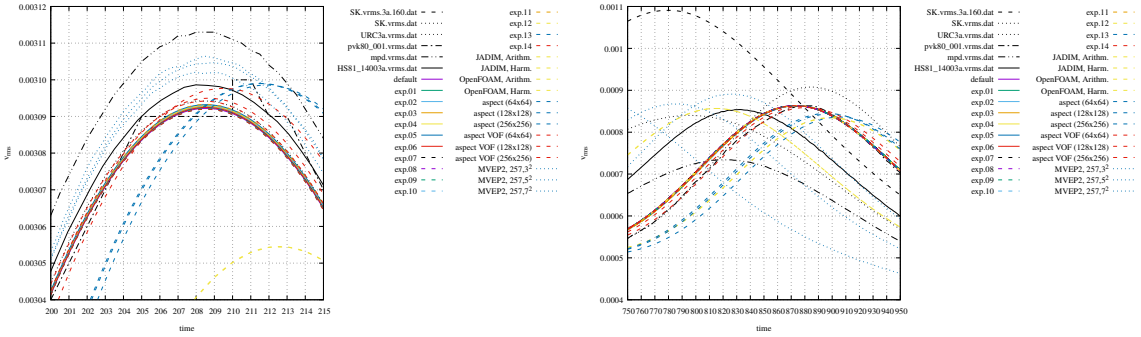


- velocity norm $|\vec{v}|$

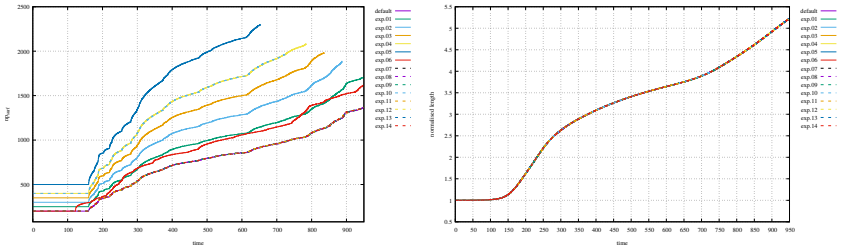


- The root mean square velocity

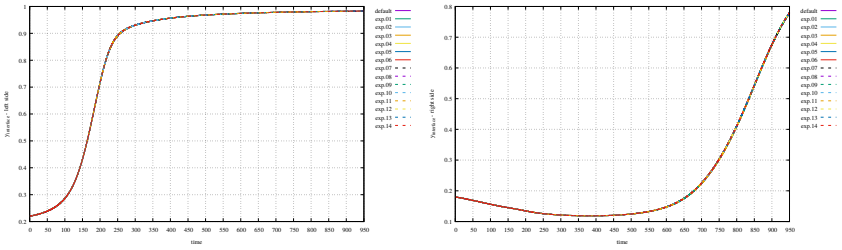




- the number of points on the interface and its length



- the position of interseaction of the interface with the left and right boundaries:



I have also plotted the vrms against those of the original paper and those of Louis-Napoleon *et al.* [220] (with codes JADIM and OpenFOAM). The main parameter which seems to govern overal mass/volume conservation is the resolution of the interface.

QUESTION: at the end of the first time step, when I write down the time associated with the measurements, should I write 0 or dt ?

QUESTION: does reduced density use change results ?

TODO: run until time=1500

TODO: viscosity ratio 10 and 100 cases

Stone 96

Blob in axisymmetric planet ($P_2 \times P_1$, $P_2^+ \times P_{-1}$)



python_codes/fieldstone_96/text.tex

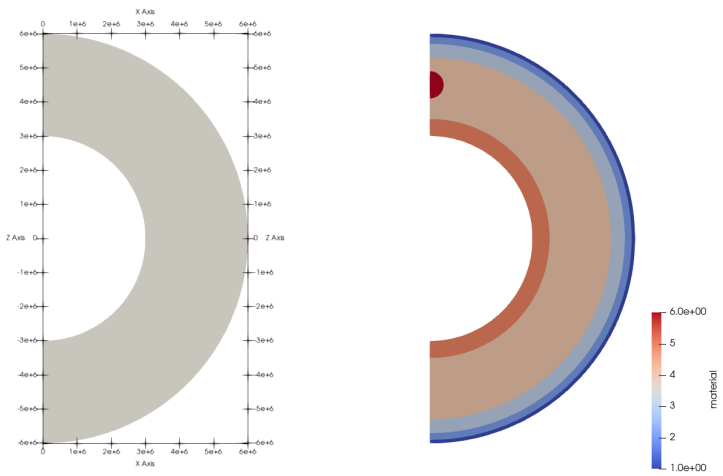
2D
axisymmetric
annulus
P2pPm1
P2P1
sparse_matrix
stokes_equation
time_stepping
use_mesher
viscous_rheology
buoyancy_driven
vrms
paraview_output
pressure_normalisation
gravity

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_96`

This stone was developed in collaboration with Marjolein Blasweiler & Bart Root.

This stone is abandoned and has been superseded by stone 152.

The domain is a half annulus and there are 6 materials in the domain: 5 concentric layers and a 'blob', a half disc which center lies on the z-axis, as shown in the following figure.



Example with $R_{inner} = 3000$ km and $R_{outer} = 6000$ km.

The thicknesses of the layers, their density, viscosity and the position, radius, density and viscosity of the blob can be adjusted.

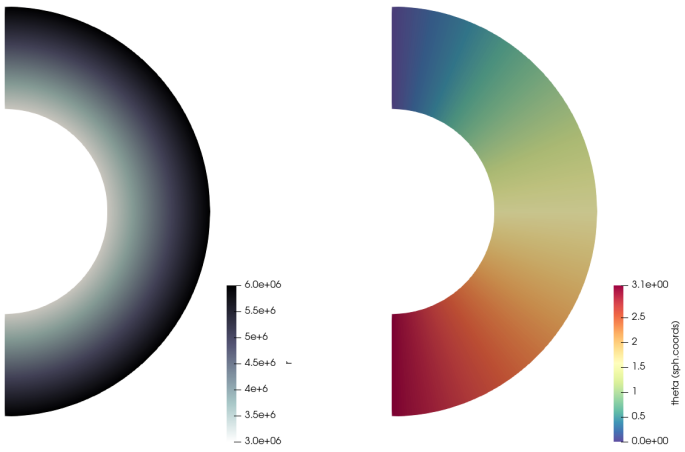
Gravity is radial and points towards the center of the planet. The code relies on $P_2 \times P_1$ elements or Crouzeix-Raviart elements. Fluids are Newtoniani, incompressible and temperature effects are neglected. Boundary conditions are either free slip or no slip.

The code carries out four main tasks:

1. it solves the Stokes equations under the assumption that the problem is axisymmetric
2. from the obtained velocity and pressure fields, it computes strain rate and stress fields
3. it computes the dynamic topography on the inner and outer boundary
4. it computes the static gravity field as generated by the density field (again under the axisymmetric assumption)

Results are then exported to ascii and vtu file(s).

The r and θ spherical coordinates are defined as follows:



Example with $R_{inner} = 3000$ km and $R_{outer} = 6000$ km.

How is mesh built

The convex hull is built with points which are (on average) distant by h, a parameter specified by the user (`hhh`). The same is done for the material interfaces. The resulting set of nodes is then passed to the *triangle* mesher, as explained in STONE XXX. This returns a mesh of triangles. However, we need mid-edge nodes for the Stokes equation so this is what the `mesh_P1_to_P2` does. In the case of the Crouzeix-Raviart element, mid-triangle nodes are also added.

include here figure with nodes

The pressure solution p is projected onto the velocity mesh and is then called q.

About pressure normalisation

In spherical coordinates, we actually want to compute

$$\langle p \rangle_\Gamma = \frac{1}{4\pi R_o^2} \iiint \delta(r - R_o) p(\theta) r^2 \sin\theta \, dr d\theta d\phi = \frac{1}{2} \int_0^\pi p(\theta) \sin\theta \, d\theta$$

The integral is then carried out by finding the elements for which two vertices are on the outside boundary, computing the angular opening $d\theta$, the average pressure along the edge and the value of $\sin\theta$ at the middle of the edge. The obtained value is then subtracted from the pressure.

From Cylindrical to Cartesian coordinates

We will now convert it to Cartesian coordinates using the equation from `https://www.brown.edu/Departments/Engineering/Courses/En221/Notes/Polar_Coords/Polar_Coords.htm`, where $\boldsymbol{T}$ is a tensor:

$$\boldsymbol{T}_{Cyl} = \begin{pmatrix} T_{rr} & T_{r\theta} & T_{rz} \\ T_{\theta r} & T_{\theta\theta} & T_{\theta z} \\ T_{zr} & T_{z\theta} & T_{zz} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{T}_{Cart} = \begin{pmatrix} T_{xx} & T_{xy} & T_{xz} \\ T_{yx} & T_{yy} & T_{yz} \\ T_{zx} & T_{zy} & T_{zz} \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} T_{rr} & T_{r\theta} & T_{rz} \\ T_{\theta r} & T_{\theta\theta} & T_{\theta z} \\ T_{zr} & T_{z\theta} & T_{zz} \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In our case the calculation is taking place in the (x, z) plane, i.e. $\theta = 0$ so that the rotation matrices above are in fact identity matrices. We then simply have $\dot{\boldsymbol{\varepsilon}}_{Cart} = \dot{\boldsymbol{\varepsilon}}_{Cyl}$.

A proxy for elasto-viscous lithospheric rheology

The implementation of elastic behaviour in a velocity-pressure code is not trivial and is presented in Section 15.26. In the end the lithospheric viscosity will be an effective viscosity given by

$$\eta_{eff} = \frac{\mu \delta t}{1 + \mu/\eta \delta t}$$

This is only used if `use_ev` is true.

Computing the total mass of the planet based on radial profiles

REMARK: for now the code does not use radial profiles.

The mass of the shell is given by

$$M = \iiint_\Omega \rho(r) r^2 \sin\theta dr d\theta d\phi = 4\pi \int_{Ri}^{Ro} \rho(r) r^2 dr$$

The density profile is known on N intervals ($N+1$ points) between Ri and Ro so that

$$M = 4\pi \sum_{i=1}^{N} \int_{r_i^-}^{r_i^+} \rho(r) r^2 dr$$

where r_i^- and r_i^+ are the bounds of the interval i under consideration. Over this interval the density is linear and goes from $\rho(r_i^-) = \rho_i^-$ to $\rho(r_i^+) = \rho_i^+$. We have then $\rho(r) = ar + b$ with

$$\rho(r) = \frac{\rho_i^+ - \rho_i^-}{r_i^+ - r_i^-}(r - r_i^-) + \rho_i^- \qquad a = \frac{\rho_i^+ - \rho_i^-}{r_i^+ - r_i^-} \qquad b = -\frac{\rho_i^+ - \rho_i^-}{r_i^+ - r_i^-} r_i^- + \rho_i^-$$

so that the mass of a shell corresponding to interval i bounded by r_i^- and r_i^+ is

$$\begin{aligned} M_i &= \int_{r_i^-}^{r_i^+} (ar+b)r^2 dr \\ &= \int_{r_i^-}^{r_i^+} (ar^3+br^2) dr \\ &= a\int_{r_i^-}^{r_i^+} r^3 dr + b\int_{r_i^-}^{r_i^+} r^2 dr \\ &= \frac{a}{4}\left[(r_i^+)^4-(r_i^-)^4\right] + \frac{b}{3}\left[(r_i^+)^3-(r_i^-)^3\right] \end{aligned} \tag{96.1}$$

We can now look at each provided density profile for the planet and compute the corresponding mass.

Elliptic blob

The eccentricity e of an ellipse is defined as

$$e = \sqrt{1-\frac{b^2}{a^2}}$$

where a is the length of the semi-major axis and b is the length of the semi-minor axis.

The eccentricity of ellipse is less than 1. The eccentricity of a ellipse helps us to understand how circular it is with reference to a circle. Eccentricity also measures the ovalness of the ellipse and eccentricity close to one refers to high degree of ovalness.

In our case we assume that the semi-major axis is along the x-axis. We will set $b = R_{blob}$ and vary the eccentricity so as to obtain a circle ($e = 1$) or an ellipse $e < 1$. We have

$$e^2 = 1-\frac{b^2}{a^2} \quad \rightarrow \quad 1-e^2 = \frac{b^2}{a^2} \quad \rightarrow \quad \frac{1}{1-e^2} = \frac{a^2}{b^2} \quad \rightarrow \quad a^2 = b^2\frac{1}{1-e^2} \quad \rightarrow \quad a = b\sqrt{\frac{1}{1-e^2}}$$

The volume of the blob is then

$$V_{blob} = \frac{4\pi}{3}a^2 b$$

Gravity calculations

This is an axisymmetric problem so the mesh is generated for a single value of the cylindrical coordinates θ angle. This makes gravity calculations a bit more difficult because each triangle of the mesh corresponds in 3D to a ring of matter with this triangular cross section. I have therefore chosen to divide this ring in `nel_phi` segments. Each segment has a given volume and therefore mass, and having computed its center its generated gravity field can be added. The volume of such a block is $\iiint_{block} r dr d\theta dz = \iint_{\triangle} r dr dz \cdot \int d\theta$. The first integral is carried out for each triangle and the result is stored in the `arear` array.

Look at Morgan (1965) [238] for gravity anomalies above sinking sphere.

Open question: what is an optimal value for `nel_phi` ?

Basic facts about the planet Mars

Facts about planet Mars[1]:

- average radius $R = 3389.5 \pm 0.2$km
- equatorial radius 3396.2 ± 0.1km
- polar radius 3376.2 ± 0.1km
- volume $1.6318 \cdot 10^{20}\mathrm{m}^3$
- mass $6.4171 \cdot 10^{23}$kg
- mean density $\langle \rho \rangle = 3934\mathrm{kg\,m}^{-3}$
- moment of inertia $I = 0.3644 \pm 0.0005$
- surface gravity $g = 3.72076\mathrm{m\,s}^{-2}$

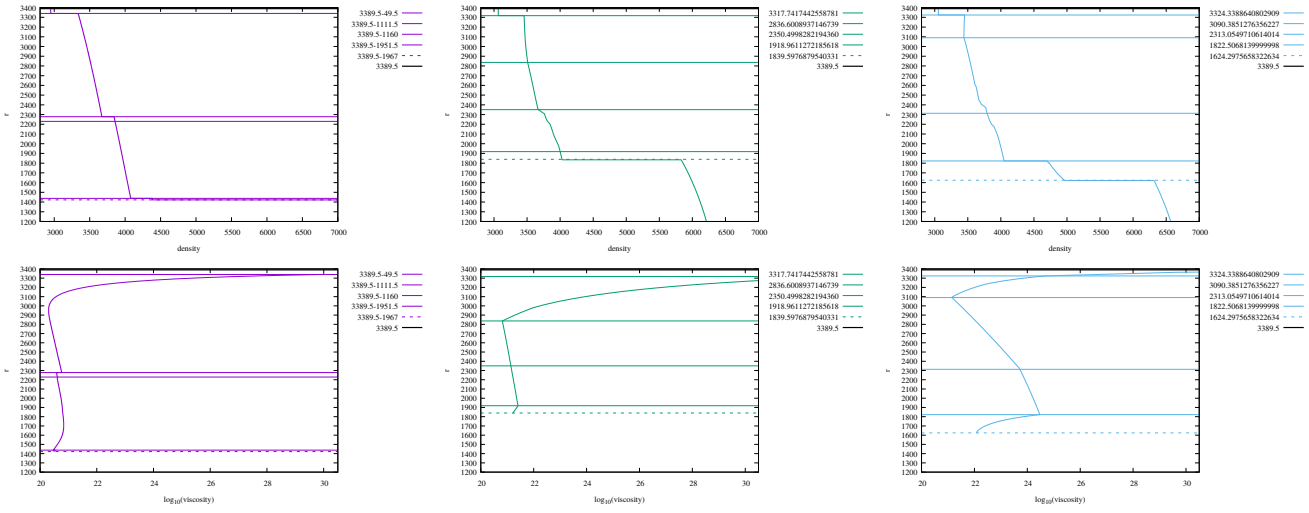
The surface gravity can be obtained with

$$g_{surf} = \frac{\mathcal{G}M}{R^2} = \frac{6.67430 \cdot 10^{-11}\ 6.4171 \cdot 10^{23}}{(3.3895 \cdot 10^6)^2} \simeq 3.727977$$

Internal structure of the planet Mars

The internal structure of the planet is not settled although it is now widely accepted that the planet has a core. News alert: [187, 304, 196].

B. Steinberger was kind enough to communicate to us the density and viscosity profiles used in Steinberger *et al.* (2010) [309] [2]:

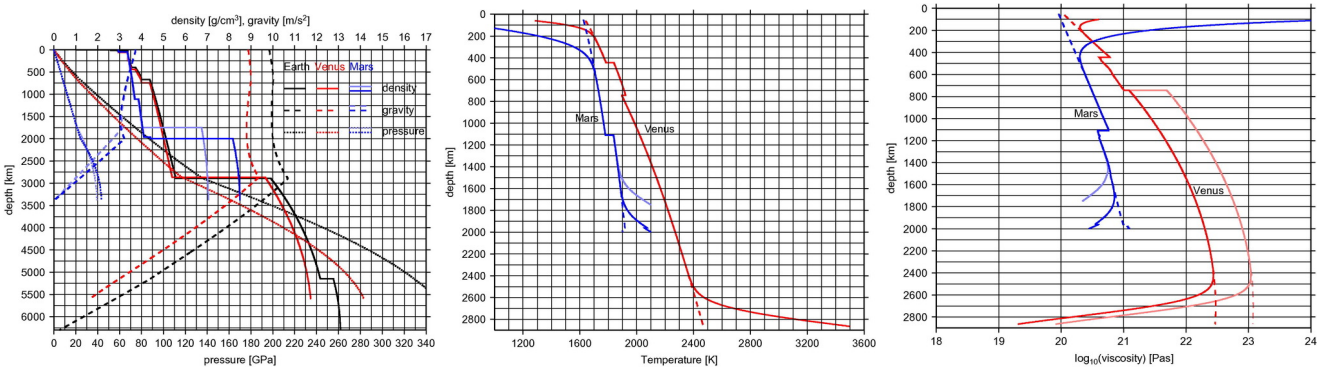


Data curtesy of B. Steinberger, from [309]. Samuel data via private communication.

in python_codes/fieldstone_96/data/

[1] `https://en.wikipedia.org/wiki/Mars`

[2] Files sent to us we slightly altered for the purpose of this work. The density profile was missing the 50km near the center of the planet so padding was used. The viscosity profile starts below the moho at 50km and stopped at the cmb.



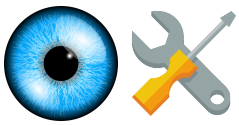
Taken from Steinberger *et al.* (2010) [309]

In table 1 of Steinberger *et al.* [309]: crust thickness is set to 50km. The core radius is set to 1389.5km. However in the data set we were sent it seems that the cmb is at 1422km radius. To simplify things we take $R = 3389$km and $R_{cmb} = 1422$km in the code.

Sort this out

Stone 97

using the CRUST 1.0 dataset



```
paraview_output
dataset
spherical_shell
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_97`

All CRUST 1.0 data are available at `https://igppweb.ucsd.edu/~gabi/crust1.html`. The files `crust1.rho` and `crust1.bnds` count 64,800 lines, corresponding to 360x180 points, i.e. a 1° resolution.

As explained on the website: "The model is defined from 89.5 to -89.5 deg latitude (so that in fact the data is the co-latitude) and -179.5 to 179.5 deg longitude. Longitudes are the inner loop, i.e. all longitudes are stored for each latitude, then the next latitude is given. The model starts at 89.5 N and 179.5 W."

The structure of the CRUST 1.0 data is as follows:

```
============================================================
 crust1.rho | layer | name              | crust1.bnds
 -----------------------------------------|----0-------
 1          | 0     | water             |
 -----------------------------------------|----1-------
 2          | 1     | ice               |
 -----------------------------------------|----2-------
 3          | 2     | upper sediments   |
 -----------------------------------------|----3-------
 4          | 3     | mid sediments     |
 -----------------------------------------|----4-------
 5          | 4     | lower sediments   |
 -----------------------------------------|----5-------
 6          | 5     | upper crust       |
 -----------------------------------------|----6-------
 7          | 6     | mid crust         |
 -----------------------------------------|----7-------
 8          | 7     | lower crust       |
 -----------------------------------------|----8-------MOHO
 9          | 8
============================================================
```

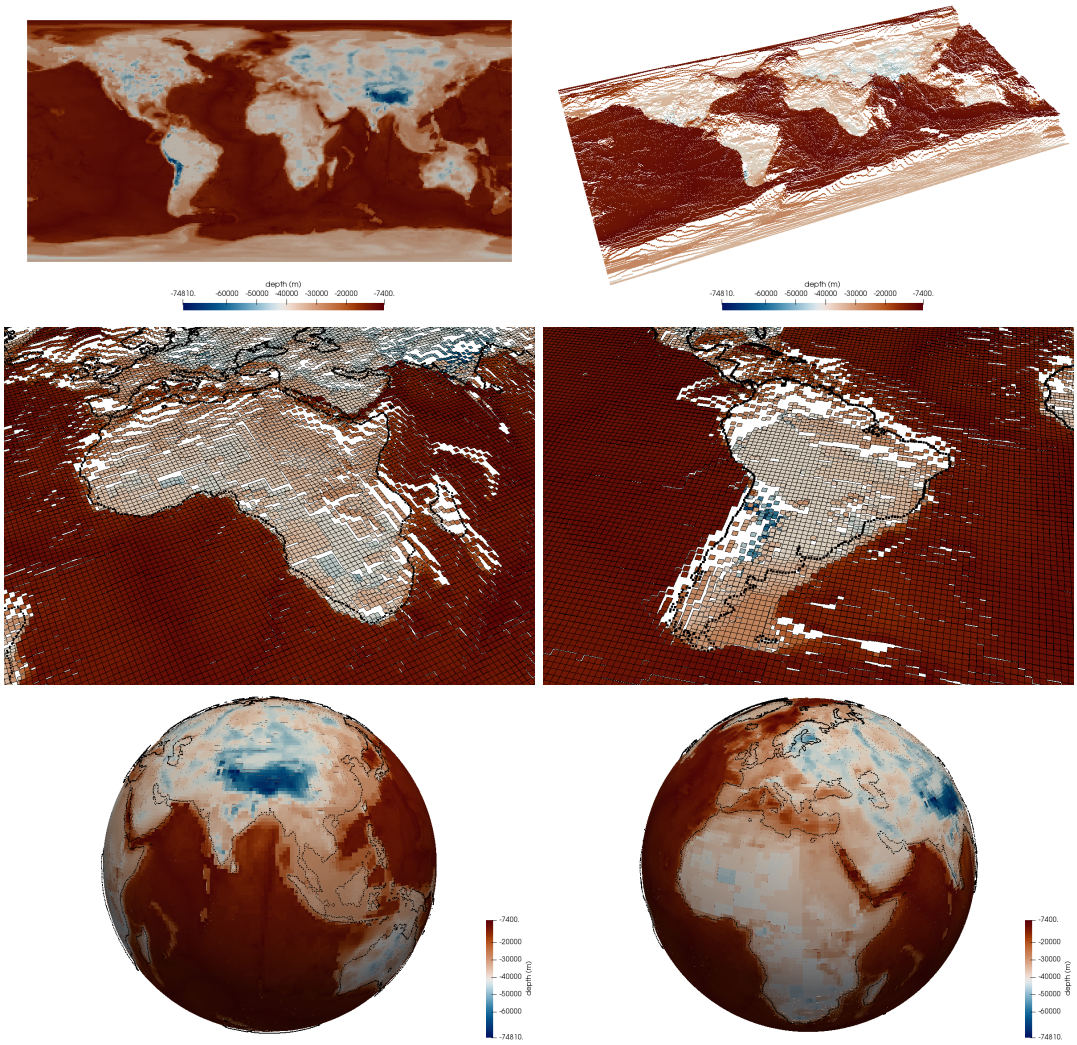
This stone allows the user to choose a minimum and maximum radius, as well as a number of shells in between `nrad`, and rewrites the data in spherical coordinates as the ascii file `crust1p0_full_for_aspect.ascii` which is in the format that ASPECT expects. Another file, `crust1p0_moho_for_aspect.ascii`, is produced. Its structure and intent is similar to the first one, except that points above the moho are assigned a single density $\rho = 2900\text{kg}\,\text{m}^{-3}$ while points below the moho are assigned $\rho = 3300\text{kg}\,\text{m}^{-3}$.

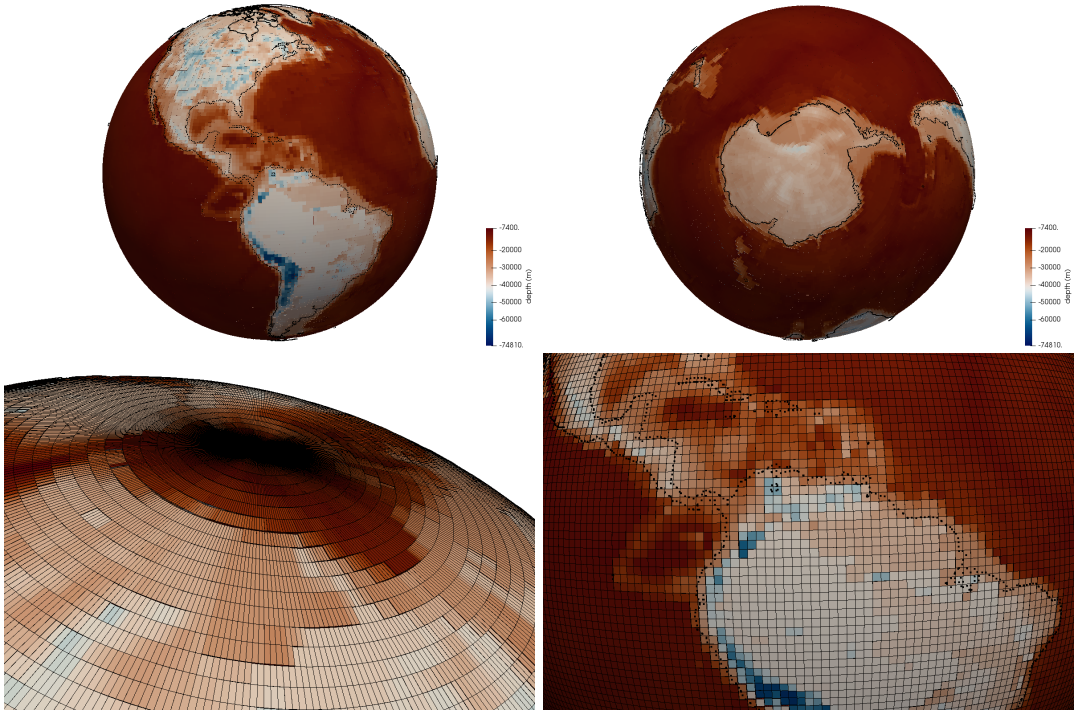
Very importantly, ASPECT does not read in densities but temperatures. This stone then converts the density values into temperature using the simple relationship

$$\rho = \rho_0(1 - \alpha(T - T_0)) \qquad \text{or,} \qquad T = \frac{1}{\alpha}\left(1 - \frac{\rho}{\rho_0}\right)$$

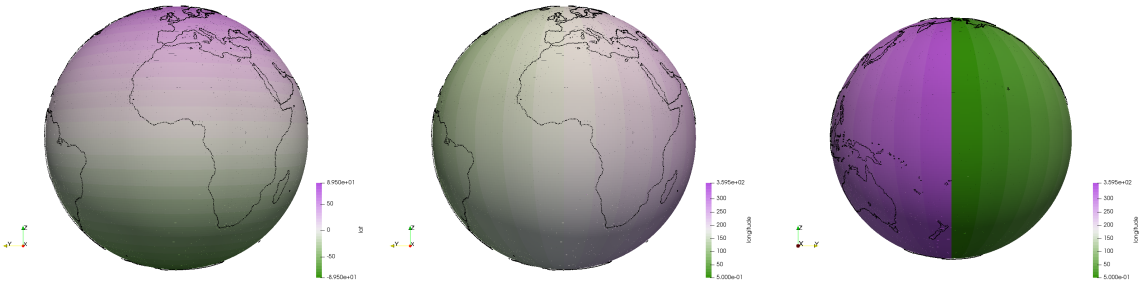
and we here take $\alpha = 3 \cdot 10^{-5}\text{K}^{-1}$, $T_0 = 0\text{K}$ and $\rho_0 = 3300\text{kg}\,\text{m}^{-3}$. This temperature is meaningless, but when used in the material model of ASPECT , it yields the expected density field.

CRUST 1.0 also comes with the file `depthtomoho.xyz` which name is self-explanatory. Values are stored at coordinates $x.5$, $y.5$ (with $x \in [0, 359]$ and $y \in [-89 : 89]$) so I assign a 1x1degree cell around it and give the whole cell the associated depth. The data is exported as vtu file. One is a 2D lat-lon map (the vertical elevation of the map follows the moho depth) named `moho_map.vtu`, the other is a spherical mesh where each cell is assigned the radius of the Earth minus the moho depth, named `moho_shell.vtu`.





colormap=vik, see `http://www.fabiocrameri.ch/vik.php`. Continent boundaries are from Stone 69.

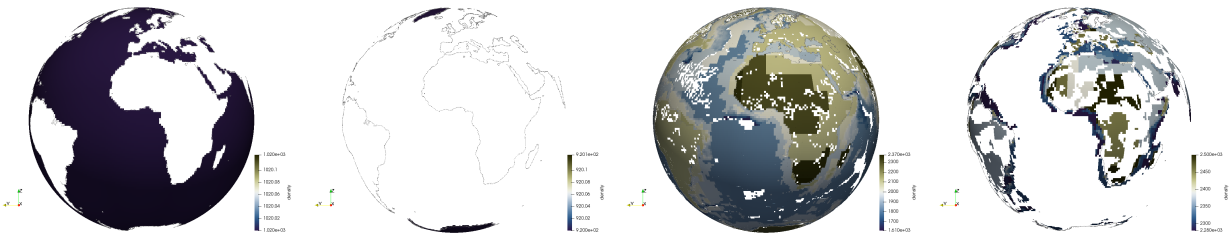


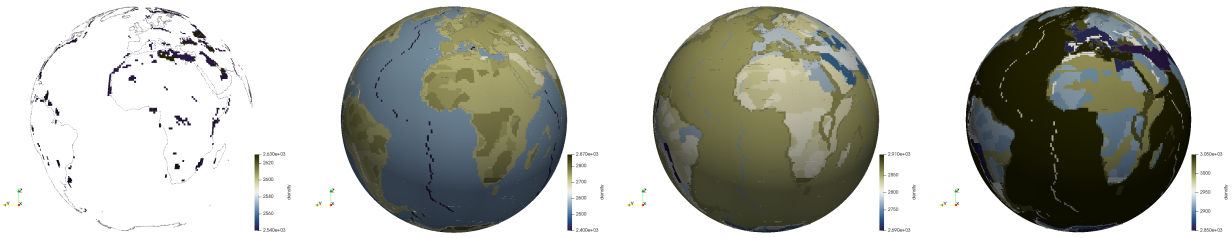
Note that latitudes start at 0 and go to 360, with 180 corresponding to Greenwich.

Subsequently the code uses the same data, loops over each cell and assesses for each layer if it has non zero thickness. In this case it is added to the layer list. We find:

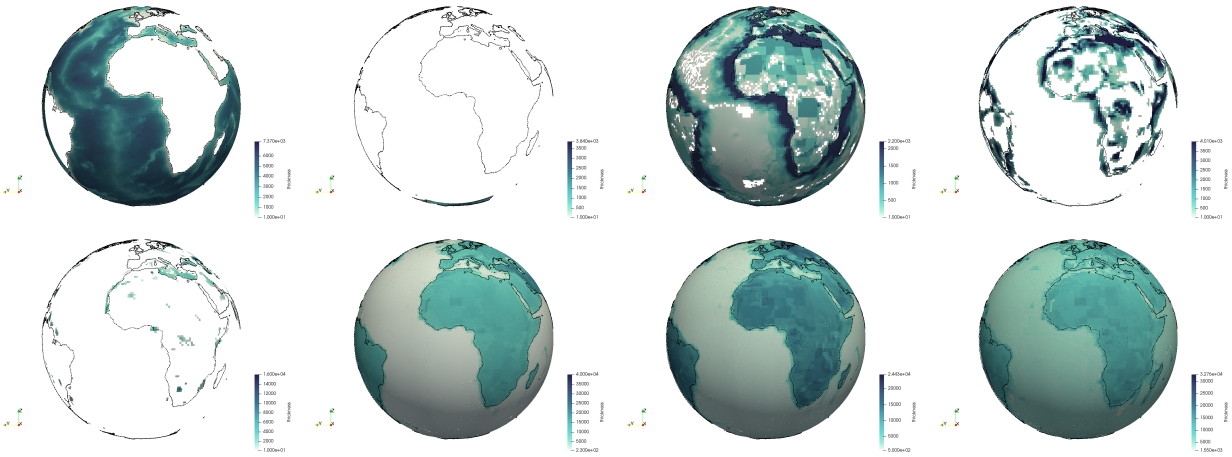
```
water      : 42549 cells out of 64800
ice        : 7550  cells out of 64800
up. seds.  : 60650 cells out of 64800
mid. seds. : 14355 cells out of 64800
low. seds. : 2471  cells out of 64800
up. crust  : 64800 cells out of 64800
mid. crust : 64800 cells out of 64800
low. crust : 64800 cells out of 64800
```

From the lat-lon coordinates and the depths of the layer boundaries the coordinates of the 8 corners of the cell are computed and transformed into x, y, z coordinates and exported to its respective vtu file (one per layer). These 8 files (water, ice, upper-middle-lower sediments and upper-middle-lower crust) are shown hereunder:

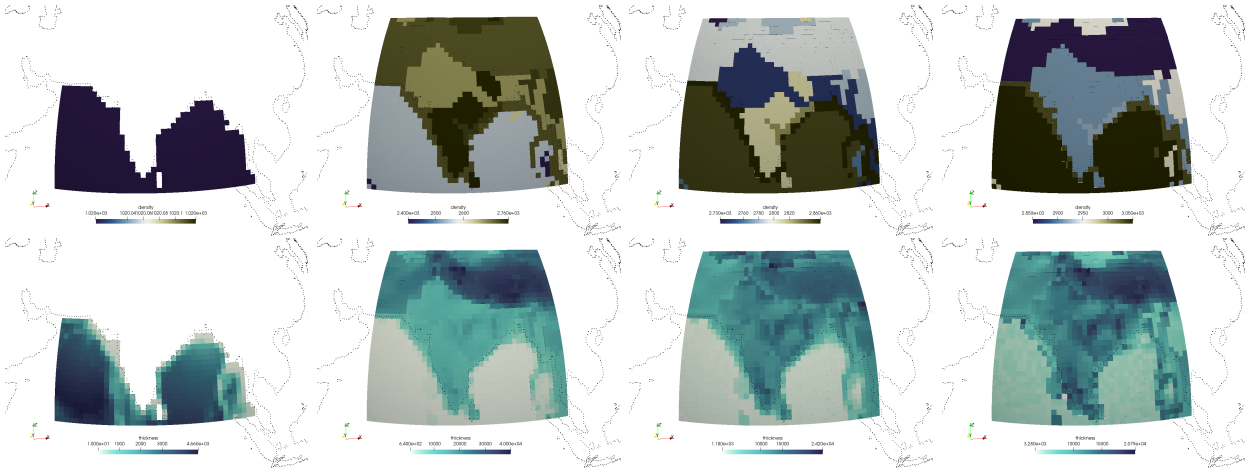




From left to right, top to bottom, layers 0 to 7. colormap=vik[1]. Continent boundaries are from Stone 69.



From left to right, top to bottom, layers 0 to 7. Continent boundaries are from Stone 69.

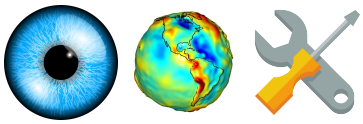


Water, upper, middle and lower crust layers, for longitudes between 60+180 and 100+180 and latitudes between 5 and 40.

[1] http://www.fabiocrameri.ch/vik.php

Stone 98

Single WINTERC layer gravity benchmark



```
3D
spherical_shell
gravity
paraview_output
numerical_benchmark
```

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_98`

This stone serves three purposes:

- read in the ascii data file `rho_56km_SH_W32.txt`, which is a single layer of a version of the WINTERC model, and rewrite in a the right format for ASPECT , while also transforming the density data into temperatures;
- make a 3D shell vtu file of `rho_56km_SH_W32.txt`;
- compute the gravity acceleration and potential of the density distribution.

This experiment is carried out in Root, Sebera, Szwillus, Thieulot, Martinec, and Fullea [268].

Part 1

Here the data file is read in, latitudes and longitudes are converted to spherical coordinates, densities are converted to temperatures, and the ASPECT data ascii file `bench2.txt` is generated. Because the density does not vary in the radial direction, only two values are stored in the ascii file, one slightly below and one slightly above the desired shell.

Part 2

The dataset has a half-degree resolution, i.e. there are 720*380=259,200 cells and to each cell a single density value is assigned. Denoting L_c an l_c the longitude and latitude of the cell center (respectively), and $R_c^- = 6371e3 - 80e3$ and $R_c^+ = 6371e3 - 56e3$ the inner and outer radii respectively then we can define 8 points which make up the bounds of the hexahedral cell:

	lon	lat	radius
0	$L_c - 0.25$	$l_c - 0.25$	R_c^-
1	$L_c + 0.25$	$l_c - 0.25$	R_c^-
2	$L_c + 0.25$	$l_c + 0.25$	R_c^-
3	$L_c - 0.25$	$l_c + 0.25$	R_c^-
4	$L_c - 0.25$	$l_c - 0.25$	R_c^+
5	$L_c + 0.25$	$l_c - 0.25$	R_c^+
6	$L_c + 0.25$	$l_c + 0.25$	R_c^+
7	$L_c - 0.25$	$l_c + 0.25$	R_c^+

The eight coordinates are then translated to Cartesian coordinates x_i, y_i, z_i with $i \in [0, 7]$ and these now form a new hexahedral cell which spans the surface of the shell. This cell geometry is called a tesseroid[1]. Note that 1) an equidistant lat-lon sampling always leads to an oversampling near the poles when the values are projected on a sphere 2) the cell touching the poles are degenerate and become triangular prims.

The total volume of the shell is

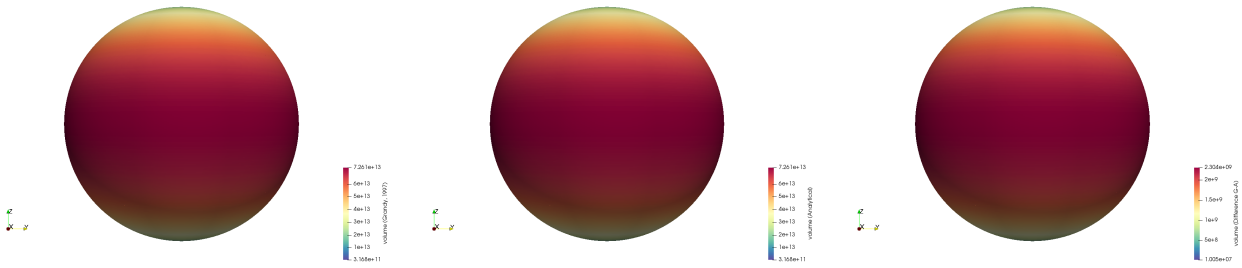
$$V = \frac{4}{3}\pi((R^+)^3 - (R^-)^3) \simeq 1.1981639394630371 \cdot 10^{19}\mathrm{m}^3$$

Having done so, we can now compute the volume of each cell (in the Cartesian coordinates space). I first started by re-using a function I already had, based on the technical report by Grandy (1997) [138]. The volume of a cell is arrived at through geometrical considerations and is stored in the `vol1` array.

Another way of computing the volume of a cell is by computing the following integral

$$\begin{aligned} V_c &= \int_{R^-}^{R^+}\int_{\phi^-}^{\phi^+}\int_{\theta^-}^{\theta^+} r^2 \sin\theta dr d\theta d\phi \\ &= \frac{1}{3}[(R^+)^3 - (R^-)^3](\phi^+ - \phi^-)[-\cos\theta]_{\theta^-}^{\theta^+} \\ &= \frac{1}{3}[(R^+)^3 - (R^-)^3](\phi^+ - \phi^-)(\cos\theta^- - \cos\theta^+) \end{aligned} \tag{98.1}$$

Note that $\phi^+ - \phi^- = 0.5°$ (which should be converted to radian). Results are stored in the array `vol2`.



Left to right: cell volume as obtained with Grandy's formula, analytical calculation, difference between the two.

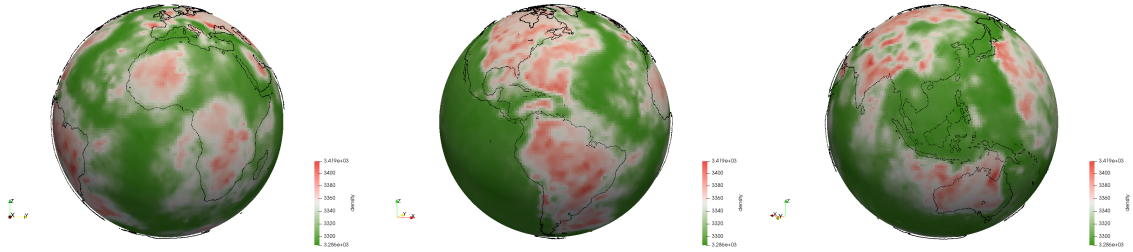
Unsurprisingly, we see that the analytical formula gives more accurate results:

```
vol1 (m/M): 316812248681.0113 72607558204572.3
vol2 (m/M): 316822301633.22815 72609862156914.12
total vol1: 1.1981259210377495e+19 (anal: 1.1981639394630371e+19)
total vol2: 1.1981639394630423e+19 (anal: 1.1981639394630371e+19)
total vol rel. error: 0.00317305704465769 %
total vol rel. error: 4.2732048857141803e-13 %
```

Having obtained the volume of each cell, I then compute their mass by multiplying their `vol1` and `vol2` values by their density, to obtain `mass1` and `mass2` respectively:

```
mass1 (m/M/total): 1050831686835569.9 2.47169850817731e+17 3.980924983467316e+22
mass2 (m/M/total): 1050865031381870.6 2.471776939069733e+17 3.9810513044962e+22
```

[1] `https://tesseroids.readthedocs.io/en/stable/theory.html`



Density field

Given a cell, one can then ask the question of the location of its center of mass. As explained for example on Wikipedia[2] the center of mass coordinates $\vec{R}_c$ of the cell is given by:

$$\iiint_V (\vec{r} - \vec{R}_c)\rho(\vec{r})dV = \vec{0}$$

or,

$$\vec{R}_c = \frac{\iiint_V \rho(\vec{r})\vec{r}dV}{\iiint_V \rho(\vec{r})dV} = \frac{\iiint_V \vec{r}dV}{\iiint_V dV}$$

since the density is constant within a cell. The denominator is simply the volume of the cell V_c which we have previously computed.

$$\begin{aligned}
\vec{R}_c|_x &= \frac{1}{V_c}\iiint_V x dV \\
&= \frac{1}{V_c}\int_{R^-}^{R^+}\int_{\phi^-}^{\phi^+}\int_{\theta^-}^{\theta^+}(r\sin\theta\cos\phi)r^2\sin\theta dr d\theta d\phi \\
&= \frac{1}{V_c}\int_{R^-}^{R^+}\int_{\phi^-}^{\phi^+}\int_{\theta^-}^{\theta^+} r^3\sin^2\theta\cos\phi\ dr d\theta d\phi \\
&= \frac{1}{V_c}\frac{1}{4}[(R^+)^4 - (R^-)^4]\cdot I_1 \cdot I_2
\end{aligned} \tag{98.2}$$

with

$$\begin{aligned}
I_1 &= \int_{\theta^-}^{\theta^+}\sin^2\theta\ d\theta = \left[\frac{\theta}{2} - \frac{1}{4}\sin 2\theta\right]_{\theta^-}^{\theta^+} \\
I_2 &= \int_{\phi^-}^{\phi^+}\cos\phi d\phi = \sin\phi^+ - \sin\phi^-
\end{aligned}$$

and then

$$\begin{aligned}
\vec{R}_c|_y &= \frac{1}{V_c}\iiint_V y\ dV \\
&= \frac{1}{V_c}\int_{R^-}^{R^+}\int_{\phi^-}^{\phi^+}\int_{\theta^-}^{\theta^+}(r\sin\theta\sin\phi)r^2\sin\theta dr d\theta d\phi \\
&= \frac{1}{V_c}\frac{1}{4}[(R^+)^4 - (R^-)^4]\cdot I_1 \cdot I_3 \\
\vec{R}_c|_z &= \frac{1}{V_c}\iiint_V z\ dV \\
&= \frac{1}{V_c}\iiint_V (r\cos\theta)r^2\sin\theta\ dr d\theta d\phi \\
&= \frac{1}{V_c}\iiint_V r^3\cos\theta\sin\theta\ dr d\theta d\phi \\
&= \frac{1}{V_c}\frac{1}{4}[(R^+)^4 - (R^-)^4]\cdot I_4 \cdot (\phi^+ - \phi^-)
\end{aligned} \tag{98.3}$$

[2] https://en.wikipedia.org/wiki/Center_of_mass

with

$$I_3 = \int_{\phi^-}^{\phi^+} \sin\phi \; d\phi = -\cos\phi^+ + \cos\phi^-$$

$$I_4 = \int_{\theta^-}^{\theta^+} \sin\theta\cos\theta \; d\theta = \left[-\frac{1}{2}\cos^2\theta\right]_{\theta^-}^{\theta^+}$$

Each cell has now a volume/mass associated to it and we assume that it is a point mass at the location of its center of mass.

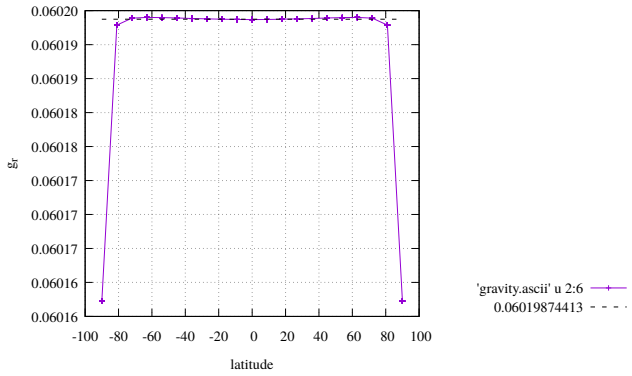
Part 3a

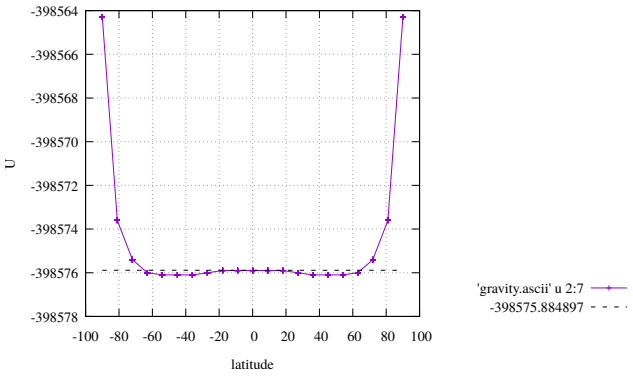
The gravity acceleration and potential are computed on a longitude/latitude grid at a height of 250km above the Earth surface at 6371km, i.e. $R_g = 6621\text{km}$ using a simple summation over all cells.

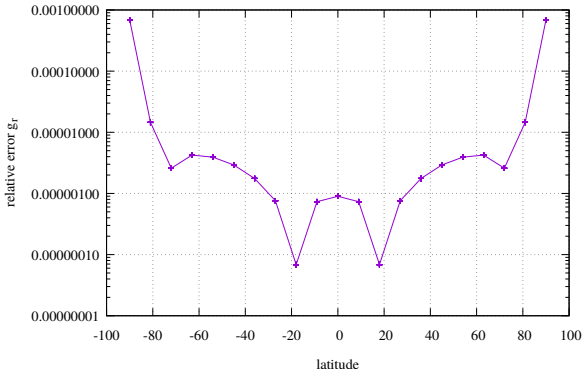
Before I compute the gravity fields generated by the WINTERC layer, I set the density in the shell to a constant value $\rho_0 = 3300\text{kg}\,\text{m}^{-3}$. In the interest of time I compute gravity on a $(18+1)\times(9+1)$ grid. We expect

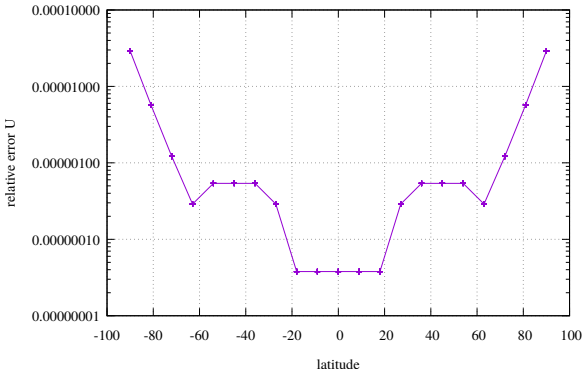
$$g = \frac{\mathcal{G}M_{shell}}{R_g^2} = \mathcal{G}\frac{4}{3}\pi\frac{(R^+)^3-(R^-)^3}{R_g^2} \simeq 0.06019874413$$

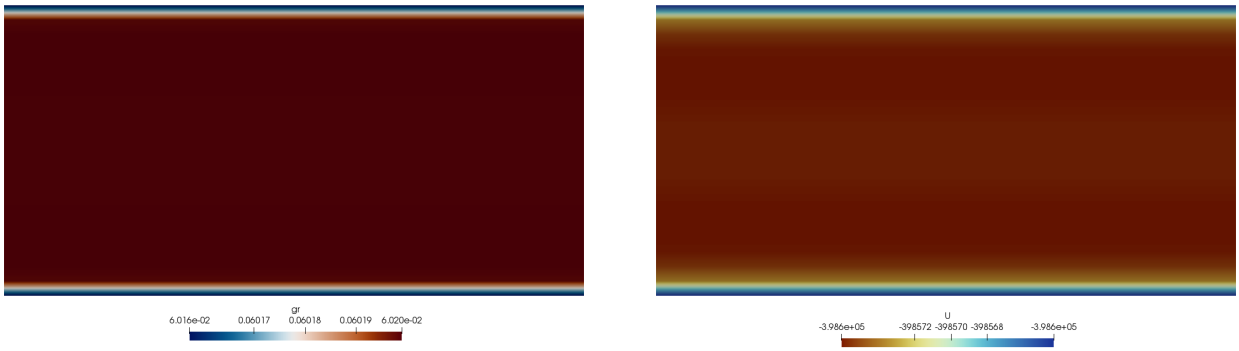
$$U = -\frac{\mathcal{G}M_{shell}}{R_g} = -\mathcal{G}\frac{4}{3}\pi\frac{(R^+)^3-(R^-)^3}{R_g} \simeq -398575.884897$$







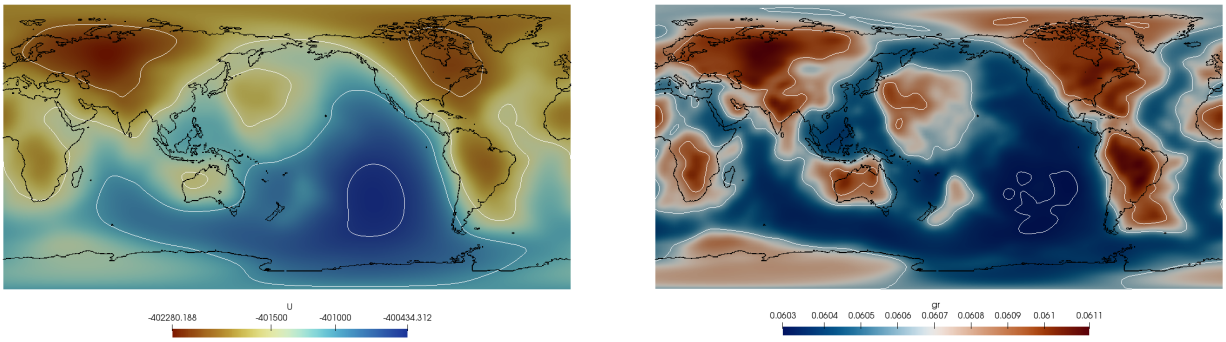




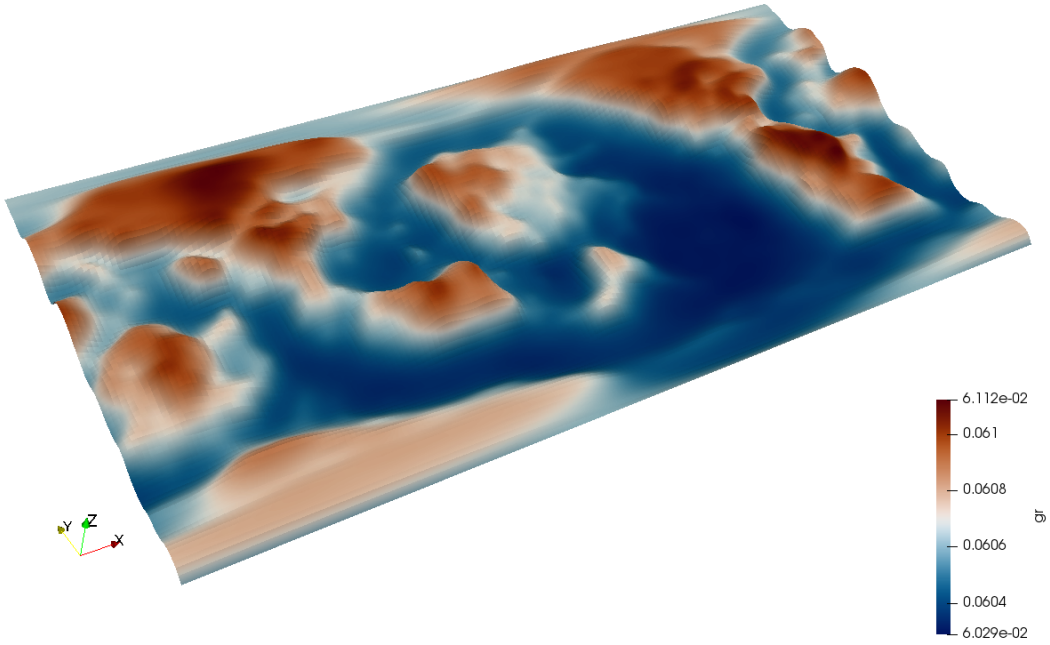
We sse that results are quite accurate, except near the poles, which is to be attributed to the shape of the cells/tesseroids.

Part 3b

We can now compute the gravity fields on the WINTERC shell:



Resolution of measurement grid is 181x91. It took about 19,100 seconds to run, averaging 1.16s per measurement point. Potential isocontours at -400.5e3, -401e3, -401.5e3, -402e3. Radial acceleration contours at 0.0603, 0.0606 and 0.0609



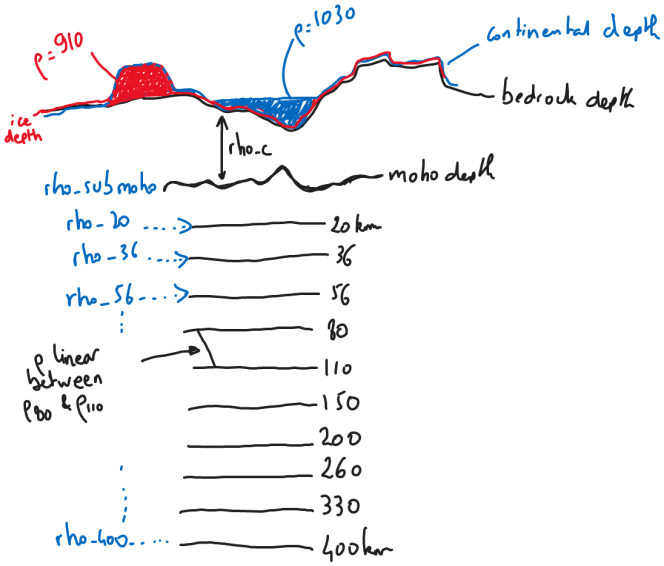
Radial acceleration, warped in Paraview, factor 25000.

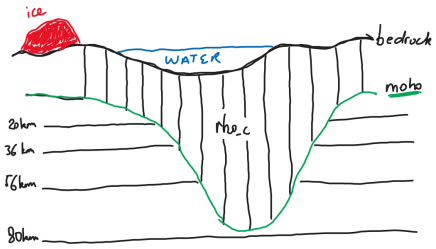
As mentioned earlier, the cells, when placed in a 3D space, are tesseroids. As explained at `https://tesseroids.readthedocs.io/en/stable/theory.html` the gravitational potential of a tesseroid can be calculated.

Stone 99

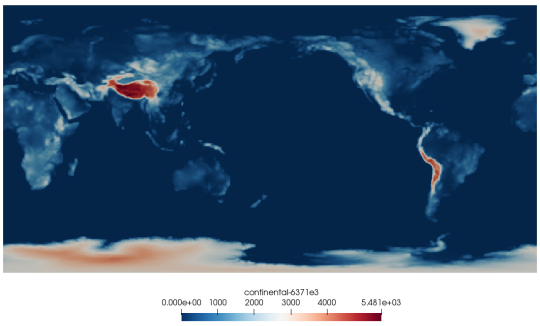
reading in the whole WINTERC 5.4 dataset

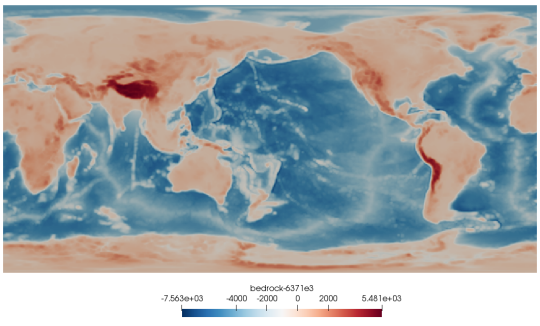
The data format of WINTERC 5.4 is quite complicated and is best summarised as follows:

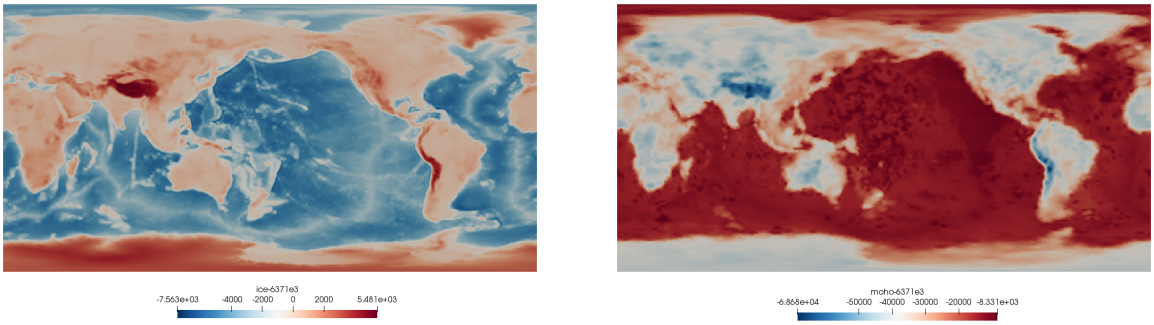




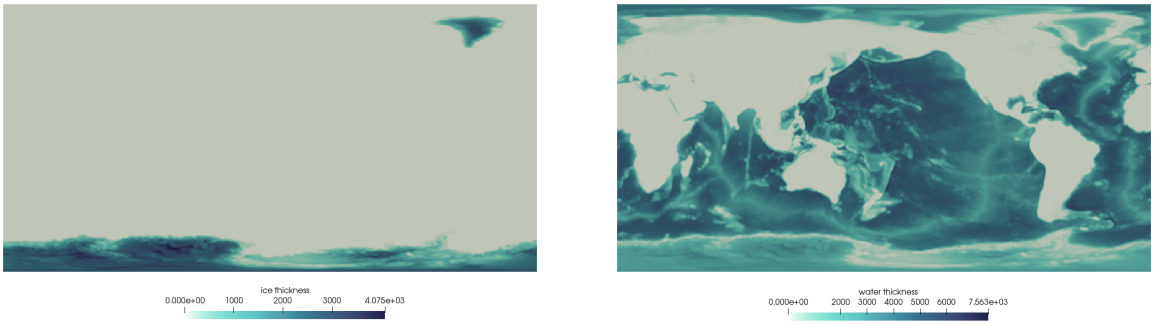
Below 20km depth densities are given at various depths (36, 56, 80, 110, 150, 200, 260, 330 and 400km) and values in between should be interpolated linearly. There are four files ETOPO2_km_continental.xyz, ETOPO2_km_depth_Bed.xyz, ETOPO2_km_depth_Ice.xyz and Global_Moho_CRUST1.0.xyz which contain the location of these four boundaries at 1/4 degree resolution.





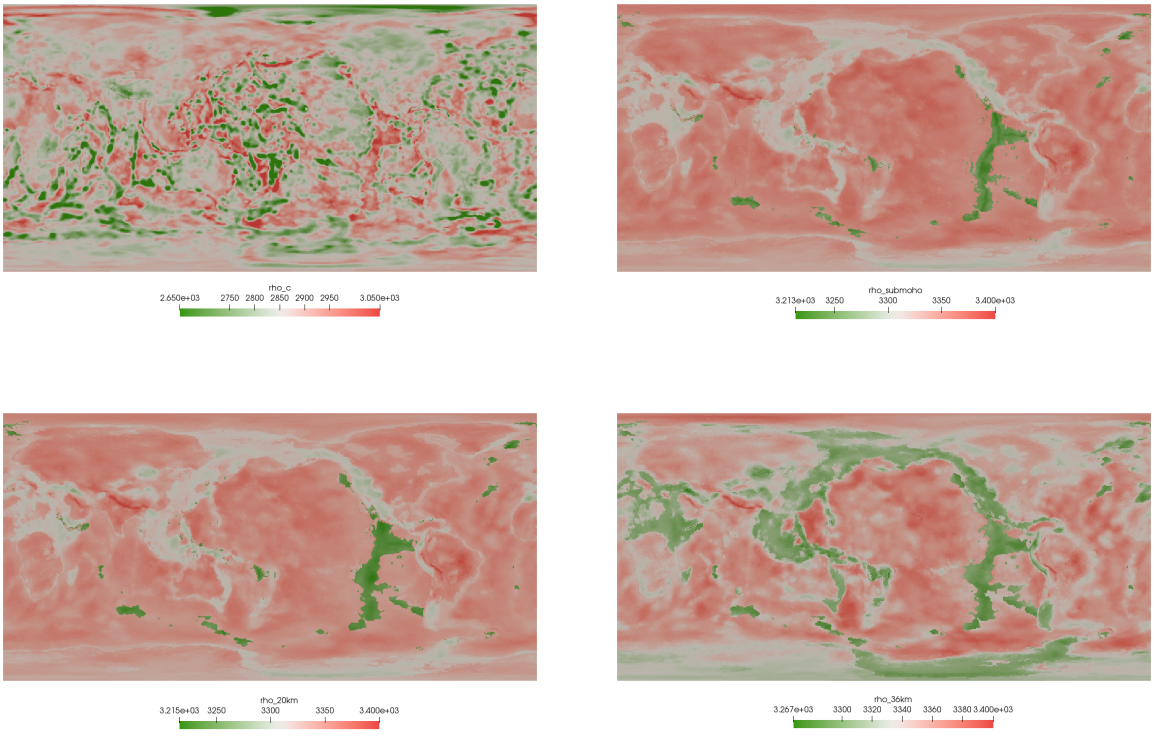


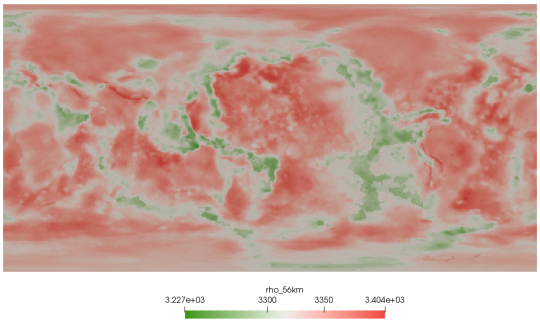
Topography of the four key layers.

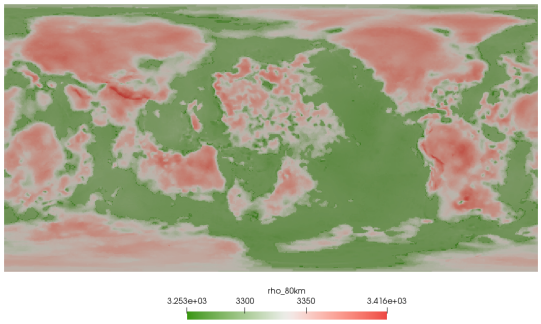


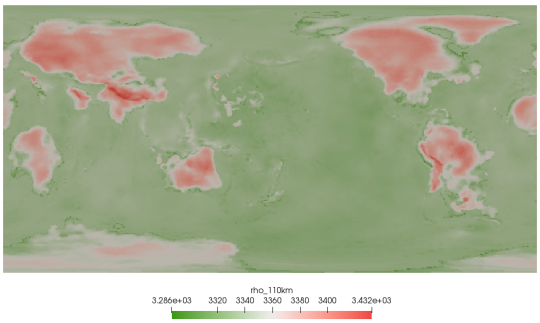
computed thicknesses of ice and water.

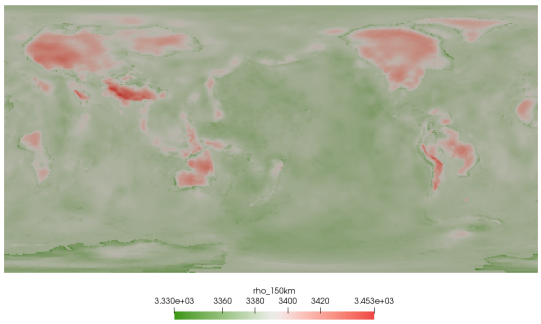
Between the bedrock and the moho the density is radially constant but changes laterally. It is stored in rho_c_out.xyz. The moho has its own density file rho_submoho_out.xyz Water has a density of $1030\,\mathrm{kg\,m^{-3}}$ and ice of $910\,\mathrm{kg\,m^{-3}}$.

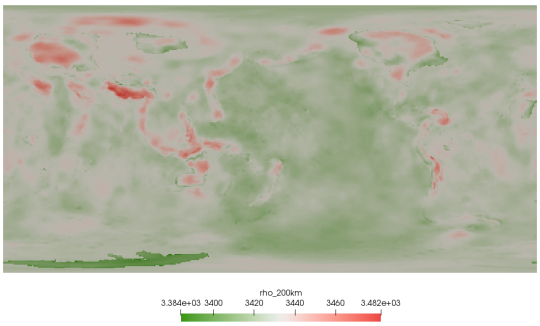


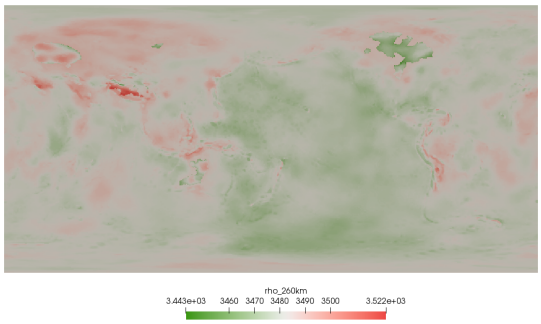


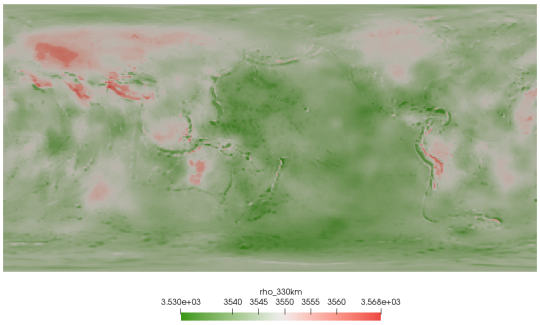


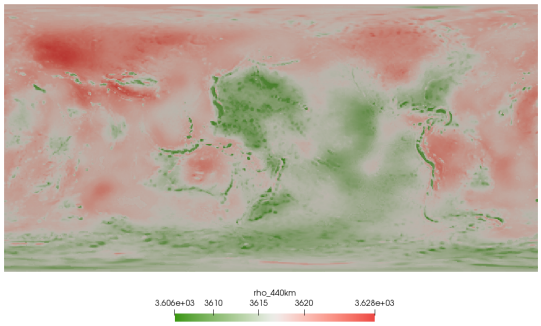












Density values at depths 20, 36, 56, 80, 110, 150, 200, 260, 330, 400km.

Stone 100

the MOLA Mars topography & computing gravity



python_codes/fieldstone_100/text.tex

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_100`

This fieldstone was developed in collaboration with B. Root.

Very high resolution topography datasets of Mars are available at `https://astrogeology.usgs.gov/search/details/Mars/GlobalSurveyor/MOLA/Mars_MGS_MOLA_DEM_mosaic_global_463m/cub`. However the data format is not very convenient so Bart Root helped me out and exported the data to ascii format.

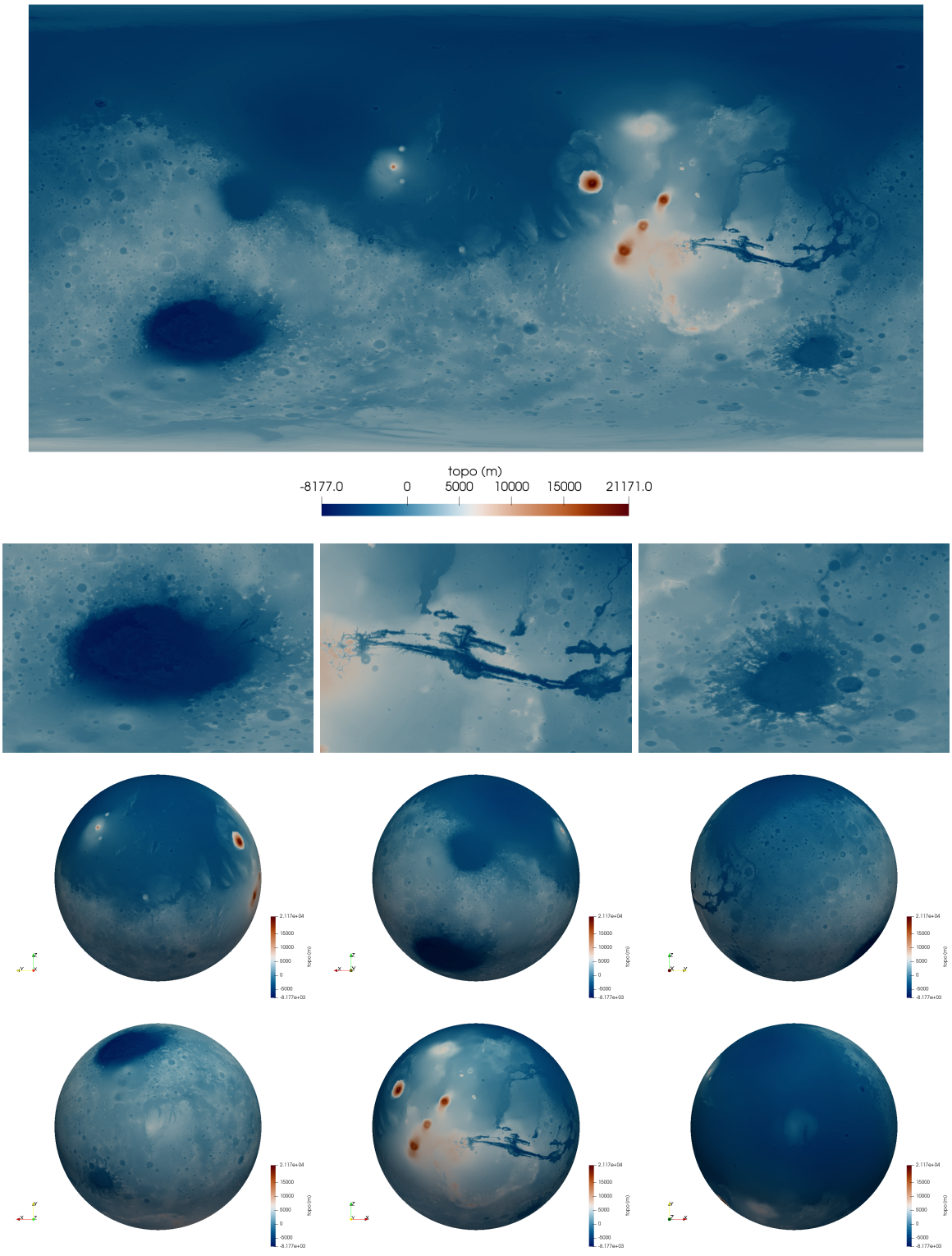
The file *MOLA_1deg.txt* counts 64,800 lines, i.e. 360x180 points. Longitude values range from 0.5 to 359.5 and latitudes from -89.5 to +89.5.

The file *MOLA_0.5deg.txt* counts 4 times as many lines, i.e. 720x360 points=259,200. Longitude values range from 0.25 to 359.75 and latitudes from -89.75 to +89.75.

The file *MOLA_0.25deg.txt* counts 4 times as many lines, i.e. 1,036,800 points.

The file *MOLA_0.0625deg.txt* counts 16 times as many lines, i.e. 16,588,800 points.

We follow a similar approach as in STONE 97 and use the data to produce two vtu files, one on a 2D plane, one on a 3D sphere:



Based on the 0.0625degree resolution file.

Note that the files corresponding to 1/2, 1/4, and 1/16 of degree are not on github since they are quite large. Please contact me.

Using the topography to compute its gravity signal

Each cell is characterised by a min/max latitude, longitude. We can convert these to spherical coordinates with ϕ being the longitude and θ being the colatitude. The topography is given with respect to the Mars

radius R_{mars}. Since the minimum of the topography is about -8.177 km we define the inner radius of the hollow planet as $R_{min} = R_{mars} - t$ with $t = 8.2$ km. The topography associated to the cell is the one in its middle so we define $R_{max} = R_{mars} + h$ where h is the topography in the cell middle. Each cell then has a volume

$$\begin{aligned} V &= \int_{\phi_{min}}^{\phi_{max}} \int_{\theta_{min}}^{\theta_{max}} \int_{R_{min}}^{R_{max}} dV \\ &= \int_{\phi_{min}}^{\phi_{max}} \int_{\theta_{min}}^{\theta_{max}} \int_{R_{min}}^{R_{max}} r^2 \sin\theta dr d\theta d\phi \\ &= \int_{\phi_{min}}^{\phi_{max}} d\phi \cdot \int_{\theta_{min}}^{\theta_{max}} \sin\theta d\theta \cdot \int_{R_{min}}^{R_{max}} r^2 dr \\ &= (\phi_{max} - \phi_{min})(\cos\theta_{min} - \cos\theta_{max}) \frac{1}{3}(R_{max}^3 - R_{min}^3) \end{aligned} \tag{100.1}$$

We will assume for simplicity that the upper crust has a constant density ρ_c.

The last step is to define the center of the cell. The θ_c and ϕ_c coordinates of the cell middle point are trivial, they are those from the topography file. In the radial direction, we shall simply take

$$r_c(\theta_c, \phi_c) = \frac{1}{2}(R_{min} + R_{max}) = \frac{1}{2}(R_{mars} + h(\theta_c, \phi_c) + R_{mars} - t) = R_{mars} + \frac{1}{2}[h(\theta_c, \phi_c) - t]$$

On Mars, 1degree $= 2\pi R_{mars}/360 \simeq 59$ km, so that the highest resolution corresponds to a cell of side ~ 3.7 km!

idea/todo: -divide columns radially so as to obtain almost cubic cells? -use tesseroid formula ? -implement GLQ?

gravity data at `https://pgda.gsfc.nasa.gov/products/57` with format explaination at `https://pds-geosciences.wustl.edu/dataserv/gravity_model_desc.htm`

We can now proceed to compute the gravity field(s) resulting from this thin crustal layer at a point $\vec{r} = (x, y, z)$, as explained in Section 16.7.

Gravity calculations - 1degree resolution

Gravity fields g_x, g_y, g_z and U are computed at satellite height Rsat on a regular grid of latitudes and longitudes counting nlat2,nlon2 points. The Cartesian coordinates of these measurement points xM,yM,zM are computed as follows:

```
xM =np.zeros((nlat2,nlon2),dtype=np.float64)
yM =np.zeros((nlat2,nlon2),dtype=np.float64)
zM =np.zeros((nlat2,nlon2),dtype=np.float64)
for ilat2 in range(0,nlat2):
    for ilon2 in range(0,nlon2):
        phi2=360/nlon2*(ilon2+0.5)  * np.pi/180
        theta2=180/nlat2*(ilat2+0.5)* np.pi/180
        xM[ilat2,ilon2]=Rsat*np.sin(theta2)*np.cos(phi2)
        yM[ilat2,ilon2]=Rsat*np.sin(theta2)*np.sin(phi2)
        zM[ilat2,ilon2]=Rsat*np.cos(theta2)
```

We now have the coordinates of all measurement points and the coordinates of all the cells xC,yC,zC which have a volume cell_volume. As explained in Section 16.7, the gravity fields can be computed as follows:

$$g_x(x,y,z) = \mathcal{G} \sum_{e=1}^{Ncell} \rho_e V_e \frac{x - x_e}{|\vec{r} - \vec{r}_e|^3} \tag{100.2}$$

$$g_y(x,y,z) = \mathcal{G} \sum_{e=1}^{Ncell} \rho_e V_e \frac{y - y_e}{|\vec{r} - \vec{r}_e|^3} \tag{100.3}$$

$$g_z(x,y,z) = \mathcal{G} \sum_{e=1}^{Ncell} \rho_e V_e \frac{z - z_e}{|\vec{r} - \vec{r}_e|^3} \tag{100.4}$$

$$U(x,y,z) = -\mathcal{G} \sum_{e=1}^{Ncell} \rho_e V_e \frac{1}{|\vec{r} - \vec{r}_e|} \tag{100.5}$$

where

$$|\vec{r}-\vec{r}_e| = \sqrt{(x-x_e)^2+(y-y_e)^2+(z-z_e)^2}$$

Since the gravity is assumed constant and equal for all cells, ρ_e becomes ρ_0 and can be taken out of the summations. The corresponding code is then

```
gx =np.zeros((nlat2,nlon2),dtype=np.float64)
gy =np.zeros((nlat2,nlon2),dtype=np.float64)
gz =np.zeros((nlat2,nlon2),dtype=np.float64)
UU =np.zeros((nlat2,nlon2),dtype=np.float64)
for ilat2 in range(0,nlat2):
   for ilon2 in range(0,nlon2): # loop over measurement points
       for ilat in range(0,nlat):
           for ilon in range(0,nlon): # loop over cells
               distx=xM[ilat2,ilon2]-xC[ilat,ilon]
               disty=yM[ilat2,ilon2]-yC[ilat,ilon]
               distz=zM[ilat2,ilon2]-zC[ilat,ilon]
               dist=np.sqrt(distx**2+disty**2+distz**2)
               K=cell_volume[ilat,ilon]/dist**3
               gx[ilat2,ilon2]+=K*distx
               gy[ilat2,ilon2]+=K*disty
               gz[ilat2,ilon2]+=K*distz
               UU[ilat2,ilon2]-=cell_volume[ilat,ilon]/dist
x*=Ggrav*rho0
y*=Ggrav*rho0
z*=Ggrav*rho0
U*=Ggrav*rho0
```

Let us call the algorithm above Alg-1. From experience we know that these calculations often take a lot of time since Python is particularly not suited for imbricated for loops. In general, accessing and updating a cell of a two-dimensional array has a cost higher than updating a scalar. The last four lines inside the inner most for loop will be executed nlon*nlat times per measurement point so we can design a slightly modified algorithm coined Alg-2:

```
for ilat2 in range(0,nlat2):
   for ilon2 in range(0,nlon2): # loop over measurement points
       ggx=0
       ggy=0
       ggz=0
       UUU=0
       for ilat in range(0,nlat):
           for ilon in range(0,nlon): # loop over cells
               distx=xM[ilat2,ilon2]-xC[ilat,ilon]
               disty=yM[ilat2,ilon2]-yC[ilat,ilon]
               distz=zM[ilat2,ilon2]-zC[ilat,ilon]
               dist=np.sqrt(distx**2+disty**2+distz**2)
               K=cell_volume[ilat,ilon]/dist**3
               ggx+=K*distx
               ggy+=K*disty
               ggz+=K*distz
               UUU-=cell_volume[ilat,ilon]/dist
           #end for
       #end for
       gx[ilat2,ilon2]=ggx
       gy[ilat2,ilon2]=ggy
       gz[ilat2,ilon2]=ggz
       UU[ilat2,ilon2]=UUU
   #end for
end for
```

The algorithms above can be split into loops over measurement points and loops over the cells. With reusability in mind we could create a function which computes the gravity at a single point. This function needs as arguments the coordinates of the measurement point, and the coordinates of all cells and their volumes:

```
def compute_g_at_point(x,y,z,nlat,nlon,xC,yC,zC,volume):
    ggx=0.
    ggy=0.
    ggz=0.
    UUU=0.
    for ilat in range(0,nlat):
        for ilon in range(0,nlon):
```

```
                distx=x-xC[ilat,ilon]
                disty=y-yC[ilat,ilon]
                distz=z-zC[ilat,ilon]
                dist=np.sqrt(distx**2+disty**2+distz**2)
                K=volume[ilat,ilon]/dist**3
                ggx+=K*distx
                ggy+=K*disty
                ggz+=K*distz
                UUU-=volume[ilat,ilon]/dist
            #end for
        #end for
        return ggx,ggy,ggz,UUU
```

so that the gravity calculations now elegantly write:

```
for ilat2 in range(0,nlat2):
    for ilon2 in range(0,nlon2):
        gx[ilat2,ilon2],gy[ilat2,ilon2],gz[ilat2,ilon2],UU[ilat2,ilon2]=\
          compute_g_at_point(xM[ilat2,ilon2],yM[ilat2,ilon2],zM[ilat2,ilon2],nlat,nlon,xC,yC,zC,
    cell_volume)
```

This is Alg-3.

Finally, we could then load the Numba module[1] and add a sinple decorator before the function:

```
import numba
from numba import jit

@jit(nopython=True)
def compute_g_at_point(x,y,z,nlat,nlon,xC,yC,zC,volume):
    [...]
    return ggx,ggy,ggz,UUU
```

As explained on the website "Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN." This is Alg-4.

We load the 1degree-resolution file and set nlat2=16,nlon2=32. We time the gravity calculations and find:

Alg-1	183.005 s
Alg-2	165.774 s
Alg-3	123.866 s
Alg-4	0.442 s

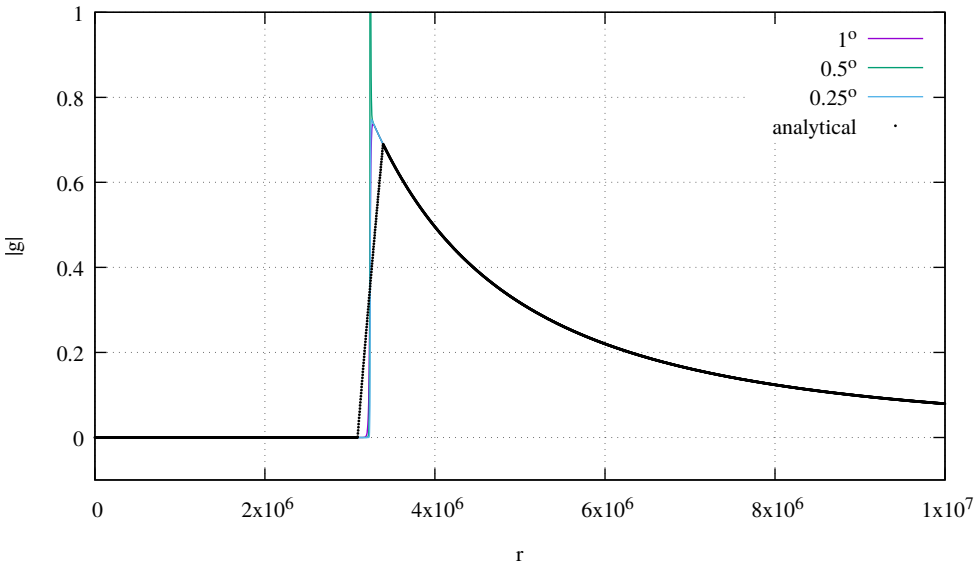
We find that Alg-2 is a little bit faster than Alg-1. Alg-3 is about 20% faster (why?!), but Alg-4 is more than 100 times faster!

All four algorithms obviously yield the same results so in what follows we only use Alg-4. Rsat is set to Rmars+100 km and nlat2=nlat and nlon2=nlon, i.e. the resolution of the measurements is as high as the resolution of the data.

[1] https://numba.pydata.org/

Analytical benchmark

In this case the topography is not used, only the coordinates of the cells. The shell is 300 km thick, i.e. $R_{mars} - 300 \leq r \leq R_{mars}$. Gravity is computed on 5000 points on a line $r \in [0, 4R_{mars}]$ parameterised by $\theta = \phi = 0.123456789$ rad (so as to avoid any cancelling due to symmetry).

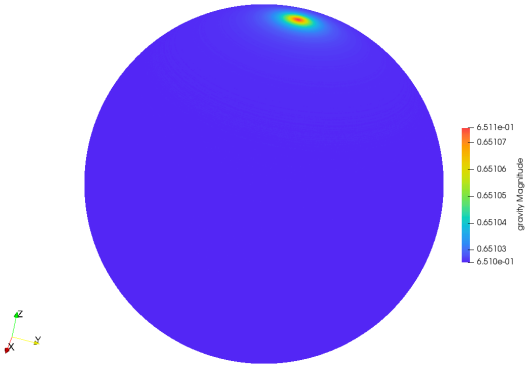


Results agree with the analytical solution (see appendix A of Thieulot [327] (2018)) except inside the shell since it is only composed of a single cell (represented by a point mass!) in the r direction.

The gravity is also measured at $R_{mars} + 100$ km:

resolution	min(g)	max(g)	min(U)	max(U)
1 °	0.651021	0.651818	-2271938.557588	-2271751.401358
1/2 °	0.651024	0.651225	-2271804.046129	-2271757.177753
1/4 °	0.651025	0.651075	-2271770.343922	-2271758.621926
analytical	0.65102561831		-2271759.10332	

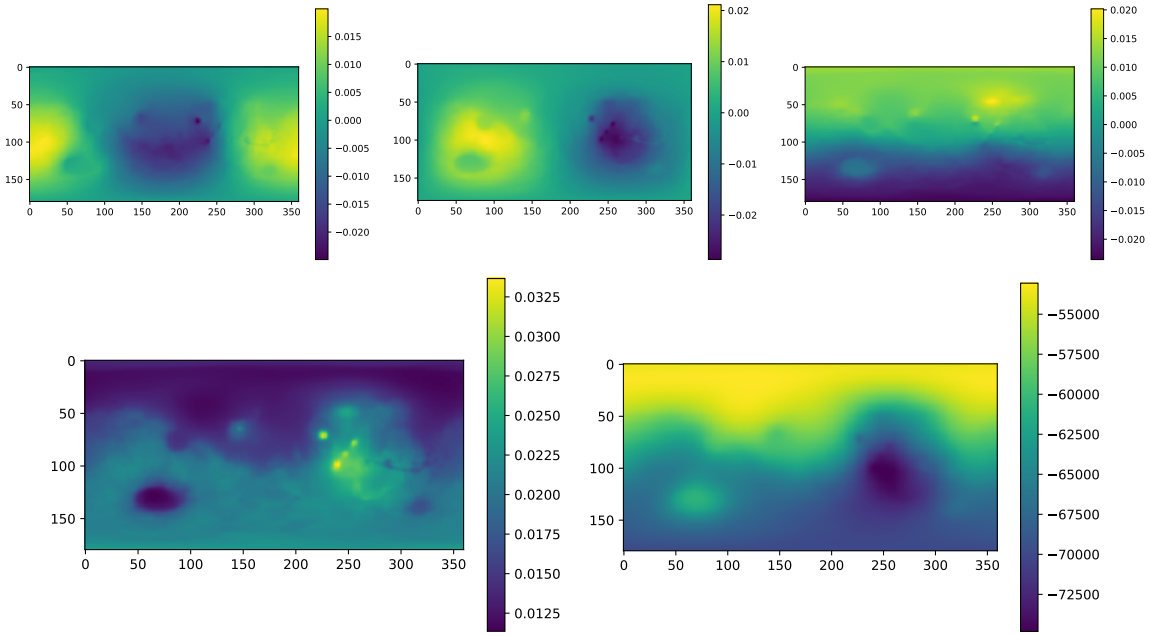
We see that the errors are concentrated above the poles:



Measured gravity $|\vec{g}|$ for 1/4-degree resolution dataset.

1 degree resolution data

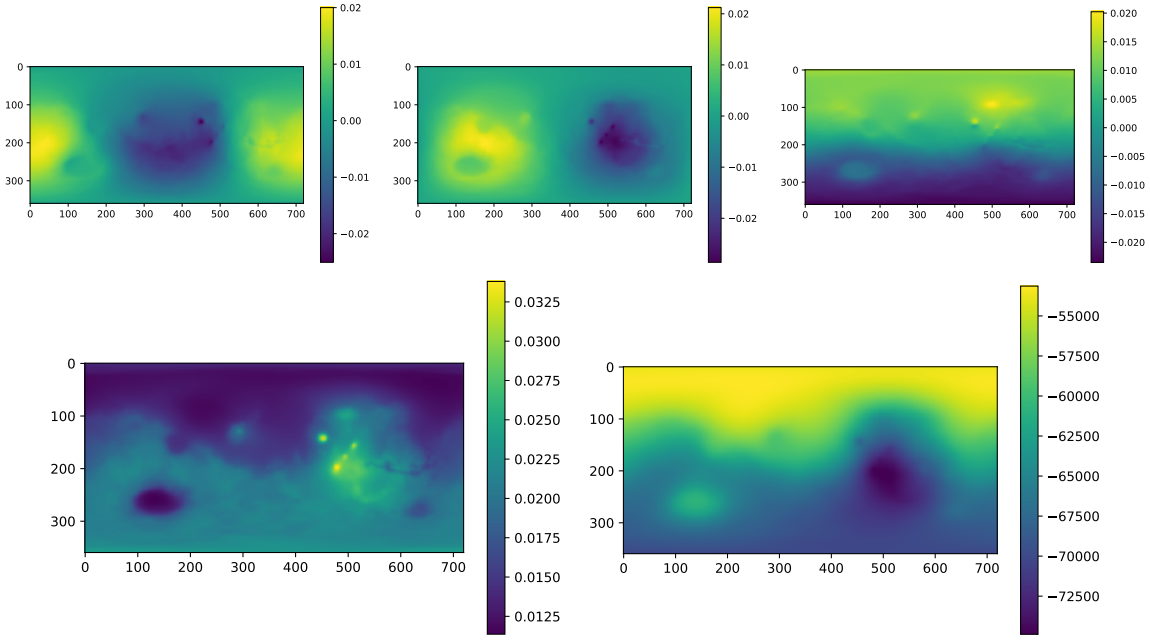
Compute time is about 21s.



Top row: g_x, g_y, g_z; bottom row: gravity vector norm and potential

1/2 degree resolution data

Compute time is about 344s.



Top row: g_x, g_y, g_z; bottom row: gravity vector norm and potential

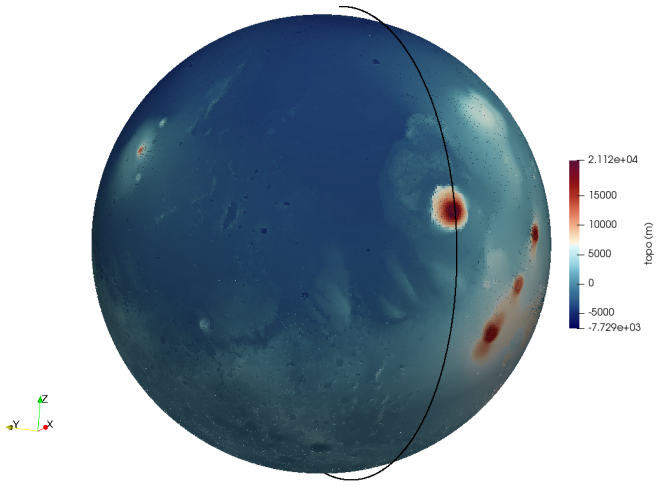
1/4 degree resolution data

about 5800s

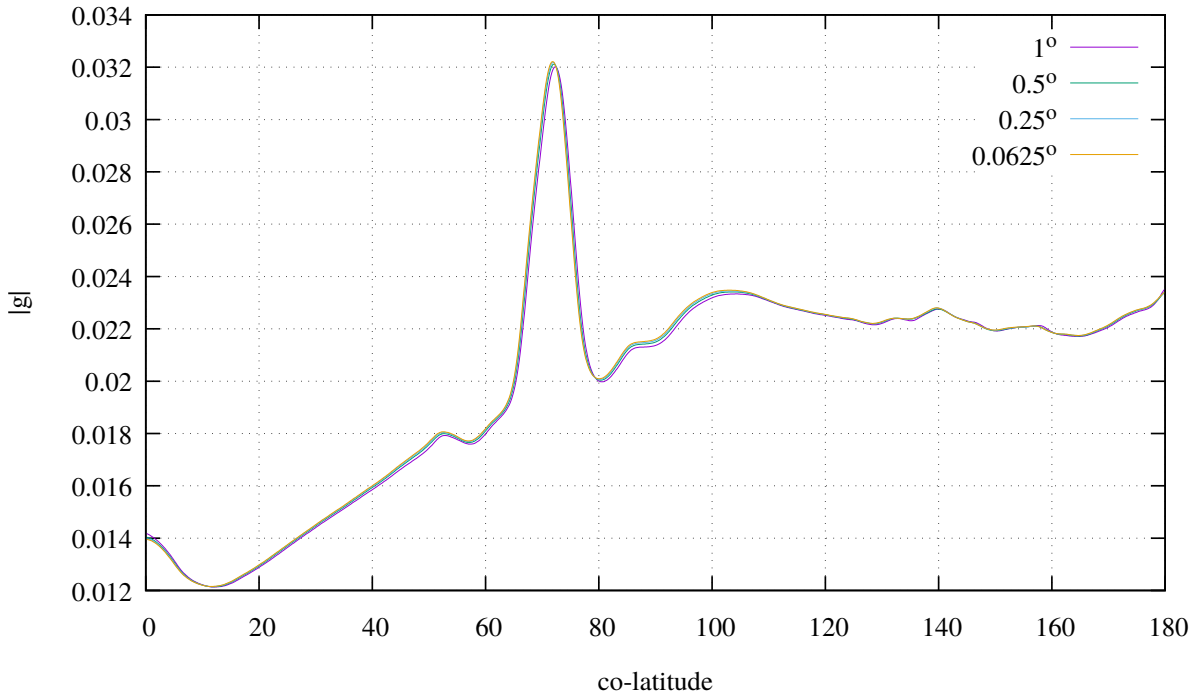
Flying above Mount Olympus

Olympus Mons[2] is an enormous shield volcano on Mars. The volcano has a height of over 21.9 km as measured by the Mars Orbiter Laser Altimeter (MOLA). Olympus Mons is about two and a half times Mount Everest's height above sea level. It is the largest and highest mountain and volcano of the Solar System, and is associated with the Tharsis Montes, a large volcanic region on Mars. Its coordinates are 18°39′N, 226°12′E.

We then design a satellite flight path that follows the circle passing by the poles and above the volcano, i.e. we fix $lon = 226\ °12'$, starting at the north pole and ending at the south pole, i.e. co-latitudes between 0° and 180°:



The measured gravity on this line for all 4 resolutions is as follows:

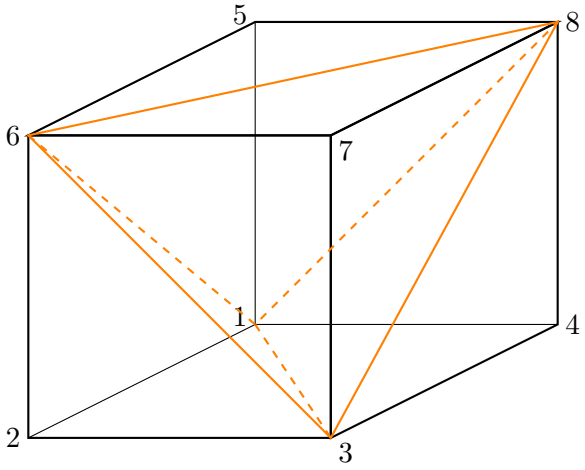


Measured gravity $|\vec{g}|$ as a function of the co-latitude.

TODO: plot topography underneath. Also plot g_r and not $|\vec{g}|$?

[2] `https://en.wikipedia.org/wiki/Olympus_Mons`

Using tetrahedra



tetrahedron	node 1	node 2	node 3	node 4
1	2	3	6	1
2	4	1	8	3
3	5	1	6	8
4	7	8	6	3
5	3	6	1	8

Stone 101

Y12M solver for $Q_1 \times P_0$+penalty Fortran codes



2D
box
Q1P0
penalty
sparse_matrix
analytical_benchmark
stokes_equation
Fortran

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_101`

This stone explores the possibilities offered by the Y12M direct solver[1]. It is identical to STONE 01. Despite its age (it was published in 1981), this solver remains interesting because it is rather compact, has a simple interface, fits in a single file and relies on a sparse storage for the FE matrix.

The documentation [374] states: "The Y12M is a package of Fortran subroutines for the solution of large and sparse systems of linear algebraic equations developed at the Regional Computing Centre at the University of Copenhagen (RECKU). Gaussian elimination and pivotal interchanges are used to factorize the matrix of the system into two triangular matrices L and U. An attempt to control the magnitude of the non-zero elements in order to avoid overflows or underflows and to detect singularities is carried out during the process of factorization, lterative refinement of the first solution may be performed. It is verified (by a large set of numerical examples) that iterative refinement combined with a large drop-tolerance and a large stability factor is often very successful when the matrix of the system is sparse, Not only is the accuracy improved but the factorization time is also considerably reduced so that the total computing time for the solution of the system with iterative refinement is less than that without iterative refinement (in some examples the total computing time was reduced by more than three times). The storage needed can often be reduced also."

The Y12M solver comes in two versions Y12MAE (single precision) and Y12MAF (double precision). We obviously use the latter in the code. This solver requires the matrix to be stored in COO coordinates, not CSR nor CSC. The non-zero entries are then stored in two integer arrays SNR and RNR of length Nfem:

- SNR - INTEGER array. On entry SNR(j) must contain the column number of the non-zero element stored in A(j). The content of array SNR is modified by subroutine Y12MA.

[1] `http://www.netlib.org/y12m/doc`

- RNR - INTEGER array. On entry RNR(i) must contain the row number of the non-zero element stored in A(i). The content of array RNR is modified by subroutine Y12MA.

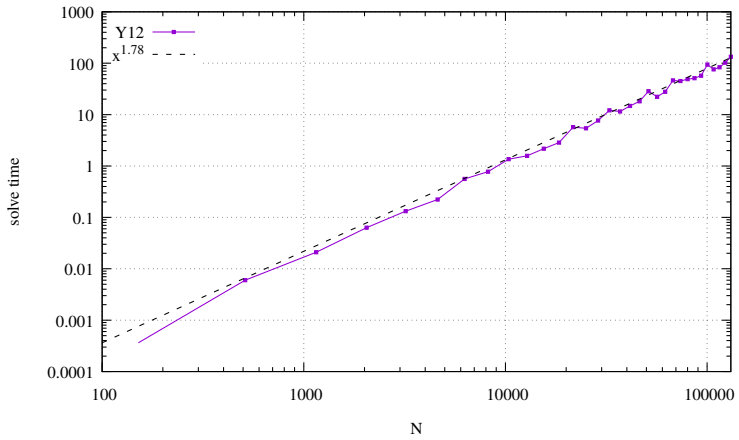
Note that since I had the code for building the CSR arrays I build these and use them to fill SNR and RNR:

```
nz=0
snr(1)=1
rnr(1)=1
ia(1)=1
do j1=1,nny
do i1=1,nnx
   ip=(j1-1)*nnx+i1 ! node number
   do k=1,ndof
      ii=2*(ip-1) + k ! address in the matrix
      nsees=0
      do j2=-1,1 ! exploring neighbouring nodes
      do i2=-1,1
         i=i1+i2
         j=j1+j2
         if (i>=1 .and. i<= nnx .and. j>=1 .and. j<=nny) then ! if node exists
            jp=(j-1)*nnx+i  ! node number of neighbour
            do l=1,ndof
               jj=2*(jp-1)+l  ! address in the matrix
               nz=nz+1
               snr(nz)=ii
               rnr(nz)=jj
               ja(nz)=jj
               nsees=nsees+1
            end do
         end if
      end do
      end do
      ia(ii+1)=ia(ii)+nsees
   end do ! loop over ndofs
end do
end do
```

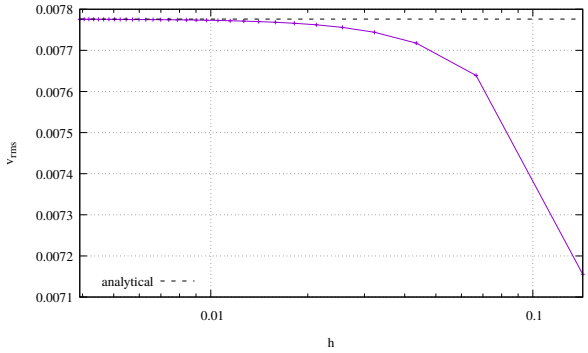
This algorithm only works when the mesh is structured and linear elements are used. Also, despite the matrix being symmetric it must be stored in its entirety here.

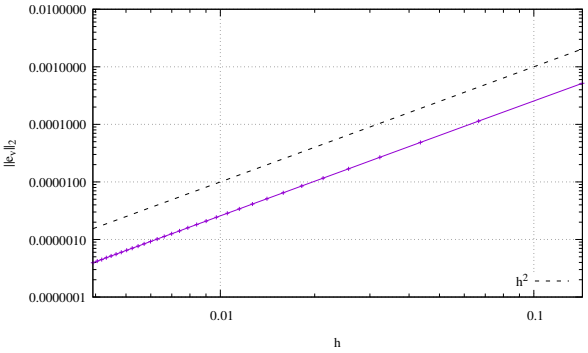
The code can be compiled by means of the *compile* script (you may need to modify it to make it compatible with your machine) and the code is executed by running the generated *simplefem* executable.

There is a built-in loop in the code which varies resolutions from 8×8 to 256×256. We can then look at the solve time as a function of velocity degrees of freedom:



It carries out a solve on a 256×256 mesh in about 130s. This is of course rather slow when compared to MUMPS performance, but it allows for plenty of testing. The code also computes the root mean square velocity and we can see that it becomes more and more accurate with an increase in resolution (i.e. a decrease in element size h), which is also confirmed by looking at the velocity error convergence, found to be quadratic as expected (see STONE 01):


0.0078
0.0077
0.0076
0.0075
0.0074
0.0073
0.0072
0.0071
v_{rms}
analytical
0.01
0.1
h


0.0100000
0.0010000
0.0001000
0.0000100
0.0000010
0.0000001
$\|\epsilon_v\|_2$
h^2
0.01
0.1
h

Stone 102

conformal mesh refinement in 2D



2D
box
Q1P0
penalty
analytical_benchmark
stokes_equation
vrms
paraview_output
Fortran

 Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_102`

The topic of Conformal Mesh Refinement is introduced in Section 15.7.6. This stone is a 'simple' approach to the implementation of Conformal Refinement. The code was written a few years prior to my working on fieldstone, so it is in Fortran. It is a simple Finite Element code which relies on $Q_1 \times P_0$ elements. It solves the manufactured solution problem of Section 18.2.

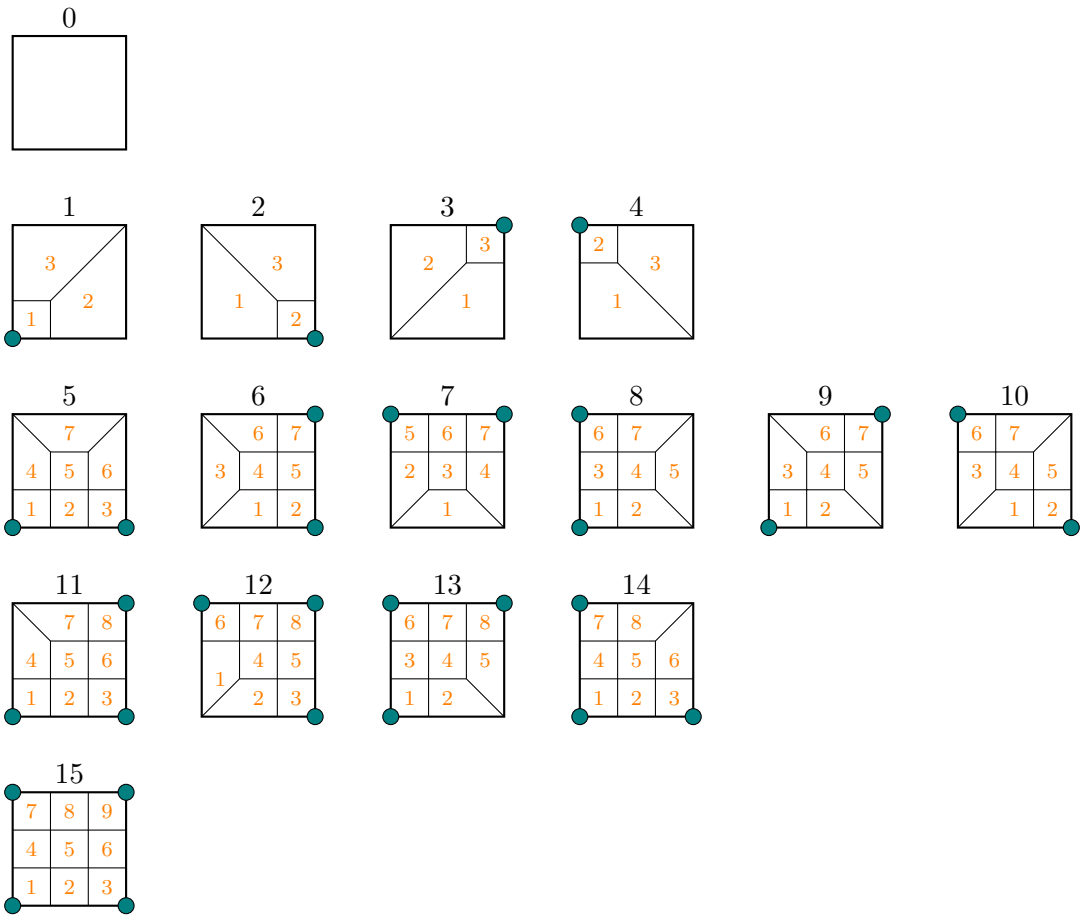
Let us start with a very simple test: the mesh counts $9 \times 7 = 63$ elements. The greyed elements are those to be refined. We then first flag the nodes which compose them and these are shown in teal color. Based on this information we can then consider each element in the mesh and assign it a type, based on whether or not one or more nodes are flagged:

(tikz_cr_1.tex)

0	0	0	0	0	0	0	0	0
0	0	2	5	5	1	0	0	0
0	0	6	15	15	14	1	0	0
0	2	11	15	15	15	8	0	0
0	6	15	15	15	13	10	5	5
0	3	7	7	7	4	6	15	15
0	0	0	0	0	0	6	15	15

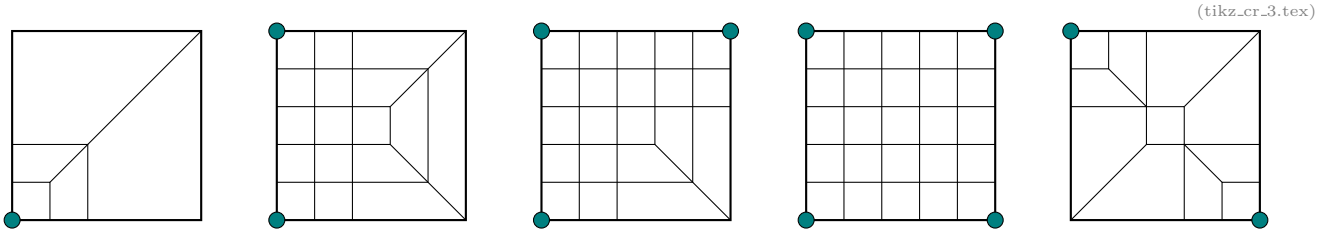
The type is stored in the elemental array `crtype` while the nodes belonging to elements to be refined are stored in `crnode`. There are 15 types of elements (the element type is tied to how many and which vertices are flagged nodes):

(tikz_cr_2.tex)



Note that the refinement above is based on a 3x3 refinement of elements. One could also carry out a 5x5-type refinement which would then rely on the following elements (and their rotated versions)[1]:

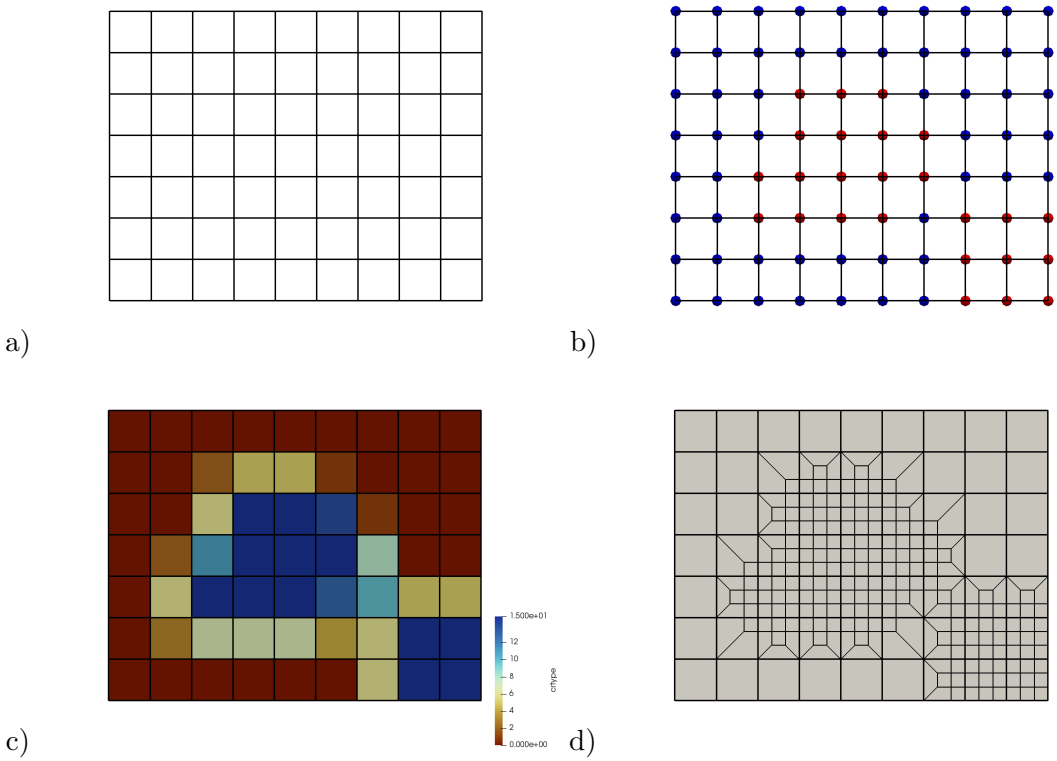
(tikz_cr_3.tex)



After running the code, various vtu files are to be found in the OUT folder. *solution.vtu* contains the refined mesh with the calculated solution.

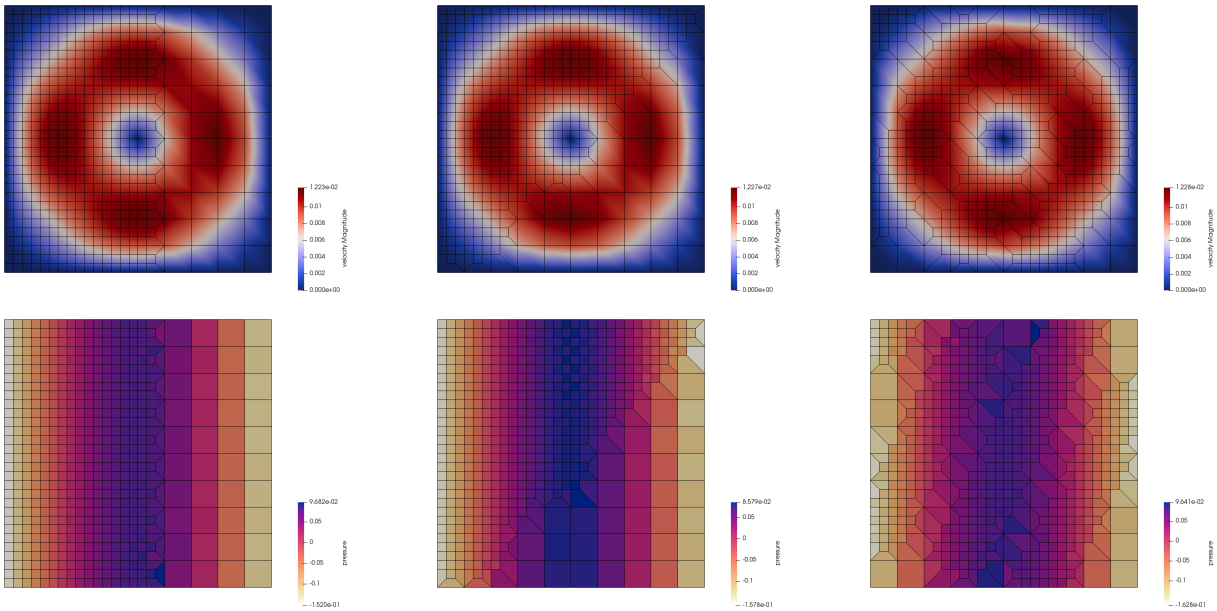
[1] I have never seen a concrete example of these...

Case test=0 The domain size is 9x7 and nelx=9, nely=7, so that all elements are square. The code solves the FE system but it is pointless as this test is about demonstrating the resulting mesh from the flagged elements at the beginning of this stone.



a) original mesh. b) flagged nodes. c) element types. d) refined mesh.

Case test=1,2,3



Initial resolution 10x10. Left: The left half of elements are flagged for refinement. Middle: The elements above the diagonal are flagged for refinement. Right: Every 7 elements is flagged for refinement.

Stone 103

conformal mesh refinement in 3D

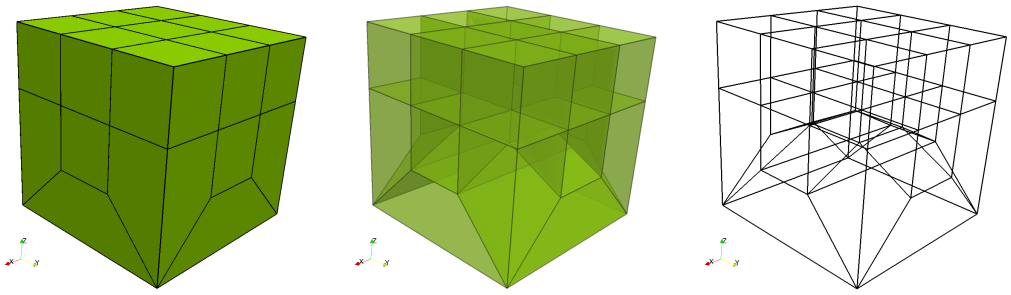


3D
box
Q1P0
paraview_output
Fortran

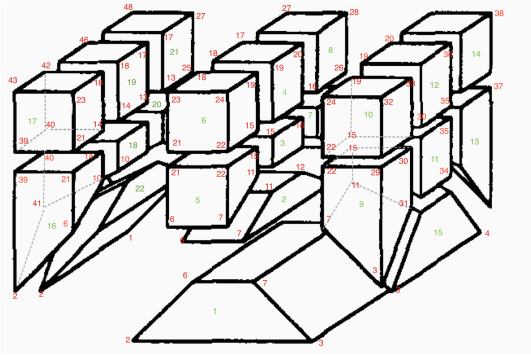
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_103`

The topic of Conformal Mesh Refinement is introduced in Section 15.7.6. This code is in Fortran for the same reason STONE 102 is. However, it is not nearly as complete as STONE 102: It can only refine above a certain height, as the different types of subdivisions are not implemented. Furthermore, there is no FE solver in it either. It is just an example of simple 3D conformal refinement meshing.

The *prog.f90* code showcases the decomposition of one element and generates the *refined.vtu* file:

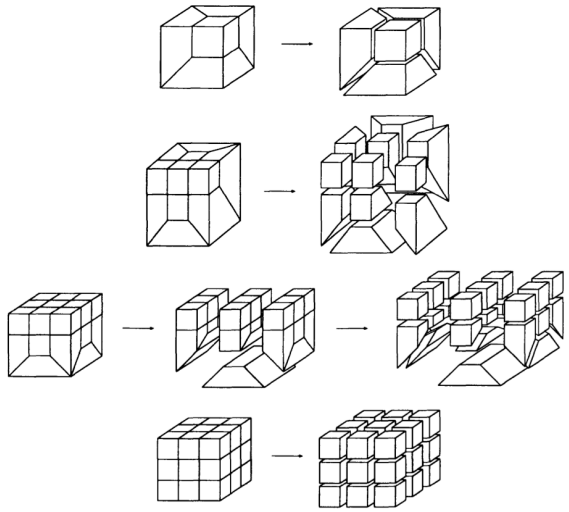


The internal numbering goes as follows:



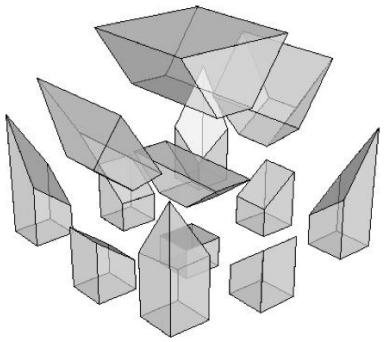
Taken from ? Numbers added by me. There are additional handwritten notes in the images folder.

In order to finish the *stone.f90* code, one would need to implement all four (+rotations) of these subdivisions:



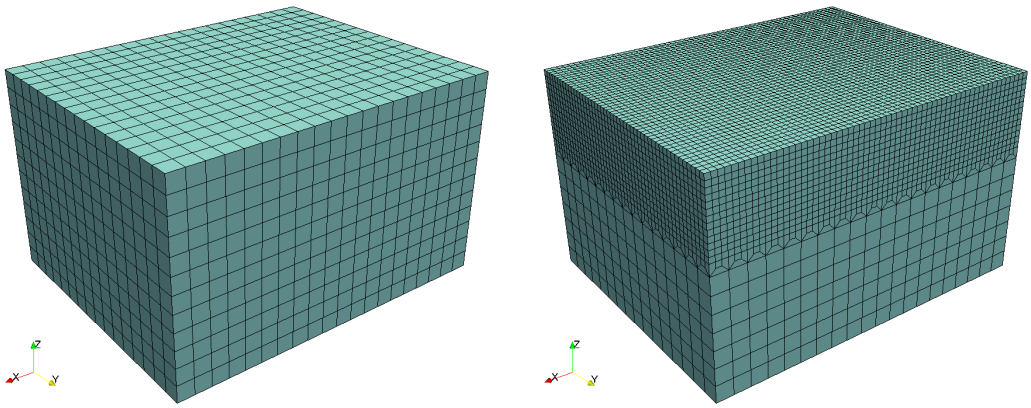
Taken from ?

Note that there is another way to subdivide an element:



Taken from [150].

Finally, we can run the code on a $20 \times 16 \times 12$ original mesh:



Stone 104

grad-div stabilisation $(Q_2 \times Q_1,\ Q_2 \times P_{-1})$



Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_104`

Note to self: normalisation does not work for Q2Q1.

All the following benchmarks are isoviscous with $\eta = 1$ and are valid in the unit square domain. As such the scaling coefficient between blocks $\mathbb{K}$ and $\mathbb{G}$ of the Stokes matrix (see Section **??**) is 1 and won't play any role in the results.

REDO all with aspect

According to Jenkins *et al.* (2014) [170], Grad-div stabilization results from adding $\vec{0} = -\gamma\vec{\nabla}(\vec{\nabla}\cdot\vec{\mathbf{v}})$ to the continuous Stokes equations, i.e.:

$$-\nabla p + \nabla\cdot(2\eta\dot{\boldsymbol{\varepsilon}}) + \rho\vec{g} = -\gamma\vec{\nabla}(\vec{\nabla}\cdot\vec{\mathbf{v}})$$

We immediately notice that the aditional term is in fact identical to the penalty term and its weak form and discretisation is worked out in Section **??**, so that the matrix $\mathbb{K}$ then becomes:

$$\mathbb{K} = \int \boldsymbol{B}^T\cdot[\gamma\boldsymbol{K} + \eta\boldsymbol{C}]\cdot\boldsymbol{B}\ dV$$

with

$$\boldsymbol{C} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \boldsymbol{K} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The implementation is then trivial. Surprisingly in the papers on the grad-div stabilisation the penalty method is never mentioned (?) although John *et al.* state that "[...] it shall be emphasized that large contributions of the grad-div stabilization result in linear systems of equations with large condition numbers", which is a well know problem of the penalty method.

In this case $\gamma > 0$ but another parallel can be drawn when $\gamma = -\frac{2}{3}\eta$. In that case

$$\nabla\cdot(2\eta\dot{\boldsymbol{\varepsilon}}) + \gamma\vec{\nabla}(\vec{\nabla}\cdot\vec{\mathbf{v}}) = \nabla\cdot(2\eta\dot{\boldsymbol{\varepsilon}}) - \frac{2\eta}{3}\vec{\nabla}(\vec{\nabla}\cdot\vec{\mathbf{v}}) = \nabla\cdot 2\eta(\dot{\boldsymbol{\varepsilon}} - \frac{1}{3}(\vec{\nabla}\cdot\vec{\mathbf{v}})\mathbf{1}) = \nabla\cdot 2\eta\dot{\boldsymbol{\varepsilon}}^d$$

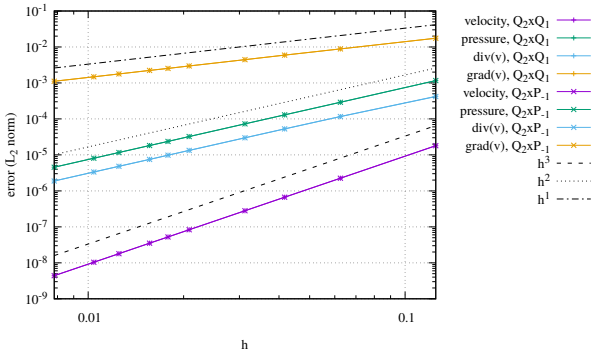
Obviously γ should not be 'too negative', otherwise terms on the diagonal of $\gamma\boldsymbol{K} + \eta\boldsymbol{C}$ will be negative and this will cause problems.

In what follows I have implemented what I think the grad-div method is, with rather limited/puzzling (?) results. Rather importantly John *et al.* conclude that "the grad-div stabilization might improve the pressure-robustness in certain situations but it is not a remedy".

Relevant Literature: [246]

Experiment=1: Donea & Huerta manufactured solution

I carry out this benchmark to make sure that the code behaves as expected and that the error rates are what we expect: 3rd order for velocity, second order for pressure for both $Q_2 \times Q_1$ and $Q_2 \times P_{-1}$ elements (see for instance Thieulot & Bangerth [328]).



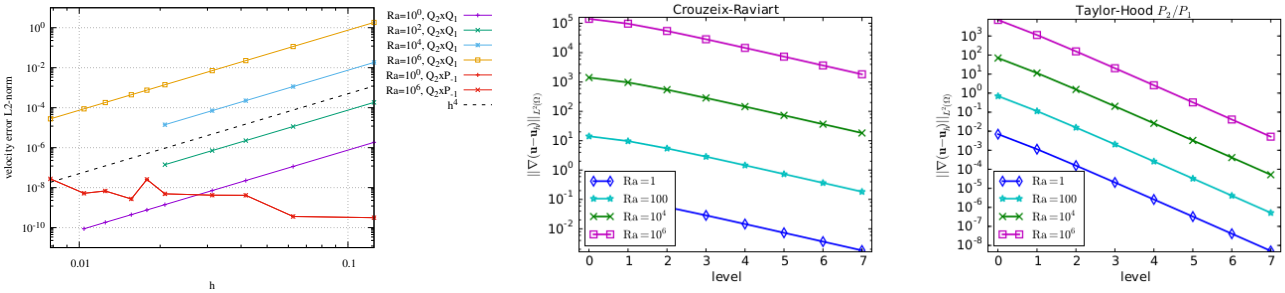
Experiment=2: example 1.1 in John *et al.* (2017)

Following John *et al.* (2017) [172] we postulate the following velocity and pressure fields:

$$\begin{aligned} \vec{\mathbf{v}}(x,y) &= \vec{0} \\ p(x,y) &= \mathsf{Ra}(y^3 - y^2/2 + y - 7/12) \end{aligned}$$

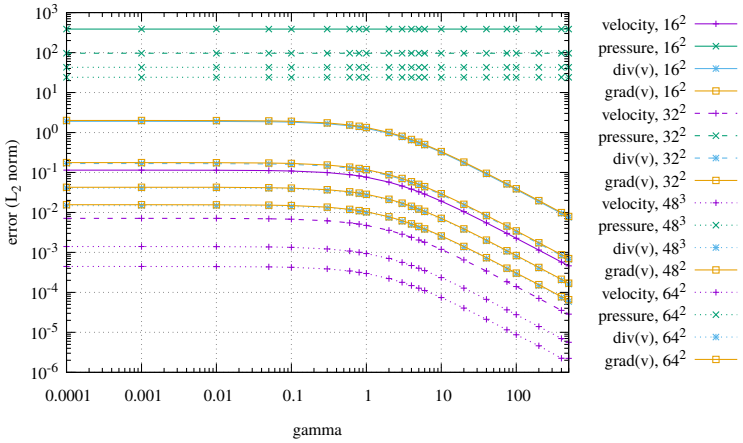
with $\eta = 1$ and the buoyancy force vector is

$$\vec{b} = \begin{pmatrix} 0 \\ \mathsf{Ra}(1 - y + 3y^2) \end{pmatrix}$$



Left: this stone; Right: taken from [172]

We find that the velocity error magnitude for the $Q_2 \times Q_1$ element scales with the Rayleigh number, and more surprisingly (?) that its convergence is quartic. However, the velocity error magnitude for the $Q_2 \times P_{-1}$ remains very low independently of the Rayleigh number



Experiment=3,4,5:

These three manufactured solutions come from Jenkins *et al.* (2014) [170]. The velocity is the same in all cases, $\vec{\mathrm{v}} = (\cos 2\pi y, \sin 2\pi x)$ while the three pressure fields are

$$\begin{aligned} p_1(x,y) &= \sin(2\pi y) \\ p_2(x,y) &= \sin(8\pi y) \\ p_3(x,y) &= 10^4 \sin(2\pi y) \end{aligned}$$

We need to compute the vector $\vec{b}$ from these. The strain rate tensor is then

$$\dot{\boldsymbol{\varepsilon}}(\vec{\mathrm{v}}) = \begin{pmatrix} 0 & -\pi \sin 2\pi y + \pi \cos 2\pi x \\ -\pi \sin 2\pi y + \pi \cos 2\pi x & 0 \end{pmatrix}$$

and then the full stress tensor:

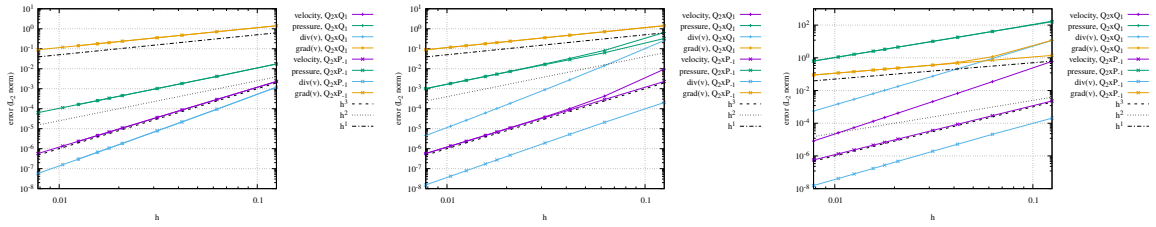
$$\boldsymbol{\sigma} = -p\mathbf{1} + 2\eta\dot{\boldsymbol{\varepsilon}} = \begin{pmatrix} -p & -2\pi\eta(\sin 2\pi y - \cos 2\pi x) \\ -2\pi\eta(\sin 2\pi y - \cos 2\pi x) & -p \end{pmatrix}$$

Finally

$$-\vec{b} = \vec{\nabla} \cdot \boldsymbol{\sigma} = \begin{pmatrix} -\partial_x p - 4\pi^2 \eta \cos 2\pi y \\ -\partial_y p - 4\pi^2 \eta \sin 2\pi x \end{pmatrix}$$

We see that we have $\partial_x p_{1,2,3} = 0$ so

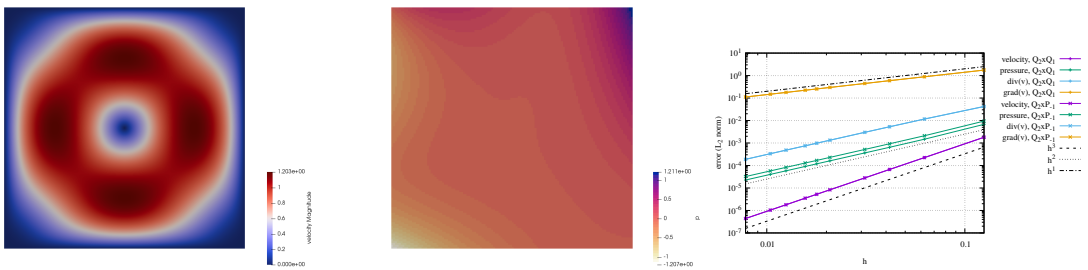
$$\vec{b} = \begin{pmatrix} 4\pi^2 \eta \cos 2\pi y \\ \partial_y p + 4\pi^2 \eta \sin 2\pi x \end{pmatrix}$$

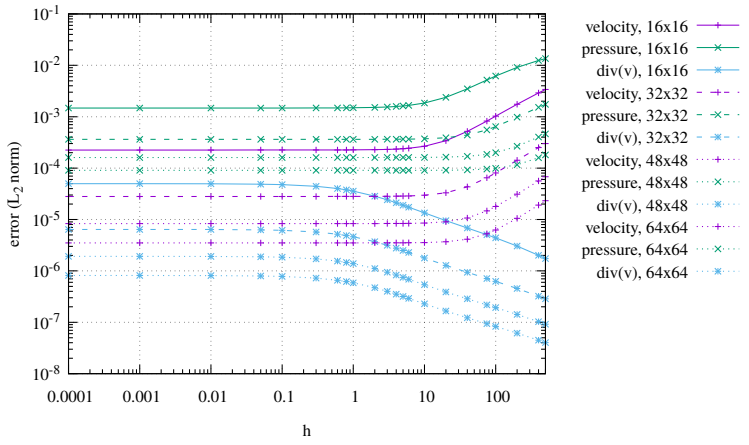


We see that the Q2Q1 velocity error is much larger than the Q2P1 error in benchmark 3 but that it converges again in a quartic manner. Pressure errors are however identical.

Experiment=6: manufactured solution in John *et al.* (2017) [172]

See Section **??** for details.





errors as a function of γ.

Relevant Literature: Heister and Rapin [152]

Stone 105

solving the flexure equation with FDM



1D
elastic_rheology
finite differences

Code at https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_105

This stone was developed in collaboration with Zoltán Erdős.

Under many simplifying approximations (See Section 3.9 of Turcotte & Schubert book [335]), the simplest time-independent flexure equation is

$$D\frac{d^4w(x)}{dx^4} + P\frac{d^2w(x)}{dx^2} + (\rho_m - \rho_c)gw(x) = q(x)$$

where $w(x)$ is the plate deflection (m),

$$D = \frac{Et^3}{12(1-\nu^2)}$$

is the flexural rigidity, with t the thickness of the plate (m), E is Young's modulus, ν is the Poisson ratio, P is a horizontal force per unit length (kg/m^{2}sec^2), q is the load (Pa), ρ_m is the mantle density and ρ_c is the density of the crust (kg m^{-3}). This is a 4th-order ODE. In order to solve it, we will need 4 boundary conditions.

Redo Fig. 3.24 of T&S.

We wish to solve this equation on the domain $[0, L_x]$ with the Finite Difference method. Indeed the presence of a 4th-order derivative poses quite a few problems for the Finite Element method and in this case the FDM is more straighforward to implement (see also discussion in Section 2.3 of Buiter [48]). The domain is discretised with n_p nodes equally spaced by $h = L_x/(n_p - 1)$.

We will need the 2nd- and 4th-order derivatives[1]:

$$\frac{d^2w}{dx^2} = \frac{w_{n-1} - 2w_n + w_{n+1}}{h^2} + \mathcal{O}(h^3) \tag{105.1}$$

$$\frac{d^4w}{dx^4} = \frac{w_{n-2} - 4w_{n-1} + 6w_n - 4w_{n+1} + w_{n+2}}{h^4} + \mathcal{O}(h^5) \tag{105.2}$$

Both are second-order accurate. We then obtain

$$D\frac{w_{n-2} - 4w_{n-1} + 6w_n - 4w_{n+1} + w_{n+2}}{h^4} + P\frac{w_{n-1} - 2w_n + w_{n+1}}{h^2} + (\rho_m - \rho_c)gw_n = q_n$$

[1] https://en.wikipedia.org/wiki/Finite_difference_coefficient

or,

$$\boxed{\frac{D}{h^4}w_{n-2} + \left(-\frac{4D}{h^4} + \frac{P}{h^2}\right) w_{n-1} + \left(\frac{6D}{h^4} - \frac{2P}{h^2} + (\rho_m - \rho_c)g\right) w_n + \left(-\frac{4D}{h^4} + \frac{P}{h^2}\right) w_{n+1} + \frac{D}{h^4}w_{n+2} = q_n}$$

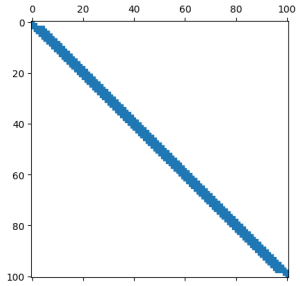
One could also multiply this equation by h^4 and obtain:

$$Dw_{n-2} + \left(-4D + Ph^2\right) w_{n-1} + \left(6D - 2Ph^2 + (\rho_m - \rho_c)gh^4\right) w_n + \left(-4D + Ph^2\right) w_{n+1} + Dw_{n+2} = q_n h^4$$

We will formulate this equation for every node $i \in [2, n_p - 3]$ in a systematic manner and we will therefore obtain a linear system of equations of the form

$$\boldsymbol{A} \cdot \vec{w} = \vec{b}$$

where $\boldsymbol{A}$ is an $n_p \times n_p$ matrix (sparse since pentadiagonal) and $\vec{b}$ the right-hand side vector. The vector $\vec{w}$ contains the n_p deflection unknowns at the nodes of the mesh.



The four missing lines are addressed when we consider the boundary conditions: We assume the plate to be very long, and that the load(s) will be mostly concentrated around $L_x/2$. We therefore prescribe $w = 0$ and $dw/dx = 0$ at $x = 0$ and $x = L_x$ (i.e. the deflection and the slope at the edges of the beam both tend to zero).

The first condition is trivial to impose: set 1 on the diagonal of the matrix for $i = 0$ and $i = n_p - 1$ and the corresponding entries of the rhs to zero. The second will have us use the backward derivative on the left for node $i = 1$ and the forward derivative on the right for node $n_p - 2$:

$$\begin{aligned} \frac{dw}{dx}(x = 0) &\simeq \frac{-w_0 + w_1}{h} \\ \frac{dw}{dx}(x = L_x) &\simeq \frac{-w_{n_p-2} + w_{n_p-1}}{h} \end{aligned} \tag{105.3}$$

The two equations will replace the second and before last lines in the matrix.

The flexure equation which allows for D to be a function of space is

$$\frac{d^2}{dx^2}\left(D(x)\frac{d^2w(x)}{dx^2}\right) + P\frac{d^2w(x)}{dx^2} + (\rho_m - \rho_c)gw(x) = q(x)$$

Instead of using the expression for a 4th-order derivative we then use the expression for a 2nd-order derivative twice. Let us temporarily coin $\phi = D(x)\frac{d^2w}{dx^2}$. Then

$$\frac{d^2}{dx^2}\left(D(x)\frac{d^2w(x)}{dx^2}\right) \tag{105.4}$$

$$= \frac{d^2\phi}{dx^2} \tag{105.5}$$

$$\simeq \frac{\phi_{n-1} - 2\phi_n + \phi_{n+1}}{h^2}$$

$$\simeq \frac{1}{h^2}\left(D_{n-1}\frac{w_{n-2} - 2w_{n-1} + w_n}{h^2} - 2D_n\frac{w_{n-1} - 2w_n + w_{n+1}}{h^2} + D_{n+1}\frac{w_n - 2w_{n+1} + w_{n+2}}{h^2}\right)$$

$$= \frac{D_{n-1}}{h^4}w_{n-2} - 2\frac{D_{n-1} + D_n}{h^4}w_{n-1} + \frac{D_{n-1} + 4D_n + D_{n+1}}{h^4}w_n - 2\frac{D_n + D_{n+1}}{h^4}w_{n+1} + \frac{D_{n+1}}{h^4}w_{n+2}$$

Obviously if $D_{n-1} = D_n = D_{n+1}$ we recover the expression above derived with a constant flexural rigidity D. The full flexural equation with the variable $D(x)$ then reads as

$$\frac{D_{n-1}}{h^4}w_{n-2} + \left(-2\frac{D_{n-1}+D_n}{h^4} + \frac{P}{h^2}\right)w_{n-1} + \left(\frac{D_{n-1}+4D_n+D_{n+1}}{h^4} - \frac{2P}{h^2} + (\rho_m - \rho_c)g\right)w_n + \left(-2\frac{D_n+D_{n+1}}{h^4} + \frac{P}{h^2}\right)w_{n+1} + \frac{D_{n+1}}{h^4}w_{n+2} = q_n \quad (105.6)$$

In what follows the material parameters are those of Fig. 2.2 in Buiter [48]: $t = 10\text{km}$, $\nu = 0.25$, $W = 10^{11}\text{Pa}$, $\rho_m = 3250\text{kg m}^{-3}$, $\rho_c = 2800\text{kg m}^{-3}$.

Relevant Literature: Section 11 in Simpson's book [298]

Benchmark 1: line load

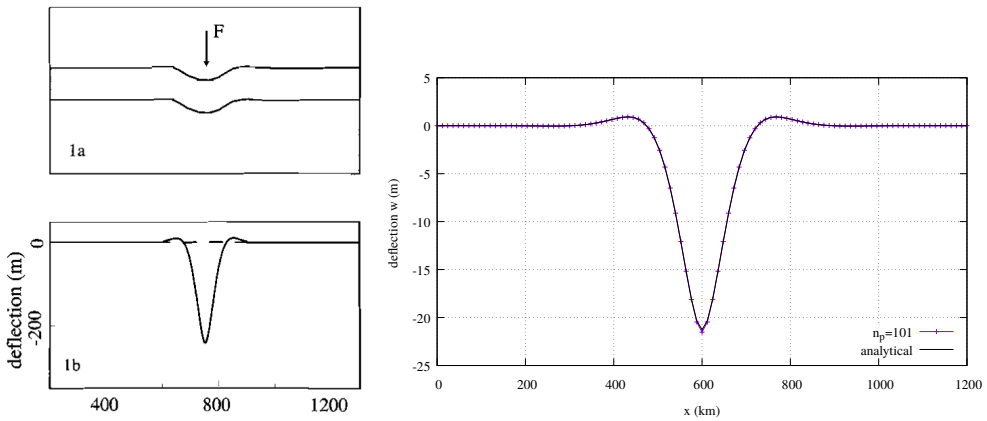
Following Buiter (2000) [48] we consider the case of a lineload. The lineload F is applied in the middle of the domain and the solution is therefore symmetric around $x = L_x/2$. The analytical solution is given by[2]

$$w_{analytical} = \frac{F\alpha^3}{8D}\exp -\frac{|x|}{\alpha}\left(\cos\frac{|x|}{\alpha} + \sin\frac{|x|}{\alpha}\right)$$

with the so-called flexural parameter

$$\alpha = \left(\frac{4D}{(\rho_m - \rho_c)g}\right)^{1/4}$$

which measures the wavelength of the flexure.



Left: Taken from [48]

Benchmark 2: periodic loading

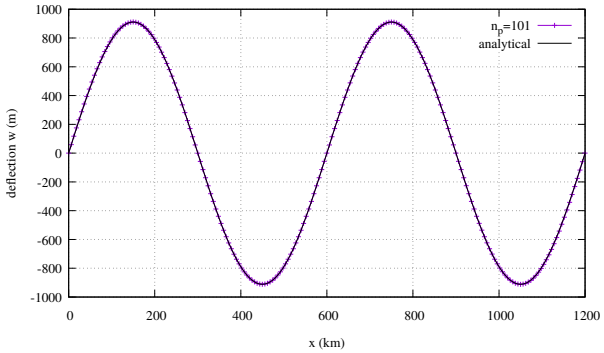
For this benchmark we only prescribe $w = 0$ at both extremities. For the second and before last lines of the matrix we resort to using forward and backward expressions for the 4th-order derivative for simplicity. The load is given by:

$$q(x) = \rho_c g h_0 \sin 2\pi\frac{x}{\lambda}$$

The solution is $w(x) = w_0 \sin 2\pi x/\lambda$, with

$$w_0 = \frac{h_0}{\frac{\rho_m}{\rho_c} - 1 + \frac{D}{\rho_c g}(2\pi/\lambda)^4}$$

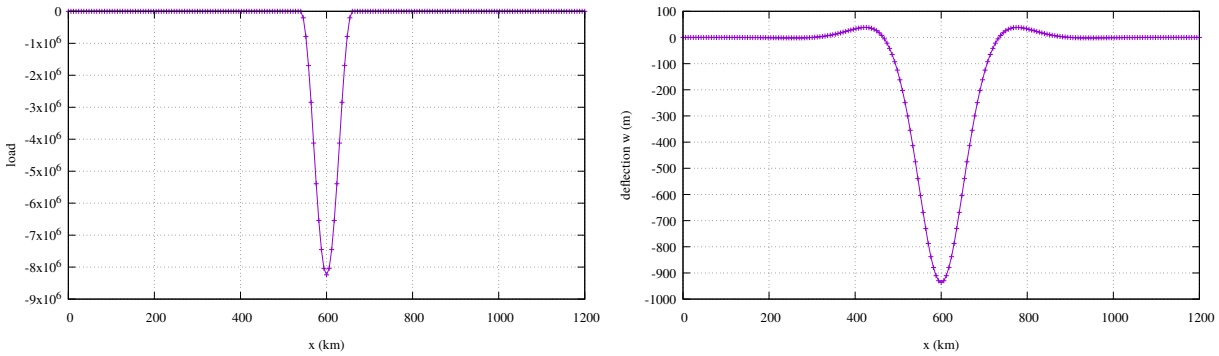
[2] I believe the absolute values were missing in the thesis.



Obtained with $\lambda = 0.5Lx$

Experiment 3: island

The load is given by $q(x) = -\rho_c g h_0 \left(1 + \cos 2\pi \frac{|x - L_x/2|}{\lambda}\right)$ for $|x - L_x/2| < \lambda/2$



Obtained with $\lambda = 0.1$. Left: load; Right: deflection.

Relevant Literature: J.-C. de Bremaecker. "Is the oceanic lithosphere elastic or viscous?" In: *J. Geophys. Res.: Solid Earth* 82.14 (1977), pp. 2001–2004

Christopher Beaumont. "The evolution of sedimentary basins on a viscoelastic lithosphere: theory and examples". In: *Geophy. J. Int.* 55.2 (1978), pp. 471–497. DOI: 10.1111/j.1365-246X.1978.tb04283.x

Stone 106

plumes in axisymmetric chunk ($Q_2 \times Q_1$)

2D
axisymmetric
annulus
chunk
Q2Q1
sparse_matrix
numerical_benchmark
stokes_equation
energy_equation
thermo_mechanical
time_stepping
viscous_rheology
buoyancy_driven
vrms
rayleigh_number
paraview_output
pressure_scaling
pressure_normalisation
steady_state

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_106`

Relevant Literature:

W.S. Kiefer and B. Hager. "Geoid anomalies and dynamic topography from convection in cylindrical geometry: applications to mantle plumes on Earth and Venus". In: *Geophy. J. Int.* 108 (1992), pp. 198–214. DOI: 10.1111/j.1365-246X.1992.tb00850.x;

Scott D King. "Geoid and topographic swells over temperature-dependent thermal plumes in spherical-axisymmetric geometry". In: *Geophys. Res. Lett.* 24.23 (1997), pp. 3093–3096. DOI: 10.1029/97GL53154;

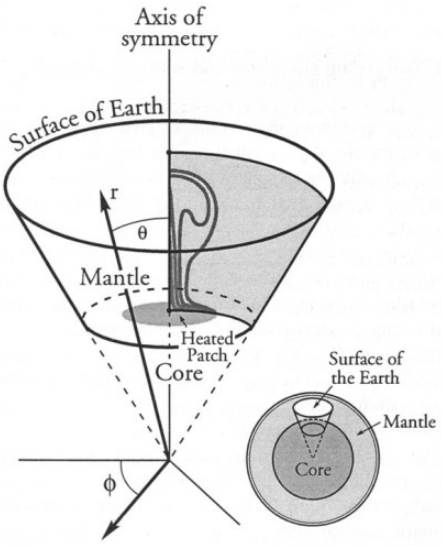
Hannah L Redmond and Scott D King. "A numerical study of a mantle plume beneath the Tharsis Rise: Reconciling dynamic uplift and lithospheric support models". In: *J. Geophys. Res.: Planets* 109.E9 (2004). DOI: 10.1029/2003JE002228;

Walter S Kiefer. "Melting in the Martian mantle: Shergottite formation and implications for present-day mantle convection on Mars". In: *Meteoritics & Planetary Science* 38.12 (2003), pp. 1815–1832. DOI: 10.1111/j.1945-5100.2003.tb00017.x;

Qingsong Li and Walter S Kiefer. "Mantle convection and magma production on present-day Mars: Effects of temperature-dependent rheology". In: *Geophys. Res. Lett.* 34.16 (2007). DOI: 10.1029/2007GL030544;

Wei Leng and Michael Gurnis. "Shape of thermal plumes in a compressible mantle with depth-dependent viscosity". In: *Geophys. Res. Lett.* 39.5 (2012). DOI: 10.1029/2012GL050959.

This stone is inspired by Kellogg and King [184] (1997).



Taken from [184].

I set out to reproduce the results of the original publication. The necessary information is spread out throughout the first pages. In Section 3 it is stated that the angular opening is $\pi/4$ and $r_i/r_o = 0.55$. However, it is not clear what r_i or r_o are, although Section 3 mentions r values of 1.7 which is somewhat in the 'middle' of the cone. Because the equations in the paper are in dimensionless form, and because it is common to choose the depth of the mantle as reference length, then we probably have $r_o - r_i = 1$. In this case we find $r_i = 1.22\ldots$ and $r_o = 2.22\ldots$ (1st guess: confirmed by author S.K., June 2021. Also consistent with King [193]).

Free-slip boundary conditions are imposed on all boundaries. A patch on the inner boundary of the shell is heated by maintaining it at a constant dimensionless temperature of $T_{patch} = 1$; elsewhere the base of the shell is insulating ($\partial T/\partial r = 0$). At the top of the shell, a constant dimensionless temperature of $T_{top} = 0$ was maintained. Unfortunately it is not specified how wide the patch is, so I choose $\pi/8$ which seems probable when looking at Figure 3 of the paper (2nd guess: confirmed by S.K., June 2021). The sides of the cone are insulating ($\partial T/\partial\theta = 0$). Because of how FE works the insulating boundary conditions are automatically enforced when Dirichlet boundary conditions are absent.

The viscous rheology is temperature-dependent:

$$\eta(T) = \eta_0 \exp\left[E\left(\frac{1}{T+T_0} - \frac{1}{1+T_0}\right)\right]$$

with $E = \{0, 0.25328, 3\}$. T_0 is defined as "the surface temperature divided by the temperature drop across the shell" but its value is not specified. On Earth the surface temperature is ~ 300K while the temperature drop across the mantle is ~ 3000K so I take $T_0 = 0.1$ (3rd guess: S.K. says that ΔT can be recovered from the Rayleigh number and using standard Earth values for all other parameters, but given the uncertainty about the Rayleigh number itself, this parameter is lost forever). The value for η_0 is also not specified but if this value is used as reference viscosity value during the adimensionalisation process then $\eta_0 = 1$ (4th guess).i As indicated in the paper, the viscosity is "clipped" to $1000\eta_0$. The initial temperature in the domain is set to $T = 0.25$.

As seen in Section 3.11.2, the dimensionless heat transport equation is void of any coefficient if κ/h is used as a reference velocity (where h is a reference length) so we take the dimensionless value of κ to be 1 in Eq. 5 of the paper.

Looking at Eq. 1 we find that there probably is a factor 2 missing before the viscosity μ (5th guess: confirmed by S.K., June 2021). In Section 2.2 it is stated that the SUPG method is used (see Section 9.6) but how the stabilisation parameter is chosen is not specified (6th guess).

In the end the set of equations solved by the code is as follows:

$$-\vec{\nabla}p + \vec{\nabla}\cdot(2\eta\dot{\vec{\varepsilon}}) = \mathsf{Ra}\, T\vec{e}_r \tag{106.1}$$

$$\vec{\nabla}\cdot\vec{v} = 0 \tag{106.2}$$

$$\frac{\partial T}{\partial t} + \vec{v}\cdot\vec{\nabla}T = \Delta T \tag{106.3}$$

Given the above boundary conditions, initial temperature and Rayleigh number value $\mathsf{Ra} = 10^7$ these equations can be solved.

Concerning the value of Ra, it is said to be 10^7 but preliminary runs (in 2D) yield much narrower plumes for the isoviscous case than in the paper. However using 10^6 instead seems to visually match the paper's results. I asked S.K. whether there could not be a mistake in the reported Ra in the paper and he kindly answered the following: "A mistake is always possible. Here's a possibility: the $\mathsf{Ra} = 10^7$ was what was input into the input file but we did not do a proper internal scaling to account for this, i.e. r_o^3 was used rather than $(r_o - r_i)^3$ with $r_o^3 = 2.22^3 \sim 10$. Usually in our codes what we put in as the Rayleigh number is really rho-g-alpha and all the other factors are input as 1 (or implicitly assumed to be 1). At any rate, I suspect 10^6 could be a result of not accounting properly for the radius not being 1.0. I found some files one of my students used circa 2007 and the Ra was set to 10^7 in the input file while the grid (different file) was $1.2 \rightarrow 2.2$. I could see how one would forget when it comes to writing up the paper."

The paper mentions dimensioned quantities twice: in the caption of Fig. 3, we learn that $t = 0.000561$ corresponds to $1.5 \cdot 10^8$yr, $t = 0.000730$ corresponds to $2.0 \cdot 10^8$yr, and $t = 0.016353$ corresponds to $4.4 \cdot 10^9$yr. This yields a reference time of about 270e9, which is close to the one obtained as $t_{ref} = h^2/\kappa = (2891e3)^2/10^{-6} \sim 8.36 \cdot 10^{18} s \sim 265 \cdot 10^9$yr (see Section 3.11.2).

This stone relies on $Q_2 \times Q_1$ elements while the orginal paper relies on the cheaper $Q_1 \times P_0$ element with a penalty formulation (the value of the penalty parameter is also not mentioned).

The user chooses the number of elements in the radial direction *nelr* and the number of elements in the tangential direction is automatically computed as $nelt = 1.5 \cdot nelr$. Elements on the boundaries are flagged by means of the $flag_el_X$ array ($X = 1, 2, 3, 4$ corresponding to left, right, bottom and top). Similar boolean arrays are setup for nodes on the boundaries: $flag_X$. Free-slip boundary conditions are more difficult to impose than for Cartesian domains because the boundaries of the computational domain are either curved and/or not aligned with the Cartesian axis. The timestep is computed by means of a CFL criterion, but it is limited to a $2\times$ increase from one time step to the next to insure a somewhat smooth transition at the beginning when the plume is rising faster and faster. Because the lithostatic pressure does not contribute to the calculations and has effectively been removed from the equations the pressure that is being solved for is the dynamic pressure. We can therefore run models with open boundary conditions on the side walls. This is controlled by the $use_fs_on_sides$ parameter. If this parameter is true then there is a rotational nullspace and it is removed by setting $\vec{\mathrm{v}} = \vec{0}$ at $x = 0$ and $y = r_i$. This option was not explored in the original paper.

The equations are solved in their cylindrical axisymmetric form, see Section 3.14. Their FE discretisation is presented in Section 10.8.6 and Section 9.5.

Remark. *LK filter and/or SUPG should be tested (at the moment the temperature us thresholded in the material model). Explore resolution, initial T in the mantle, try scaled-down Steinberger average viscosity profile, look at dynamic topography, strain-rate dependent power-law rheology, different Ra number, ... finish scaling up to real Earth like dimensions. Would be nice to normalise the pressure on the top surface, not only a corner. Also remove all bc on sides and use reduced density only?*

Computing surface tractions

Pressure normalised to be zero in the upper right corner so as to remove the nullspace. It is then projected onto the velocity nodes. This quantity is then normalised so that the average pressure at the surface is zero.

Because of the axisymmetry the deviatoric stress tensor and stress tensor are given by

$$\boldsymbol{\tau} = \begin{pmatrix} \tau_{rr} & \tau_{r\theta} & \tau_{rz} \\ \tau_{\theta r} & \tau_{\theta\theta} & \tau_{\theta z} \\ \tau_{zr} & \tau_{z\theta} & \tau_{zz} \end{pmatrix} = \begin{pmatrix} \tau_{rr} & 0 & \tau_{rz} \\ 0 & \tau_{\theta\theta} & 0 \\ \tau_{zr} & 0 & \tau_{zz} \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\sigma} = -p\mathbf{1} + \boldsymbol{\tau}$$

The traction at the surface is given by

$$\vec{t}_n = \boldsymbol{\sigma} \cdot \vec{n} = \begin{pmatrix} \sigma_{rr} n_r + \sigma_{rz} n_z \\ 0 \\ \sigma_{zr} n_r + \sigma_{zz} n_z \end{pmatrix}$$

and the normal component of the traction by

$$t_n = \vec{t}_n \cdot \vec{n} = \sigma_{rr} n_r^2 + 2\sigma_{rz} n_r n_z + \sigma_{zz} n_z^2$$

Scaling it back up

What follows is an attempt at using the regular equations with Earth-like values for parameters...

As explained in Section 3.11.2 we choose four reference quantities:

- the reference length $L_{ref} = r_o - r_i = 2891\text{km}$ (trivial choice) with $r_i = 3480\text{km}$ and $r_o = 6371\text{km}$.
- the reference temperature $T_{ref} = \Delta T = 3000\text{K}$ (trivial choice)
- the thermal diffusion coefficient κ_{ref}. From Warren *et al.* [346]: $\rho_0 = 3250\text{kg}\,\text{m}^{-3}$, $k = 2.25\text{W}\,\text{m}^{-1}\,\text{K}^{-1}$, $C_p = 1250\text{J}\,\text{K}^{-1}$ so
$$\kappa = \frac{k}{\rho C_p} = \frac{2.25}{3250 \cdot 1250} \simeq 5.5 \cdot 10^{-7}$$
so we take $\kappa_{ref} = 5.5 \cdot 10^{-7}\text{m}^2\,\text{s}^{-1}$.
- the reference viscosity η_{ref}. The Rayleigh number is defined by
$$\mathsf{Ra} = \frac{\rho_0 g \alpha \Delta T (r_o - r_i)^3}{\eta_0 \kappa} = \frac{\rho_0^2 C_p g \alpha \Delta T (r_o - r_i)^3}{\eta_0 k}$$
with $\alpha = 3 \cdot 10^{-5}\text{K}^{-1}$ and $g = 9.81\text{m}\,\text{s}^{-2}$ so
$$\eta_{ref} = \eta_0 = \frac{\rho_0^2 C_p g \alpha \Delta T (R_{outer} - R_{inner})^3}{\mathsf{Ra}\ k} \simeq 1.25 \cdot 10^{22}\text{Pa}\,\text{s}$$
which is a very acceptable value for the mantle viscosity, see Section 25.3.

The dimensionless viscosity is either constant ($\eta_0' = 1$) or temperature-dependent

$$\eta'(T') = \eta_0' \exp\left[\frac{E}{R\Delta T}\left(\frac{1}{T' + t_0} - \frac{1}{1 + t_0}\right)\right]$$

where "t_0 is the surface temperature T_{surf} divided by the temperature drop across the shell ΔT". Assuming that the authors used

$$T' = \frac{T - T_{surf}}{T_{patch} - T_{surf}} = \frac{T - T_{surf}}{\Delta T}$$

then (after multiplying the equation above by η_{ref}):

$$\eta(T) = \eta_0 \exp\left[\frac{E}{R}\left(\frac{1}{T - T_{surf} + T_{surf}} - \frac{1}{T_{patch} - T_{surf} + T_{surf}}\right)\right] = \eta_0 \exp\left[\frac{E}{R}\left(\frac{1}{T} - \frac{1}{T_{patch}}\right)\right]$$

so that $\eta(T_{patch}) = \eta_0$. The authors investigate three cases:

$$\frac{E}{R\Delta T} = \{0, 0.25328, 3\}$$

With $R = 8.31$ and $\Delta T = 3000\text{K}$ then $E = \{0, 6317.6, 74829.6\}$ which are rather low values for an activation energy.

Note that the rheology can be written:

$$\eta(T) = \frac{1}{2} \underbrace{2\eta_0 \exp\left(-\frac{E}{RT_{patch}}\right)}_{A^{-1}} \exp\frac{E}{RT}$$

which is effectively a diffusion creep type viscosity. We find that for $T_{patch} = 3000 + 273 = 3273$:

$$A^{-1} = \{2.5e22, 1.98e22, 1.6e21\}$$

or,

$$A = \{4 \cdot 10^{-24}, 5.05 \cdot 10^{-24}, 6.26 \cdot 10^{-23}\}$$

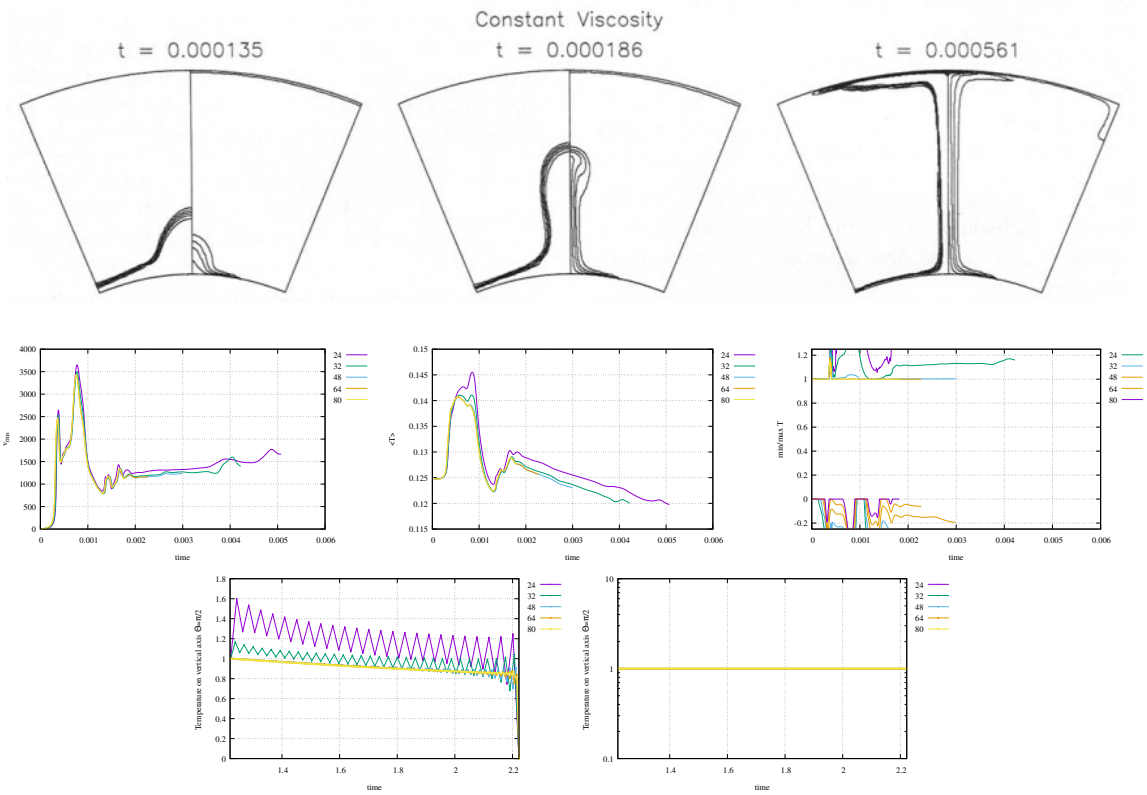
From the four reference values above we can construct additional ones, such as the reference stress, which we will need to look at dynamic topography. From Section 3.11.2:

$$\sigma_{ref} = \eta_{ref}\dot{\varepsilon}_{ref} = \eta_{ref}\frac{\kappa_{ref}}{L_{ref}^2} = \eta_0\frac{\kappa}{(r_o - r_i)^2} = \frac{\rho_0 g\alpha\Delta T(r_o - r_i)^3}{\mathsf{Ra}\ \kappa}\frac{\kappa}{(r_o - r_i)^2} = \frac{\rho_0 g\alpha\Delta T(r_o - r_i)}{\mathsf{Ra}} \simeq 829.55$$

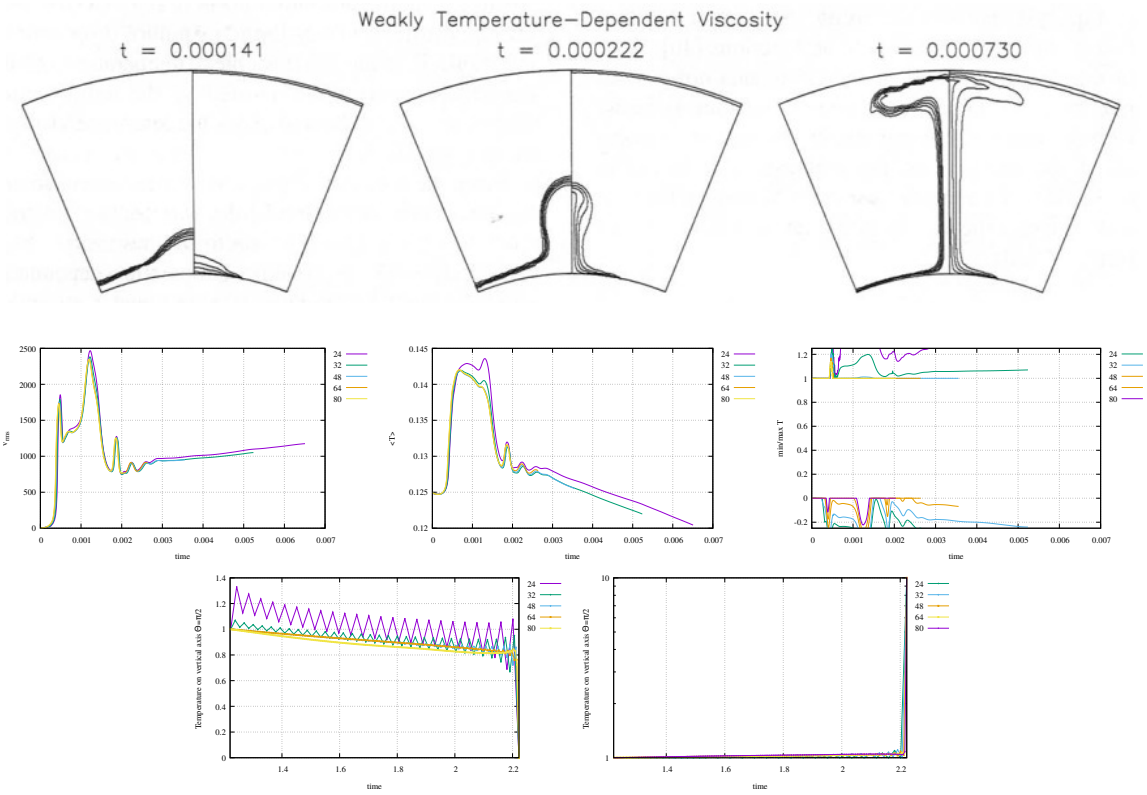
Of course this exercise is somewhat silly as the values of most material parameters vary with depth, temperature, etc ... and the choices I have made so far are rather arbitrary.

The following results obtained on various meshes. No SUPG. The CFL number is set to 1. $Q_2 \times Q_1$ elements are used.

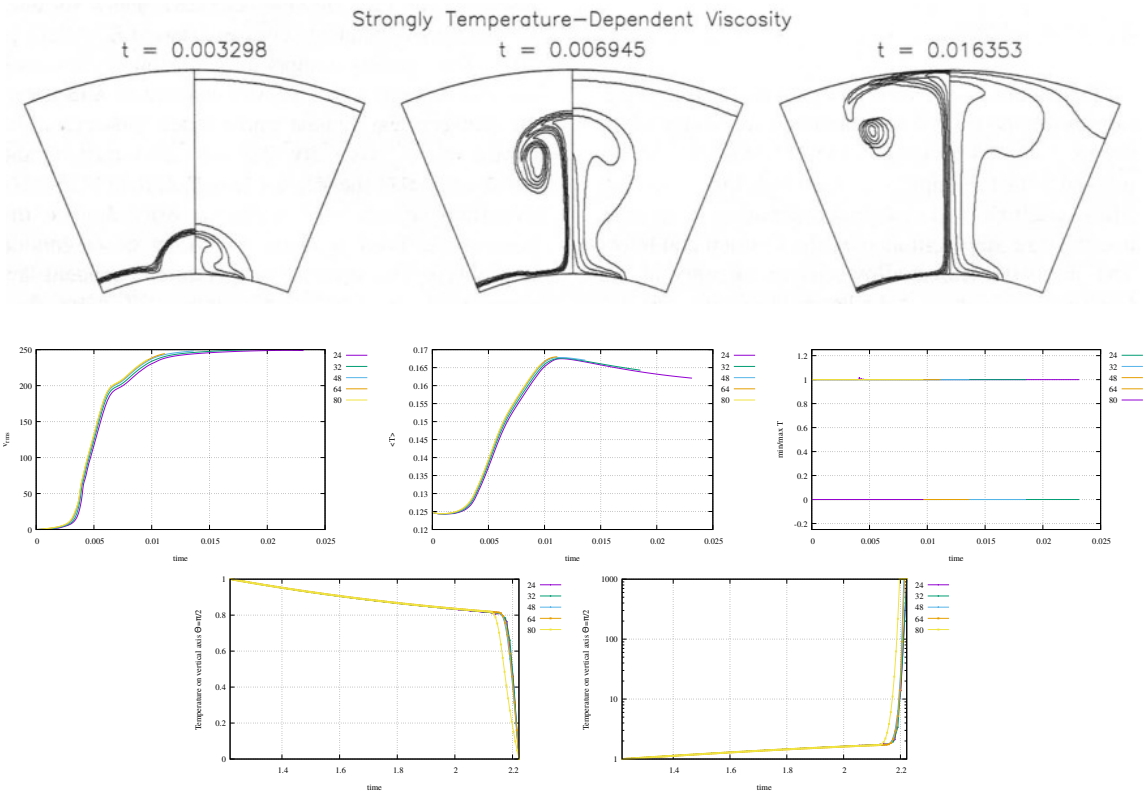
Isoviscous model - $Ra = 10^6$



Weakly temperature-dependent model - $Ra = 10^6$



Strongly temperature-dependent model - $\mathsf{Ra} = 10^6$



Stone 107

Convection in a porous medium

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_107`

This stone was developed in collaboration with Zoltán Erdős.

Outdated... being worked on. Nov 2024.

Manufactured solution

Before we start using the code for convection experiments, we wish to benchmark it with a manufactured solution. We start from the velocity of the Donea & Huerta benchmark in the unit square:

$$u(x,y) = x^2(1-x)^2(2y-6y^2+4y^3) \tag{107.1}$$
$$v(x,y) = -y^2(1-y)^2(2x-6x^2+4x^3) \tag{107.2}$$

which we know to be divergence-free. We postulate $p(x,y)=-x^3y^3+0.0625$ (so that it is on average zero over the domain) and set $\rho=1$, $\boldsymbol{K}=\mathbf{1}$, $\eta_f=1$ so that

$$\begin{aligned} g_x(x,y) &= -u(x,y)-\partial_x p \\ &= -u(x,y)+3x^2y^3 \\ g_y(x,y) &= -v(x,y)-\partial_y p \\ &= -v(x,y)+3x^3y^2 \end{aligned}$$

Stone 108

topography above lower crustal bordering the Tibetan plateau

REFDM

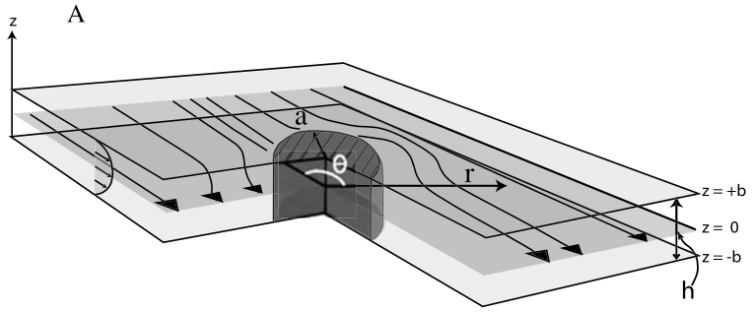
1D
elastic_rheology
finite differences

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_108`

This stone was developed in collaboration with Paul Pitard.

The article by Clark *et al.* (2005) [77] about "Dynamic topography produced by lower crustal flow against rheological strength heterogeneities bordering the Tibetan Plateau" is rather popular and has been cited over 300 times. It uses an analytical model to compute the dynamic pressure generated by a flow around an obstacle (the Sichuan Basin) which is then used as the driving load for the flexure equation in order to compute a vertical deflection. In what follows we go through their derivations first and then proceed to replicate their results.

We assume that the lower crust is ductile and flows in a flat channel. A single obstacle is present in the middle of the domain corresponding to a rigid block or sub-regional crustal fragment modelled as a cylinder with its axis perpendicular to the direction of flow. Its viscosity is taken to be infinite to make it rigid. We prescribe a unidirectional far-field flow U within the channel (far from the obstacle) and examine the effect of an impermeable rigid obstacle.



Taken from Clark *et al.* (2005). The center of the cylindrical coordinates axis system is in the middle of the cylinder. The far field velocity is aligned with the x-axis.

In the absence of/far from the obstacle the flow is characterised by no-slip boundary conditions on the top and bottom of the channel ($z = \pm b$) and flow symmetry produces parabolic channel flow $\vec{\mathrm{v}}$:

$$\vec{\mathrm{v}}(z) = -\frac{b^2}{2\eta}\vec{\nabla}P\left(1-\frac{z^2}{b^2}\right) \tag{108.1}$$

where $2b$ is the height of the channel. The obstacle has a radius a. We can compute the depth-averaged flow:

$$\langle\vec{\mathrm{v}}\rangle = \frac{1}{2b}\int_{-b}^{+b}\vec{\mathrm{v}}(z)\,dz = -\frac{b^2}{2\eta}\vec{\nabla}P\frac{1}{2b}\int_{-b}^{+b}\left(1-\frac{z^2}{b^2}\right)dz = -\frac{b^2}{2\eta}\vec{\nabla}P\frac{1}{2b}\left(2b-\frac{1}{3b^2}2b^3\right) = -\frac{b^2}{3\eta}\vec{\nabla}P = -\kappa\vec{\nabla}P \tag{108.2}$$

where κ is the effective permeability of the lower crust. Note that this is a Darcy-type flow equation.

No-slip boundary conditions are applied on the cylinder obstacle. The appropriate solution for a 2D potential flow around a circle/cylinder is given by[1]

$$\langle\mathrm{v}_r(r,\theta)\rangle = U\left(1-\frac{a^2}{r^2}\right)\cos\theta \tag{108.3}$$

$$\langle\mathrm{v}_\theta(r,\theta)\rangle = -U\left(1+\frac{a^2}{r^2}\right)\sin\theta \tag{108.4}$$

where U is the is the scalar value of the vertically averaged far-field flow, and the $\langle\cdot\rangle$ brackets indicate that these are vertically averaged fields. [2] We of course find that when $r\to a$ then the velocity is zero (no-slip boundary conditions on the obstacle) and when $r\to\infty$ then the velocity tends to $(U\cos\theta, -U\sin\theta)$ as expected since θ is the angle made with the direction of far-field flow.

The corresponding pressure field (excluding lithostatic pressure) is:

$$p(r,\theta) = -\frac{1}{\kappa}U\left(r+\frac{a^2}{r}\right)\cos\theta$$

since it is trivial to verify that inserting this expression in Eq. (108.2) yields the velocity components of Eqs. (108.3) and (108.4) (remember that in polar coordinates the gradients operator is $\vec{\nabla} = (\partial_r, \frac{1}{r}\partial_\theta)$.

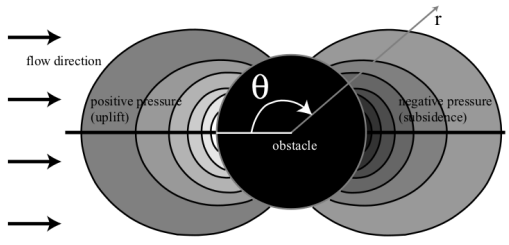
We find that the pressure field is only defined in the fluid, i.e. $r\geq a$ and that it depends on θ on the surface of the cylinder. Far from the cylinder this expression is a bit more problematic as it grows with r. We then split the pressure field as follows:

$$p(r,\theta) = -\frac{1}{\kappa}Ur\cos\theta - \frac{Ua^2}{\kappa r}\cos\theta$$

Clark *et al.* (2005) attribute the first one to the background (far-field) Darcy-type flow ($U\sim -\kappa p/r$) and the second to a local pressure field developed due to flow being diverted around the rigid obstacle. They then proceed to discard the first and call the second the dynamic pressure p_{dyn} which they use to drive a flexure equation.

$$p_{dyn}(r,\theta) = -\frac{Ua^2}{\kappa r}\cos\theta$$

The following figure shows the pressure field around the obstacle. We find that the pressure is null on a direction perpendicular to the flow direction passing through the center of the obstacle ($\theta=\pi/2$) and its maximum positive and negative amplitude is reached on the line given by $\theta=0$.



Taken from Clark *et al.* (2005).

[1] `https://en.wikipedia.org/wiki/Potential_flow_around_a_circular_cylinder`

[2] If $\mathrm{v} = \mathrm{v}_0(1-z^2/b^2)$ then $U = \frac{1}{2b}\int_{-b}^{+b}\mathrm{v}\,dz = 2\mathrm{v}_0/3$

Under many simplifying approximations (See Section 3.9 of Turcotte & Schubert book [335]), the simplest time-independent flexure equation is

$$D\frac{d^4w(x)}{dx^4} + P\frac{d^2w(x)}{dx^2} + (\rho_m - \rho_c)gw(x) = q(x)$$

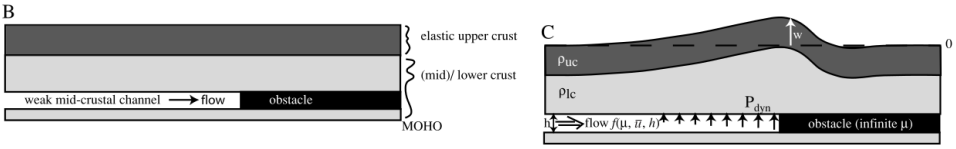
where $w(x)$ is the plate deflection (m),

$$D = \frac{Et^3}{12(1-\nu^2)}$$

is the flexural rigidity, with t the thickness of the plate (m), E is Young's modulus, ν is the Poisson ratio, P is a horizontal force per unit length (kg/m^{2}sec^2), q is the load (Pa), ρ_{lc} is the lower crust density and ρ_{uc} is the density of the upper crust (kg m^{-3}). This is a 4th-order Ordinary Differential Equation. In order to solve it, we will need 4 boundary conditions.

In the book the problem at hand is such that t is the real thickness of the bending plate. In geophysics this quantity is replaced by the so-called Effective Elastic Thickness Te. To quote Burov & Diament (1995): "The physical meaning and significance of the effective elastic thickness for continents are still enigmatic, because for continental lithosphere estimates of Te bear little relation to specific geological or physical boundaries." See also Tesauro *et al.* (2012) [322] for values of the effective elastic thickness of the continental lithosphere on Earth.

The approach in Clark *et al.* is rather peculiar because they do not really address the question of the thickness of the upper crust, although they draw it containing a viscous part and an elastic part:



Taken from Clark *et al.* (2005).

Instead the only upper crust property that is needed in the equation above is the effective elastic thickness.

Clark *et al.* (2005) neglect the P term, i.e. horizontal forces so that the equation becomes:

$$D\frac{d^4w(x)}{dx^4} + \delta\rho\, gw(x) = q(x)$$

where $\delta\rho$ is the density difference between the two layers (more on this later). They then assume that the dynamic pressure calculated above is the sole source of load on this elastic plate and then obtain

$$D\frac{d^4w(x)}{dx^4} + \delta\rho\, gw(x) = p_{dyn}(x)$$

which is their Eq. (10). Note that in fact x should be understood as the coordinate r. Because p_{dyn} depends on θ they can compute the deflection on any 1D line passing through the center of the obstacle.

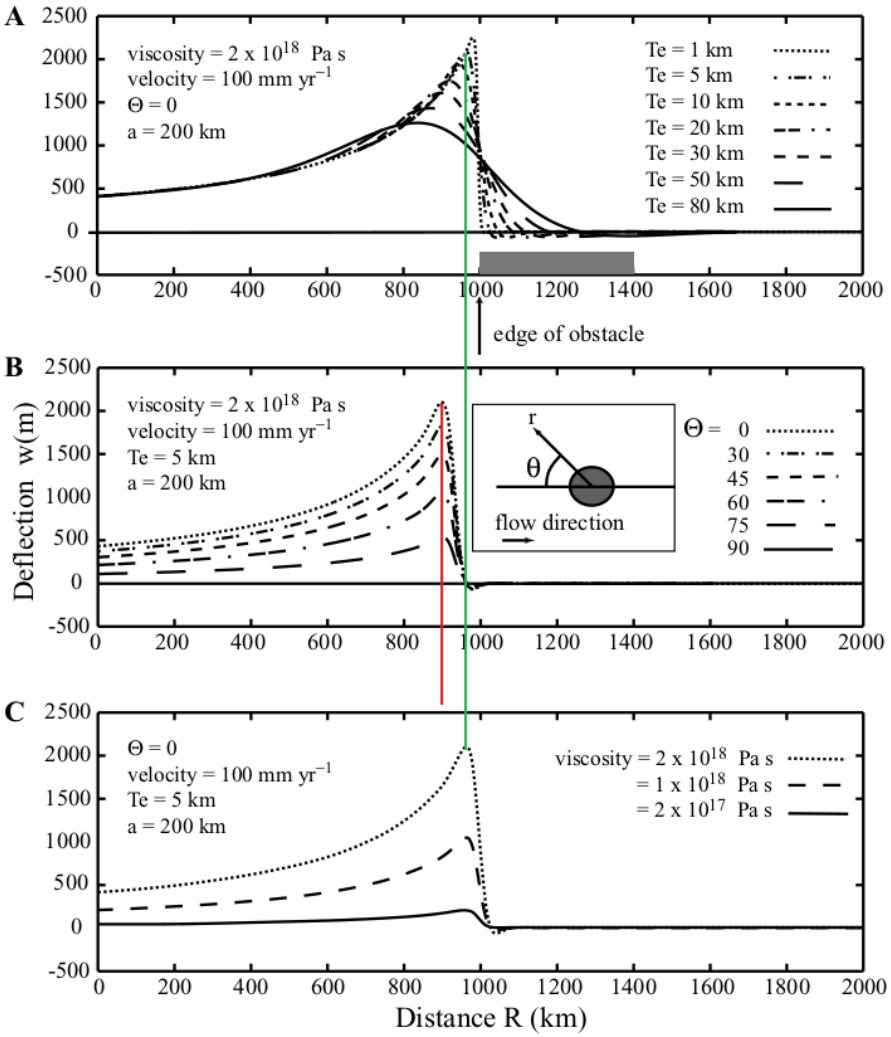
Solving this equation is rather trivial (see STONE 105) BUT we are missing the following things from their paper:

- boundary conditions on flexure equation. If domain is (very) large, we can assume standard ones.
- $\delta\rho$ value
- they provide values for the effective elastic thickness Te (t in the equations above) but fail to report E and ν so I cannot compute D. They however refer to Burov & Diament [55] and we find in there $E = 6.5 - 8 \cdot 10^{10}$N m^{-2} and $\nu = 0.25$. Taking $E = 7 \cdot 10^{10}$ yields $D = 6.222 \cdot 10^9 Te^3$
- the size of the domain (!)

The inescapable conclusion is that the paper is not replicable. Let us however make an attempt. We fix $\nu = 0.25$ and $E = 70 \cdot 10^9$ Pa. We Take a ridiculously large domain of 20,000 km so as to avoid any interference of the boundaries on the results. Boundary conditions are $w = 0$ and $w' = 0$ on the left and

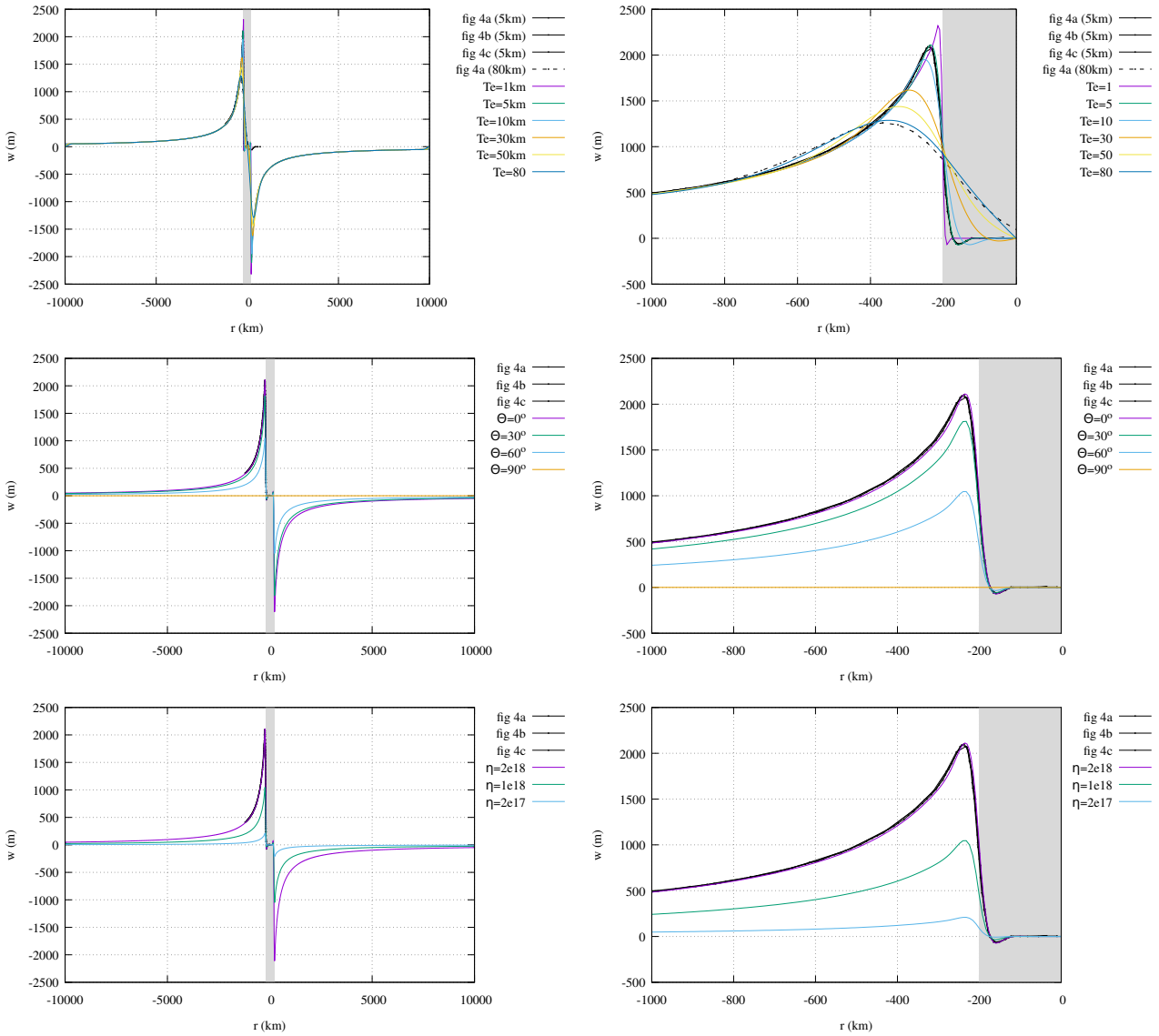
right boundaries. Turning now to $\delta\rho$ one must recall that in STONE 105 it is $\rho_m - \rho_c$ with a load being prescribed atop the crust. In our case the load is prescribed at the bottom of the crustal layer so the air above is the 'mantle layer' of STONE 105, so that $\delta\rho = \rho_{uc} - \rho_{air} = \rho_{uc}$ and we take $\rho_{uc} = 2850\ \mathrm{kg\,m^{-3}}$. Gravity is set to $g = 9.81\ \mathrm{m\,s^{-2}}$.

The code is not even 100 lines long. It solves the flexure equation, see STONE 105. The load is zero below the obstacle which is placed in the middle of the domain and equal to p_{dyn} outside of it. We then run the same models as Clark *et al.* (2005) and hope to reproduce the results of their Fig. 4 shown here under:



Taken from Clark *et al.* (2005). Note that all three subplots contain the same plot for $\eta = 2 \cdot 10^{18}$ Pa s, $v = 100$ mm yr^{-1}, $\theta = 0$, $a = 200$ km and $Te = 5$ km (our reference case) and yet the location of the maximum deflection in figure B is different than the one in A or C...? The measured offset between the red and green line was 63 km! The gray block indicates the obstacle.

It is worth noting that in the paper they do not show the right part of the model, i.e. the region where p_{dyn} is negative. Also I suspect that the authors did not run their model with the negative pressure at all since the deflection w is not similar to the positive pressure deflection on the right of the obtacle, i.e. for $R > 1400$ on the figure above.



Data from the paper were obtained with WebPlotDigitizer[3]. As mentioned above data from Fig 4b were shifted by 63km. The $r = 0$ value corresponds to the center of the obstacle. The left column corresponds to the obtained solution over the whole domain, and the right column to the solution obtained zoomed in on the same range as Fig. 4 of their paper. The gray area corresponds to the obstacle.

We find that we get near perfect match between our results and those of Clark *et al.* (2005).

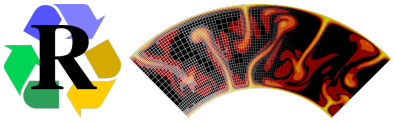
look at P again?

prescibe pdyn only on left

[3] https://automeris.io/WebPlotDigitizer/

Stone 109

Poiseuille flow with obstacle in 3D ($Q_2 \times Q_1$)



3D
box
Q2Q1
sparse_matrix
numerical_benchmark
stokes_equation
paraview_output
pressure_scaling
pressure_normalisation

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_109`

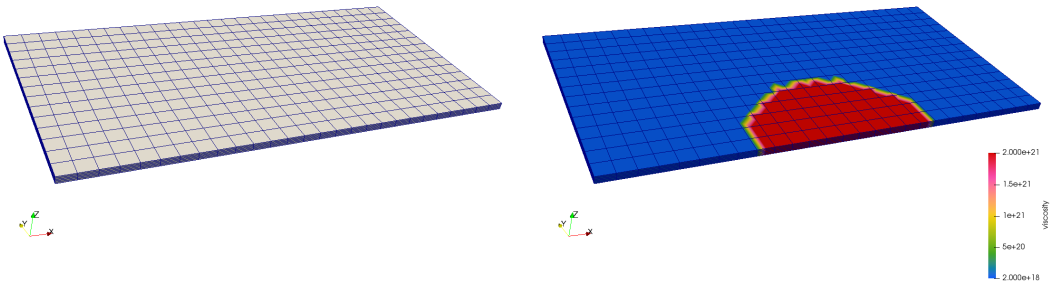
This stone reproduces the lower crust experiment explained in STONE 108 coming from Clark, Bush, and Royden [77] (2005).

The domain is $1000 \times 500 \times 15$ km. Gravity is set to zero so density is irrelevant. Boundary conditions are no slip at the top and bottom, free slip on $y = 0$ and $y = L_y$ and a parabolic Poiseuille flow is prescribed on $x = 0$ and $x = L_x$:

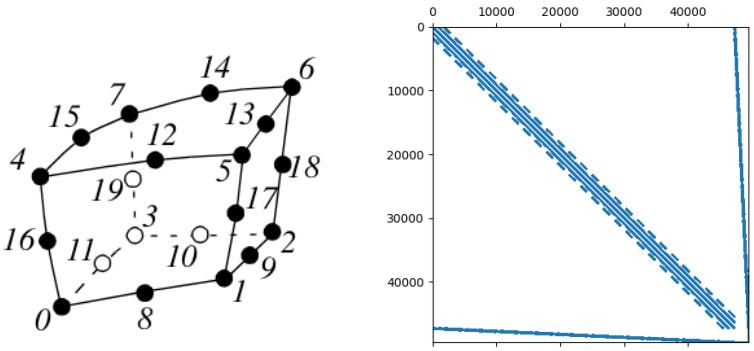
$$u(z) = U_0 \frac{4z(L_z - z)}{L_z^2}$$

with $U_0 = 80$ mm yr^{-1}.

The obstacle is centered on $(L_x/2, 0)$ and has a radius of $a = 200$ km (because of the symmetry of the problem I only model half of the domain). It is assigned a viscosity $\eta = 2 \cdot 10^{21}$ Pa s which is 1000 times larger than the viscosity of the channel $\eta = 2 \cdot 10^{18}$ Pa s.



Given the boundary conditions there is a pressure nullspace which is removed by enforcing that pressure is volume normalised, i.e. $\int_\Omega p\, dV = 0$. The element used is the Taylor-Hood pair $Q_2 \times Q_1$ (see Section **??**). The numbering of the nodes follows the one of the VTK format as shown hereunder. In retrospect it is not the most straightforward one but it is irrelevant with regards to the calculations. A complete layout of the nodes is present in the images folder of this stone.



Left: Node numbering of the 20 first nodes; Right: matrix structure

Because of the direct solver the resolution is rather limited and the maximum is about $27 \times 14 \times 5$ (the solver then claims about 26+ Gb). The analytical pressure

$$p(r,\theta) = -\frac{1}{\kappa} U r \cos\theta - \frac{U a^2}{\kappa r}\cos\theta$$

is also computed and projected onto the mesh (pressure is then set to zero inside the obstacle) (where $U = 2U_0/3$, see footnote in STONE 108).

Benchmarking

By setting the viscosity of the obstacle to the same value as the rest of the domain the problem becomes a classical Poiseuille flow problem, as described in Section **??**. In this case the pressure gradient Π is related to the velocity profile as follows:

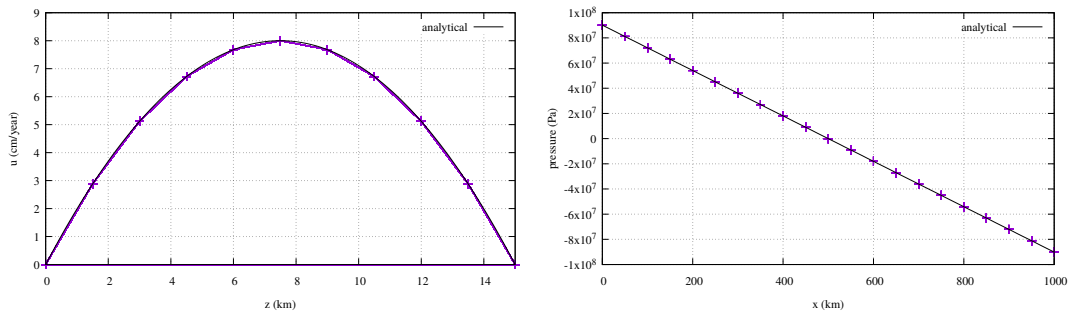
$$u(z) = \frac{1}{2}\frac{\Pi}{\eta_0}(z^2 - zH)$$

with $\Pi = \frac{\partial p}{\partial x} < 0$, i.e. there is more pressure applied to the left than to the right of the channel.

Looking above at our applied boundary velocity we find that

$$\frac{1}{2}\frac{|\Pi|}{\eta_0} = \frac{4U_0}{L_z^2}$$

or, $|\Pi| \simeq 18.03$. The domain is 1000 km long so $\Delta P = |\Pi| L_x \simeq 18,027,001$ i.e. we expect the pressure to be $90,135,005$ Pa at $x = 0$ and $-90,135,005$ Pa at $x = L_x$.

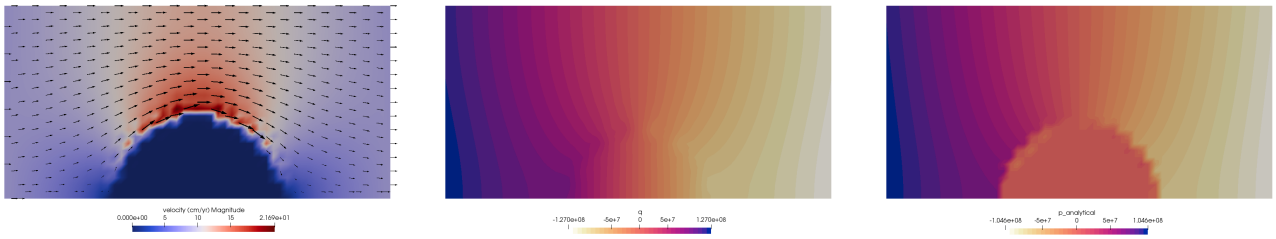


Computed velocity and pressure fields for the whole domain.

The computed velocity and pressure fields match their analytical counterparts so we can now turn to the problem at hand.

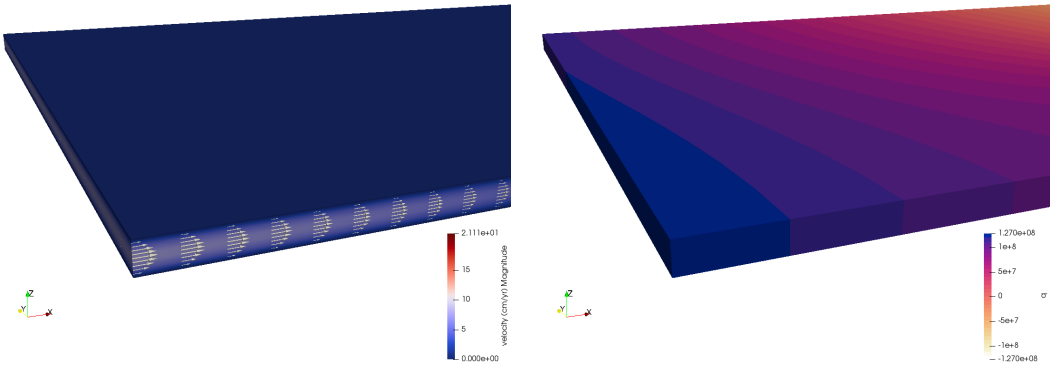
Application

We find that the computed pressure is rather similar to the analytical pressure outside the obstacle, although amplitudes differ by about 20%. This is likely due to the finite size of the domain while the analytical solution assumes an infinite domain around the obstacle.



Velocity, pressure and analytical pressure fields.

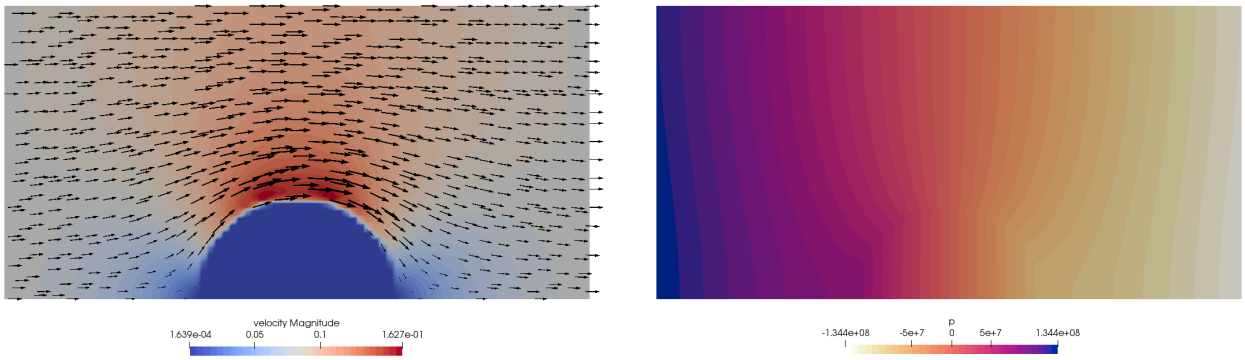
Note that the computed pressure does not depend on the z-coordinate (while of course the velocity does):



Velocity and pressure fields. We see that the aspect ratio of the elements is rather large and this is likely to alter the accuracy of the calculations.

The recovered pressure is rather similar to the analytical one. Indeed we find that an increase in resolution and an increase in domain size (especially L_z) brings the computed pressure closer and closer to the analytical one. At the max resolution that fits on my 32Gb laptop pressures are off by less than 10%.

I have also run this experiment with the ASPECT code and the prm file is present in the folder of this stone. Resolution was higher than above ($160 \times 80 \times 16 = 204,800$ elements).



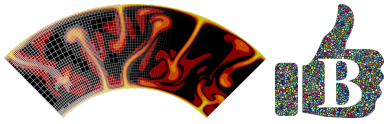


We find that the results obtained with ASPECT do not differ substantially from those obtained with this stone, despite a much higher resolution.

We find that the pressure min and max are quite larger than the isoviscous case (about 12.35MPa instead of 9.014MPa).

Stone 110

Convection in a 2D box - BA vs EBA ($Q_2 \times Q_1$)



2D
box
Q2Q1
numerical_benchmark
thermo_mechanical
buoyancy_driven
vrms
nusselt_number
rayleigh_number
paraview_output
steady_state
defect_picard

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_110`

The setup is taken from Blankenbach *et al.* (1989) [23] (See STONE 3, and Section **??**). The equations that we are solving are presented in Section **??**. Boundary conditions are free slip on all sides. There is therefore a pressure nullspace so a normalisation condition is needed. $Q_2 \times Q_1$ elements are used in what follows, although the code also implements the $Q_3 \times Q_2$ and $Q_4 \times Q_3$ elements, although it must be said that the postprocessors will most likely not work with these (yet).

Since we are only interested in the steady state and not the path to steady state, then the $\partial_t T$ term in the energy equation is zeroed and Picard iterations are used (with a relaxation parameter of 0.5) to arrive at the steady state. The convergence criterion is based on two factors: when the temperature field and the Nusselt number relative difference fall below the tolerance $tol = 10^{-7}$ then iterations stop. Note that the relaxation parameter can be close to 1 when the Rayleigh number is low.

The parameters are:

$$L_x = L_y = 1 \qquad \alpha = 2.5 \cdot 10^{-3} \qquad k = 1 \qquad C_p = 10^{-2} \qquad \rho_0 = 20 \qquad T_0 = 0 \qquad \vec{g} = (0,-1) \qquad \Delta T = 1$$

The Rayleigh number Ra is an input of the code (10^4, 10^5 or 10^6), so we choose the viscosity accordingly:

$$\eta_0 = \frac{\alpha g_y L_y^3 \rho_0^2 C_p \Delta T}{k \mathsf{Ra}}$$

This choice of parameters yields the following dissipation number:

$$\mathsf{Di} = \frac{\alpha g_y L_y}{C_p} = 0.25$$

The Boussinesq Approximation only relies on the Rayleigh number but the Extended Boussinesq Approximation relies on both Rayleigh and Dissipation numbers[1]. Because we target a specific value of the dissipation number Di we cannot choose the parameters entering the Rayleigh number as freely as before.

The shear heating term is $\Phi = 2\eta_0\dot{\epsilon} : \dot{\epsilon}$ since the flow is incompressible. With regards to solving the energy equation this term is trivial to implement since it does not depend on temperature at all and it ends up in the right hand side.

The adiabatic term is $\alpha T\vec{\mathrm{v}} \cdot \vec{\nabla} p$. We see that it depends on temperature so it is discretised in such a way that it contributes to the FE matrix via a mass matrix. The pressure is bilinear so its gradient at the quadrature points is easy to compute. Also because it is the gradient and not the pressure itself that enters this term then the normalisation constant does not influence results. Note that this term is often linearised by assuming that the pressure is mostly hydrostatic so that it then becomes: $-\alpha T\rho_0\vec{\mathrm{v}} \cdot \vec{g}$. Both expressions are exported to the vtu file for visual comparison.

Since no work is done on the domain and there are no internal heat sources/sinks, we then expect that energy conservation yields a zero heat flux balance on the boundary of the domain. Since no temperature is imposed on the sides this implies a zero heat flux. We can then monitor the bottom and top heat fluxes and sum them, expecting zero.

As explained in Section **??**, the above choice of parameters yields the following fundamental reference quantities:

- a length $L_{ref} = 1$
- a temperature $T_{ref} = 1$
- a viscosity $\eta_{ref} = \eta_0 = 10^{-6,7,8}$ (corresponding to $\mathsf{Ra} = 10^{4,5,6}$)
- a thermal diffusion coefficient $\kappa_{ref} = k/\rho_0 C_p = 5$

and the derivative reference quantity $\mathrm{v}_{ref} = L_{ref}/t_{ref} = \kappa_{ref}/L_{ref} = 5$. When written to file the root mean square velocity is divided by v_{ref} so as to allow for a direct comparison with existing published values. In what follows we monitor:

- the dimensionless root mean square velocity $\mathrm{v}_{rms}/\mathrm{v}_{ref}$,
- the average temperature $\langle T \rangle$,
- the dimensionless Nusselt number Nu,
- the heat flow balance $\vec{q}_{top} + \vec{q}_{bottom}$,
- the steady state average vertical temperature and velocity profiles.

The way the code is written now means that resolution is limited to 80x80 elements because of the memory cost (approx. 30Gb). Higher resolutions (96x96 were run on my super-desktop machine shrek with 128Gb RAM).

Note that the ASPECT results may not be true steady state, but the first 3 digits are good. Also ASPECT relies on a stabilisation scheme for the advection term of the energy equation unlike this STONE . Turning the stabilisation off was found not to alter results in any significant way.

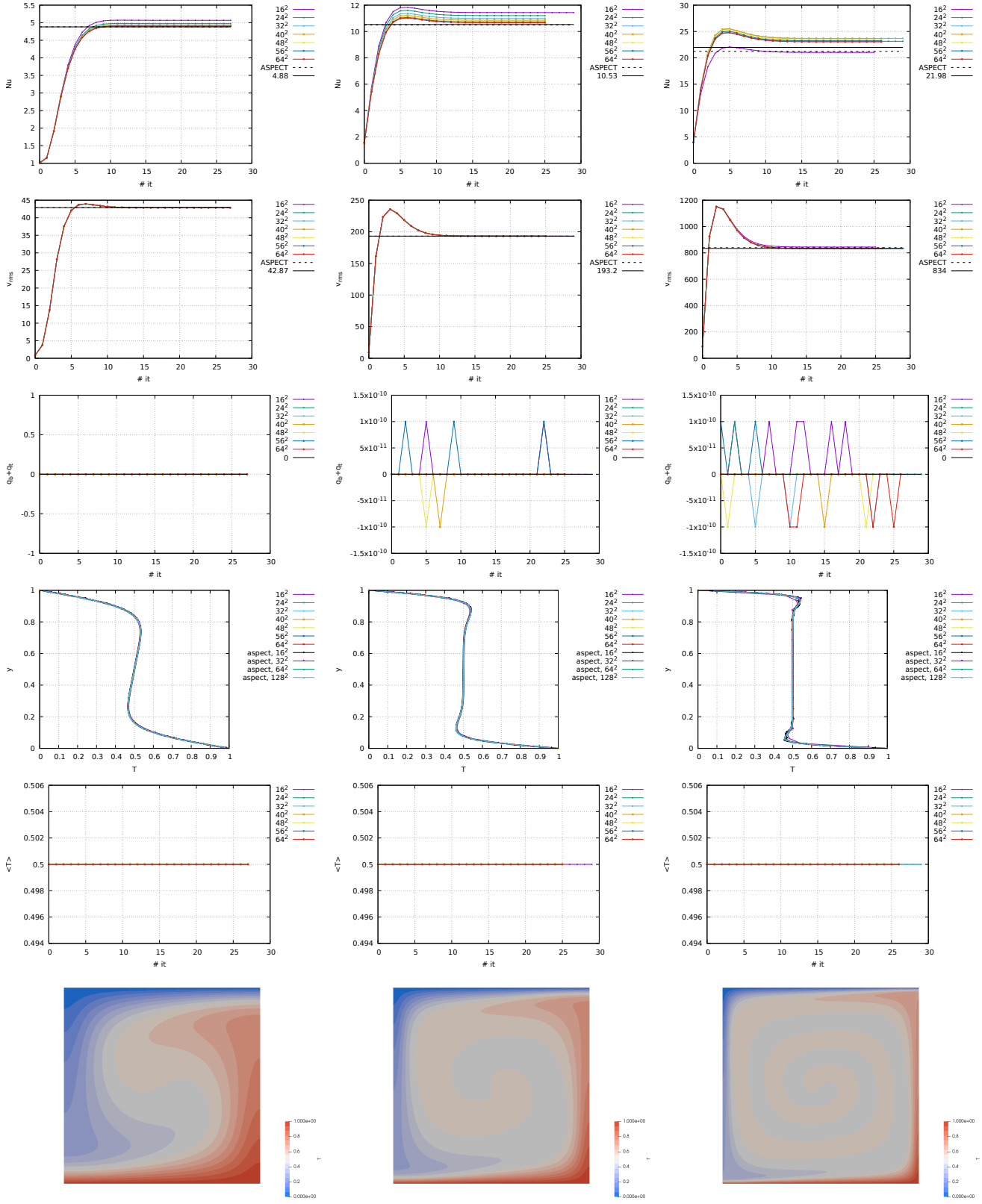
Vertical average profiles are computed as follows:

$$\langle T \rangle(y) = \int_0^{Lx} T(x,y)\, dx \qquad \langle \mathrm{v} \rangle(y) = \int_0^{Lx} \sqrt{u(x,y)^2 + v(x,y)^2}\, dx$$

The integrals are carried out inside each element on the bottom face (nodes 0,1,2), the 'middle face' (nodes 3,4,5) and the top face (nodes 6,7,8) and then summed together to arrive at the profile (the profiles arrays are nny long). Note that in STONE 1 I had implemented a simple nodal average per row of nodes and this approach turned out to be very inaccurate here.

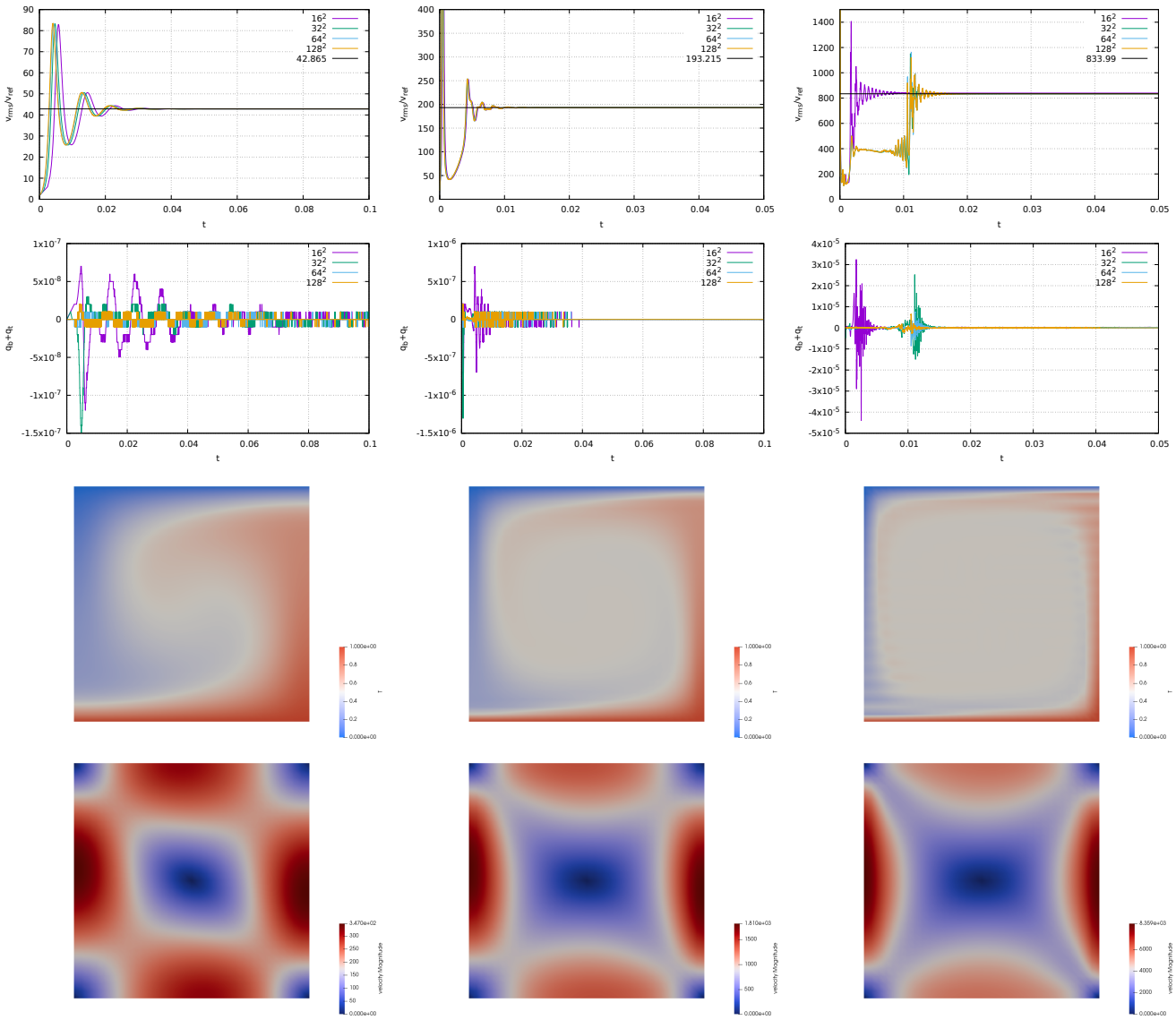
[1] Note that the temperature at the surface is also necessary to define the setup in a unique manner. This observation has been a major source of headache in efforts to replicate benchmark papers.

Boussinesq Approximation (BA)



Left: $\mathsf{Ra} = 10^4$, middle: $\mathsf{Ra} = 10^5$, right: $\mathsf{Ra} = 10^6$

ASPECT results for 16x16, 32x32, 64x64 and 128x128 meshes



Left: Ra = 10^4, middle: Ra = 10^5, right: Ra = 10^6

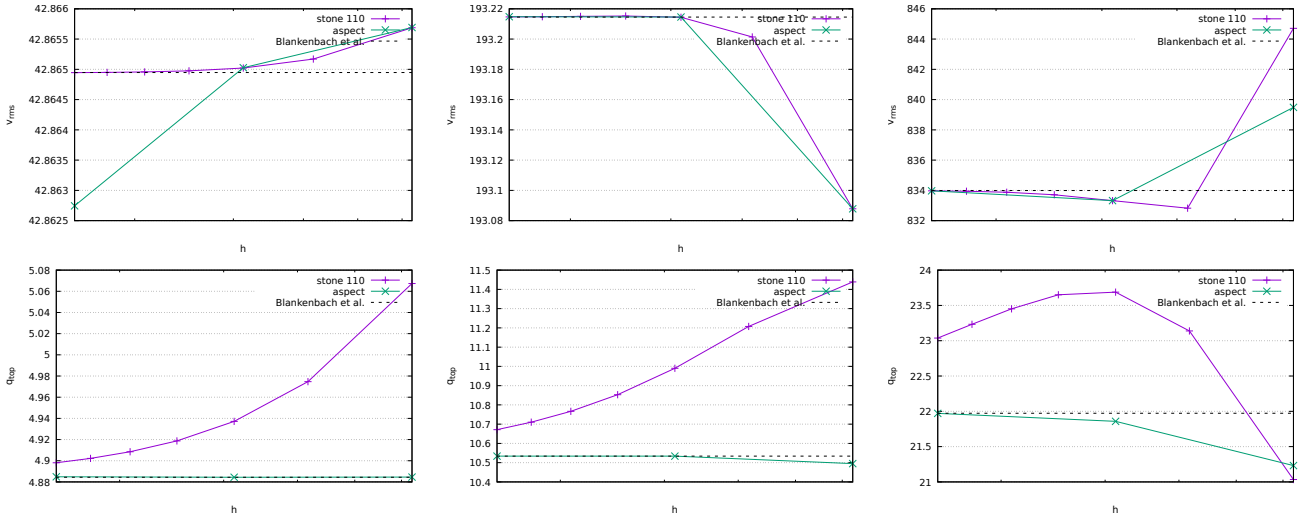
Ra		ASPECT (16x16)	ASPECT (32x32)	ASPECT (64x64)	STONE 110 (32x32)	STONE 110 (64x64)	Blankenbach *et al.* (1989)
10^4	ν_{rms}	42.8656918	42.865026	42.8627448	42.8650211917	42.8649453947	42.864947
	q_{bot}	-4.88459727	-4.88443000	-4.88484574	-4.9371243062	-4.8980781972	4.884409
	q_{top}	+4.88459726	+4.88443000	+4.88484574	+4.9371243062	+4.8980781972	
10^5	ν_{rms}	193.087842	193.214561	193.2147926	193.2144168167	193.2146484435	193.21454
	q_{bot}	-10.4951125	-10.5340940	-10.5341271	-10.9901799691	-10.6715594312	10.534095
	q_{top}	+10.4951125e	+10.5340940	+10.5341271	+10.9901799691	+10.6715594312	
10^6	ν_{rms}	839.48771	833.32106	833.972142	833.3211308177	833.9721673108	833.98977
	q_{bot}	-21.2315844	-21.8584326	-2.19710401	-23.6879005419	-23.0372569642	21.972465
	q_{top}	+21.2315844	+21.8584327	+2.19710400	+23.6879005419	+23.0372569642	

Velocity values are dimensionless in order to compare with Blankenbach *et al.* (1989) but not the heat flux values.

Ra		ASPECT (32x32)	ASPECT (64x64)	ASPECT (128x128)	STONE 110 (32x32)	STONE 110 (64x64)	STONE 110 (80x80)	STONE 110 (96x96)	Blankenbach *et al.* (1989)
10^4	ν_{rms}	42.865026	42.8649502	42.864945					
	q_{bot}	-4.88443000	-4.88441060	-4.88440926					
	q_{top}	4.88443000	4.88441060	4.88440926					
10^5	ν_{rms}	193.214561	193.2147924	193.2145528					
	q_{bot}	-10.5340940	-10.5341271	-10.5340972					
	q_{top}	10.5340940	10.5341271	10.5340972					
10^6	ν_{rms}	833.321056	833.972142	833.990308					
	q_{bot}	-21.8584327	-21.9710401	-21.9725040					
	q_{top}	21.8584326	21.9710400	21.9725040					

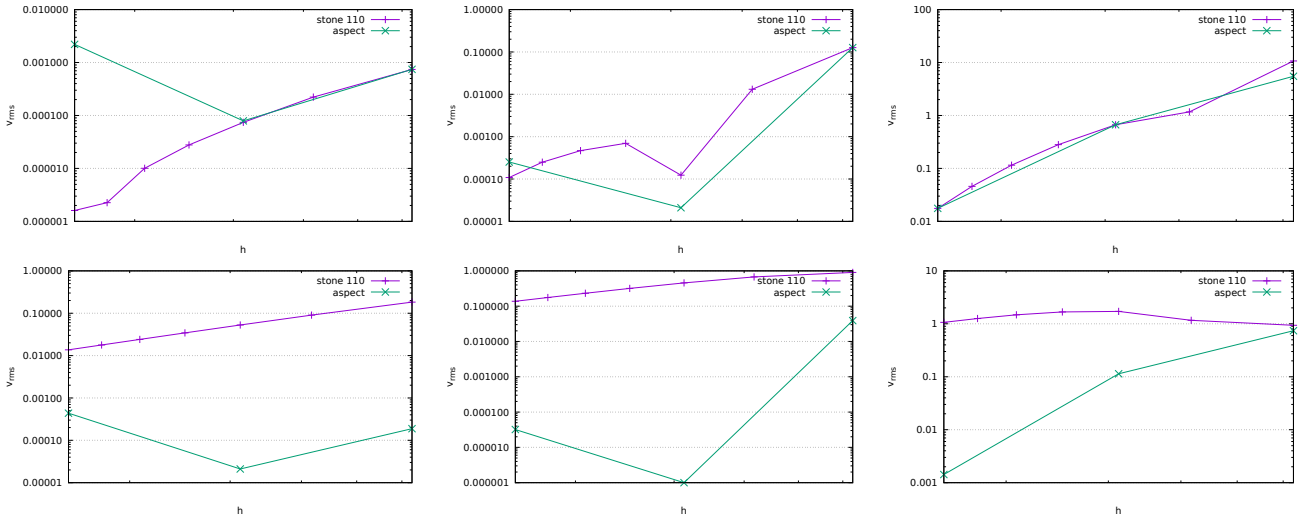
It looks like the heat flux calculations are much more accurate in ASPECT , most likely due to the CBF algorithm. Also ASPECT relies on the entropy stabilisation for the advection and it is likely to slightly alter the temperature field (additional diffusion in zones of high gradients).

On the following figures I report the steady state ν_{rms} and q_{top} values for both codes and for all three Rayleigh numbers:



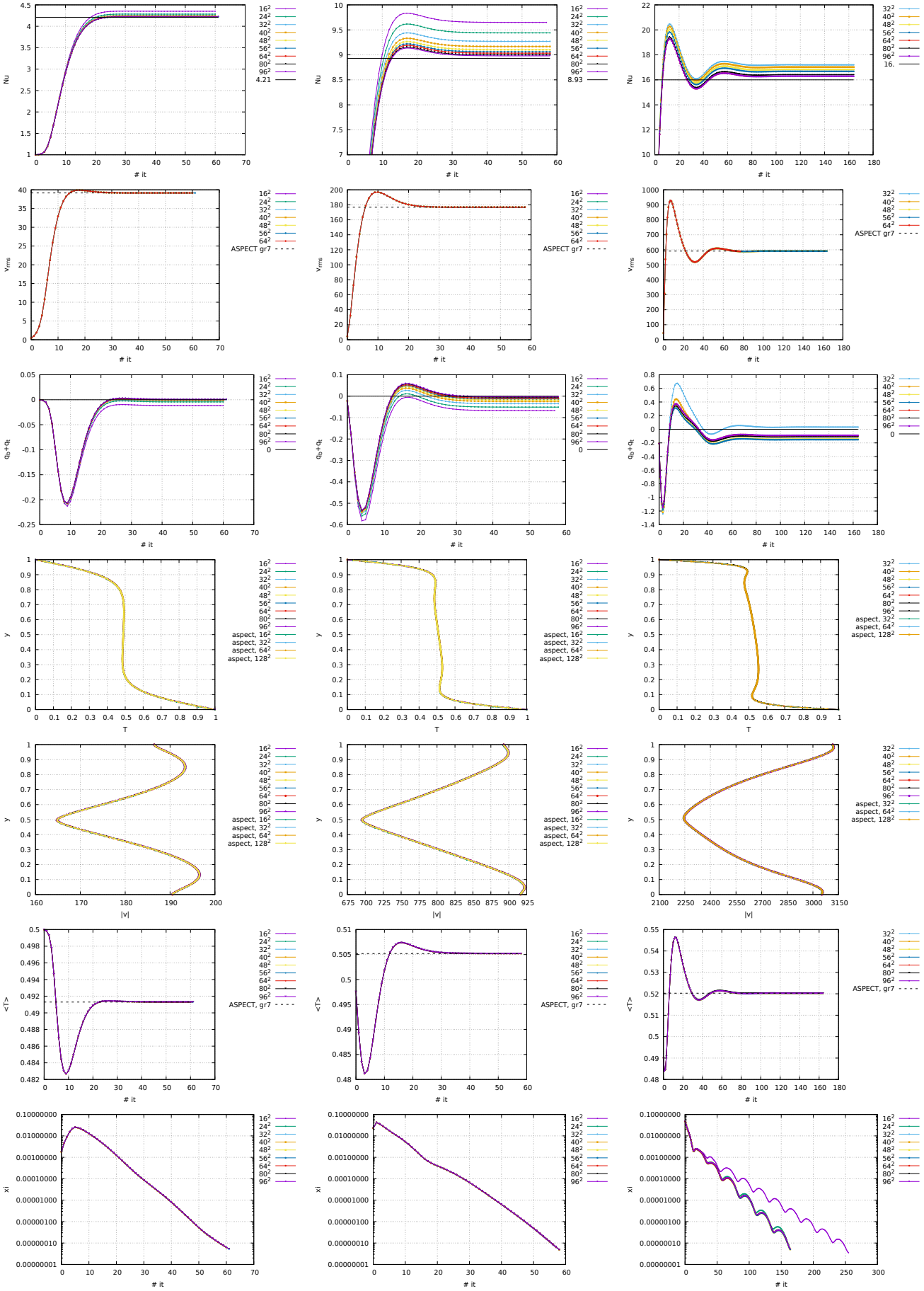
Left: Ra $= 10^4$, middle: Ra $= 10^5$, right: Ra $= 10^6$

The heat flux/Nusselt number results obtained with this STONE seem to converge to the expected values but ASPECT results are much more accurate even at low resolution... Looking at the vtu files they look virtually identical. since the vrms are also near identical I am def leaning towards the CBF for the reason why aspect>stone.

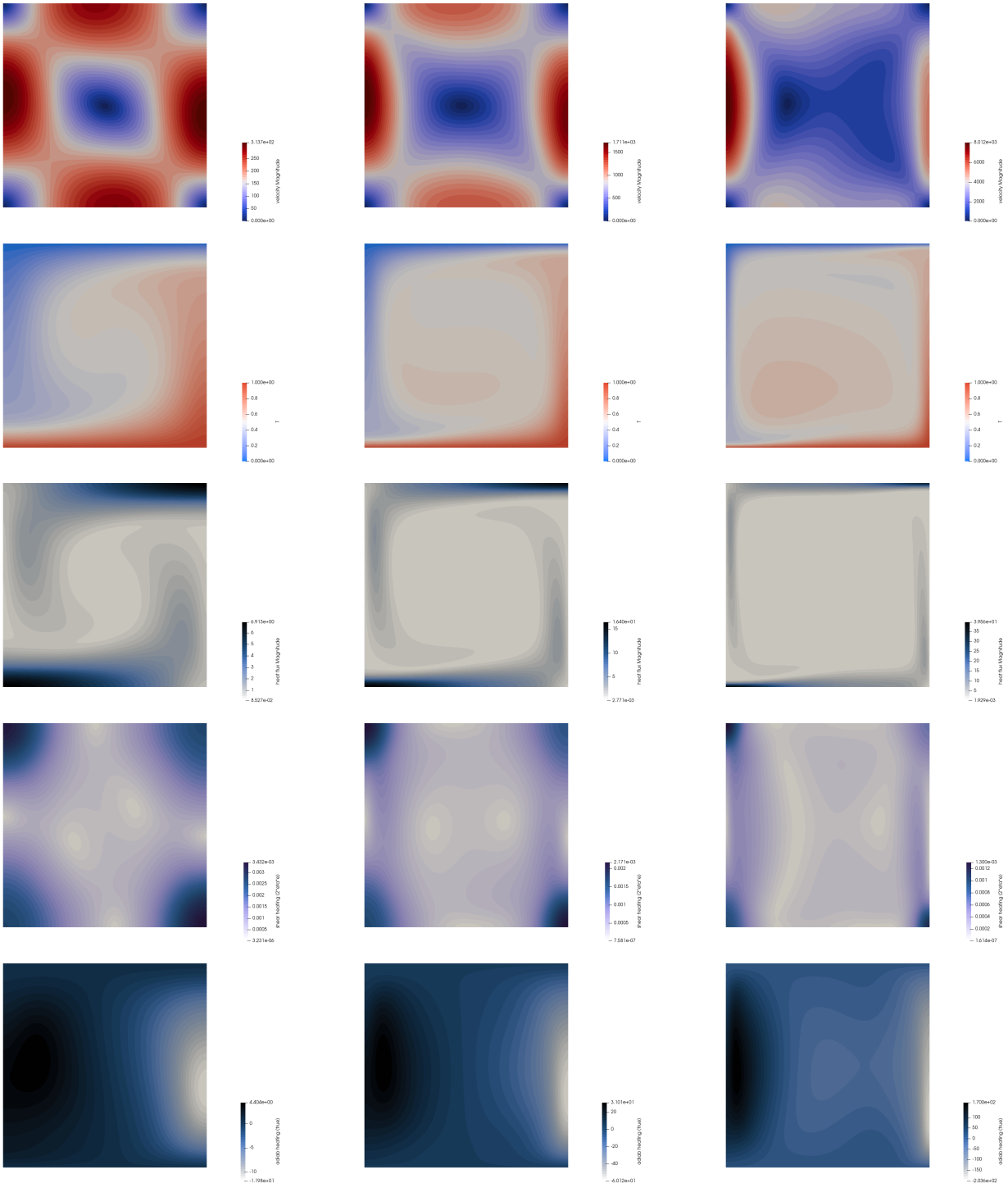


Left: Ra $= 10^4$, middle: Ra $= 10^5$, right: Ra $= 10^6$

Extended Boussinesq Approximation (EBA)

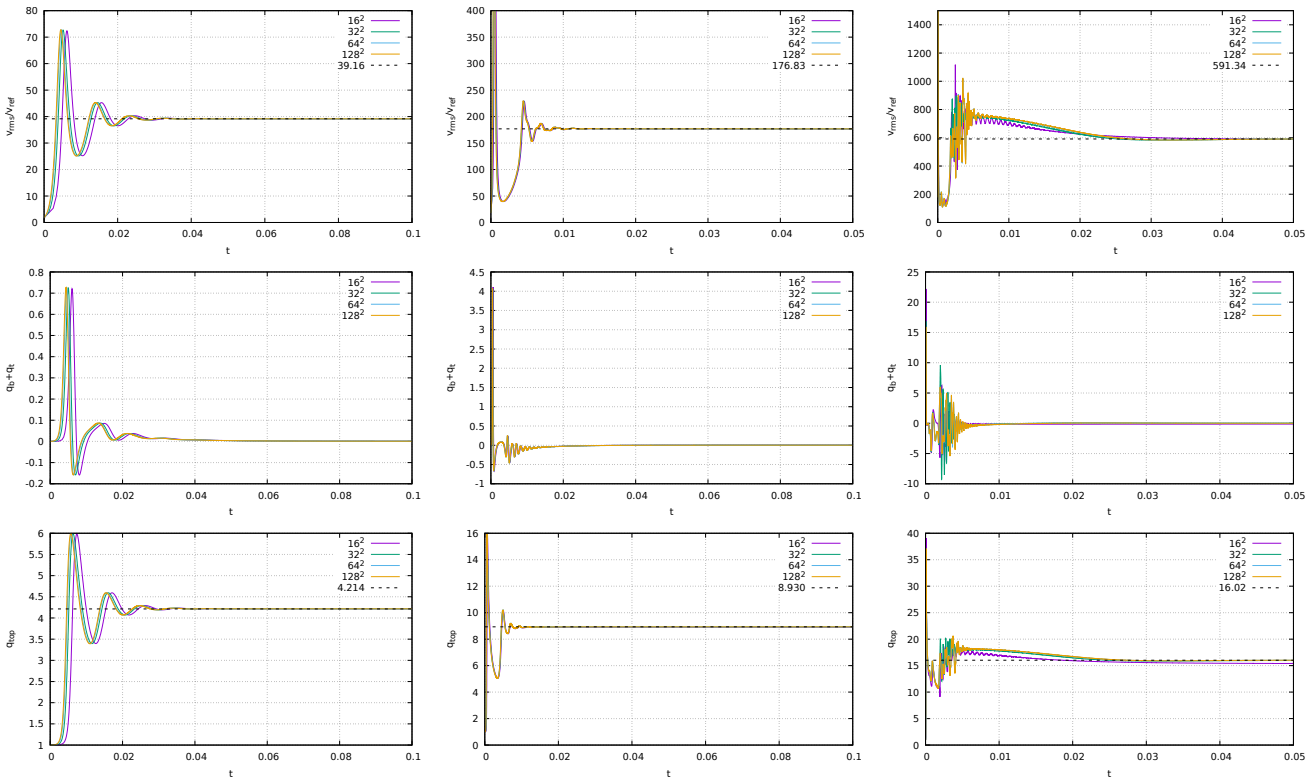


Left: $\mathrm{Ra} = 10^4$, middle: $\mathrm{Ra} = 10^5$, right: $\mathrm{Ra} = 10^6$



Left: $\mathsf{Ra} = 10^4$, middle: $\mathsf{Ra} = 10^5$, right: $\mathsf{Ra} = 10^6$

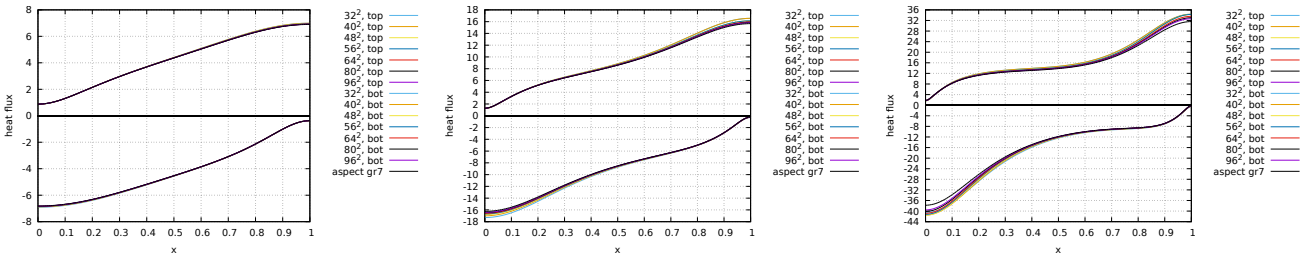
ASPECT results for 16×16, 32×32, 64×64 and 128×128 meshes



Left: Ra $= 10^4$, middle: Ra $= 10^5$, right: Ra $= 10^6$

Ra		ASPECT (32x32)	ASPECT (64x64)	ASPECT (128x128)	STONE 110 (32x32)	STONE 110 (64x64)	STONE 110 (80x80)	STONE 96 ×
10^4	v_{rms}	39.159131	39.1590718	39.1637782	39.1220715117	39.1427811339	39.1637762658v	39.16377
	q_{bot}	-4.21461224	-4.21459920	-4.21497114	-4.2526699758	-4.2241736047	-4.2219231712v	-4.21980
	q_{top}	+4.21466572	+4.21465118	+4.21595249	+4.2561018386	4.2260501790	4.2224541900v	4.22047
10^5	v_{rms}	176.8339244	176.7353244	176.8536186	176.3717700756	176.6073765137	176.8532981837v	176.8532
	q_{bot}	-8.93023781	-8.90386799	-8.93084342	-9.2311968388	-9.0088339160	-9.0034702435v	-8.98186
	q_{top}	+8.93021403	+8.90519202	+8.93331108	+9.2584172686	9.0239878253	8.9981639734v	8.97884
10^6	v_{rms}	590.707916	591.340132	591.408084v	556.3611676986	570.1830724535	591.4131298027v	591.4206
	q_{bot}	-15.9490236	-16.0182326	-16.0198829v	-16.1081289993	-16.1217624933	-16.5082181546v	-16.3811
	q_{top}	+15.9661980	+16.0184006	16.0247946v	+16.7479398662	16.2626758191	16.3927306028v	16.29137

Note that the vrms values of aspect in the statistics file must be divided by the reference velocity (i.e. 5).



Top and bottom heat flux obtained with STONE 110 and ASPECT . There is a very good match for Ra $= 10^{4,5}$ but still a small discrepancy at Ra $= 10^6$.

Implementing the CMB algorithm in the STONE would most likely yield much better results.

Stone 111

Manufactured solution for compressible elastic deformation in a disc



2D
disk
P1
sparse_matrix
compressible
analytical_benchmark
error_convergence
elastic_rheology
vrms
paraview_output
displacement_code

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_111`

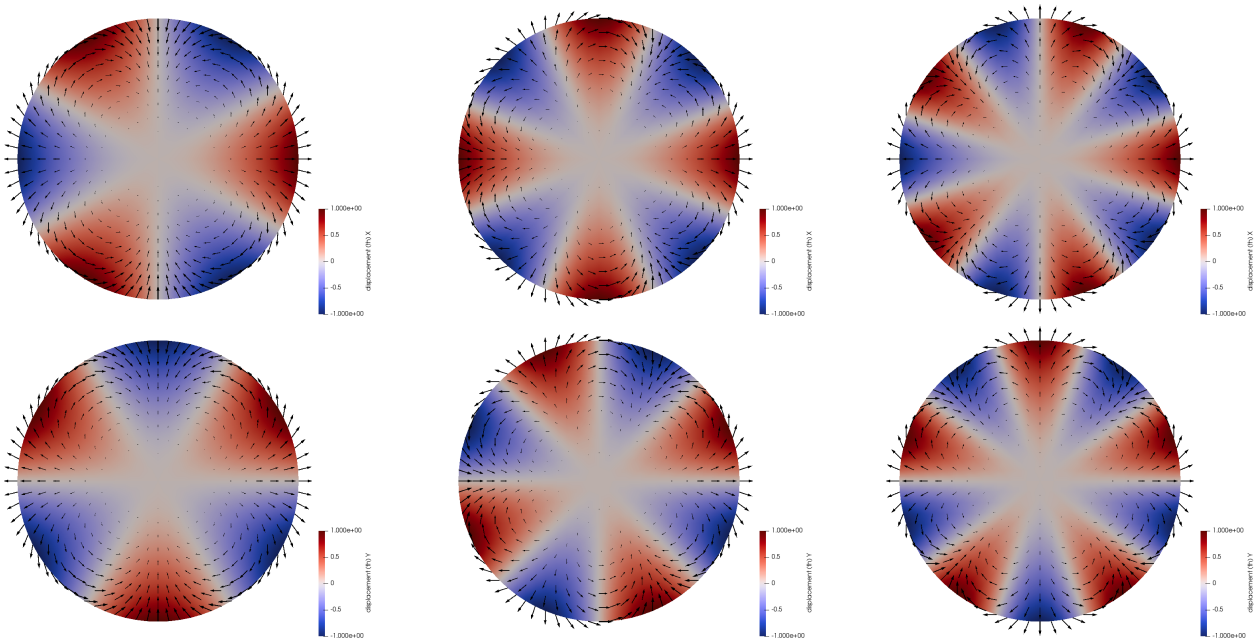
The derivations for this benchmark are found in Section **??**. We here recall the fields for $n \geq 2$ and $k \geq 1$:

$$
\begin{aligned}
\nu_r &= \nu_0 r^n \cos(k\theta) \\
\nu_\theta &= \nu_0 r^n \sin(k\theta) \\
\varepsilon_{rr} &= \nu_0 n r^{n-1} \cos(k\theta) \\
\varepsilon_{\theta\theta} &= \nu_0 r^{n-1} (k+1) \cos(k\theta) \\
\varepsilon_{r\theta} &= \frac{\nu_0}{2} r^{n-1} (n-1-k) \sin(k\theta) \\
\nu_{rms} &= 2\pi \nu_0^2 \frac{R^{2n+2}}{2n+2} \\
\sigma_{rr} &= \lambda(\vec{\nabla}\cdot\vec{\nu}) + 2\mu\varepsilon_{rr} \\
&= \nu_0 r^{n-1} \cos(k\theta)[\lambda(n+k+1) + 2\mu n] \\
\sigma_{\theta\theta} &= \lambda(\vec{\nabla}\cdot\vec{\nu}) + 2\mu\varepsilon_{\theta\theta} \\
&= \nu_0 r^{n-1} \cos(k\theta)[\lambda(n+k+1) + 2\mu(k+1)] \\
\sigma_{r\theta} &= 2\mu\sigma_{r\theta} \\
&= \mu\nu_0 r^{n-1} (n-1-k) \sin(k\theta)
\end{aligned}
$$

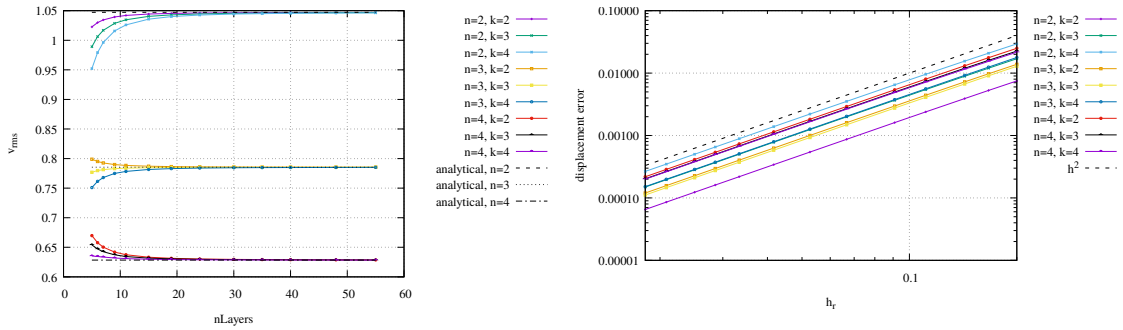
Note that the ν_{rms} is independent of k.

In what follows the disc has radius $R = 1$. We set ν_0 and choose $\mu = 1$, the Poisson ratio $\nu = 0.25$ so that $\lambda = 2\mu\nu/(1 - 2\nu) = 1$. The mesher is borrowed from STONE 58. The mesh consists of `nLayers` concentric layers of elements. We choose as 'diameter' of elements the value $h_r = R/nLayers$. The code relies on first-order triangular elements (P_1). The analytical displacement field is prescribed on the boundary. Compared to STONE 58, this manufactured solution has the advantage that it provides displacement, strain and stress components everywhere in the disk: these fields have a simple analytical expression and are finite.

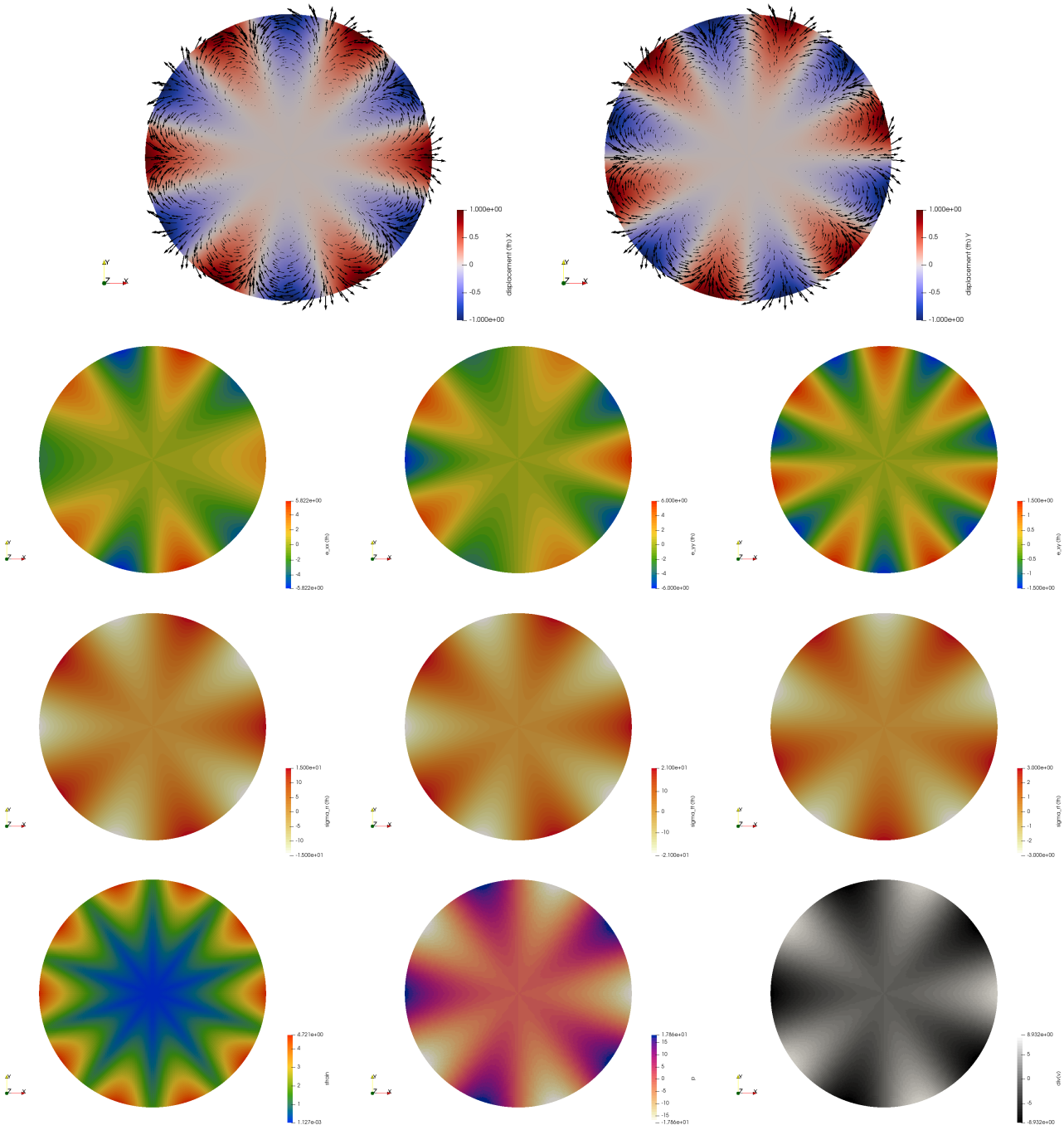
The parameter n influences the dependence in r of the fields while the parameter k influences the fields in the θ direction.



u and v fields for $n = 2$. Left to right: $k = 2, 3, 4$.



Root mean square displacement and discretisation errors obtained with $n = 2, 3, 4$ and $k = 2, 3, 4$. Quadratic convergence is recovered as expected with linear elements.



Various fields for $n = 3$ and $k = 5$. Just because it is pretty.

Stone 112

MINI, $P_2 \times P_1$, $Q_2 \times Q_1$, $Q_2 \times P_{-1}$ and C-R elts



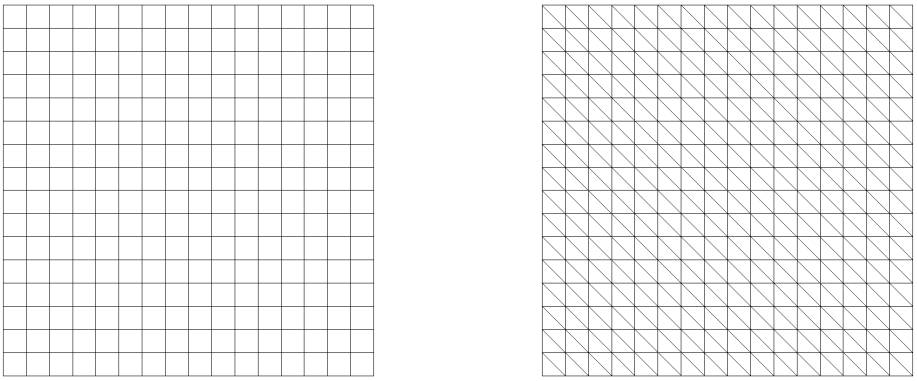
2D
box
Q2Q1
Q2Pm1
MINI
P2pPm1
P2P1
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
vrms
paraview_output
pressure_normalisation
pressure_bc

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_112`

This STONE showcases the MINI (Section 10.5.7), $P_2 \times P_1$ (Section 10.5.8), $Q_2 \times Q_1$ (Section 10.4.2), $Q_2 \times P_{-1}$ (Section 10.4.3 - unmapped approach, see STONE 76) and Crouzeix-Raviart (Section 10.5.9) elements. All experiments take place in the unit square (except SolVg).

The mesh is composed of $nel_x \times nel_y$ cells which are cut in half via a diagonal (NW-SE or SO-NE) into 2 triangles for simplicity[1] when triangular elements are used:

[1] I revisit this for SolVi



Quadrilaterals and triangles mesh for a resolution of 16 × 16 cells.

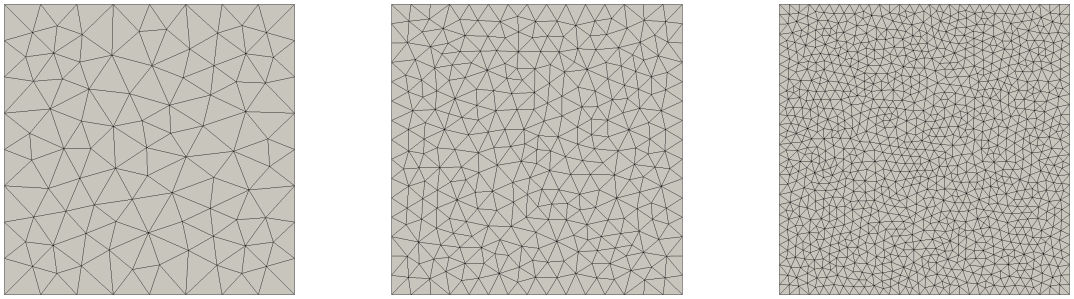
The last pressure dof is fixed to remove the nullspace and the obtained pressure field is then normalised so that $\int_\Omega p dV = 0$.

Resolutions are ranging from 8×8 to maximum 320×320. The high resolution runs take hours because python is slow. 320×320 seems to be the limit on my 32Gb-RAM laptop (I build and feed the entire Stokes matrix to the solver).

These are the important numerical parameters which allow to control the type of calculation in the code:

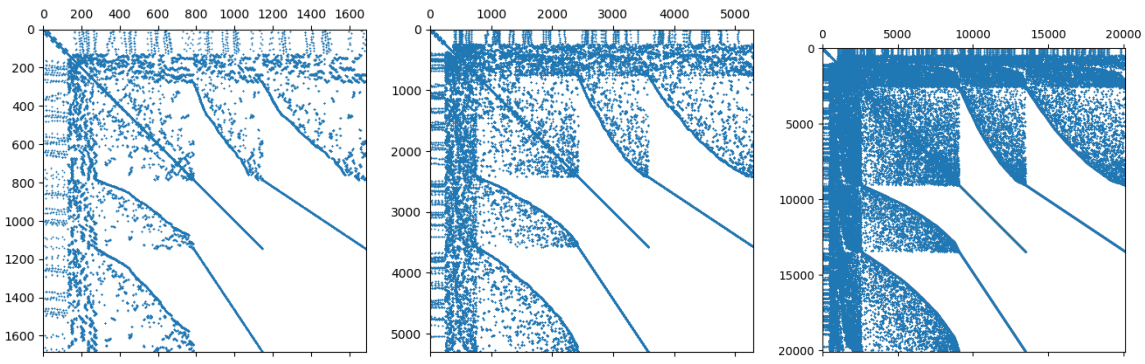
- elt=MINI, P2P1, CR, Q2Q1 or Q2P1.
- experiment: 1→ Donea & Huerta manufactured solution; 2 → high viscosity sinker; 3→ SolCx; 4→ SolCx; 5→ SolVi; 6→ SolVg (viscosity grooves).
- nelx,nely: the number of cells in each direction
- randomize_mesh allows to add a small random perturbation to the nodes

In order to test simplicial meshes with a topology that is not as regular as the one generated by splitting quadrilaterals I have created 5 meshs with gmsh (see Appendix **??**).



gmsh meshes at level 1,2,3. 4 and 5 are not shown.

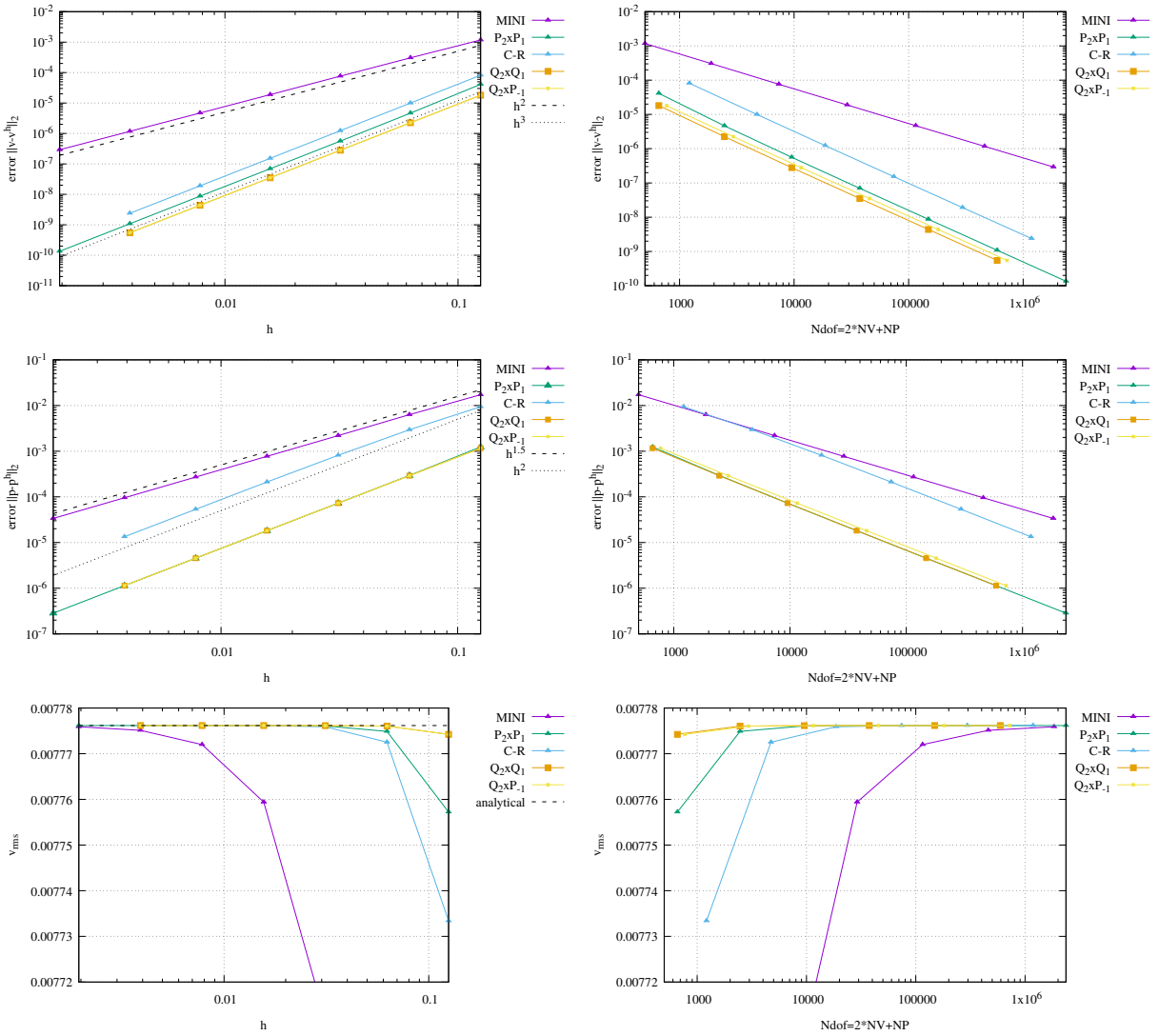
These meshes are truly unstructured ('random' if you prefer) and the resulting sparsity pattern of the resulting Stokes matrix reflects this fact



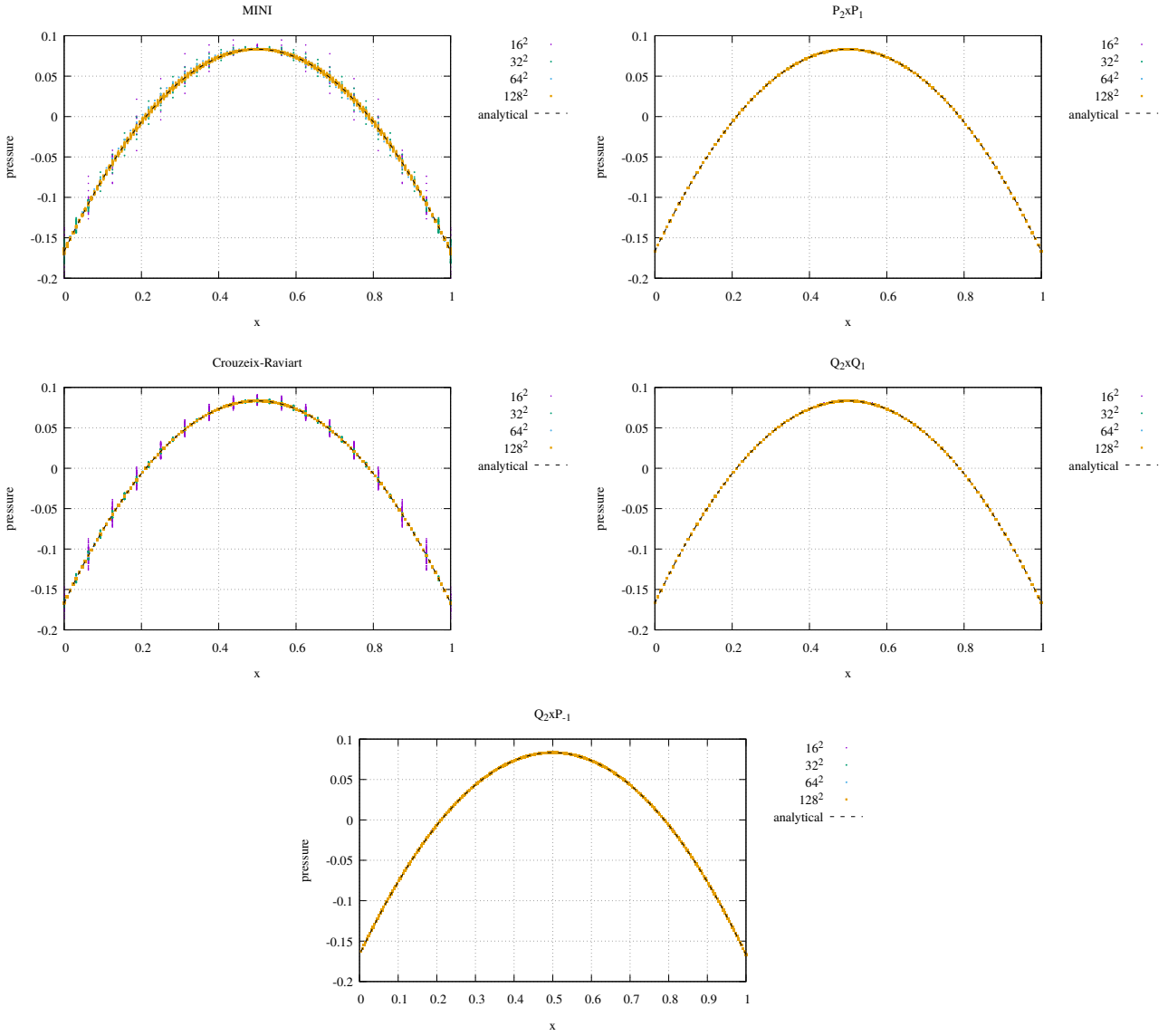
Matrix sparsity pattern corresponding to meshes above for C-R element.

Donea & Huerta manufactured solution

We start with the manufactured solution problem "Donea & Huerta" (see Section **??**).

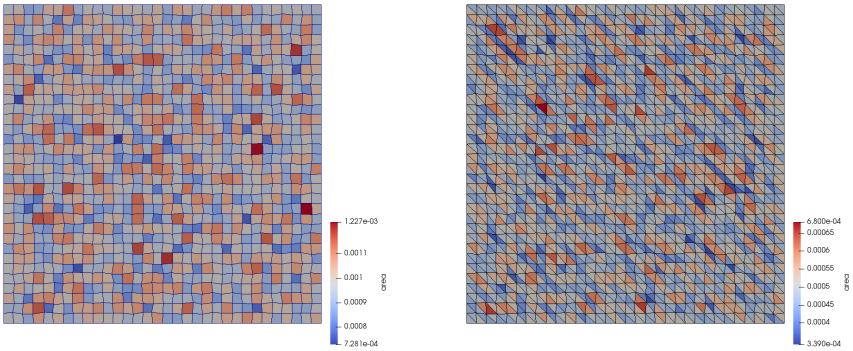


Velocity L_2 error (top row), pressure L_2 error (middle row) and root mean square velocity (bottom row) as a function of the element size h (left column) and the total number of degrees of freedom (right column).

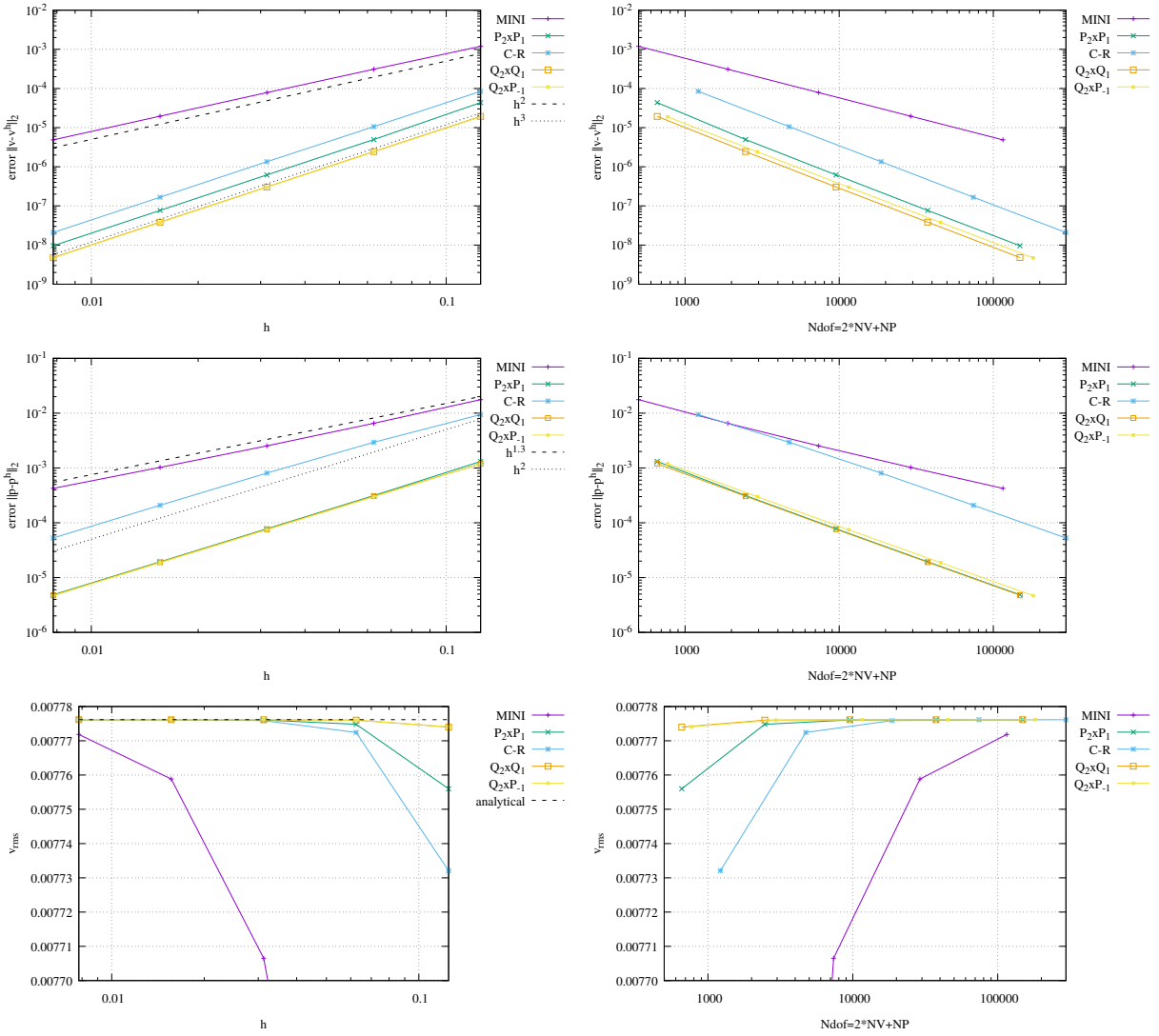


Pressure field as a function of the x-coordinate on a 16×16, 32×32 and 64×64 mesh for all 5 elements.

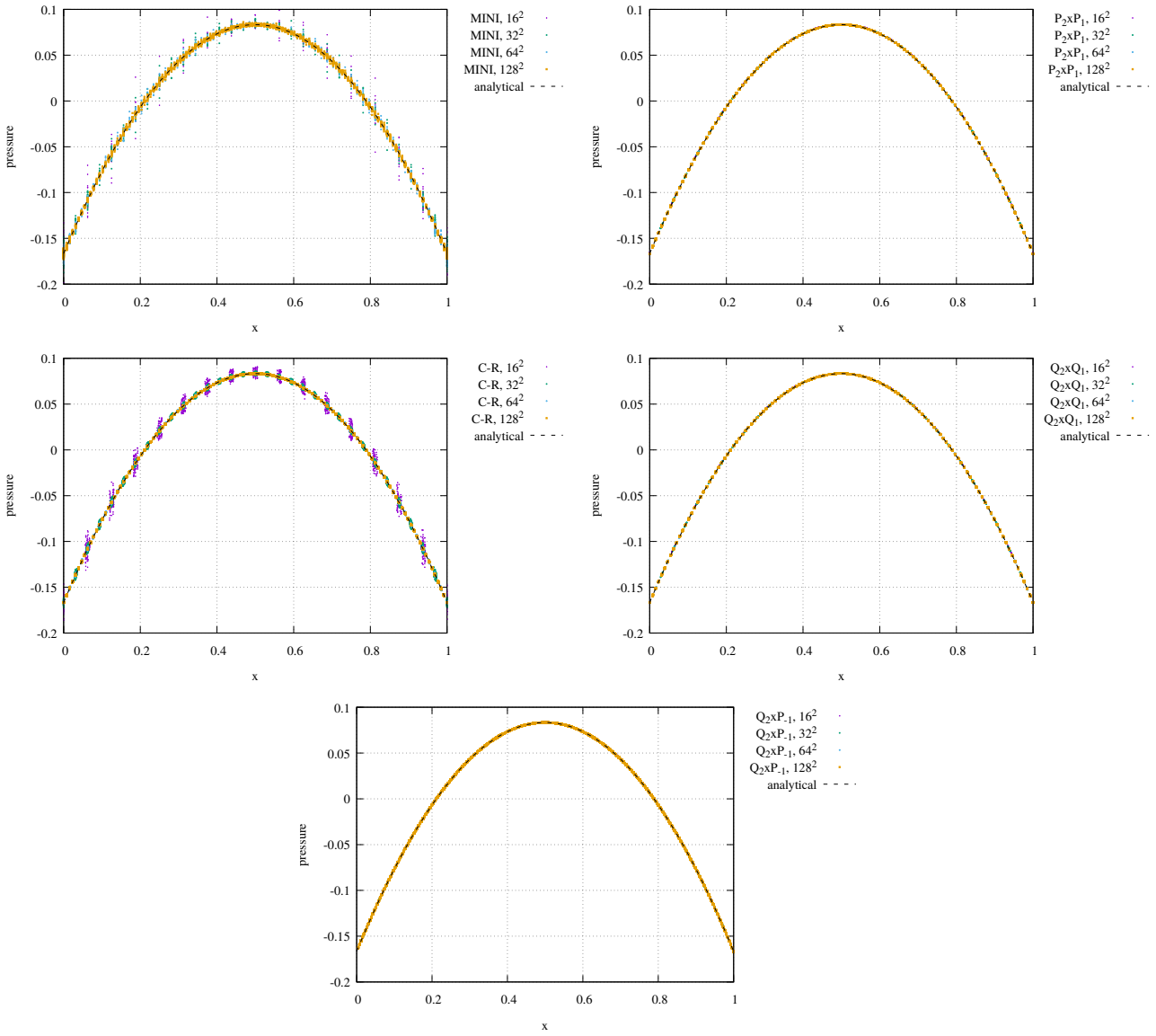
Influence of mesh quality: A random perturbation is added to the interior nodes.



Mesh and element areas for quadrilaterals and triangles on a 32x32 cell mesh.

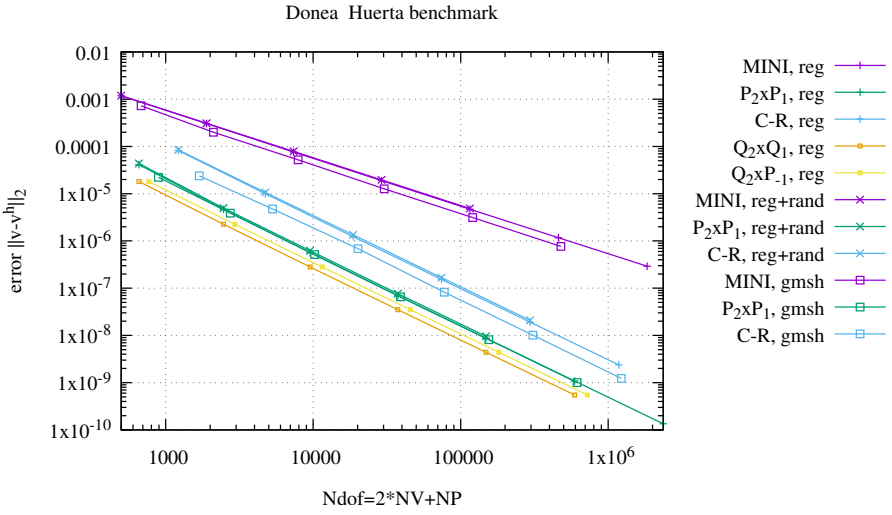


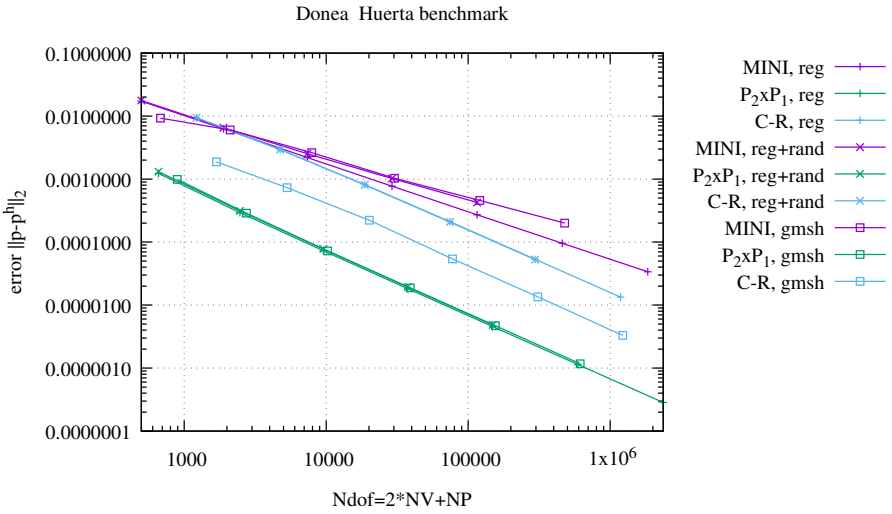
Velocity L_2 error (top row), pressure L_2 error (middle row) and root mean square velocity (bottom row) as a function of the element size h (left column) and the total number of degrees of freedom (right column).



Pressure field as a function of the x-coordinate on a 16×16, 32×32 and 64×64 mesh for all 5 elements.

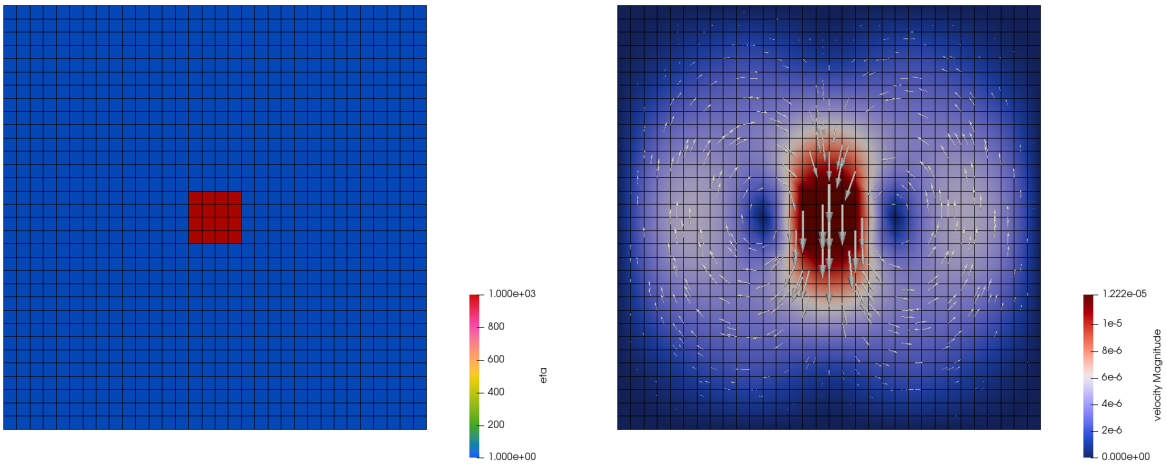
Comparison of triangular elements only on 3 types of mesh: regular, regular+random perturbation and gmsh obtained.





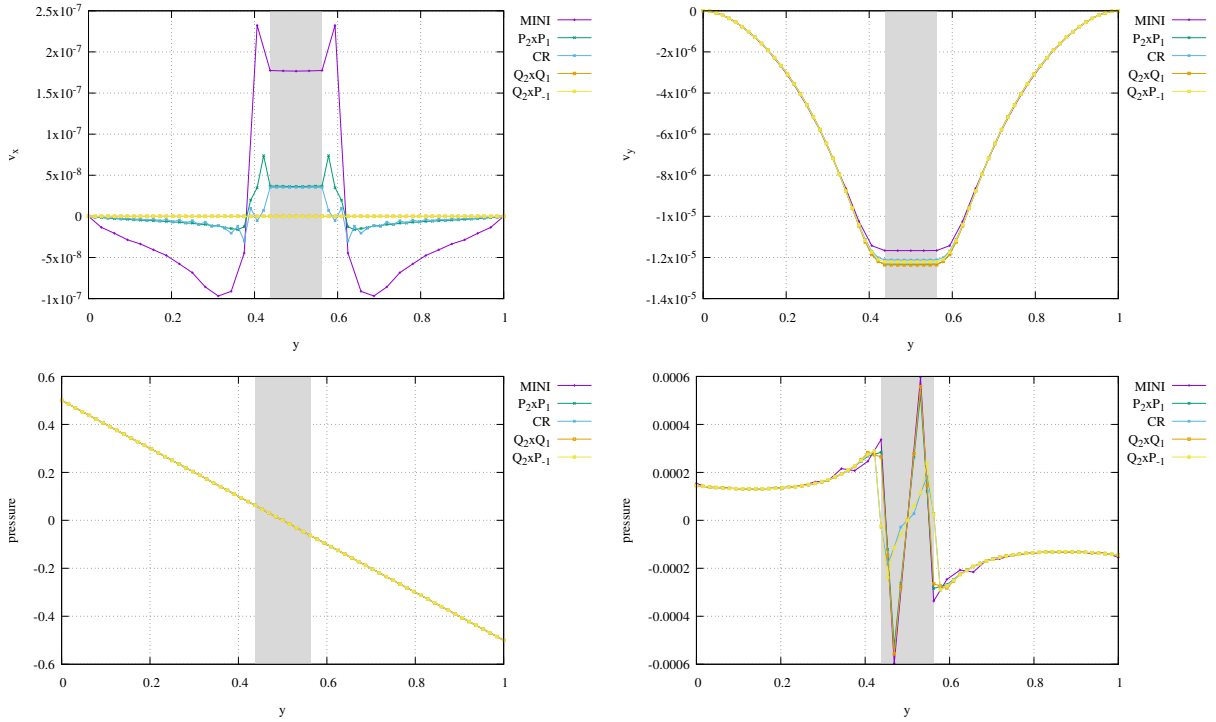
Sinking block

The setup is described in Section **??**. No-slip boundary conditions case. A sinker of size 0.125×0.125 is placed in the middle of the domain. It has density $\rho = 1.01$ while the surrounding fluid has density $\rho = 1$. Their respective viscosity is $\eta = 10^3$ and η_1. Gravity is vertical pointing downwards with $|\vec{g}| = 1$.

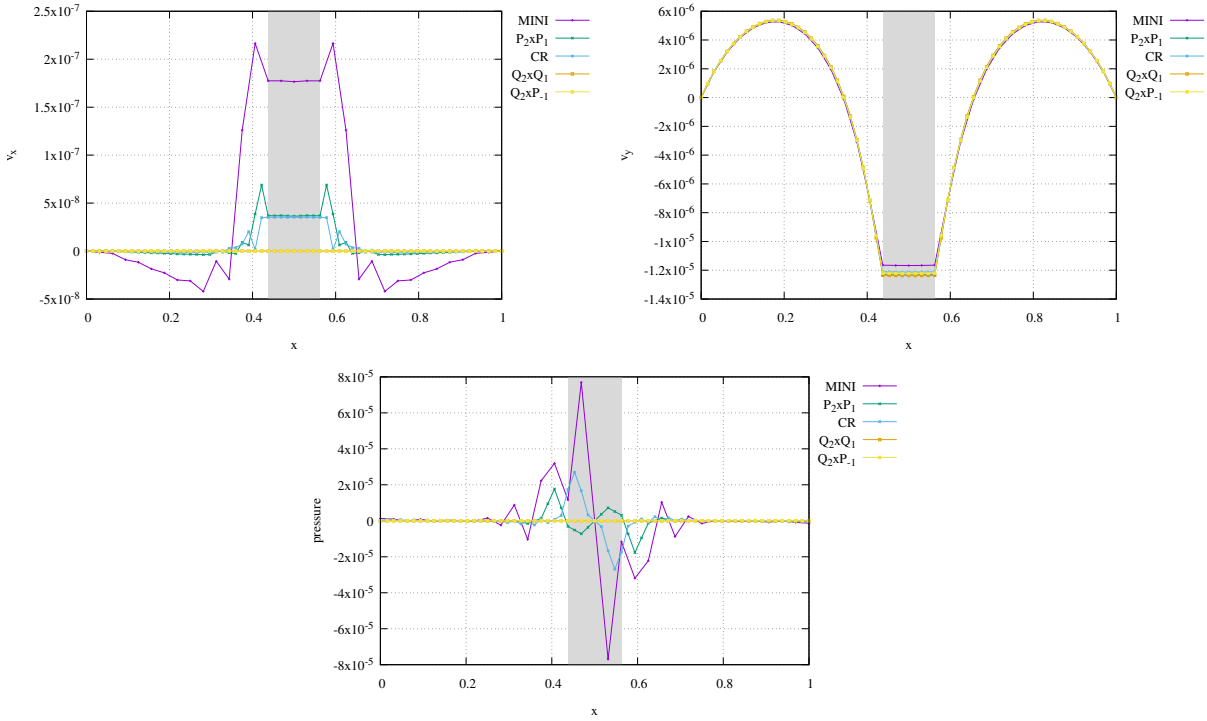


32×32 grid, $Q_2 \times P_{-1}$ element.

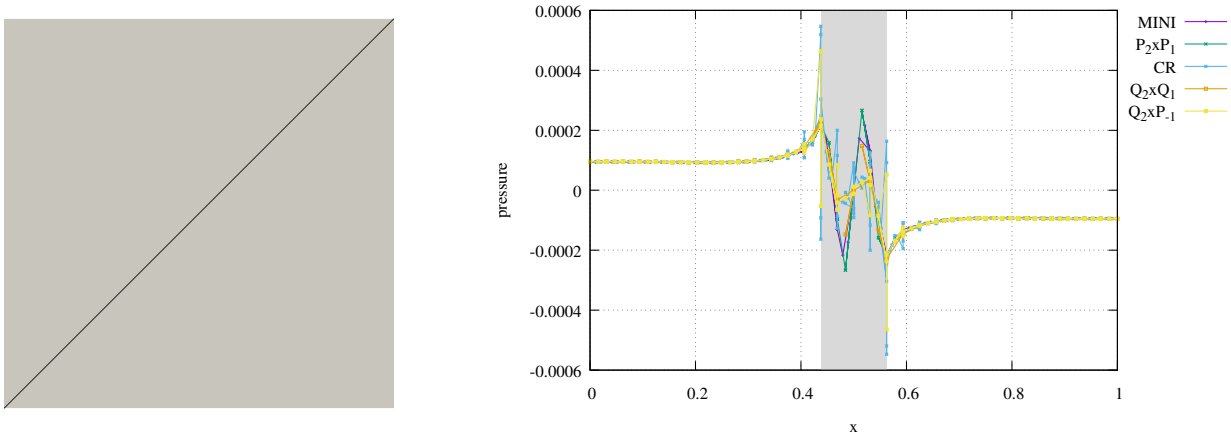
We run this experiment on a 32×32 cell mesh (these cells are then cut in half in the case of triangular elements). Velocity and pressure are measured on two profiles, one vertical and one horizontal passing through the middle of the domain.



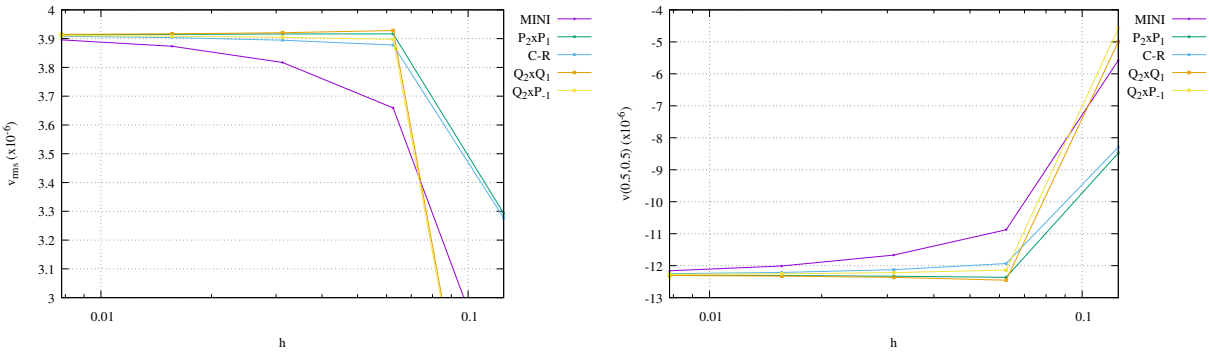
Vertical profiles. Pressure is projected onto the velocity nodes.



Horizontal profiles. Pressure is projected onto the velocity nodes.



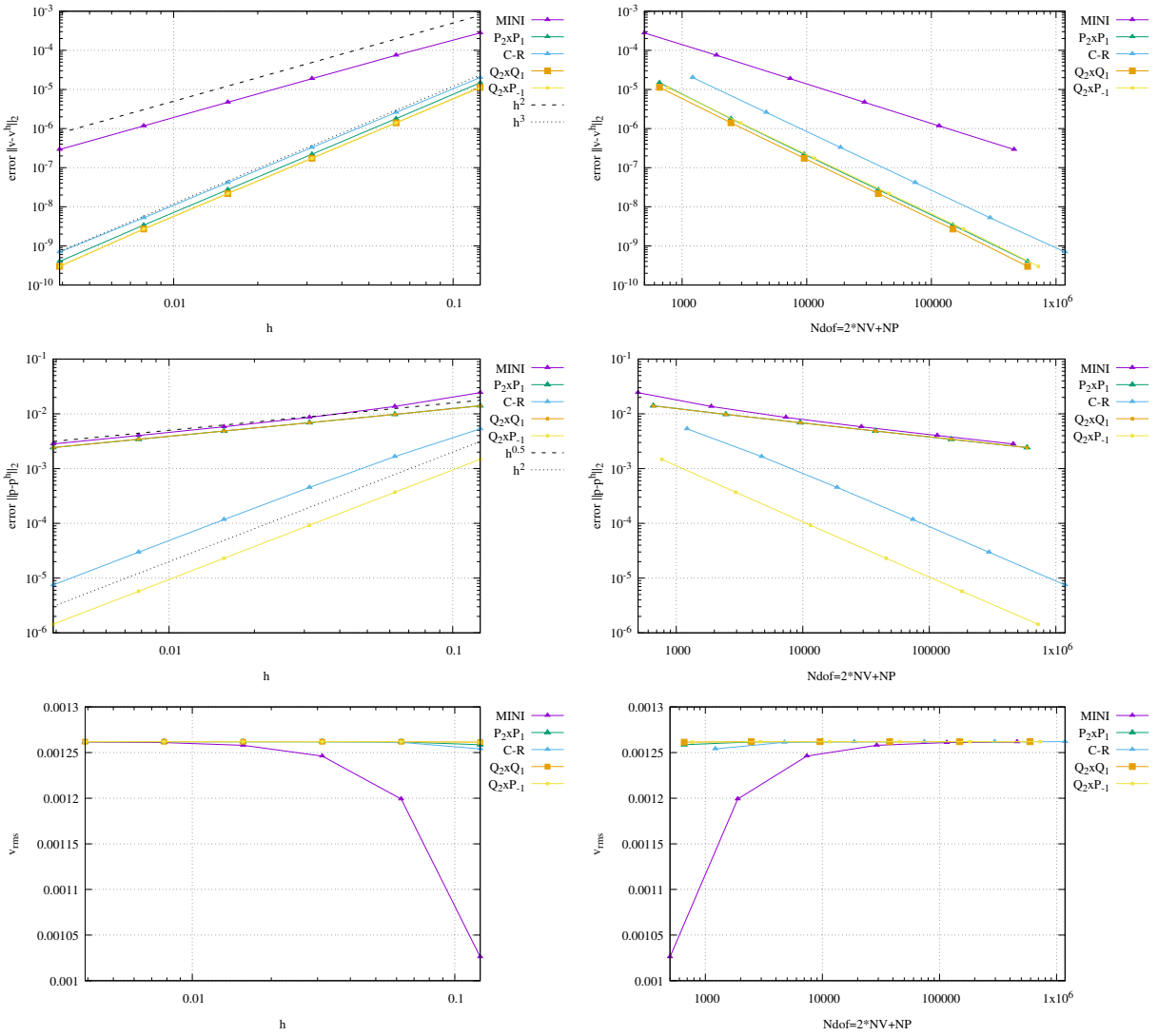
Dynamic pressure $p - p_{lith}$ field on SW-NE diagonal



Left: root mean square velocity; Right: vertical velocity in the middle of the domain/sinker.

SolCx

The setup is described in Section **??**. The viscosity is discontinuous along the vertical line $x = 1/2$ so discontinuous pressure elements perform better:

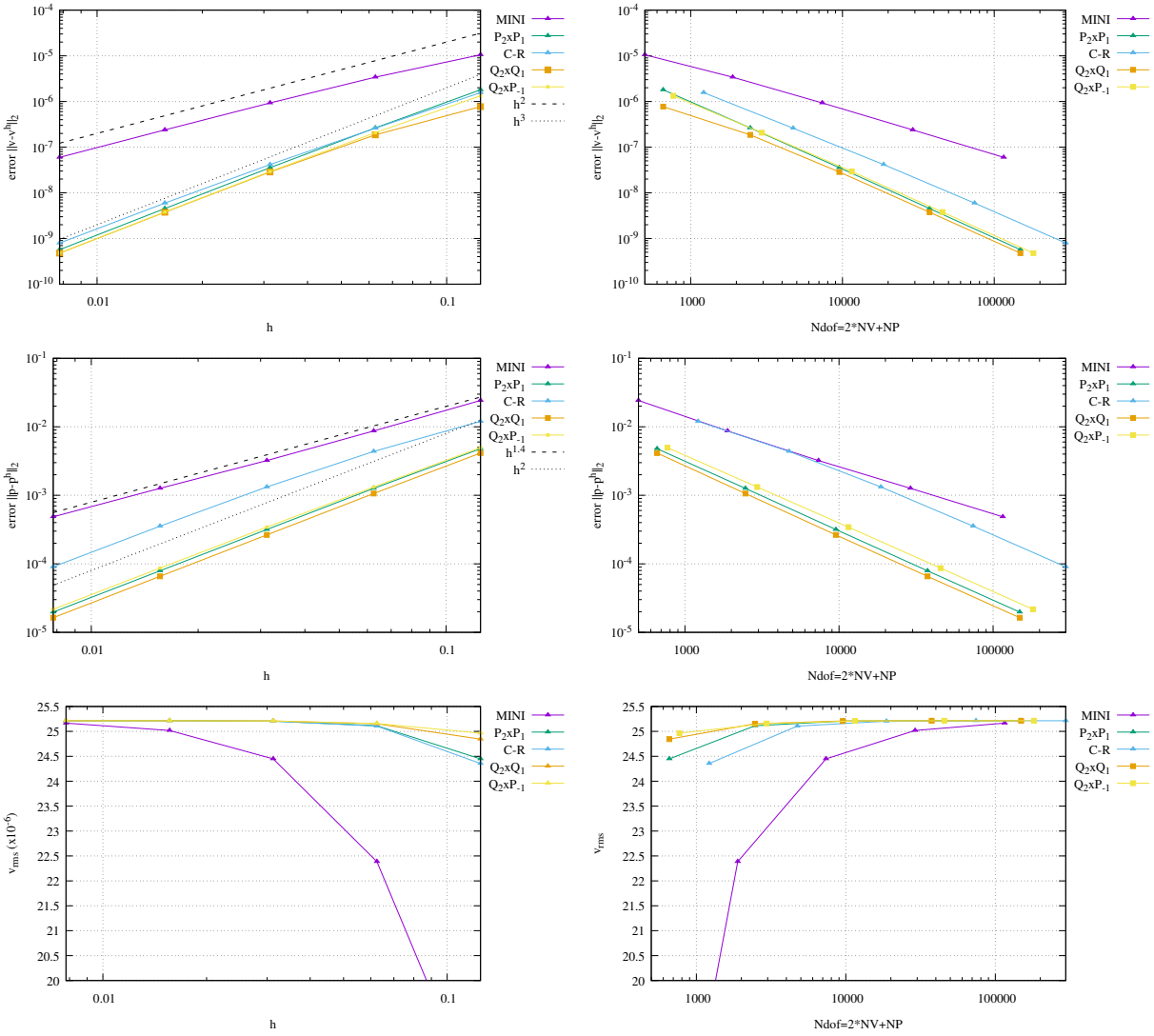


Velocity L_2 error (top row), pressure L_2 error (middle row) and vrms (bottom row) as a function of the element size h (left column) and the total number of dofs (right column).

Thieulot & Bangerth (2021) [328] report $\mathcal{O}(h^3)$ for velocity for both $Q_2 \times Q_1$ and $Q_2 \times P_1$ and $\mathcal{O}(h^{0.5})$ for $Q_2 \times Q_1$ and $\mathcal{O}(h^2)$ for $Q_2 \times P_{-1}$ for pressure. Our measurements are similar here.

SolKz

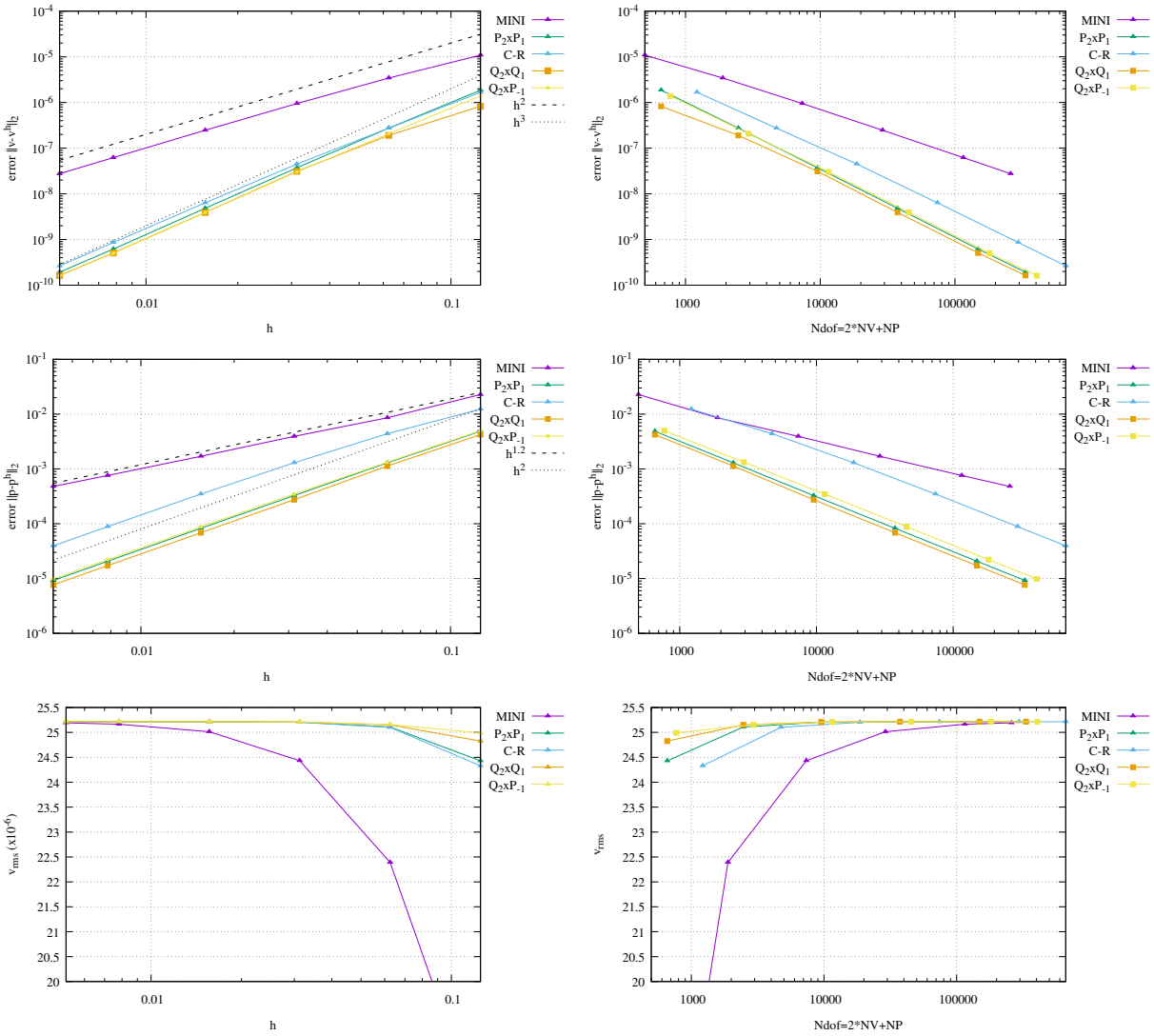
The setup is described in Section **??**.



Velocity L_2 error (top row), pressure L_2 error (middle row) and vrms (bottom row) as a function of the element size h (left column) and the total number of dofs (right column).

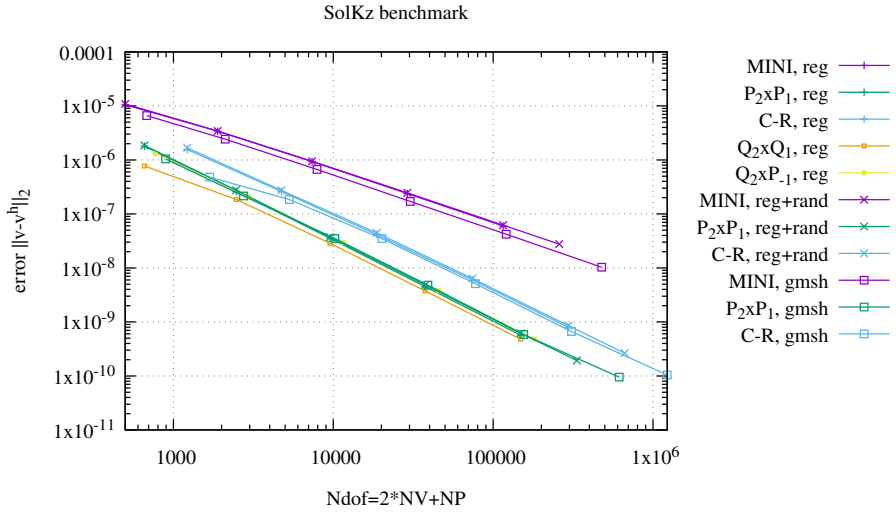
Note the 1.4 convergence of pressure for MINI

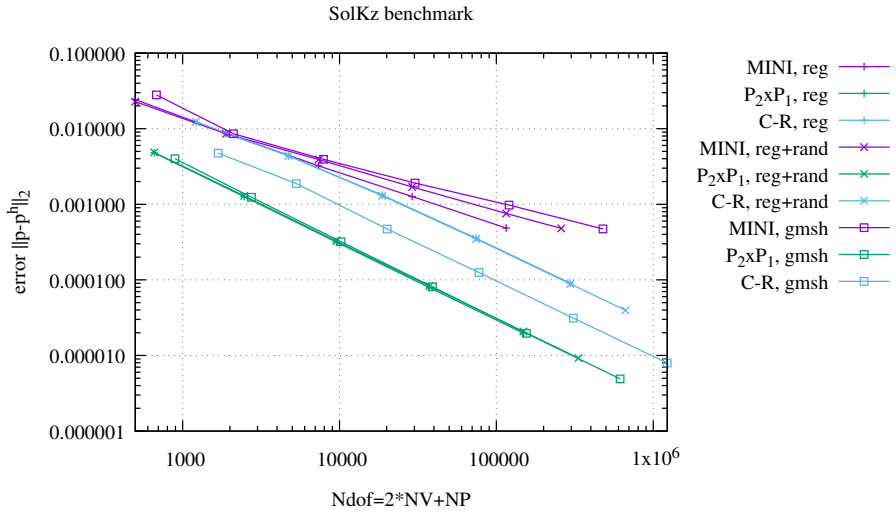
Influence of mesh quality: A random perturbation is added to the interior nodes.



Velocity L_2 error (top row), pressure L_2 error (middle row) and root mean square velocity (bottom row) as a function of the element size h (left column) and the total number of degrees of freedom (right column).

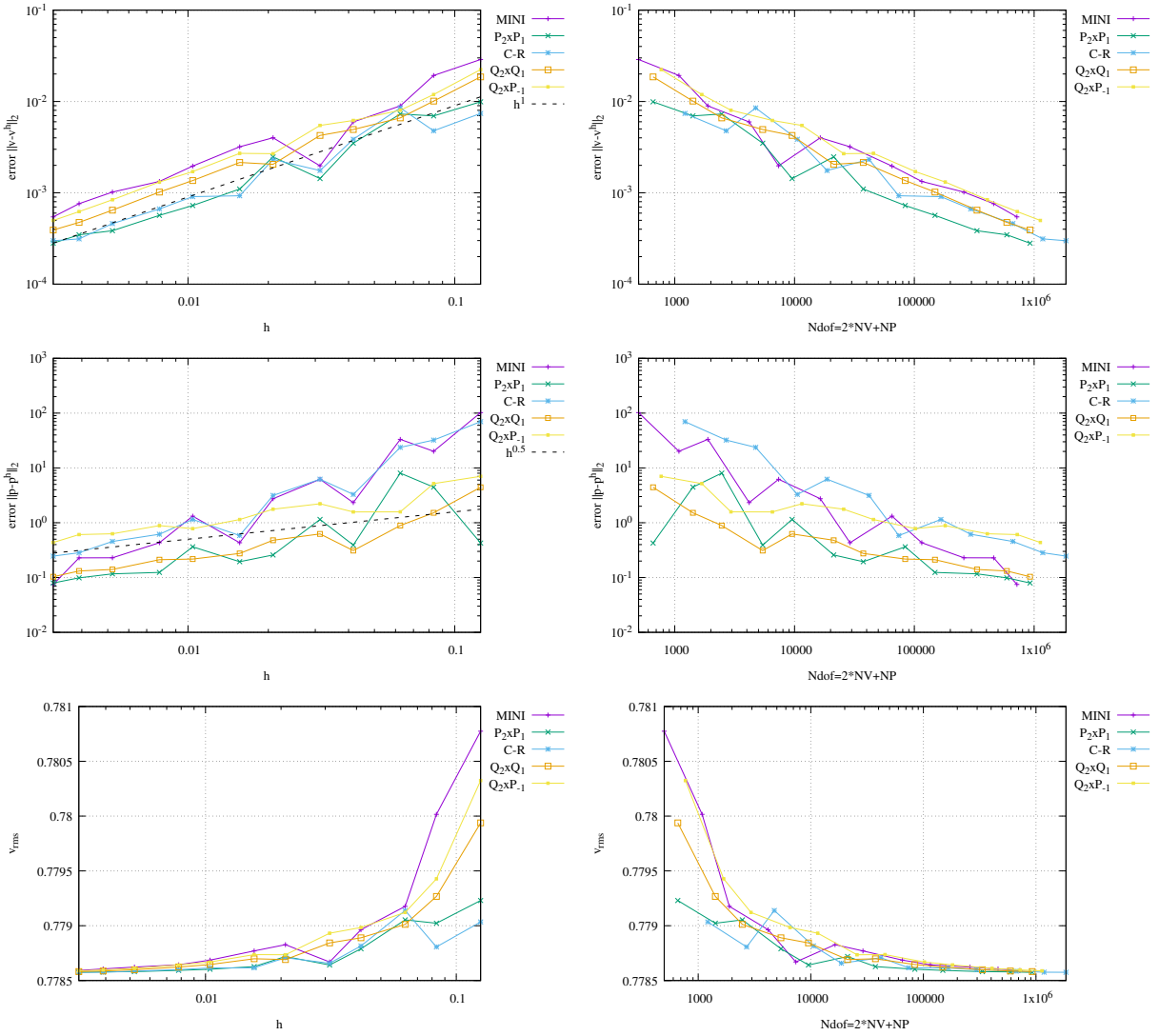
Comparison of triangular elements only on 3 types of mesh: regular, regular+random perturbation and gmsh obtained.





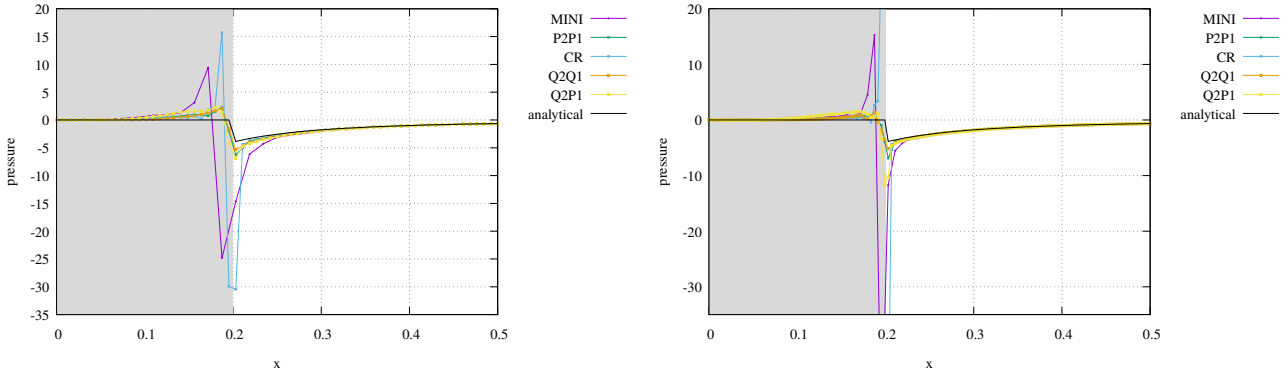
SolVi

The setup is described in Section **??**.



Velocity L_2 error (top row), pressure L_2 error (middle row) and vrms (bottom row) as a function of the element size h (left column) and the total number of dofs (right column).

Thieulot & Bangerth (2021) [328] report $\mathcal{O}(h)$ convergence for velocity and $\mathcal{O}(h^{0.5})$ for pressure across all four elements that they tested. Our measurements are similar here.



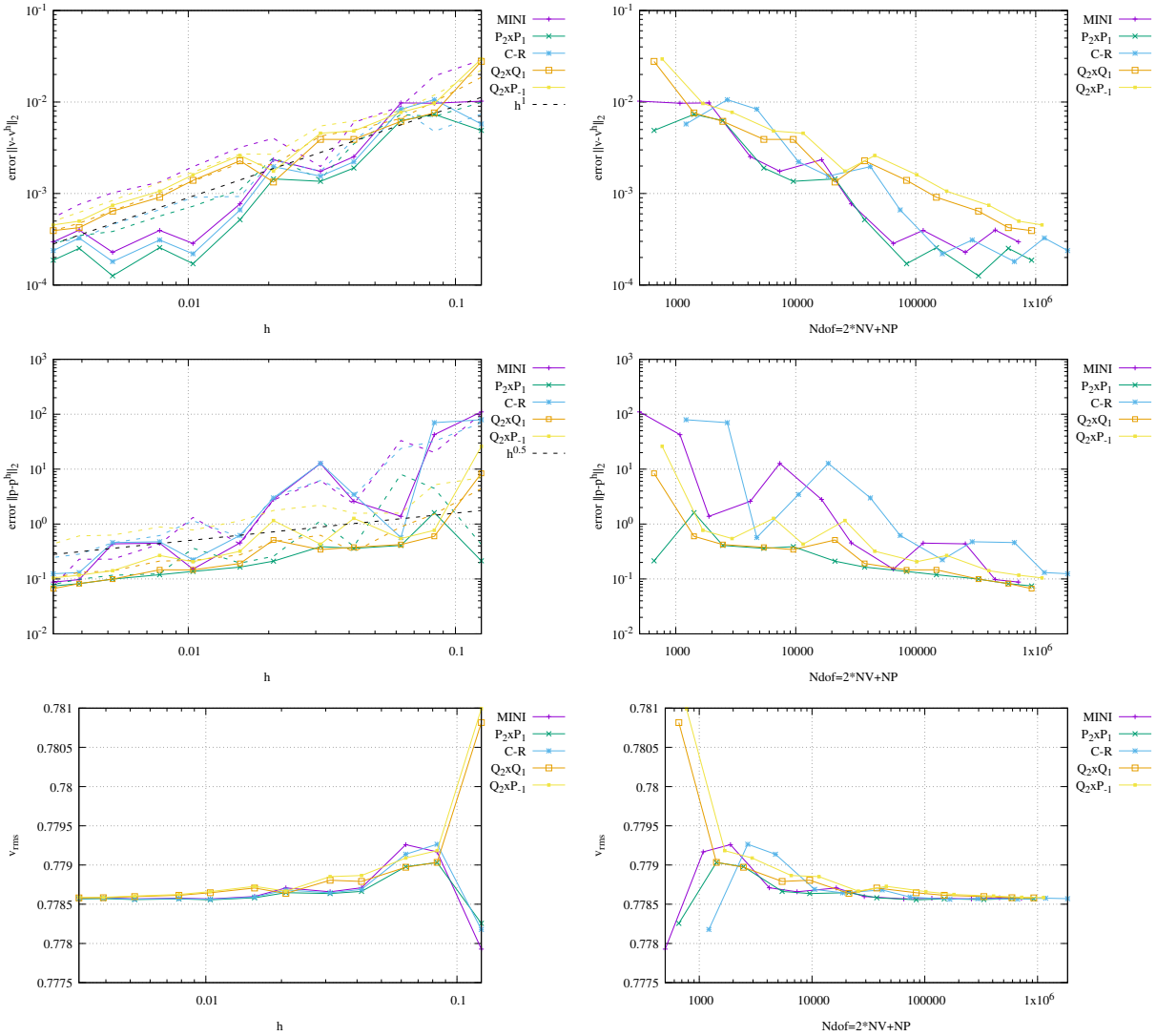
Pressure at the bottom of the domain. Grey area corresponds to the high viscosity disc. Left: 64x64 cells; right: 128x128 cells.

SolVi - A quick look at how viscosity is prescribed

The interface between the inclusion and the matrix is never aligned with element edges so that inside the elements cut by this interface some quadrature points are assigned $\eta_q = \eta_i = 10^3$ and other quadrature points are assigned $\eta_q = \eta_m = 1$.

It is well known that such an approach often yields less accurate results than when the viscosity in the element is either constant or linear (see current work by Gerry and also ASPECT manual).

In order to avoid the discussion of which averaging is best suited, I have simply here explored the possibility of assigning a constant viscosity inside each element based on whether or not its barycenter is inside or outside the inclusion.

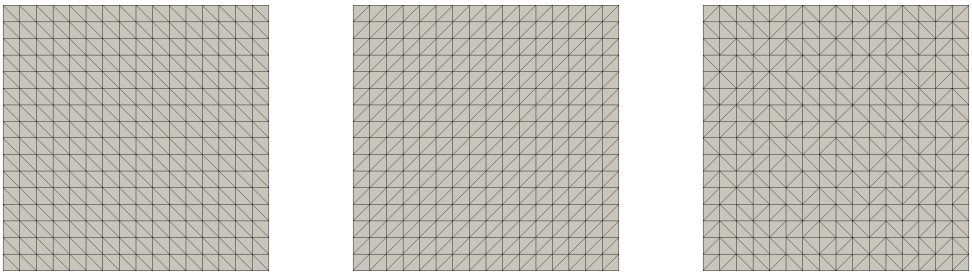


Velocity L_2 error (top row), pressure L_2 error (middle row) and vrms (bottom row) as a function of the element size h (left column) and the total number of dofs (right column). Dashed lines correspond to the reference case (see previous section) while continuous lines correspond to elemental viscosities.

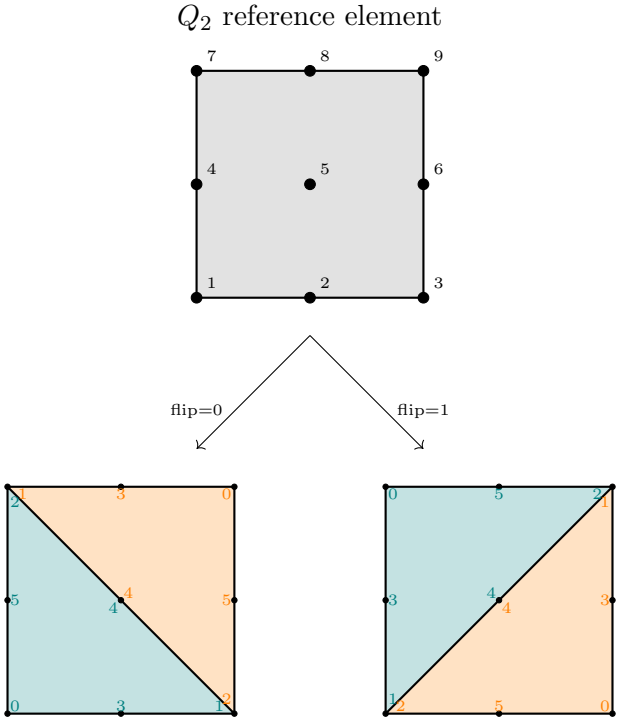
We find that results seem to be more accurate but the convergence rates for both velocity and pressure are not significantly improved.

SolVi - focus on triangular elements and how quadrilaterals are split

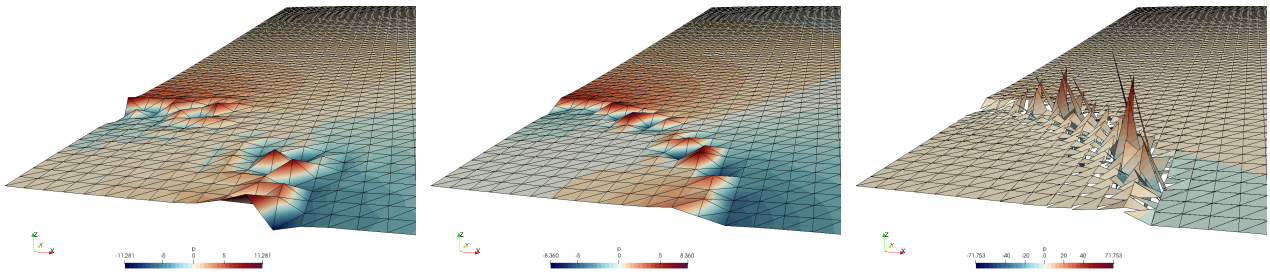
The mesh consists of $nelx \times nely$ cells. When triangular elements are used, each quadrilateral cell can be split either along one diagonal or the other. There are then three options in the code: if `tridiag`=0, then all cells are split along the NW-SE diagonal, if `tridiag`=1, then all cells are split along the SW-NE diagonal, and if `tridiag`=2, then cells are split randomly across one or the other diagonal, as shown in the following figure:



From left to right: tridiag=0,1,2. 16x16 cells.



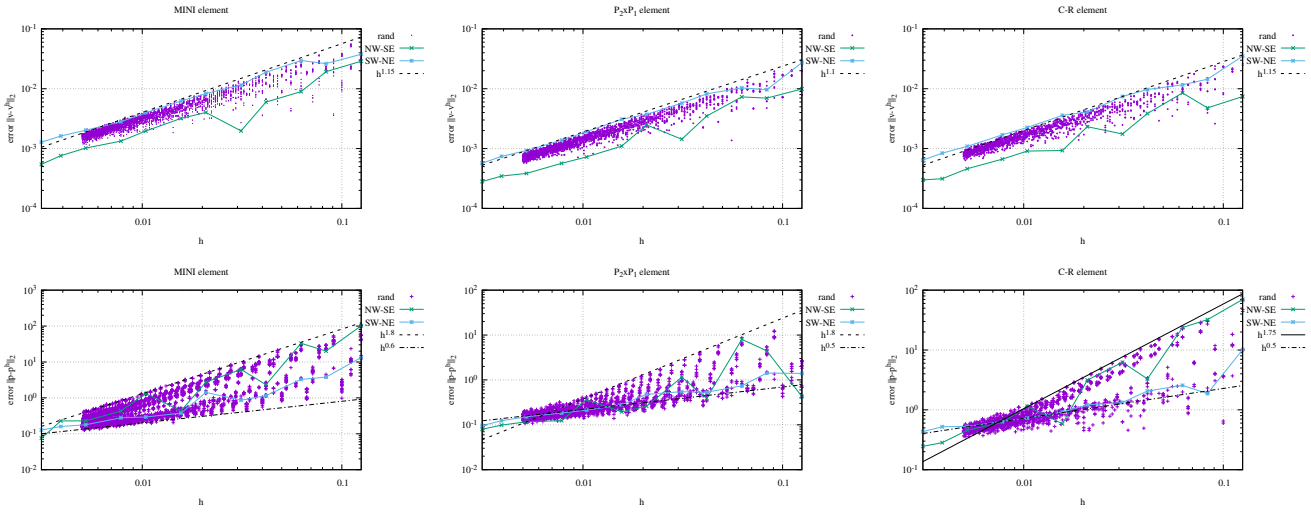
How the biquadratic quadrilateral is split into two quadratic triangles. Numbers correspond to internal numbering.



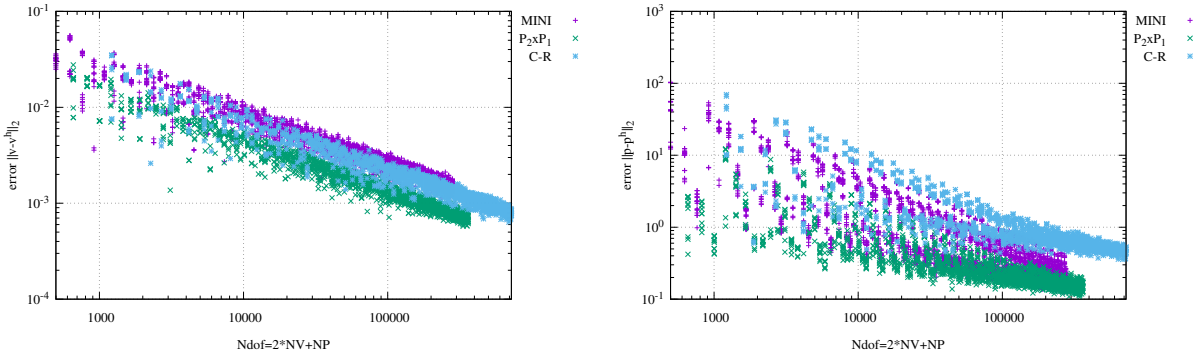
From left to right: pressure field obtained with MINI, P2P1 and CR elements on 64x64 mesh with SE-NW splitting.

The following plot are similar to those before but each element/resolution combination is (*still*) being run 250 times so as to generate 250 different meshes with different random cell splits. The important conclusion are that a) the maximum of the error across the 250 meshes converges faster for the pressure than for velocity up to about 100×100 resolutions b) it then takes converges at a lower rate, i.e. $\propto h^{0.5}$.

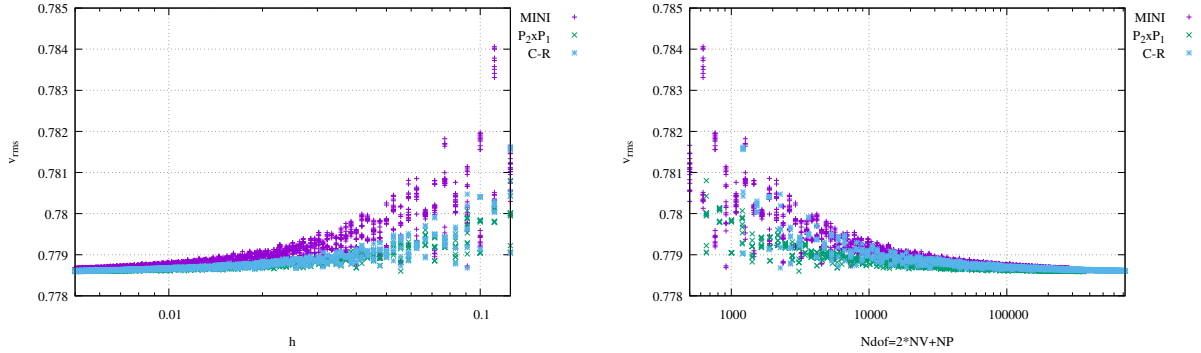
Resolutions on the following plots range from 8x8 to 200x200. I will run up to 300x300 during the xmas break.



Velocity (top row) and pressure (bottom row) error as a function of the element size for the MINI (left), $P_2 \times P_1$ (middle) and C-R element (right).



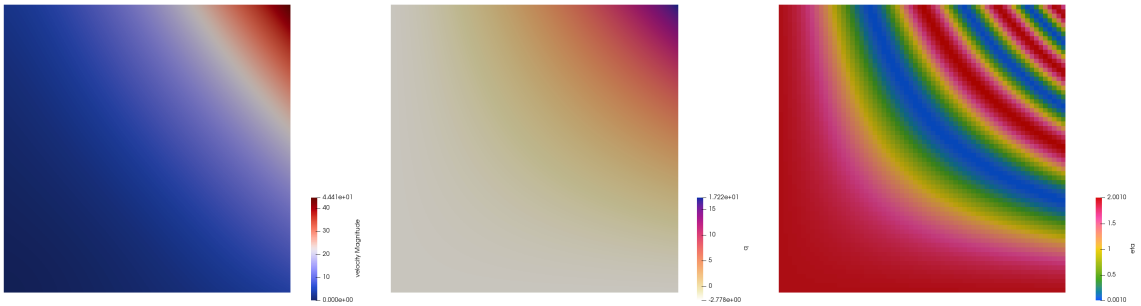
Velocity L_2 error (left), pressure L_2 error (right) as a function of the total number of degrees of freedom.



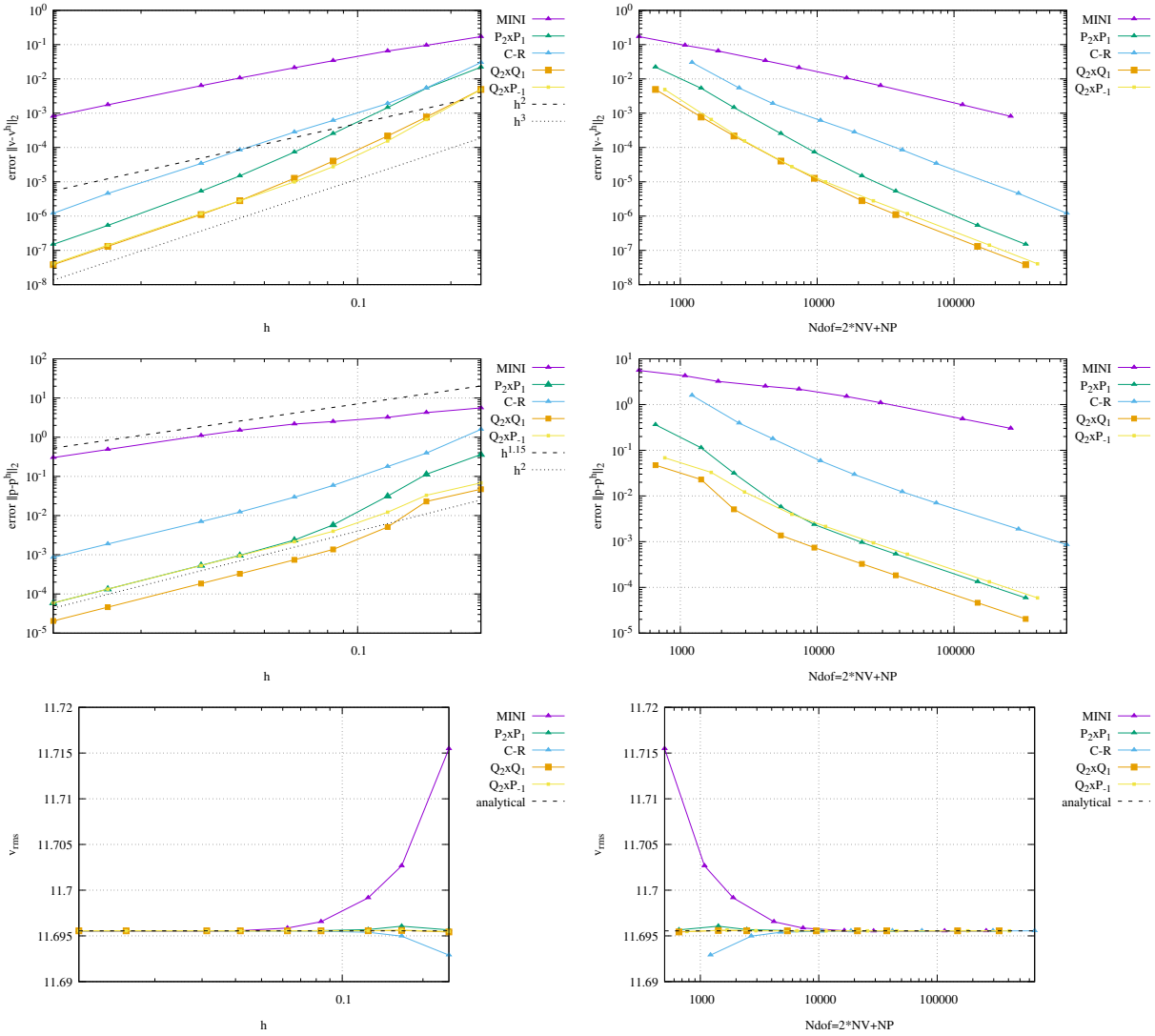
vrms as a function of the element size h (left) and the total number of dofs (right).

SolVg (Viscosity grooves)

This manufactured solution is presented in Section **??** and is also implemented in ASPECT . Domain is 2x2. The ϵ parameter controls the viscosity amplitude and is set to 0.001.



Velocity, pressure and (elemental) viscosity field for $\epsilon = 0.001$ on a 64x64 mesh



Velocity L_2 error (top row), pressure L_2 error (middle row) and vrms (bottom row) as a function of the element size h (left column) and the total number of dofs (right column).

Assuming that $\|\boldsymbol{u} - \boldsymbol{u}_h\|_{L_2} = \mathcal{O}(h^{\alpha_v})$ and $\|p - p_h\|_{L_2} = \mathcal{O}(h^{\alpha_p})$ I have gathered the exponent values in the following table.

	MINI				$P_2 \times P_1$				C-R				$Q_2 \times Q_1$				$Q_2 \times P_1$			
experiment	α_v	r	α_p	r	α_v	r	α_p	r	α_v	r	α_p	r	α_v	r	α_p	r	α_v	r	α_p	r
D & H	2	4	1.5	3	3	2	2	1	3	3	2	2	3	1	2	1	3	1	2	1
D & H (rand)	2	5	1.3	3	3	2	2	1	3	3	2	2	3	1	2	1	3	1	2	1
SolCx	2	4	0.5	4	3	2	0.5	3	3	3	2	2	3	1	0.5	3	3	1	2	1
SolKz	2	4	1.4	5	3	2	2	2	3	3	2	4	3	1	2	1	3	1	2	3
SolKz (rand)	2	4	1.2	5	3	2	2	2	3	3	2	4	3	1	2	1	3	1	2	3
SolVi	1	-	0.5	3	1	-	0.5	1	1	-	0.5	4	1	-	0.5	2	1	-	0.5	5
SolVi (multi)																				
SolVg	2	4	1.15	4	3	2	2	2	3	3	2	3	3	1	2	1	3	1	2	2
total	-	?	-	?	-	?	-	?	-	?	-	?	-	?	-	?	-	?	-	?

The r value corresponds to the **r**anking. For instance if Q2Q1 is the most accurate for velocity on D & H then it gets a 1, then P2P1 gets a 2 because it was the second best accurate element, etc ... In the end I would sum all the r values per column (replacing the '?') and the smaller the value the better the element.

Note that SolVi measurements will be updated when all the runs have completed.

Preliminary Conclusions

- the MINI element is very appealing (a single bubble added to P_1) and cheap. However it is also the least accurate element, even looking at the errors as function of number of dofs and not h
- P2P1 always more accurate than CR except for SolCx where disc pressure elements are way more accurate.
- P2P1 about as accurate as Q2Q1 in all benchmarks

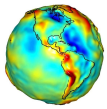
Open questions

- at the moment I build the entire Stokes matrix in a sparse format and send it to the default python solver. This means that no iterative solver is used (inner or outer) so that the discussion on the nb of FGMRES iterations of our paper cannot be carried out here.
- Right now viscosity is computed on quadrature point. No averaging. It probably only really matters for SolVi and SolVg (and may be a bit for SolKz). Should I implement elemental averaging and try?
- are there other elements you wish to add to this list? $P_3 \times P_2$?
- I plot now the velocity and pressure errors as a function of the total number of dofs. Does this make sense? or should I plot the velocity errors as a fct of the velocity dofs and the pressure errors as a fct of pressure dofs?
- any thoughts about my ranking system ?like it ? hate it ?
- am I right to think that $Q_2 \times P_{-1}$ in ASPECT is of the unmapped type ? (Boffi and Gastaldi [26] (2002))
- should I repeat the same statistical study with 250 different triangular meshes per element for SolVg ?

note to myself: check and probably remove stone 46, 47 ?
This stone also supports grid obtained with quilt, see stone 117.

Stone 113

Gravity of a const. density tetrahedron & hexahedron following Werner & Scheeres (1997)



P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_113`

Last revision: Sept. 26th, 2024.

Note that there are two python files in the folder: stone_tetrahedron.py and stone_hexahedron.py.

This STONE based on the article by Werner & Scheeres (1997) [351] in which the gravity fields generated by a polyhedron *of constant density*.

Before we dive in the calculations it is worth highlighting a few excerpts from the paper. First, the authors present three methods to compute the gravity field of an irregular-shaped body:

1. harmonic expansion. They write:

> Harmonic expansions have several drawbacks. The first is that the harmonic expansion is always an approximation to a gravity field due to the finite truncation of the series expansion. The truncation error grows when evaluating the gravity field close to the model's radius of convergence. Additional terms are necessary in the expansion to maintain a given accuracy. The second major drawback is that the same form of the exterior harmonic expansion is no longer guaranteed to converge inside the circumscribing sphere, and indeed often diverges. Another drawback is that the harmonic expansion yields no information about whether a field point is outside or inside the body.

2. mass concentrations ('mascons').

> A second, commonly used approach for evaluating asteroid gravitation is to fill the body with point masses ('mascons' - mass concentrations) on an evenly spaced grid (Geissler et al., 1995). [...] for a given computational effort, the mascon approach is less accurate than a harmonic approach (in its region of convergence).

3. polyhedron. This is the method presented in the paper.

> we investigate a third approach, which is to model the asteroid as a constant-density polyhedron. The polyhedron can have concavities in its surface (e.g. craters), overhangs, interior

> voids (caves), and even holes all the way through (torus). There is no special penalty for including small details, i.e. the entire body does not have to be modeled at a uniformly high resolution. [...] The polyhedral approach remedies several drawbacks outlined above. The gravity field is exact for the given shape and density. polyhedron gravitation is a valid and exact solution up to the surface of the body. There is no region of divergence. [...] Another way our work differs is that our ultimate formulas are expressed intrinsi- cally using vectors and distances. Other papers are cluttered with special coordinate systems and angles.

The authors explain what a polyhedron is in this context:

> By polyhedron we mean a three-dimensional solid body whose surface consists of planar faces meeting along straight edges or at isolated points called vertices. Exactly two faces meet at each edge. Three or more edges and a like number of faces meet at each vertex. Note: the vertex coordinates of a polyhedron alone are insufficient to describe it. The connective topology must also be described – edges connect which vertex pairs and bound which face pairs.

The volume integrals are transformed via the Gauss divergence theorem, into surface integrals, e.g.:

$$U = \frac{1}{2}\mathcal{G}\rho_0 \iint_S \vec{n} \cdot \frac{\vec{r}}{r}\, dS$$

Ultimately this integral will involve face integrals and edges integrals. The paper is somewhat complex but well written. There is however no point re-typing all the derivations and I present in the next section the finalised equations of their Section 2.6.

Tetrahedron: the theory

In what follows $e = 1, 2, ...6$ stands for an edge while $f = 1, 2, 3, 4$ stands for a face. Each polyhedron face has an outward-pointing face normal vector $\vec{n}_f$ and a face dyad $\boldsymbol{F}_f = \vec{n}_f \vec{n}_f$. Each edge of each face has an outward-pointing edge normal vector $\vec{n}_{e,f}$ perpendicular to both $\vec{n}_f$ and the edge. For the edge connecting vertices 1 and 2 shared by faces A and B, the edge dyad is $\boldsymbol{E}_{12} = \vec{n}_A \vec{n}_{12,A} + \vec{n}_B \vec{n}_{21,B}$.

Let $\vec{r}_i$ represent the vector from the variable field-point M at location $\vec{r}_M$ to polyhedron vertex P_i, and let $r_i = ||\vec{r}_i||$ be its length.

For the polyhedron edge connecting vertices i and j of length l_{ij}, the dimensionless per-edge factor L_e is

$$L_e = L_{ij} = \int_e \frac{1}{r} ds = \int_{P_i}^{P_j} \frac{1}{r} ds = \ln \frac{r_i + r_j + l_{ij}}{r_i + r_j - l_{ij}}$$

For a triangular face f bounded by vertices i, j, k the dimensionless per-face factor ω_f is

$$\omega_f = \iint_{triangle} \frac{\Delta z}{r^3} dS = 2 \arctan \frac{\vec{r}_i \cdot (\vec{r}_j \times \vec{r}_k)}{r_i r_j r_k + r_i(\vec{r}_j \cdot \vec{r}_k) + r_j(\vec{r}_k \cdot \vec{r}_i) + r_k(\vec{r}_i \cdot \vec{r}_j)}$$

The gravity vector $\vec{g}$ generated by the mass inside the tetrahedron at point M is given by

$$-\vec{g} = \vec{\nabla} U = \underbrace{-\mathcal{G}\rho_0 \sum_e L_e \boldsymbol{E}_e \cdot \vec{r}_e}_{\vec{g}_e} + \underbrace{\mathcal{G}\rho_0 \sum_f \omega_f \boldsymbol{F}_f \cdot \vec{r}_f}_{\vec{g}_f} = -\vec{g}_e + \vec{g}_f$$

with

$$\begin{aligned} \vec{g}_e &= \mathcal{G}\rho_0 \sum_e L_e \boldsymbol{E}_e \cdot \vec{r}_e \\ &= \mathcal{G}\rho_0 \left(L_{12}\boldsymbol{E}_{12} \cdot \vec{r}_{12} + L_{13}\boldsymbol{E}_{23} \cdot \vec{r}_{13} + L_{14}\boldsymbol{E}_{34} \cdot \vec{r}_{14} \right. \\ &\quad \left. + L_{23}\boldsymbol{E}_{23} \cdot \vec{r}_{23} + L_{24}\boldsymbol{E}_{24} \cdot \vec{r}_{24} + L_{34}\boldsymbol{E}_{34} \cdot \vec{r}_{34} \right) \end{aligned} \tag{113.1}$$

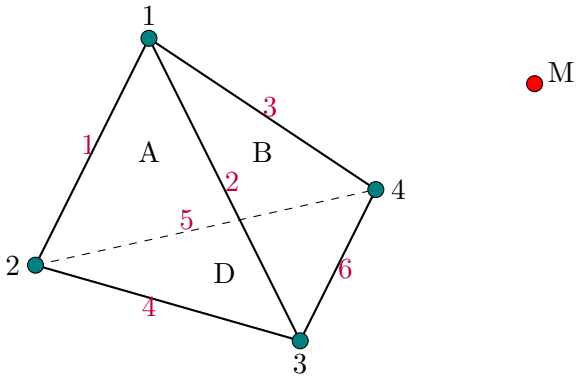
$$\begin{aligned} \vec{g}_f &= \mathcal{G}\rho_0 \sum_f \omega_f \boldsymbol{F}_f \cdot \vec{r}_f \\ &= \mathcal{G}\rho_0 \left(\omega_A \boldsymbol{F}_A \cdot \vec{r}_A + \omega_B \boldsymbol{F}_B \cdot \vec{r}_B + \omega_C \boldsymbol{F}_C \cdot \vec{r}_C + \omega_D \boldsymbol{F}_D \cdot \vec{r}_D \right) \end{aligned} \tag{113.2}$$

and the gravity potential and tensor are then given by

$$U = \frac{1}{2}\mathcal{G}\rho_0 \sum_e L_e \, \vec{r}_e \cdot \boldsymbol{E}_e \cdot \vec{r}_e - \frac{1}{2}\mathcal{G}\rho_0 \sum_f \omega_f \, \vec{r}_f \cdot \boldsymbol{F}_f \cdot \vec{r}_f \tag{113.3}$$

$$\boldsymbol{T} = \vec{\nabla}\vec{\nabla}U = -\vec{\nabla}\vec{g} = \mathcal{G}\rho_0 \sum_e L_e \cdot \boldsymbol{E}_e - \mathcal{G}\rho_0 \sum_f \omega_f \, \boldsymbol{F}_f \tag{113.4}$$

Let us consider a generic tetrahedron:



Face C is hidden in the back.

When looking at a face from the outside towards the tetrahedron the numbering of vertices is counter-clockwise.

face	vertices i,j,k	edge #1	edge #2	edge #3
A	1,2,3	12(1)	23(4)	31(2)
B	1,3,4	13(2)	34(6)	41(3)
C	1,4,2	14(3)	42(5)	21(1)
D	2,4,3	24(5)	43(6)	32(4)

edge	vertices	belongs to
1	12	A,C
2	13	A,B
3	14	B,C
4	23	A,D
5	24	C,D
6	34	B,D

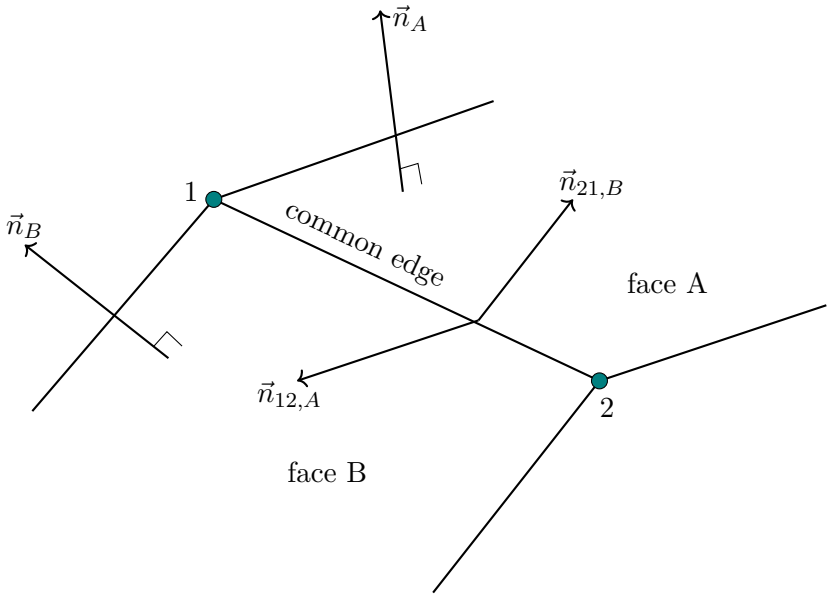


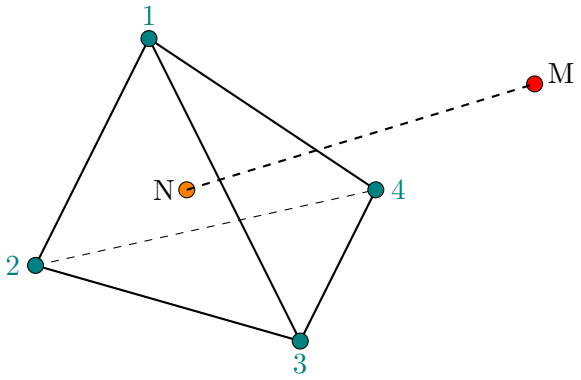
Figure made after Fig. 7 from Werner and Scheeres [351].

$\vec{n}_{1,A} = \vec{n}_{12,A} = (n_x, n_y)$ is such that it is perpendicular to $\vec{n}_A$ (it lies in the face A plane), i.e. $\vec{n}_{1,A} \cdot \vec{n}_A = 0$ and to edge #1, i.e. $\vec{n}_{1,A} \cdot \vec{l}_1 = 0$. The cross product of $\vec{n}_A$ and $\vec{l}_1$ is by definition a vector perpendicular to both. We also need to make sure it is pointing outwards. We then take the scalar product of a vector joining the center of the face and the middle of the edge under consideration. If it is negative then the normal is pointing towards the center of the face so it needs to be flipped.

Tetrahedron: Point mass approach

Let us denote by V the volume of the tetrahedron which can easily be computed:

$$V = \frac{1}{6} \left| (\vec{l}_1 \times \vec{l}_2) \cdot \vec{l}_3 \right| \tag{113.5}$$



The coordinates of its center of gravity N (under the assumption that the density is constant inside it) are given by

$$x_N = \frac{1}{4}(x_1 + x_2 + x_3 + x_4) \qquad y_N = \frac{1}{4}(y_1 + y_2 + y_3 + y_4) \qquad z_N = \frac{1}{4}(z_1 + z_2 + z_3 + z_4) \tag{113.6}$$

If the tetrahedron is considered a point mass, then the gravity field and potential are given by

$$\vec{g}_{pm} = \mathcal{G} \frac{\rho_0 V}{|\overrightarrow{MN}|^3} \overrightarrow{MN} \tag{113.7}$$

$$U_{pm} = \mathcal{G} \frac{\rho_0 V}{|\overrightarrow{MN}|} \tag{113.8}$$

with

$$\overrightarrow{MN} = \begin{pmatrix} x_N - x_M \\ y_N - y_M \\ z_N - z_M \end{pmatrix}$$

Function `compute_gravity_tetrahedron_pointmass` receives coordinates of the four vertices 1, 2, 3, 4, its density and the coordinates of the measurement point.

1. compute coordinates x_N, y_N, z_N of center of gravity with Eq. (113.6) → `xN,yN,zN`
2. compute volume V of tetrahedron with Eq. (113.5)→ `Vol`
3. compute $\vec{g}_{pm}$

Tetrahedron: the algorithm

Function `compute_gravity_tetrahedron` receives coordinates of the four vertices 1, 2, 3, 4, its density ρ_0 and the coordinates of the measurement point M.

I do not like the notation $\vec{e}_{12}$ for an edge vector since in fieldstone $\vec{e}$ is always a unit vector for the coordinate system. I then rename edge vectors $\vec{l}$.

The structure of the function goes as follows:

1. compute $\vec{l}_1$, $\vec{l}_2$,etc ... and their norms → `vec_l1`, `vec_l2`, `vec_l3`, `vec_l4`, `vec_l5`, `vec_l6`

$$\vec{l}_1 = \vec{l}_{12} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix} \qquad \vec{l}_2 = \vec{l}_{13} = \begin{pmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \end{pmatrix} \qquad \text{etc...}$$

2. compute $\vec{r}_{1,2,3,4}$. → `vec_r1`, `vec_r2`, `vec_r3`, `vec_r4`

$$\vec{r}_1 = \vec{M1} = \begin{pmatrix} x_1 - x_m \\ y_1 - y_m \\ z_1 - z_m \end{pmatrix}$$

3. compute $\vec{n}_{A,B,C,D}$. → `vec_nA` , `vec_nB`, `vec_nC`, `vec_nD`

$$\begin{aligned} \vec{n}_A &= \vec{l}_{12} \times \vec{l}_{13} / |\vec{l}_{12} \times \vec{l}_{13}| = \vec{l}_1 \times \vec{l}_2 / |\vec{l}_1 \times \vec{l}_2| && (113.9) \\ \vec{n}_B &= \vec{l}_{13} \times \vec{l}_{14} / |\vec{l}_{13} \times \vec{l}_{14}| = \vec{l}_2 \times \vec{l}_3 / |\vec{l}_2 \times \vec{l}_3| && (113.10) \\ \vec{n}_C &= \vec{l}_{14} \times \vec{l}_{12} / |\vec{l}_{14} \times \vec{l}_{12}| = \vec{l}_3 \times \vec{l}_1 / |\vec{l}_3 \times \vec{l}_1| && (113.11) \\ \vec{n}_D &= \vec{l}_{24} \times \vec{l}_{23} / |\vec{l}_{24} \times \vec{l}_{23}| = \vec{l}_5 \times \vec{l}_4 / |\vec{l}_5 \times \vec{l}_4| && (113.12) \end{aligned}$$

4. compute $\boldsymbol{F}_{A,B,C,D}$. → `mat_FA` , `mat_FB`, `mat_FC`, `mat_FD`

$$\begin{aligned} \boldsymbol{F}_A &= \vec{n}_A \vec{n}_A && (113.13) \\ \boldsymbol{F}_B &= \vec{n}_B \vec{n}_B && (113.14) \\ \boldsymbol{F}_C &= \vec{n}_C \vec{n}_C && (113.15) \\ \boldsymbol{F}_D &= \vec{n}_D \vec{n}_D && (113.16) \end{aligned}$$

5. compute $\omega_{A,B,C,D}$. → `wA` , `wB`, `wC`, `wD`

$$\begin{aligned} \omega_A &= 2 \arctan \frac{\vec{r_1} \cdot (\vec{r}_2 \times \vec{r}_3)}{r_1 r_2 r_3 + r_1(\vec{r}_2 \cdot \vec{r}_3) + r_2(\vec{r}_3 \cdot \vec{r}_1) + r_3(\vec{r}_1 \cdot \vec{r}_2)} \\ \omega_B &= 2 \arctan \frac{\vec{r_1} \cdot (\vec{r}_3 \times \vec{r}_4)}{r_1 r_3 r_4 + r_1(\vec{r}_3 \cdot \vec{r}_4) + r_3(\vec{r}_4 \cdot \vec{r}_1) + r_4(\vec{r}_1 \cdot \vec{r}_3)} \\ \omega_C &= 2 \arctan \frac{\vec{r_1} \cdot (\vec{r}_4 \times \vec{r}_2)}{r_1 r_4 r_2 + r_1(\vec{r}_4 \cdot \vec{r}_2) + r_4(\vec{r}_2 \cdot \vec{r}_1) + r_2(\vec{r}_1 \cdot \vec{r}_4)} \\ \omega_D &= 2 \arctan \frac{\vec{r_2} \cdot (\vec{r}_4 \times \vec{r}_3)}{r_2 r_4 r_3 + r_2(\vec{r}_4 \cdot \vec{r}_3) + r_4(\vec{r}_3 \cdot \vec{r}_2) + r_3(\vec{r}_2 \cdot \vec{r}_4)} \end{aligned} \qquad (113.17)$$

which translates into

```
num=np.dot(vec_r1,np.cross(vec_r2,vec_r3))
denom=(r1*r2*r3+\
       r1*np.dot(vec_r2,vec_r3)+\
       r2*np.dot(vec_r3,vec_r1)+\
       r3*np.dot(vec_r1,vec_r2))
wA=2*np.arctan2(num,denom)
```

6. compute $\vec{r}_A$, $\vec{r}_B$, $\vec{r}_C$, $\vec{r}_D$. As explained in the caption of their Figure 3, "vector $\vec{r}_f$ extends from the field point M to any point in the face plane". → `vec_rA`, `vec_rB`, `vec_rC`, `vec_rD`

7. compute $\vec{g}_f$ by means of

$$\vec{g}_f = \mathcal{G}\rho_0 \sum_f \omega_f \boldsymbol{F}_f \cdot \vec{r}_f = \mathcal{G}\rho_0 \left(\omega_A \boldsymbol{F}_A \cdot \vec{r}_A + \omega_B \boldsymbol{F}_B \cdot \vec{r}_B + \omega_C \boldsymbol{F}_C \cdot \vec{r}_C + \omega_D \boldsymbol{F}_D \cdot \vec{r}_D\right)$$

8. compute $\vec{n}_{e,f}$ for each edge of each face (12 vectors!) → `vec_nA12`, `vec_nA23`, `vec_nA13`, `vec_nB13`, etc ...

9. compute $\boldsymbol{E}_{1,2,3,4,5,6}$. → `mat_E12, mat_E13, mat_E14, mat_E23, mat_E24, mat_E34`

$$\begin{aligned}
\boldsymbol{E}_1 = \boldsymbol{E}_{12} &= \vec{n}_A\vec{n}_{12,A} + \vec{n}_C\vec{n}_{21,C} && (113.18)\\
\boldsymbol{E}_2 = \boldsymbol{E}_{13} &= \vec{n}_A\vec{n}_{13,A} + \vec{n}_B\vec{n}_{13,B} && (113.19)\\
\boldsymbol{E}_3 = \boldsymbol{E}_{14} &= \vec{n}_B\vec{n}_{14,B} + \vec{n}_C\vec{n}_{14,C} && (113.20)\\
\boldsymbol{E}_4 = \boldsymbol{E}_{23} &= \vec{n}_A\vec{n}_{23,A} + \vec{n}_D\vec{n}_{23,D} && (113.21)\\
\boldsymbol{E}_5 = \boldsymbol{E}_{24} &= \vec{n}_C\vec{n}_{24,C} + \vec{n}_D\vec{n}_{24,D} && (113.22)\\
\boldsymbol{E}_6 = \boldsymbol{E}_{34} &= \vec{n}_B\vec{n}_{34,B} + \vec{n}_D\vec{n}_{34,D} && (113.23)
\end{aligned}$$

10. compute $L_{1,2,3,4,5,6}$ → `L12`, `L13`, `L14`, ...

$$\begin{aligned}
L_1 = L_{12} &= \ln\frac{r_1+r_2+l_1}{r_1+r_2-l_1}\\
L_2 = L_{13} &= \ln\frac{r_1+r_3+l_2}{r_1+r_3-l_2}\\
L_3 = L_{14} &= \ln\frac{r_1+r_4+l_3}{r_1+r_4-l_3}\\
L_4 = L_{23} &= \ln\frac{r_2+r_3+l_4}{r_2+r_3-l_4}\\
L_5 = L_{24} &= \ln\frac{r_2+r_4+l_5}{r_2+r_4-l_5}\\
L_6 = L_{34} &= \ln\frac{r_3+r_4+l_6}{r_3+r_4-l_6}
\end{aligned}$$

11. compute $\vec{r}_{12,13,14,23,24,34}$. As specified in Section 2.1.8 of their paper "$\vec{r}_e$ is a vector from the field point to any point on edge e or its infinite extension" → `vec_r12`, `vec_r13`, `vec_r13`, etc ...

12. compute $\vec{g}_e$ by means of Eq. (113.1)

$$\begin{aligned}
\vec{g}_e &= \mathcal{G}\rho_0 \sum_e L_e \boldsymbol{E}_e \cdot \vec{r}_e\\
&= \mathcal{G}\rho_0 \left(L_{12}\boldsymbol{E}_{12}\cdot\vec{r}_{12} + L_{13}\boldsymbol{E}_{23}\cdot\vec{r}_{13} + L_{14}\boldsymbol{E}_{34}\cdot\vec{r}_{14} + L_{23}\boldsymbol{E}_{23}\cdot\vec{r}_{23} + L_{24}\boldsymbol{E}_{24}\cdot\vec{r}_{24} + L_{34}\boldsymbol{E}_{34}\cdot\vec{r}_{34}\right) \quad (113.24)
\end{aligned}$$

13. compute $\vec{g} = -\vec{g}_e + \vec{g}_f$

Except for the 4 measurements provided in Metherell and Quinn [235] I could not easily find actual values to benchmark the code against, so the following three examples aim at remedying this problem. In all what follows we set $\rho_0 = 1$ and $\mathcal{G} = 1$ for simplicity.

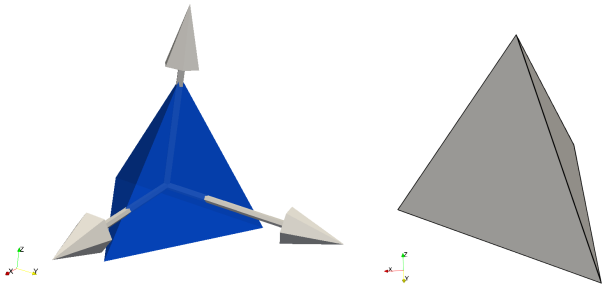
Tetrahedron test 1: the Wikipedia tetrahedron

We consider the following regular tetrahedron (i.e. all faces are equilateral triangles) centered at the origin[1]. Its edges are $l = \sqrt{8/3}$ long. We have

$$\begin{aligned}
(x_1, y_1, z_1) &= (0, 0, 1)\\
(x_2, y_2, z_2) &= (\sqrt{8/9}, 0, -1/3)\\
(x_3, y_3, z_3) &= (-\sqrt{2/9}, \sqrt{2/3}, -1/3)\\
(x_4, y_4, z_4) &= (-\sqrt{2/9}, -\sqrt{2/3}, -1/3)
\end{aligned}$$

Point 1 is at the top and points 2,3,4 are in the $z = -1/3$ plane.

[1] `https://en.wikipedia.org/wiki/Tetrahedron`

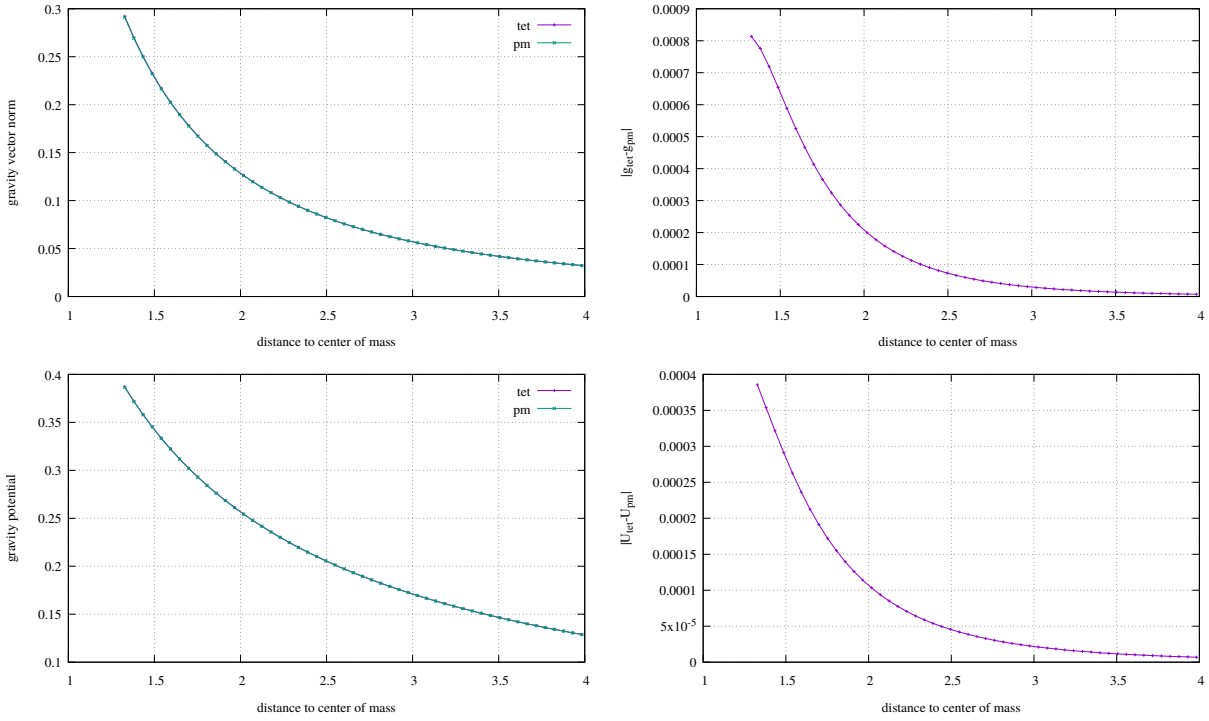


The volume of a regular tetrahedron of edge l is given by:

$$V = \frac{l^3}{6\sqrt{2}} = \frac{8}{9\sqrt{3}} \simeq 0.5132$$

and this is what we indeed recover.

```
pt_meas = np.array([0,0,10],dtype=np.float64)
pt_one=np.array([0,0,1],dtype=np.float64)
pt_two=np.array([-np.sqrt(2/9),-np.sqrt(2/3),-1/3],dtype=np.float64)
pt_three=np.array([np.sqrt(8/9),0,-1/3],dtype=np.float64)
pt_four=np.array([-np.sqrt(2/9),np.sqrt(2/3),-1/3],dtype=np.float64)
```



Tetrahedron test 2: the 111-tetrahedron

It is defined in Metherell and Quinn [235] (1986) (with a correction by Carré, Metherell, and Quinn [63] (1986)).

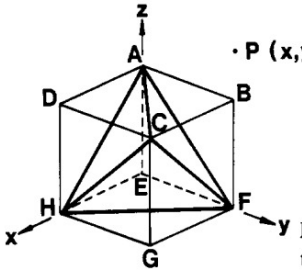
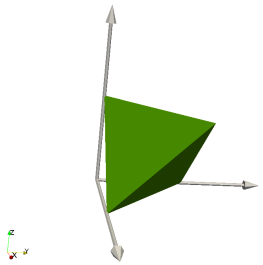


Fig. 1. The regular tetrahedron treated as a truncated cube

```
alpha=1.
pt_meas = np.array([2,0.5,0.5],dtype=np.float64)
pt_one=np.array([alpha,alpha,alpha],dtype=np.float64)
pt_two=np.array([alpha,0,0],dtype=np.float64)
pt_three=np.array([0,alpha,0],dtype=np.float64)
pt_four=np.array([0,0,alpha],dtype=np.float64)
```

In this case I choose pt1 is node C, pt2 is H, pt3 is F pt 4 is A. If the cube is a unit cube then the edge length is $\sqrt{2}$ and the volume

$$V = \frac{l^3}{6\sqrt{2}} = \frac{1}{3}$$



Metherell and Quinn [235] state: "To ensure that the above analysis can be treated with confidence we have performed numerical calculations of the field components Fx, Fy, and Fz using the expressions developed in this paper and have compared them with those obtained by direct numerical integration":

Table 4. The first three columns of this table give the coordinates of the point P (Fig. 1), with origin at E and cube side s = 1.0, where the field components $F_x/G\varrho$ are calculated. The column $(F_x/G\varrho)_1$ gives this field component as calculated by (4) whereas the column $(F_x/G\varrho)_2$ gives F_x as determined by direct numerical integration. The small numerical discrepancies between $(F_x/G\varrho)_1$ and $(F_x/G\varrho)_2$ are a result of the fact that the calculations were performed with single precision (6 digits) only

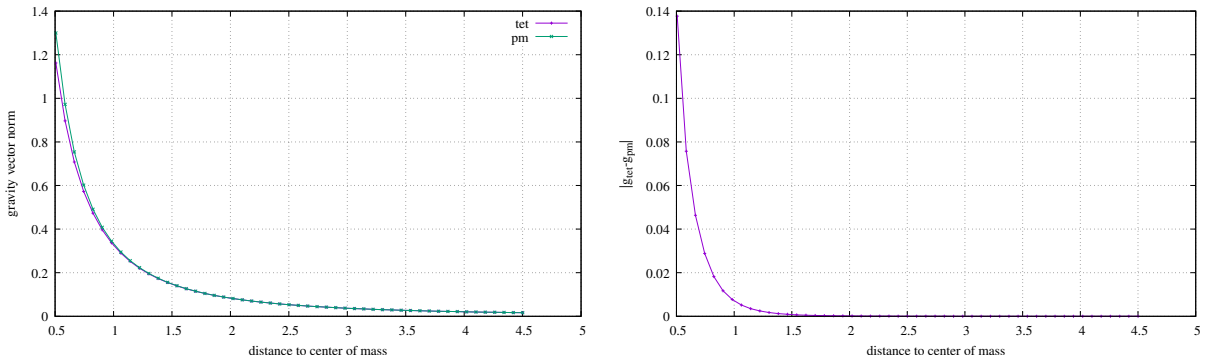
x	y	z	$(F_x/G\varrho)_1$	$(F_x/G\varrho)_2$
1.25	0.5	0.5	−0.56668	−0.56674
1.5	0.5	0.5	−0.32653	−0.32657
1.75	0.5	0.5	−0.21122	−0.21126
2.0	0.5	0.5	−0.14738	−0.14740

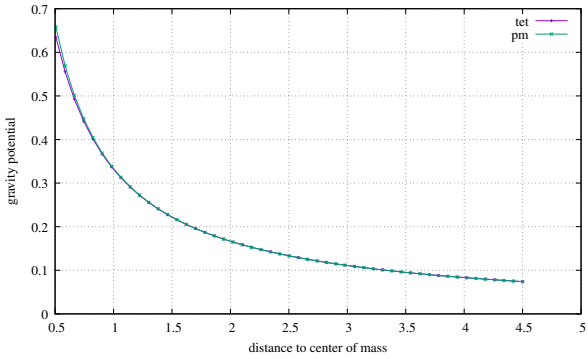
Taken from Metherell and Quinn [235].

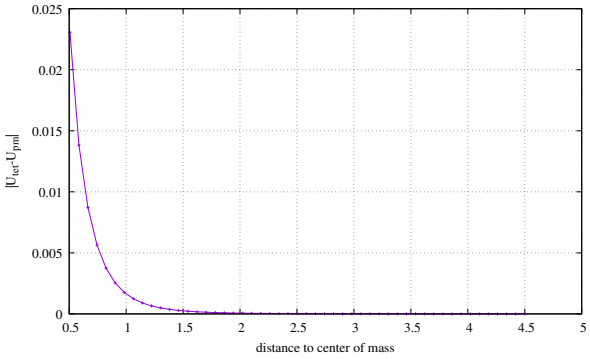
x	y	z	g_x (tet)	g_x (point mass)
1.25	0.5	0.5	0.566679144260546	0.5925925925925926
1.5	0.5	0.5	0.326526930616740	0.3333333333333333
1.75	0.5	0.5	0.211222015971550	0.2133333333333333
2.0	0.5	0.5	0.147377525744174	0.1481481481481481

Results obtained with this STONE .

Gravity fields are measured on a line starting at $(1, 0.456, 0.567)$ and ending at $(5, 0.456, 0.567)$

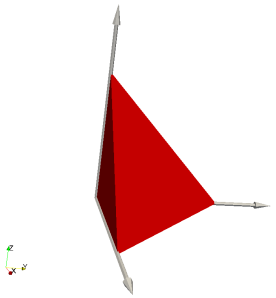






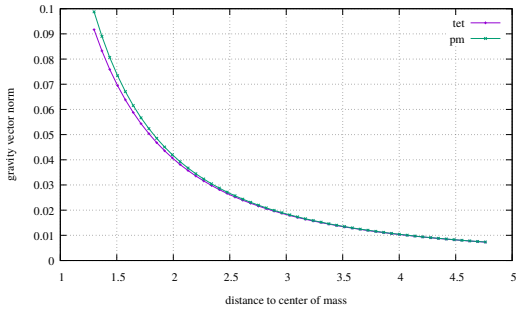
Tetrahedron test 3: the origin tetrahedron

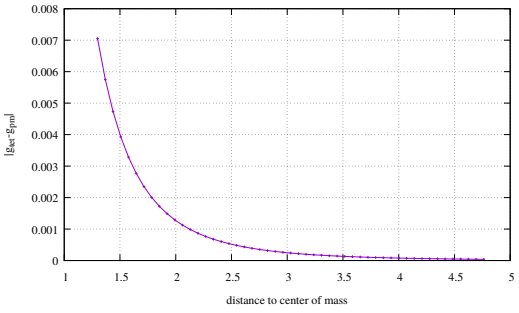
This is not a regular tetrahedron since $l_1 \neq l_4$.

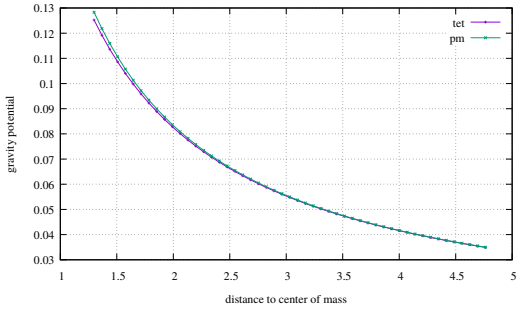


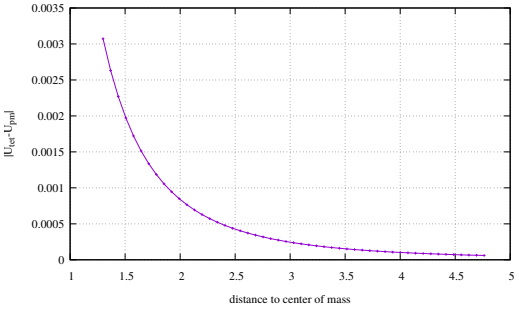
$$
\begin{aligned}
(x_1, y_1, z_1) &= (0, 0, 0) \\
(x_2, y_2, z_2) &= (1, 0, 0) \\
(x_3, y_3, z_3) &= (0, 1, 0) \\
(x_4, y_4, z_4) &= (0, 0, 1)
\end{aligned}
$$

The gravity fields are measured on a line starting at $(1, 1, 1)$ and ending at $(3, 3, 3)$:









Tetrahedron test 4: cube containing 5 tetrahedra

to do!

Hexahedron: theory

In what follows $e = 1, 2, 3, ...$ stands for an edge while $f = A, B, C, ...$ stands for a face. Each polyhedron face has an outward-pointing face normal vector $\vec{n}_f$ and a face dyad $\boldsymbol{F}_f = \vec{n}_f \vec{n}_f$. Each edge of each face has an outward-pointing edge normal vector $\vec{n}_{e,f}$ perpendicular to both $\vec{n}_f$ and the edge. For the edge connecting vertices 1 and 2 shared by faces A and B, the edge dyad is $\boldsymbol{E}_{12} = \vec{n}_A \vec{n}_{12,A} + \vec{n}_B \vec{n}_{21,B}$.

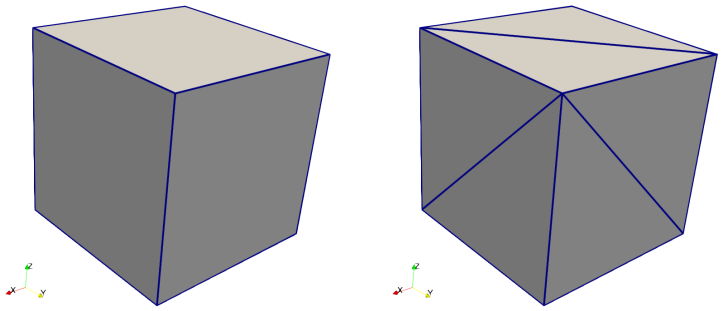
Let $\vec{r}_i$ represent the vector from the variable field-point M at location $\vec{r}_M$ to polyhedron vertex P_i, and let $r_i = ||\vec{r}_i||$ be its length.

For the polyhedron edge connecting vertices i and j of length l_{ij}, the dimensionless per-edge factor L_e is

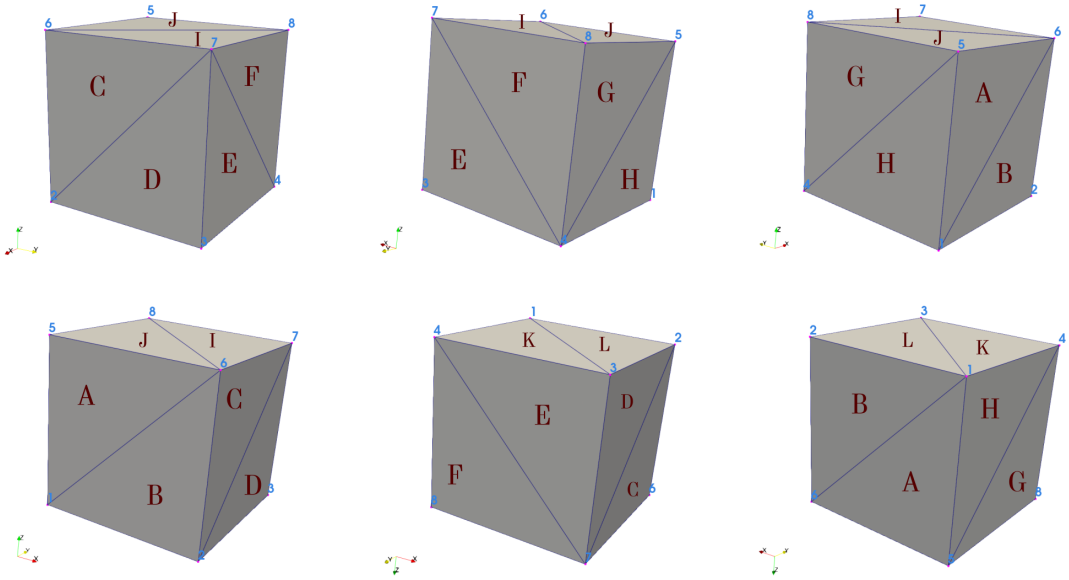
$$L_e = L_{ij} = \int_e \frac{1}{r} ds = \int_{P_i}^{P_j} \frac{1}{r} ds = \ln \frac{r_i + r_j + l_{ij}}{r_i + r_j - l_{ij}}$$

For a triangular face f bounded by vertices i, j, k the dimensionless per-face factor ω_f is

$$\omega_f = \iint_{triangle} \frac{\Delta z}{r^3} dS = 2 \arctan \frac{\vec{r}_i \cdot (\vec{r}_j \times \vec{r}_k)}{r_i r_j r_k + r_i (\vec{r}_j \cdot \vec{r}_k) + r_j (\vec{r}_k \cdot \vec{r}_i) + r_k (\vec{r}_i \cdot \vec{r}_j)}$$



Left: unit cube; Right: triangles making up the visible faces.



When looking at a face from the outside towards the tetrahedron the numbering of vertices is counter-clockwise.

face	vertices i,j,k	edge #1	edge #2	edge #3
A	5,1,6	5-1 (9)	1-6 (13)	6-5 (1)
B	2,6,1	2-6 (10)	6-1 (13)	1-2 (5)
C	6,2,7	6-2 (10)	2-7 (14)	7-6 (2)
D	3,7,2	3-7 (11)	7-2 (14)	2-3 (6)
E	3,4,7	3-4 (7)	4-7 (15)	7-3 (11)
F	8,7,4	8-7 (3)	7-4 (15)	4-8 (12)
G	8,4,5	8-4 (12)	4-5 (16)	5-8 (4)
H	1,5,4	1-5 (9)	5-4 (16)	4-1 (8)
I	7,8,6	7-8 (3)	8-6 (17)	6-7 (2)
J	5,6,8	5-6 (1)	6-8 (17)	8-5 (4)
K	4,3,1	4-3 (7)	3-1 (18)	1-4 (8)
L	2,1,3	2-1 (5)	1-3 (18)	3-2 (6)

Hexahedron: algorithm

1. compute $\vec{l}_1$, $\vec{l}_2$,etc ... and their norms → `vec_l01`, `vec_l02`, `vec_l03`, `vec_l04`, ... `vec_l18`

$$\vec{l}_1 = \vec{l}_{56} = \begin{pmatrix} x_6 - x_5 \\ y_6 - y_5 \\ z_6 - z_5 \end{pmatrix} \qquad \vec{l}_2 = \vec{l}_{67} = \begin{pmatrix} x_7 - x_6 \\ y_7 - y_6 \\ z_7 - z_6 \end{pmatrix} \qquad \text{etc...}$$

```
vec_l01=np.zeros(3,dtype=np.float64) ; vec_l01 = pt_5-pt_6 ; l01 = LA.norm(vec_l01,2)
vec_l02=np.zeros(3,dtype=np.float64) ; vec_l02 = pt_6-pt_7 ; l02 = LA.norm(vec_l02,2)
vec_l03=np.zeros(3,dtype=np.float64) ; vec_l03 = pt_7-pt_8 ; l03 = LA.norm(vec_l03,2)
...
vec_l16=np.zeros(3,dtype=np.float64) ; vec_l16 = pt_4-pt_5 ; l16 = LA.norm(vec_l16,2)
vec_l17=np.zeros(3,dtype=np.float64) ; vec_l17 = pt_6-pt_8 ; l17 = LA.norm(vec_l17,2)
vec_l18=np.zeros(3,dtype=np.float64) ; vec_l18 = pt_1-pt_3 ; l18 = LA.norm(vec_l18,2)
```

2. compute $\vec{r}_{1,2,3,4,5,6,7,8}$. → `vec_r01`, `vec_r02`, `vec_r03`, ... `vec_r08`. For example:

$$\vec{r}_1 = \vec{M1} = \begin{pmatrix} x_1 - x_m \\ y_1 - y_m \\ z_1 - z_m \end{pmatrix} \qquad r_1 = |\vec{r}_1|$$

```
vec_r01=np.zeros(3,dtype=np.float64) ; vec_r01 = pt_1-pt_M ; r01=LA.norm(vec_r01,2)
vec_r02=np.zeros(3,dtype=np.float64) ; vec_r02 = pt_2-pt_M ; r02=LA.norm(vec_r02,2)
vec_r03=np.zeros(3,dtype=np.float64) ; vec_r03 = pt_3-pt_M ; r03=LA.norm(vec_r03,2)
vec_r04=np.zeros(3,dtype=np.float64) ; vec_r04 = pt_4-pt_M ; r04=LA.norm(vec_r04,2)
vec_r05=np.zeros(3,dtype=np.float64) ; vec_r05 = pt_5-pt_M ; r05=LA.norm(vec_r05,2)
vec_r06=np.zeros(3,dtype=np.float64) ; vec_r06 = pt_6-pt_M ; r06=LA.norm(vec_r06,2)
vec_r07=np.zeros(3,dtype=np.float64) ; vec_r07 = pt_7-pt_M ; r07=LA.norm(vec_r07,2)
vec_r08=np.zeros(3,dtype=np.float64) ; vec_r08 = pt_8-pt_M ; r08=LA.norm(vec_r08,2)

r01=LA.norm(vec_r01,2)
r02=LA.norm(vec_r02,2)
r03=LA.norm(vec_r03,2)
r04=LA.norm(vec_r04,2)
r05=LA.norm(vec_r05,2)
r06=LA.norm(vec_r06,2)
r07=LA.norm(vec_r07,2)
r08=LA.norm(vec_r08,2)
```

3. compute $\vec{n}_{A,B,C,D,...K,L}$ and coordinates of face centers. → `vec_nA` , `vec_nB`, `vec_nC`, `vec_nD`, ... `vec_nK`, `vec_nL`

$$
\begin{aligned}
\vec{n}_A &= \vec{l}_{51} \times \vec{l}_{56}/|\vec{l}_{51} \times \vec{l}_{56}| = -\vec{l}_9 \times \vec{l}_1/| - \vec{l}_9 \times \vec{l}_1| \\
\vec{n}_B &= \vec{l}_{26} \times \vec{l}_{21}/|\vec{l}_{26} \times \vec{l}_{21}| = \vec{l}_{10} \times -\vec{l}_5/|\vec{l}_{10} \times -\vec{l}_5| \\
\vec{n}_C &= \vec{l}_{62} \times \vec{l}_{67}/|\vec{l}_{62} \times \vec{l}_{67}| = -\vec{l}_{10} \times \vec{l}_2/| - \vec{l}_{10} \times \vec{l}_2| \\
\vec{n}_D &= \vec{l}_{37} \times \vec{l}_{32}/|\vec{l}_{37} \times \vec{l}_{32}| = \vec{l}_{11} \times -\vec{l}_6/|\vec{l}_{11} \times -\vec{l}_6| \\
\vec{n}_E &= \vec{l}_{34} \times \vec{l}_{37}/|\vec{l}_{34} \times \vec{l}_{37}| = \vec{l}_7 \times \vec{l}_{11}/|\vec{l}_7 \times \vec{l}_{11}| \\
\vec{n}_F &= \vec{l}_{87} \times \vec{l}_{84}/|\vec{l}_{87} \times \vec{l}_{84}| = -\vec{l}_3 \times -\vec{l}_{12}/| - \vec{l}_3 \times -\vec{l}_{12}| \\
\vec{n}_G &= \vec{l}_{84} \times \vec{l}_{85}/|\vec{l}_{84} \times \vec{l}_{85}| = -\vec{l}_{12} \times \vec{l}_4/| - \vec{l}_{12} \times \vec{l}_4| \\
\vec{n}_H &= \vec{l}_{15} \times \vec{l}_{14}/|\vec{l}_{14} \times \vec{l}_{14}| = \vec{l}_9 \times -\vec{l}_8/|\vec{l}_9 \times -\vec{l}_8| \\
\vec{n}_I &= \vec{l}_{78} \times \vec{l}_{76}/|\vec{l}_{78} \times \vec{l}_{76}| = \vec{l}_3 \times -\vec{l}_2/|\vec{l}_3 \times -\vec{l}_2| \\
\vec{n}_J &= \vec{l}_{56} \times \vec{l}_{58}/|\vec{l}_{56} \times \vec{l}_{58}| = \vec{l}_1 \times -\vec{l}_4/|\vec{l}_1 \times -\vec{l}_4| \\
\vec{n}_K &= \vec{l}_{43} \times \vec{l}_{41}/|\vec{l}_{43} \times \vec{l}_{41}| = -\vec{l}_7 \times \vec{l}_8/| - \vec{l}_7 \times \vec{l}_8| \\
\vec{n}_L &= \vec{l}_{21} \times \vec{l}_{23}/|\vec{l}_{21} \times \vec{l}_{23}| = -\vec{l}_5 \times \vec{l}_6/| - \vec{l}_5 \times \vec{l}_6|
\end{aligned}
$$

```
pt_midA=np.zeros(3,dtype=np.float64) ; pt_midA = (pt_1+pt_6+pt_5)/3
pt_midB=np.zeros(3,dtype=np.float64) ; pt_midB = (pt_1+pt_2+pt_6)/3
pt_midC=np.zeros(3,dtype=np.float64) ; pt_midC = (pt_2+pt_7+pt_6)/3
...
pt_midJ=np.zeros(3,dtype=np.float64) ; pt_midJ = (pt_8+pt_5+pt_6)/3
pt_midK=np.zeros(3,dtype=np.float64) ; pt_midK = (pt_3+pt_1+pt_4)/3
pt_midL=np.zeros(3,dtype=np.float64) ; pt_midL = (pt_3+pt_2+pt_1)/3

vec_nA=np.cross( vec_l13 , vec_l09) ; vec_nA/=LA.norm(vec_nA,2)
vec_nB=np.cross( vec_l10,-vec_l05) ; vec_nB/=LA.norm(vec_nB,2)
vec_nC=np.cross(-vec_l10 , vec_l02) ; vec_nC/=LA.norm(vec_nC,2)
...
vec_nJ=np.cross( vec_l01,-vec_l04) ; vec_nJ/=LA.norm(vec_nJ,2)
vec_nK=np.cross(-vec_l07 , vec_l08) ; vec_nK/=LA.norm(vec_nK,2)
vec_nL=np.cross(-vec_l05 , vec_l06) ; vec_nL/=LA.norm(vec_nL,2)
```

4. compute $\boldsymbol{F}_{A,B,C,D}$. → `mat_FaceA` , `mat_FaceB`, `mat_FaceC`, `mat_FaceD`, ... `mat_FaceL`

$$
\begin{aligned}
\boldsymbol{F}_A &= \vec{n}_A\vec{n}_A \\
\boldsymbol{F}_B &= \vec{n}_B\vec{n}_B \\
\boldsymbol{F}_C &= \vec{n}_C\vec{n}_C \\
&... \\
\boldsymbol{F}_K &= \vec{n}_K\vec{n}_K \\
\boldsymbol{F}_L &= \vec{n}_L\vec{n}_L
\end{aligned}
$$

```
mat_FA=np.zeros((3,3),dtype=np.float64) ; mat_FA=np.outer(vec_nA,vec_nA)
mat_FB=np.zeros((3,3),dtype=np.float64) ; mat_FB=np.outer(vec_nB,vec_nB)
mat_FC=np.zeros((3,3),dtype=np.float64) ; mat_FC=np.outer(vec_nC,vec_nC)
...
mat_FJ=np.zeros((3,3),dtype=np.float64) ; mat_FJ=np.outer(vec_nJ,vec_nJ)
mat_FK=np.zeros((3,3),dtype=np.float64) ; mat_FK=np.outer(vec_nK,vec_nK)
mat_FL=np.zeros((3,3),dtype=np.float64) ; mat_FL=np.outer(vec_nL,vec_nL)
```

5. compute $\omega_{A,B,C,D}$. → `wA`, `wB`, `wC`, `wD`, ... `wK`, `wL`. In what follows $r_i = |\vec{r}_i|$.

$$\omega_A = 2\arctan\frac{\vec{r_5}\cdot(\vec{r_1}\times\vec{r_6})}{r_5r_1r_6 + r_1(\vec{r_6}\cdot\vec{r_5}) + r_6(\vec{r_5}\cdot\vec{r_1}) + r_5(\vec{r_1}\cdot\vec{r_6})} \qquad A = \{5,1,6\} = \{9,13,1\}$$

$$\omega_B = 2\arctan\frac{\vec{r_2}\cdot(\vec{r_6}\times\vec{r_1})}{r_2r_6r_1 + r_1(\vec{r_2}\cdot\vec{r_6}) + r_2(\vec{r_6}\cdot\vec{r_1}) + r_6(\vec{r_1}\cdot\vec{r_2})} \qquad B = \{2,6,1\} = \{10,13,5\}$$

$$\omega_C = 2\arctan\frac{\vec{r_6}\cdot(\vec{r_2}\times\vec{r_7})}{r_6r_2r_7 + r_2(\vec{r_7}\cdot\vec{r_6}) + r_7(\vec{r_6}\cdot\vec{r_2}) + r_6(\vec{r_2}\cdot\vec{r_7})} \qquad C = \{6,2,7\} = \{10,14,2\}$$

$$\omega_D = 2\arctan\frac{\vec{r_3}\cdot(\vec{r_7}\times\vec{r_2})}{r_3r_7r_2 + r_2(\vec{r_3}\cdot\vec{r_7}) + r_3(\vec{r_7}\cdot\vec{r_2}) + r_7(\vec{r_2}\cdot\vec{r_3})} \qquad D = \{3,7,2\} = \{11,14,6\}$$

$$\omega_E = 2\arctan\frac{\vec{r_3}\cdot(\vec{r_4}\times\vec{r_7})}{r_3r_4r_7 + r_1(\vec{r_6}\cdot\vec{r_5}) + r_6(\vec{r_5}\cdot\vec{r_1}) + r_5(\vec{r_1}\cdot\vec{r_6})} \qquad E = \{3,4,7\} = \{7,15,11\}$$

$$\omega_F = 2\arctan\frac{\vec{r_8}\cdot(\vec{r_7}\times\vec{r_4})}{r_8r_7r_4 + r_7(\vec{r_4}\cdot\vec{r_8}) + r_4(\vec{r_8}\cdot\vec{r_7}) + r_8(\vec{r_7}\cdot\vec{r_4})} \qquad F = \{8,7,4\} = \{3,15,12\}$$

$$\omega_G = 2\arctan\frac{\vec{r_8}\cdot(\vec{r_4}\times\vec{r_5})}{r_8r_4r_5 + r_4(\vec{r_5}\cdot\vec{r_8}) + r_5(\vec{r_8}\cdot\vec{r_4}) + r_8(\vec{r_4}\cdot\vec{r_5})} \qquad G = \{8,4,5\} = \{12,16,4\}$$

$$\omega_H = 2\arctan\frac{\vec{r_1}\cdot(\vec{r_5}\times\vec{r_4})}{r_1r_5r_4 + r_4(\vec{r_1}\cdot\vec{r_5}) + r_1(\vec{r_5}\cdot\vec{r_4}) + r_5(\vec{r_4}\cdot\vec{r_1})} \qquad H = \{1,5,4\} = \{9,16,8\}$$

$$\omega_I = 2\arctan\frac{\vec{r_7}\cdot(\vec{r_8}\times\vec{r_6})}{r_7r_8r_6 + r_8(\vec{r_6}\cdot\vec{r_7}) + r_6(\vec{r_7}\cdot\vec{r_8}) + r_7(\vec{r_8}\cdot\vec{r_6})} \qquad I = \{7,8,6\} = \{3,17,2\}$$

$$\omega_J = 2\arctan\frac{\vec{r_5}\cdot(\vec{r_6}\times\vec{r_8})}{r_5r_6r_8 + r_8(\vec{r_5}\cdot\vec{r_6}) + r_5(\vec{r_6}\cdot\vec{r_8}) + r_6(\vec{r_8}\cdot\vec{r_5})} \qquad J = \{5,6,8\} = \{1,17,4\}$$

$$\omega_K = 2\arctan\frac{\vec{r_4}\cdot(\vec{r_3}\times\vec{r_1})}{r_4r_3r_1 + r_3(\vec{r_1}\cdot\vec{r_4}) + r_1(\vec{r_4}\cdot\vec{r_3}) + r_4(\vec{r_3}\cdot\vec{r_1})} \qquad K = \{4,3,1\} = \{7,18,8\}$$

$$\omega_L = 2\arctan\frac{\vec{r_2}\cdot(\vec{r_1}\times\vec{r_3})}{r_2r_1r_3 + r_3(\vec{r_2}\cdot\vec{r_1}) + r_2(\vec{r_1}\cdot\vec{r_3}) + r_1(\vec{r_3}\cdot\vec{r_2})} \qquad L = \{2,1,3\} = \{5,18,6\}$$

```
# A: 516
num=np.dot(vec_r05,np.cross(vec_r01,vec_r06))
denom=(r05*r01*r06+r05*np.dot(vec_r01,vec_r06)
       +r01*np.dot(vec_r06,vec_r05)
       +r06*np.dot(vec_r05,vec_r01))
wA=2*np.arctan2(num,denom)

# B: 261
num=np.dot(vec_r02,np.cross(vec_r06,vec_r01))
denom=(r02*r06*r01+r02*np.dot(vec_r06,vec_r01)
       +r06*np.dot(vec_r01,vec_r02)
       +r01*np.dot(vec_r02,vec_r06))
wB=2*np.arctan2(num,denom)

# C: 627
num=np.dot(vec_r06,np.cross(vec_r02,vec_r07))
denom=(r06*r02*r07+r06*np.dot(vec_r02,vec_r07)
       +r02*np.dot(vec_r07,vec_r06)
       +r07*np.dot(vec_r06,vec_r02))
wC=2*np.arctan2(num,denom)

...
```

6. compute $\vec{r}_A$, $\vec{r}_B$, $\vec{r}_C$, $\vec{r}_D$. As explained in the caption of their Figure 3, "vector $\vec{r}_f$ extends from the field point M to any point in the face plane". → `vec_rA`, `vec_rB`, `vec_rC`, `vec_rD`, ... `ver_rL`

```
vec_rA = pt_midA-pt_M
vec_rB = pt_midB-pt_M
vec_rC = pt_midC-pt_M
vec_rD = pt_midD-pt_M
vec_rE = pt_midE-pt_M
vec_rF = pt_midF-pt_M
vec_rG = pt_midG-pt_M
vec_rH = pt_midH-pt_M
```

```
vec_rI = pt_midI-pt_M
vec_rJ = pt_midJ-pt_M
vec_rK = pt_midK-pt_M
vec_rL = pt_midL-pt_M
```

7. compute $\vec{g}_f$ by means of

$$\vec{g}_f = \mathcal{G}\rho_0 \sum_f \omega_f \boldsymbol{F}_f \cdot \vec{r}_f = \mathcal{G}\rho_0 \left(\omega_A \boldsymbol{F}_A \cdot \vec{r}_A + \omega_B \boldsymbol{F}_B \cdot \vec{r}_B + \omega_C \boldsymbol{F}_C \cdot \vec{r}_C + \omega_D \boldsymbol{F}_D \cdot \vec{r}_D + ... + \omega_L \boldsymbol{F}_L \cdot \vec{r}_L\right)$$

```
vec_gf=np.zeros(3,dtype=np.float64)
vec_gf=wA*np.dot(mat_FA,vec_rA) +\
       wB*np.dot(mat_FB,vec_rB) +\
       wC*np.dot(mat_FC,vec_rC) +\
       wD*np.dot(mat_FD,vec_rD) +\
       wE*np.dot(mat_FE,vec_rE) +\
       wF*np.dot(mat_FF,vec_rF) +\
       wG*np.dot(mat_FG,vec_rG) +\
       wH*np.dot(mat_FH,vec_rH) +\
       wI*np.dot(mat_FI,vec_rI) +\
       wJ*np.dot(mat_FJ,vec_rJ) +\
       wK*np.dot(mat_FK,vec_rK) +\
       wL*np.dot(mat_FL,vec_rL)
vec_gf*=Ggrav*rho0
```

8. compute $\vec{n}_{e,f}$ for each edge of each face (12 triangles*3=36 vectors!) → `vec_nA12`, `vec_nA23`, `vec_nA13`, `vec_nB13`, etc ...

```
vec_nA12 = compute_face_edge_normal(vec_nA,pt_1,pt_2,pt_midA,'faceA_n12')
vec_nA23 = compute_face_edge_normal(vec_nA,pt_2,pt_3,pt_midA,'faceA_n23')
vec_nA13 = compute_face_edge_normal(vec_nA,pt_3,pt_1,pt_midA,'faceA_n31')
```

9. compute $\boldsymbol{E}_{1,2,3,4,5,6,...18}$. → `mat_E12, mat_E13, mat_E14, mat_E23, mat_E24, mat_E34`. For the edge connecting vertices 1 and 2 shared by faces A and B, the edge dyad is $\boldsymbol{E}_{12} = \vec{n}_A \vec{n}_{12,A} + \vec{n}_B \vec{n}_{21,B}$.

edge	vertices	belongs to
1	5,6	J,A
2	6,7	I,C
3	7,8	I,F
4	8,5	J,G
5	1,2	B,L
6	2,3	D,L
7	3,4	E,K
8	4,1	H,K
9	1,5	H,A
10	2,6	B,C
11	3,7	D,E
12	4,8	F,G
13	1,6	A,B
14	2,7	C,D
15	4,7	E,F
16	4,5	G,H
17	6,8	J,I
18	1,3	L,K

The first letter is the face left of the arrow, the second letter is the face right of the arrow, when going along the arrow.

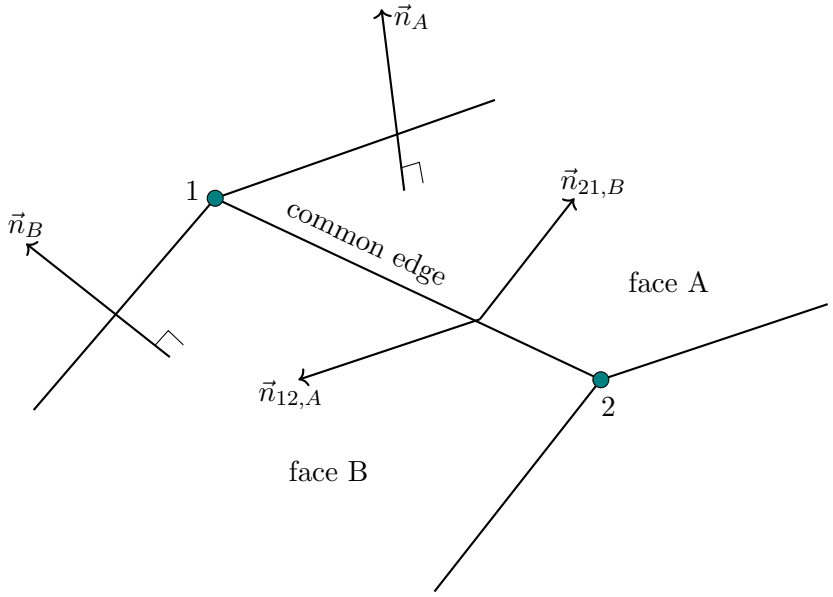


Figure made after Fig. 7 from Werner and Scheeres [351].

$$
\begin{aligned}
\boldsymbol{E}_1 = \boldsymbol{E}_{56} &= \vec{n}_A\vec{n}_{12,A} + \vec{n}_J\vec{n}_{21,J} \\
\boldsymbol{E}_2 = \boldsymbol{E}_{67} &= \vec{n}_I\vec{n}_{13,I} + \vec{n}_C\vec{n}_{13,C} \\
\boldsymbol{E}_3 = \boldsymbol{E}_{78} &= \vec{n}_F\vec{n}_{14,F} + \vec{n}_I\vec{n}_{14,I} \\
\boldsymbol{E}_4 = \boldsymbol{E}_{85} &= \vec{n}_J\vec{n}_{23,J} + \vec{n}_G\vec{n}_{23,G} \\
\boldsymbol{E}_5 = \boldsymbol{E}_{12} &= \vec{n}_B\vec{n}_{24,B} + \vec{n}_L\vec{n}_{24,L} \\
\boldsymbol{E}_6 = \boldsymbol{E}_{23} &= \vec{n}_L\vec{n}_{34,L} + \vec{n}_D\vec{n}_{34,D} \\
\boldsymbol{E}_7 = \boldsymbol{E}_{34} &= \vec{n}_E\vec{n}_{34,E} + \vec{n}_K\vec{n}_{34,K} \\
\boldsymbol{E}_8 = \boldsymbol{E}_{41} &= \vec{n}_H\vec{n}_{34,H} + \vec{n}_K\vec{n}_{34,K} \\
\boldsymbol{E}_9 = \boldsymbol{E}_{15} &= \vec{n}_A\vec{n}_{34,A} + \vec{n}_H\vec{n}_{34,H} \\
\boldsymbol{E}_{10} = \boldsymbol{E}_{26} &= \vec{n}_B\vec{n}_{34,B} + \vec{n}_C\vec{n}_{34,C} \\
\boldsymbol{E}_{11} = \boldsymbol{E}_{37} &= \vec{n}_D\vec{n}_{34,D} + \vec{n}_E\vec{n}_{34,E} \\
\boldsymbol{E}_{12} = \boldsymbol{E}_{48} &= \vec{n}_F\vec{n}_{34,F} + \vec{n}_G\vec{n}_{34,G} \\
\boldsymbol{E}_{13} = \boldsymbol{E}_{16} &= \vec{n}_A\vec{n}_{34,A} + \vec{n}_B\vec{n}_{34,B} \\
\boldsymbol{E}_{14} = \boldsymbol{E}_{27} &= \vec{n}_C\vec{n}_{34,C} + \vec{n}_D\vec{n}_{34,D} \\
\boldsymbol{E}_{15} = \boldsymbol{E}_{47} &= \vec{n}_E\vec{n}_{47,E} + \vec{n}_F\vec{n}_{74,F} \\
\boldsymbol{E}_{16} = \boldsymbol{E}_{45} &= \vec{n}_G\vec{n}_{45,G} + \vec{n}_H\vec{n}_{54,H} \\
\boldsymbol{E}_{17} = \boldsymbol{E}_{68} &= \vec{n}_I\vec{n}_{68,I} + \vec{n}_J\vec{n}_{86,J} \\
\boldsymbol{E}_{18} = \boldsymbol{E}_{13} &= \vec{n}_K\vec{n}_{13,K} + \vec{n}_L\vec{n}_{31,L}
\end{aligned}
$$

```
mat_E56=np.outer(vec_nJ,vec_nJ56)+np.outer(vec_nA,vec_nA65) #edge 5-6 ( 1) belongs to J & A
mat_E67=np.outer(vec_nI,vec_nI67)+np.outer(vec_nC,vec_nC76) #edge 6-7 ( 2) belongs to I & C
mat_E78=np.outer(vec_nI,vec_nI78)+np.outer(vec_nF,vec_nF87) #edge 7-8 ( 3) belongs to I & F
mat_E85=np.outer(vec_nJ,vec_nJ85)+np.outer(vec_nG,vec_nG58) #edge 8-5 ( 4) belongs to J & G
mat_E12=np.outer(vec_nB,vec_nB12)+np.outer(vec_nL,vec_nL21) #edge 1-2 ( 5) belongs to B & L
mat_E23=np.outer(vec_nD,vec_nD23)+np.outer(vec_nL,vec_nL32) #edge 2-3 ( 6) belongs to D & L
mat_E34=np.outer(vec_nE,vec_nE34)+np.outer(vec_nK,vec_nK43) #edge 3-4 ( 7) belongs to E & K
mat_E41=np.outer(vec_nH,vec_nH41)+np.outer(vec_nK,vec_nK14) #edge 4-1 ( 8) belongs to H & K
mat_E15=np.outer(vec_nH,vec_nH15)+np.outer(vec_nA,vec_nA51) #edge 1-5 ( 9) belongs to H & A
mat_E26=np.outer(vec_nB,vec_nB26)+np.outer(vec_nC,vec_nC62) #edge 2-6 (10) belongs to B & C
mat_E37=np.outer(vec_nD,vec_nD37)+np.outer(vec_nE,vec_nE73) #edge 3-7 (11) belongs to D & E
mat_E48=np.outer(vec_nF,vec_nF48)+np.outer(vec_nG,vec_nG84) #edge 4-8 (12) belongs to F & G
mat_E16=np.outer(vec_nA,vec_nA16)+np.outer(vec_nB,vec_nB61) #edge 1-6 (13) belongs to A & B
mat_E27=np.outer(vec_nC,vec_nC27)+np.outer(vec_nD,vec_nD72) #edge 2-7 (14) belongs to C & D
```

```
mat_E47=np.outer(vec_nE,vec_nE47)+np.outer(vec_nF,vec_nF74) #edge 4-7 (15) belongs to E & F
mat_E45=np.outer(vec_nG,vec_nG45)+np.outer(vec_nH,vec_nH54) #edge 4-5 (16) belongs to G & H
mat_E68=np.outer(vec_nJ,vec_nJ68)+np.outer(vec_nI,vec_nI86) #edge 6-8 (17) belongs to J & I
mat_E13=np.outer(vec_nL,vec_nL13)+np.outer(vec_nK,vec_nK31) #edge 1-3 (18) belongs to L & K
```

10. compute $L_{1,2,3,4,5,6} \rightarrow$ `L01`, `L02`, `L03`, ...

$$
\begin{aligned}
L_1 = L_{56} &= \ln \frac{r_5 + r_6 + l_1}{r_5 + r_6 - l_1} \\
L_2 = L_{67} &= \ln \frac{r_6 + r_7 + l_2}{r_6 + r_7 - l_2} \\
L_3 = L_{78} &= \ln \frac{r_7 + r_8 + l_3}{r_7 + r_8 - l_3} \\
L_4 = L_{85} &= \ln \frac{r_8 + r_5 + l_4}{r_8 + r_5 - l_4} \\
L_5 = L_{12} &= \ln \frac{r_1 + r_2 + l_5}{r_1 + r_2 - l_5} \\
L_6 = L_{23} &= \ln \frac{r_2 + r_3 + l_6}{r_2 + r_3 - l_6} \\
L_7 = L_{34} &= \ln \frac{r_3 + r_4 + l_7}{r_3 + r_4 - l_7} \\
L_8 = L_{41} &= \ln \frac{r_4 + r_3 + l_8}{r_4 + r_3 - l_8} \\
L_9 = L_{15} &= \ln \frac{r_1 + r_5 + l_9}{r_1 + r_5 - l_9} \\
L_{10} = L_{26} &= \ln \frac{r_2 + r_6 + l_{10}}{r_2 + r_6 - l_{10}} \\
L_{11} = L_{37} &= \ln \frac{r_3 + r_7 + l_{11}}{r_3 + r_7 - l_{11}} \\
L_{12} = L_{48} &= \ln \frac{r_4 + r_8 + l_{12}}{r_4 + r_8 - l_{12}} \\
L_{13} = L_{16} &= \ln \frac{r_1 + r_6 + l_{13}}{r_1 + r_6 - l_{13}} \\
L_{14} = L_{27} &= \ln \frac{r_2 + r_7 + l_{14}}{r_2 + r_7 - l_{14}} \\
L_{15} = L_{47} &= \ln \frac{r_4 + r_7 + l_{15}}{r_4 + r_7 - l_{15}} \\
L_{16} = L_{45} &= \ln \frac{r_4 + r_5 + l_{16}}{r_4 + r_5 - l_{16}} \\
L_{17} = L_{68} &= \ln \frac{r_6 + r_8 + l_{17}}{r_6 + r_8 - l_{17}} \\
L_{18} = L_{13} &= \ln \frac{r_1 + r_3 + l_{18}}{r_1 + r_3 - l_{18}}
\end{aligned}
$$

```
    L01=np.log((r05+r06+l01)/(r05+r06-l01)) #56
    L02=np.log((r06+r07+l02)/(r06+r07-l02)) #67
    L03=np.log((r07+r08+l03)/(r07+r08-l03)) #78
    L04=np.log((r08+r05+l04)/(r08+r05-l04)) #85
    L05=np.log((r01+r02+l05)/(r01+r02-l05)) #12
    L06=np.log((r02+r03+l06)/(r02+r03-l06)) #23
    L07=np.log((r03+r04+l07)/(r03+r04-l07)) #34
    L08=np.log((r04+r01+l08)/(r04+r01-l08)) #41
    L09=np.log((r01+r05+l09)/(r01+r05-l09)) #15
    L10=np.log((r02+r06+l10)/(r02+r06-l10)) #26
    L11=np.log((r03+r07+l11)/(r03+r07-l11)) #37
    L12=np.log((r04+r08+l12)/(r04+r08-l12)) #48
    L13=np.log((r01+r06+l13)/(r01+r06-l13)) #16
```

```
L14=np.log((r02+r07+l14)/(r02+r07-l14)) #27
L15=np.log((r04+r07+l15)/(r04+r07-l15)) #47
L16=np.log((r04+r05+l16)/(r04+r05-l16)) #45
L17=np.log((r06+r08+l17)/(r06+r08-l17)) #68
L18=np.log((r01+r03+l18)/(r01+r03-l18)) #13
```

11. compute $\vec{r}_{12,13,14,23,24,34}$. As specified in Section 2.1.8 of their paper "$\vec{r}_e$ is a vector from the field point to any point on edge e or its infinite extension". I choose the middle of the edge. → `vec_r01`, `vec_r02`, `vec_r03`, etc ...

```
vec_r01=np.zeros(3,dtype=np.float64) ; vec_r01 = 0.5*(pt_5+pt_6)-pt_M #56
vec_r02=np.zeros(3,dtype=np.float64) ; vec_r02 = 0.5*(pt_6+pt_7)-pt_M #67
vec_r03=np.zeros(3,dtype=np.float64) ; vec_r03 = 0.5*(pt_7+pt_8)-pt_M #78
vec_r04=np.zeros(3,dtype=np.float64) ; vec_r04 = 0.5*(pt_8+pt_5)-pt_M #85
vec_r05=np.zeros(3,dtype=np.float64) ; vec_r05 = 0.5*(pt_1+pt_2)-pt_M #12
vec_r06=np.zeros(3,dtype=np.float64) ; vec_r06 = 0.5*(pt_2+pt_3)-pt_M #23
vec_r07=np.zeros(3,dtype=np.float64) ; vec_r07 = 0.5*(pt_3+pt_4)-pt_M #34
vec_r08=np.zeros(3,dtype=np.float64) ; vec_r08 = 0.5*(pt_4+pt_1)-pt_M #41
vec_r09=np.zeros(3,dtype=np.float64) ; vec_r09 = 0.5*(pt_1+pt_5)-pt_M #15
vec_r10=np.zeros(3,dtype=np.float64) ; vec_r10 = 0.5*(pt_2+pt_6)-pt_M #26
vec_r11=np.zeros(3,dtype=np.float64) ; vec_r11 = 0.5*(pt_3+pt_7)-pt_M #37
vec_r12=np.zeros(3,dtype=np.float64) ; vec_r12 = 0.5*(pt_4+pt_8)-pt_M #48
vec_r13=np.zeros(3,dtype=np.float64) ; vec_r13 = 0.5*(pt_1+pt_6)-pt_M #16
vec_r14=np.zeros(3,dtype=np.float64) ; vec_r14 = 0.5*(pt_2+pt_7)-pt_M #27
vec_r15=np.zeros(3,dtype=np.float64) ; vec_r15 = 0.5*(pt_4+pt_7)-pt_M #47
vec_r16=np.zeros(3,dtype=np.float64) ; vec_r16 = 0.5*(pt_4+pt_5)-pt_M #45
vec_r17=np.zeros(3,dtype=np.float64) ; vec_r17 = 0.5*(pt_6+pt_8)-pt_M #68
vec_r18=np.zeros(3,dtype=np.float64) ; vec_r18 = 0.5*(pt_1+pt_3)-pt_M #13
```

12. compute $\vec{g}_e$ by means of Eq. (113.1)

$$\vec{g}_e = \mathcal{G}\rho_0 \sum_e L_e \boldsymbol{E}_e \cdot \vec{r}_e = \mathcal{G}\rho_0 \left(L_1 \boldsymbol{E}_1 \cdot \vec{r}_1 + L_2 \boldsymbol{E}_2 \cdot \vec{r}_2 + ... L_{17} \boldsymbol{E}_{17} \cdot \vec{r}_{17} + L_{18} \boldsymbol{E}_{18} \cdot \vec{r}_{18} \right)$$

13. compute U_e by means of Eq. (**??**)

14. compute $\vec{g} = -\vec{g}_e + \vec{g}_f$

Hexahedron: mascons

The volume of the hexahedron is computed by means of a 2^3-quadrature and since its density is constant each mascon is assigned the same mass. The number of mascons is n^3 and the mascons are spanning the entire quadrilateral by means of a linear mapping. They are evenly spread in the reference cube $[-1, 1]^3$ and then mapped out to the real hexahedron. This is all happening in the function compute_gravity_hexahedron_mascons. There is however a problem with this approach. Each mascon is assigned the same mass, which is fine if they are all equidistant. They are equidistant in the r, s, t space but not necessarily in the x, y, z after the mapping, especially if the sides of the hexahedron are not planes. This yields errors that cannot be remedied by using more points (see tests 3,4). If the opposite faces of the hexahedron are parallel and planar then this approach works fine, see tests 1&2.

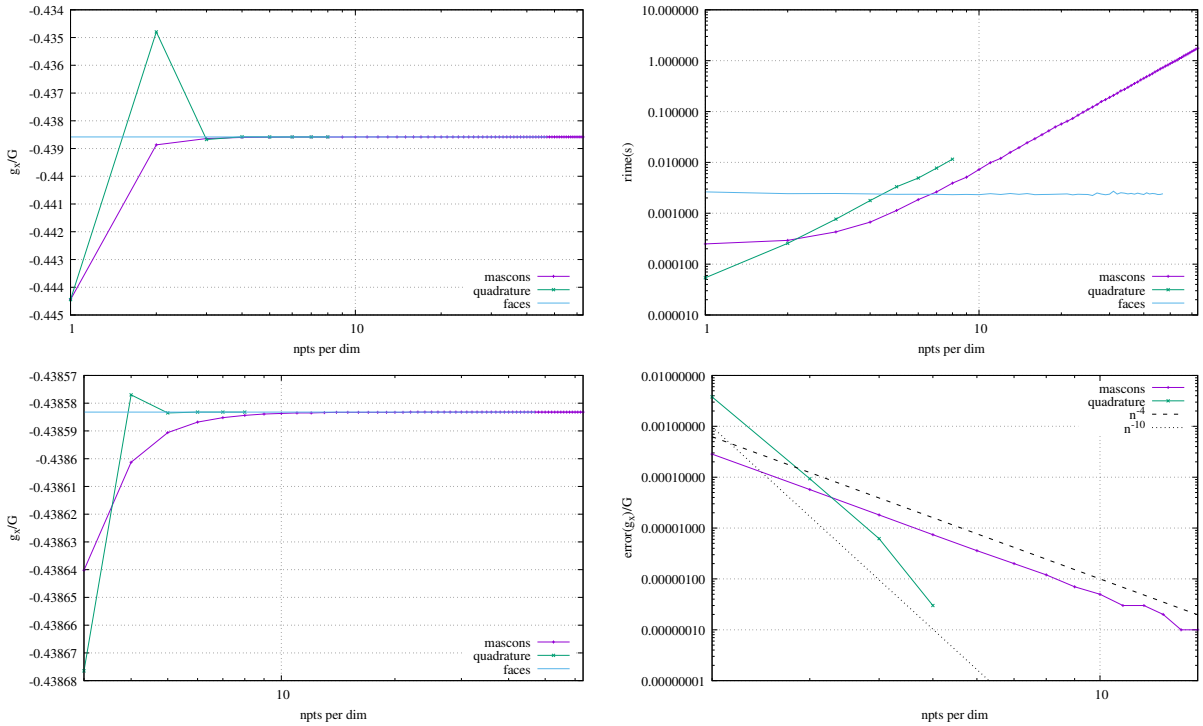
This lead me to write second function compute_gravity_hexahedron_mascons2 which first tesselates the unit cube in n_per_dim^3 sub cubes, computes the image of each node via the Q_1 mapping (and therefore the mapped out sub cube) and then compute the volume of each sub-hexahedron cell by means of the method explained in Section 15.31.

Hexahedron: Gauss quadrature

This follows a standard Gauss quadrature procedure as explained in Section 7. As we know the kernel that needs to be integrated is not a polynomial so that there is no optimal number of quadrature points. This is implemented in the function compute_gravity_hexahedron_quadrature. Quadratures from 2^3 up to 8^3 are implemented.

Hexahedron test 1: the cuboid (prism)

We first consider the unit cube $[0,1]^3$. Density is $\rho_0 = 1$ and we set $\mathcal{G} = 1$ for simplicity again. The gravity is measured at point $M = (2, 0.5, 0.5)$. Since the point is on a line perpendicular to a face that goes through its middle we expect $g_y = g_z$ by symmetry, and we indeed recover this, down to machine precision. There is no analytical solution but all three methods do converge to the same value (and it is expected that the 'faces' method is exact or at least the most accurate).



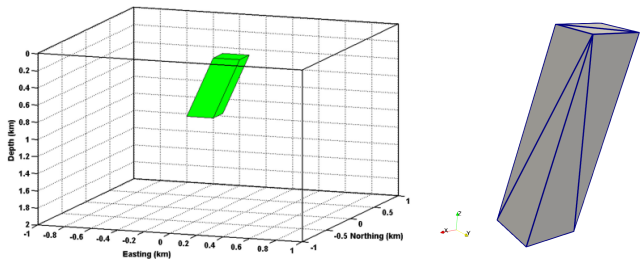
Results and cpu time for all three methods as a function of the used number of points (per dimension).

Obviously one does not need to run the compute_gravity_hexahedron_mascons function for every value of n_per_dim since it does not depend on it, but it allows for easier plotting and a sense of the expected variation in the timing measurement from call to call.

Although none of the functions have been written with efficiency in mind, we can look at the cpu time needed and we find that the faces approach is fastest for n_per_dim$\geq$ 8. Looking at the accuracy we see that the mascons is the least accurate at low n_per_dim. Gauss quadrature becomes reasonably accurate for n_per_dim=6. Looking at the error convergence (where the value obtained with the faces approach is used as reference) we see that the quadrature approach converges much faster than the mascons approach.

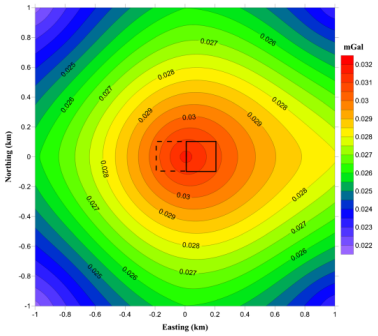
Hexahedron test 2: the dyke

This originates in Uwiduhaye, Mizunaga, and Saibi [337] (2019). It consists of a dyke with a density difference of $+170\text{kg}\,\text{m}^{-3}$, and a model space of 2km $\times$ 2km $\times$ 2km was employed. The dyke model extends from 0.2 to 0.9 km depth (the coordinates of all 8 vertices is given in the article). The 3-D view of buried dyke is shown here:



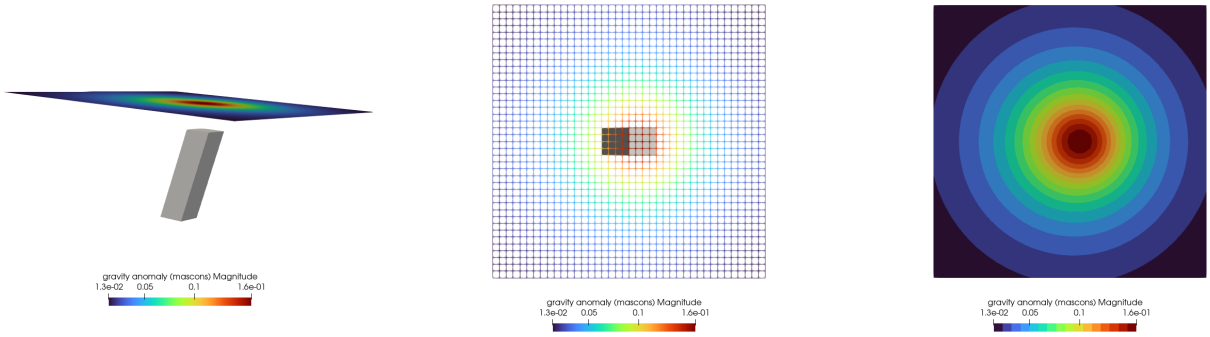
Left: figure from the paper; Right: Paraview rendering.

In the paper a 11×11 grid in the horizontal plane with observation points at 0.1-km intervals was used. The gravity anomaly was calculated using the Gauss-Legendre integration, and calculation result is shown below:



Surface gravity anomaly distribution for the inclined dyke block with the density difference of $+170\,\mathrm{kg\,m^{-3}}$.

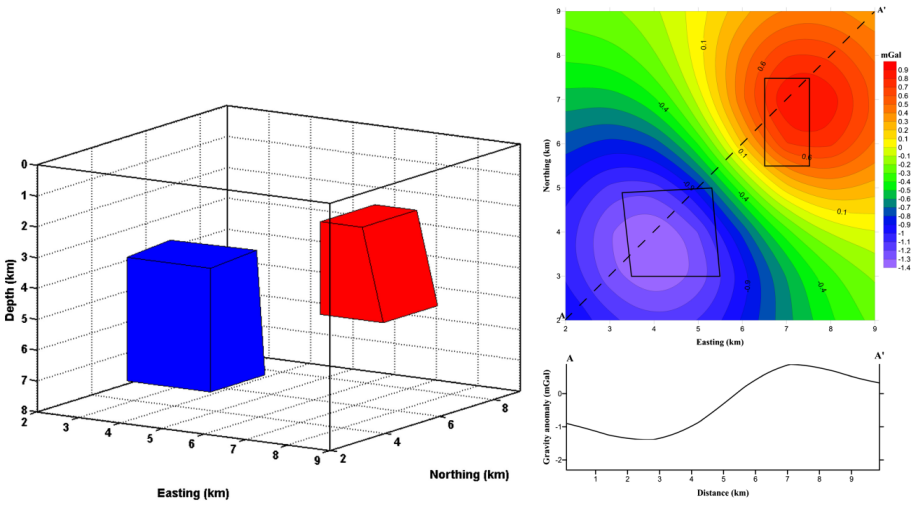
At this stage, looking at the figure above, I have my doubts: given the orientation of the hexahedron (look at the dashed line) why is gravity higher on the other side of it? (check the yellow 'ring'). Something is off, and this is corroborated by my own measurements which yield the same field for all three methods (so only one is shown):



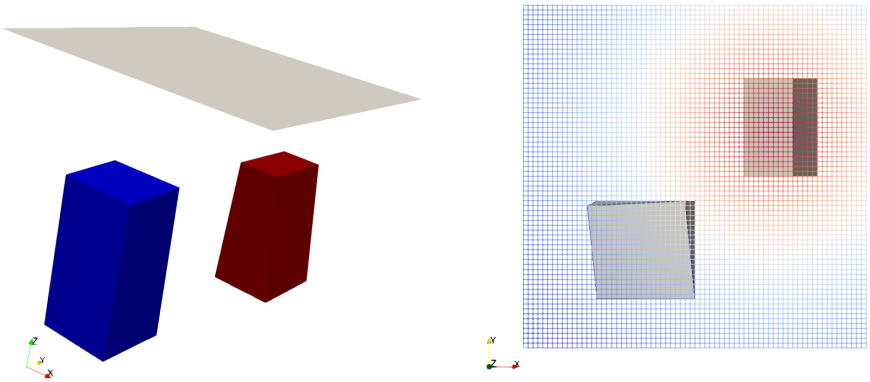
Results obtained in mGal. The asymmetry in the surface signal is not very pronounced.

Hexahedron test 3: two buried blocks

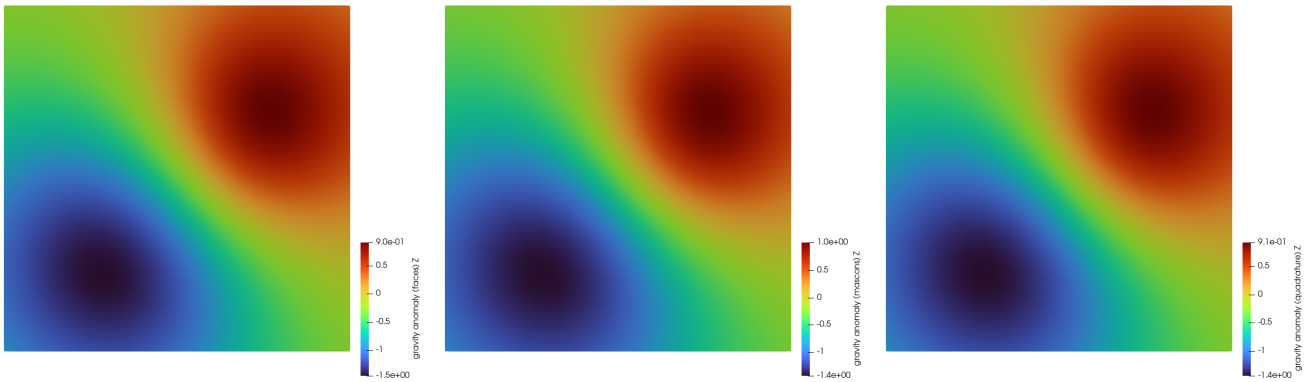
This also originates in Uwiduhaye, Mizunaga, and Saibi [337] (2019). The coordinates of the vertices for both blocks are given in Table 2 of the publication.



Left: Two blocks model with different densities. The red block has positive density difference ($+400\ \mathrm{kg\,m^{-3}}$), and blue block has negative density difference ($-400\ \mathrm{kg\,m^{-3}}$). Right: Surface gravity anomaly map (top) for the buried blocks with different densities and the gravity profile (bottom) along the A-A' line. Since the values range from negative to positive, this is likely the g_z component that is shown, not $|\vec{g}|$.

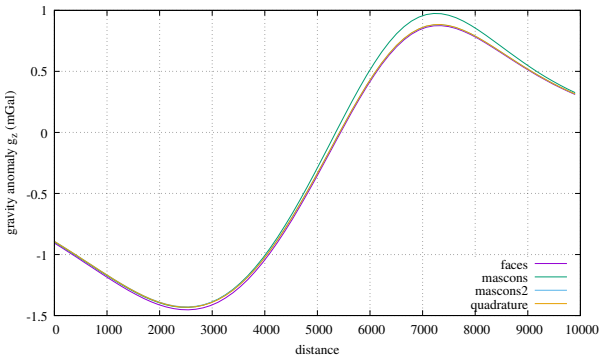


I use 10^3 mascons and 7^3 quadrature points in what follows. We find that all methods yield very similar results and that the range of values matches the one in the publication:



g_z Results obtained on a 71×71 grid. From left to right: faces, mascons, quadrature.

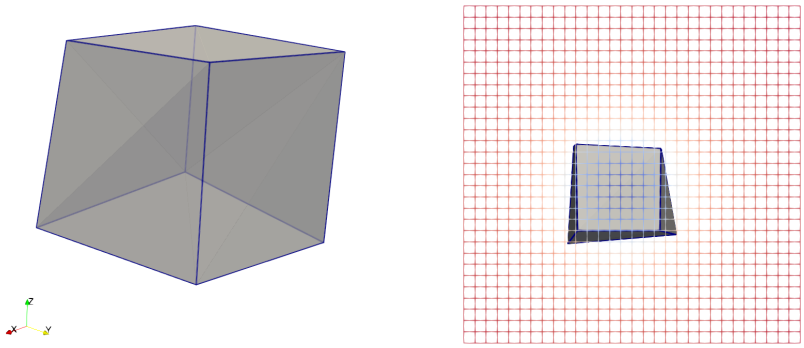
Looking at the $A - A'$ section we see that the 'naive' mascons results are a bit off with regards to the other two methods and the 'fancy' mascons method. Note that the mascons2 function is much slower than the others.



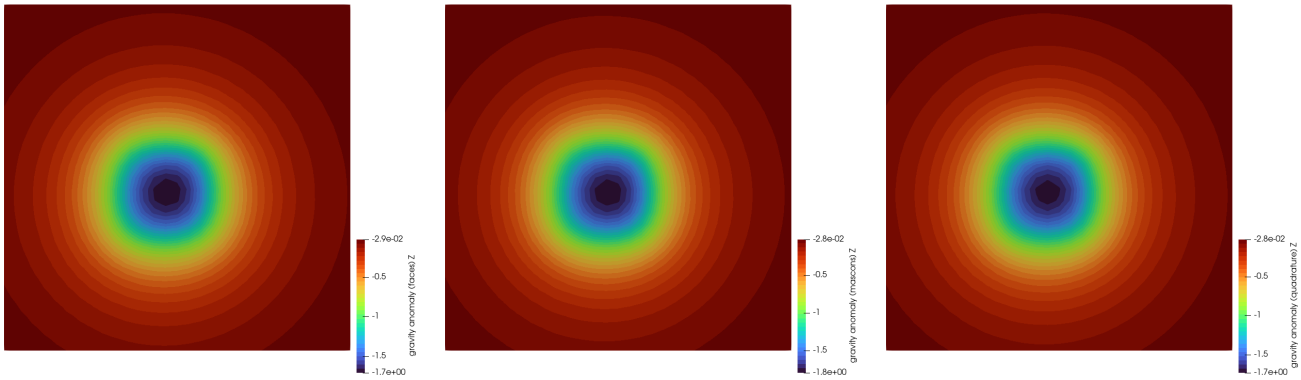
g_z on the A-A' line of the figure above.

Hexahedron test 4: the irregular hexahedron

This is a simple test which I have designed. The hexahedron is such that none of its faces is a plane, and that opposite faces are obviously not parallel. I created this test when I was investigated why the 'naive' mascons did not yield identical results as the other 2 methods and this lead me to write the 'clever' mascons approach.



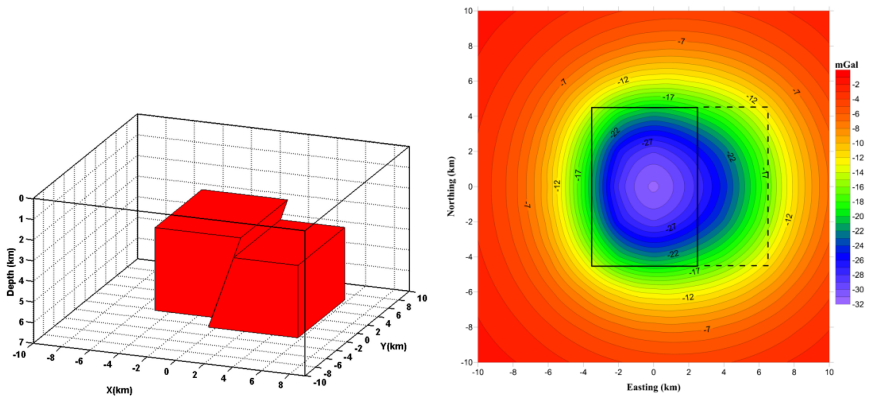
All three methods agree reasonably well.

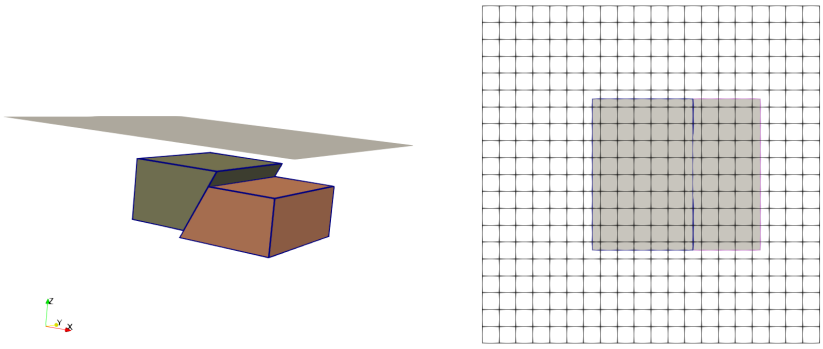


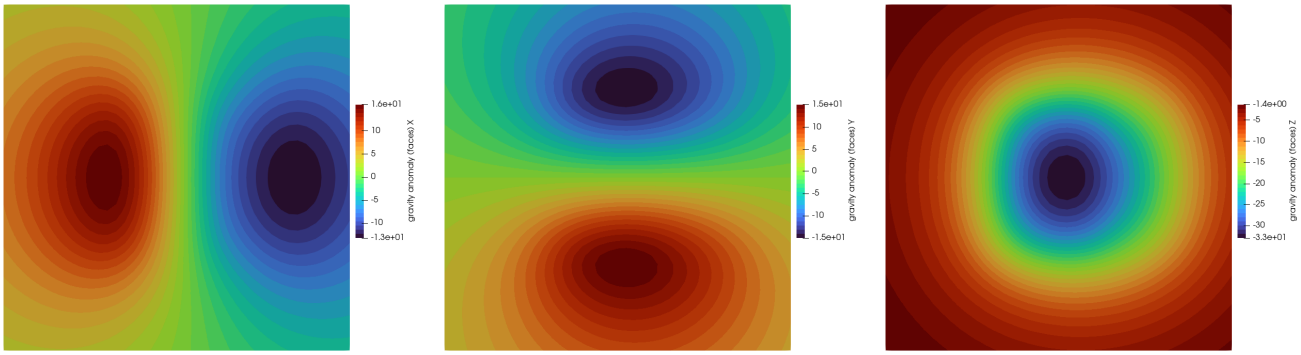
Hexahedron test 5: reverse faulted blocks

This also originates in Uwiduhaye, Mizunaga, and Saibi [337] (2019). The coordinates of the vertices for both blocks are given in Table 2 of the publication.

We are now dealing with a reverse faulted block with the hanging wall located at a depth of 2 km and the footwall at a depth of 3 km. The footwall block extends to the depth of 6.5 km, and a uniform background density of 2600 kg m^{-3} is assumed with a density contrast of -500 kg m^{-3}. The model space is $20km \times 20km \times 7km$ in the $x-$, $y-$, and $z-$directions, respectively. The gravity anomaly distribution is calculated and presented in the xy-plane.





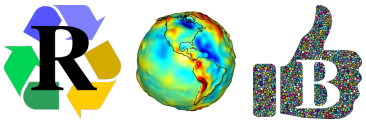


Left to right: g_x, g_y and g_z components.

TODO:
- potential is returned by all functions but never plotted above
- compute gradients for all three methods.
- find better benchmarks. For regular cuboid, compare with function in stone 84

Stone 114

Joint Gravity and Stokes inversion of a 2D circular anomaly

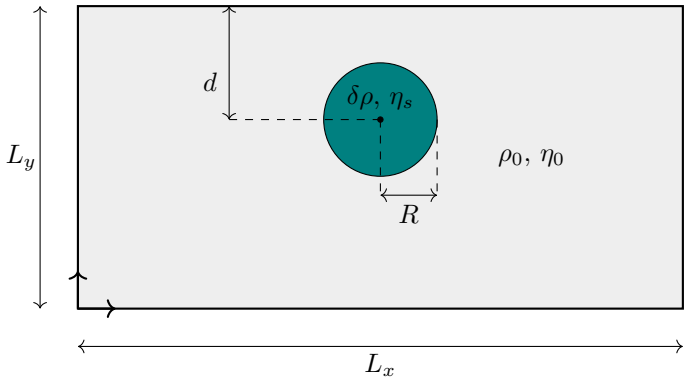


python_codes/fieldstone_114/text.tex

P Codes at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_114`

The python stone was developed in collaboration with W. Klessens. The julia stone was developed in collaboration with J. Jansen.

This STONE is inspired by Baumann, Kaus, and Popov [11] (2014). The setup is as follows:



When the sphere is lighter than its surrounding it goes up and creates a divergent flow at the surface. As explained in Baumann, Kaus, and Popov [11] an analytical solution exists for the horizontal component of the velocity at the surface in 3D. Also, the lower density of the sphere (or rather the density difference $\delta\rho$ between the sphere and its surrounding) yields a gravity field above the sphere that is lower and for which an analytical solution exists too.

Assuming we have determined what the depth of the sphere is (and assuming that it is rigid by taking its viscosity $\eta_s >> \eta_0$), then the two main +unknowns are its density difference $\delta\rho$ with regard to the surrounding mantle and its radius R.

To start with, we can compute analytically the gravity and velocity that a $\delta\rho^\dagger = -300$ and $R^\dagger = 50$ km disc generates at the surface of the model, write a simple inversion code which would operate in the $(R, \delta\rho)$-space and automatically looks for the real values of the sphere by computing a gravity misfit ξ_g and velocity misfit ξ_v and using these quantities to zero-in on the correct values of $\delta\rho$ and R.

Three ideas:

- axisymmetric
- full 3D
- try JIT? julia?

gravity forward model - analytical solution

We consider a circular inclusion at half the 2D domain of 1000 km in x- and 500 km in y-direction. Because of the evident symmetry of the problem we restrict the calculations to the right half of the domain so that now it is a square of size $L_x = L_y = 500$ km and the sphere is at coordinates $(x_c, y_c) = (0, Ly/2)$.

In the case of a disc, the gravity in the $x, y-$plane is given by

$$\vec{g}(x,y) = \mathcal{G}\frac{\pi R^2 \delta\rho}{(x-x_c)^2 + (y-y_c)^2}\begin{pmatrix} x-x_c \\ y-y_c \end{pmatrix}$$

In the case of a sphere,

$$\vec{g}(x,y,z) = \mathcal{G}\frac{\pi R^2 \delta\rho}{(x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2}\begin{pmatrix} x-x_c \\ y-y_c \\ z-z_c \end{pmatrix}$$

Gravity is computed by transforming the integral

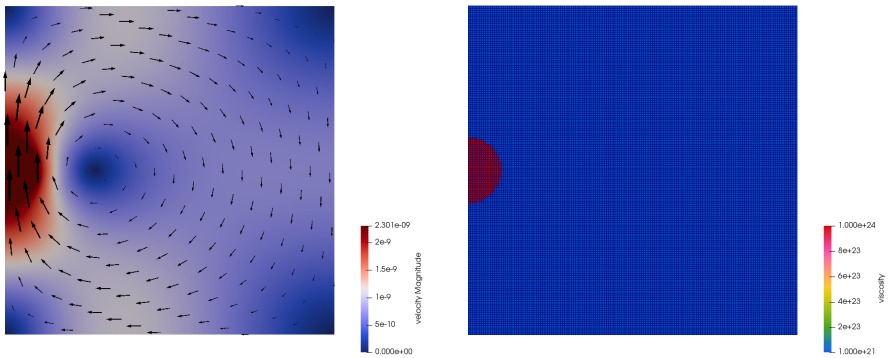
$$\vec{g}(x,y) = \iiint_V \mathcal{G}\frac{\rho(x',y',z')}{(x-x')^2 + (y-y')^2 + (z-z')^2}dx'dy'dz'$$

over the domain by a sum of elemental integrals which are then computed by means of the same quadrature as the FE matrix. Since half the disk is missing, we on-the-fly take the missing half into account by mirroring the quadrature points and adding their contribution.

Because we are only interested in the gravity signal generated by the disc/sphere, a boolean array `elt_in_sphere` is created and set to true for those elements for which at least one quadrature point is inside the sphere. This array is then used to target the contribution of the elements 'inside' the sphere.

Stokes forward model - analytical solution

The surface velocity signal of a rising sphere is computed in Appendix A of Baumann, Kaus, and Popov [11] (2014). It is based on Brenner [38] (1961), Stimson and Jeffery [311] (1926) and Pasol, Chaoui, Yahiaoui, and Feuillebois [249] (2005). See also Burgess [53] (1916). The solution is only available in 3D with axisymmetric geometry. The analytical expression is rather complex and in order to keep simple for now, we will use a different reference velocity quantity to carry out the inversion. In the folder `reference_code` a modified version of STONE 2 is placed. It computes the velocity field for the reference case at a 150x150 resolution. The root mean square velocity $\nu_{rms} = 7.740432578645655e-10$ m s^{-1} will serve as our reference value to drive the inversion.



Sphere radius is $R = 50$ km, $\rho_m = 3000$, $\delta\rho = -300$, $\eta^\star = 10^3$.

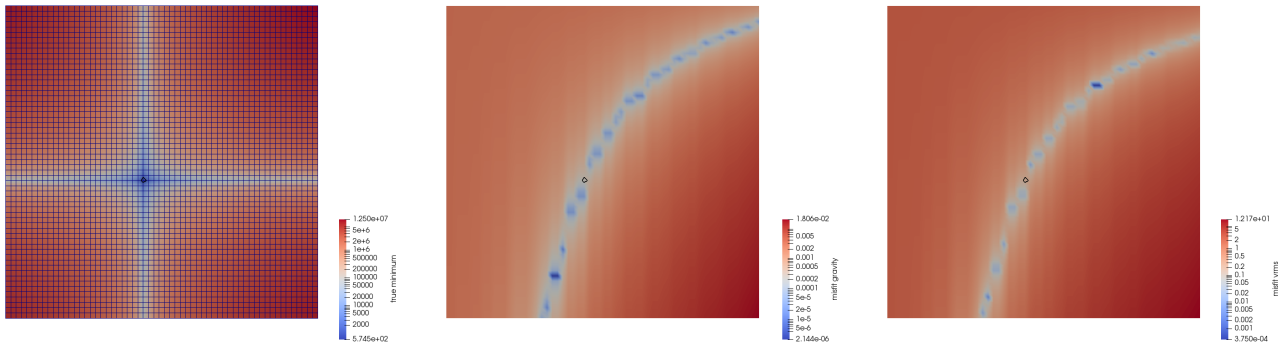
The code

In order to produce a reasonably fast Stokes solver the code relies on the $Q_1 \times P_0$ element with a penalty approach (see for example STONE 01). The Cartesian domain is overlain with a mesh counting $nel_x \times nel_y = nel$ elements which are all square, so that the Jacobian of the transformation between global and local coordinates is diagonal. Its value has been hard-wired in the code so as to save time. The entire Stokes solver + gravity calculator is inserted into a single function `compute_misfits` in the forward.py file.

Note that depending on the mesh resolution a call to the function will take betweem 1s (50x50) 2.5s (64x64), and 8s (100x100).

Inversion

The first inversion code driver1.py is very simple. The user defines the bounds of the parameter space, i.e. $\delta\rho_{min}$, $\delta\rho_{max}$, R_{min} and R_{max}. Then a regular grid composed of $n_{\delta\rho} \times n_R$ is used to compute both misfits ξ_g and ξ_v at each node of the grid and these are exported in a vtu file. No refinement or clever search takes place.



horizontal axis is radius R, vertical axis is $\delta\rho$. 60x60 points. about 2.5h. Left: location of the true minimum, i.e. the reference model; Middle: gravity misfit; Right: vrms misfit.

Stone 115

$Q_1 \times P_0$-stab element for isoviscous flow



python_codes/fieldstone_115/text.tex

2D
box
Q1P0
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
buoyancy_driven
paraview_output
pressure_normalisation

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_115`

When it comes to the stabilisation of the $Q_1 \times P_0$ elements, we have seen in Section **??** that mulitple types co-exist and that a tuning parameter ϵ plays a role. Despite the published literature on the topic, the following questions remain:

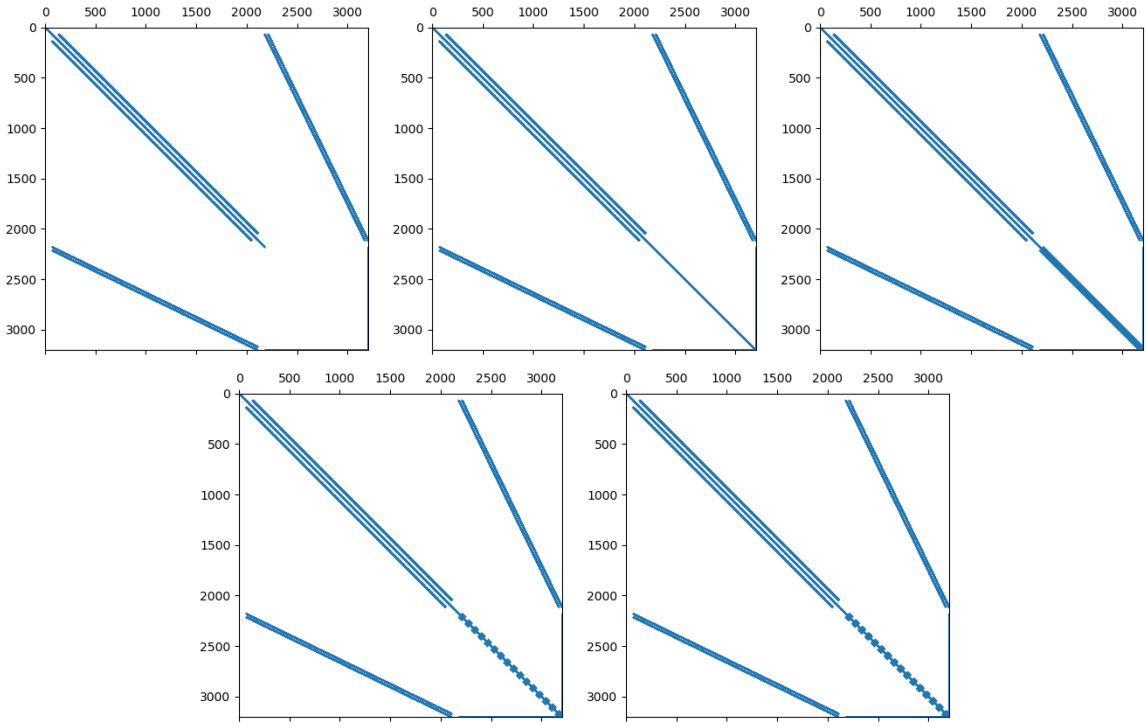
- how to deal with vicosity contrasts ?
- which stab is best for real life geodynamics ?
- how to deal with not square mesh ?
- how to choose the ϵ value ?
- look at buoyancy driven flows. lith pressure!
- I have implemented penalty, global, local and macro-element. what about the other methods, eg bodg06 ?
- concretely how would PCG converge?

The code is based on STONE 14.

Velocity divergence is measured at the center of the element.

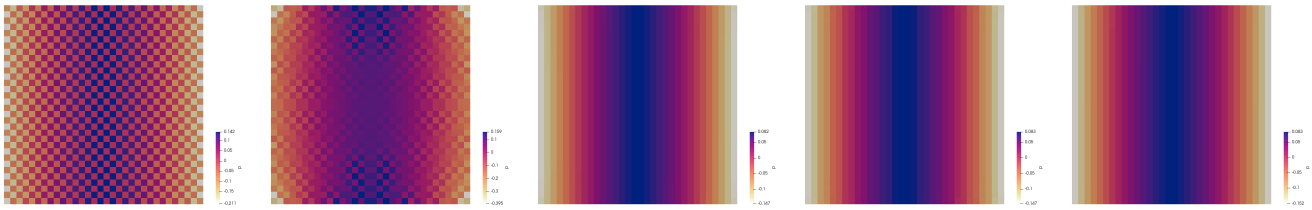
note that in case of penalty, ϵ should be $\sim 10^6$ larger than viscosity.

Donea & Huerta manufactured solution



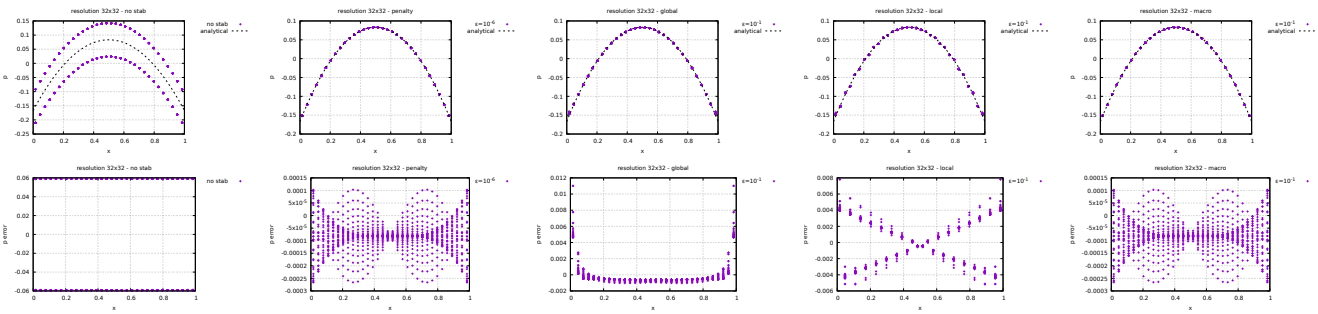
32x32. From left to right: parsity pattern of the FEM matric for no stabilisation, penalty, global, local, macro-elt.

We find that all three stabilisation $\mathbb{C}$ matrices are enough to suppress the chequerboard mode:



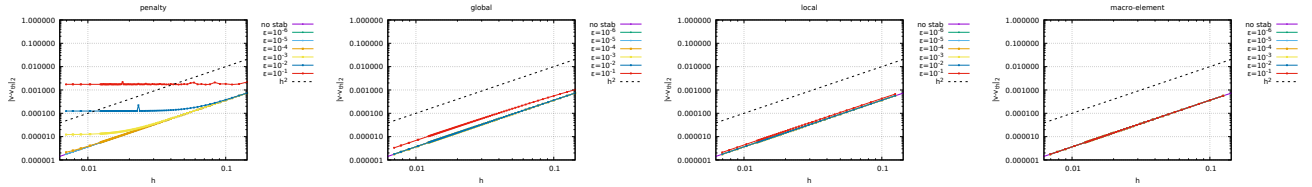
resolution 32×32. From left to right: no stabilisation, penalty, global, local, macro-elt. $\epsilon = 10^{-1}$

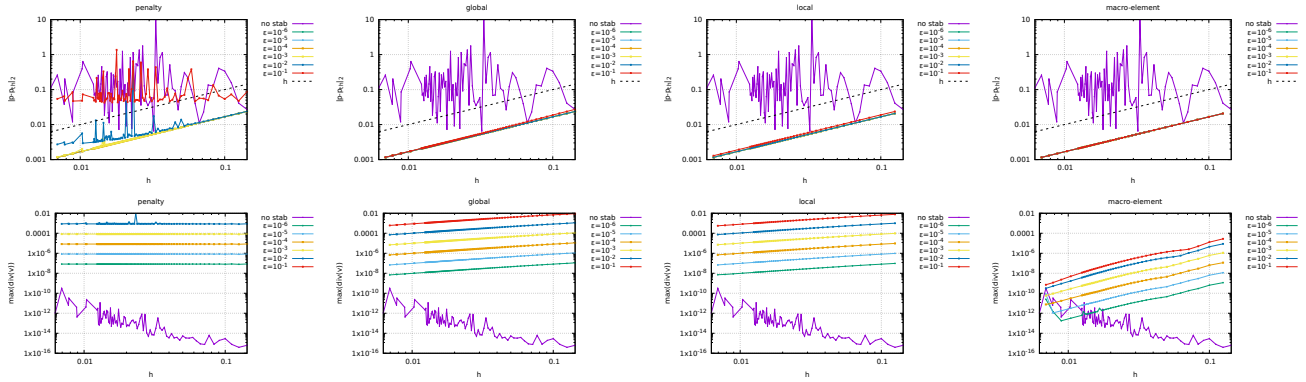
We can now explore the influence of the stabilisation parameter ϵ. Obviously, if too low it won't have any effect as the $\mathbb{C}$ matrix then tends to zero. If too high, then we expect it to perturb the solution too much. The pressure field is shown in the following figure for various values of ϵ,



32x32. pressure profile.

Let us now turn to the velocity and pressure error convergence rates:

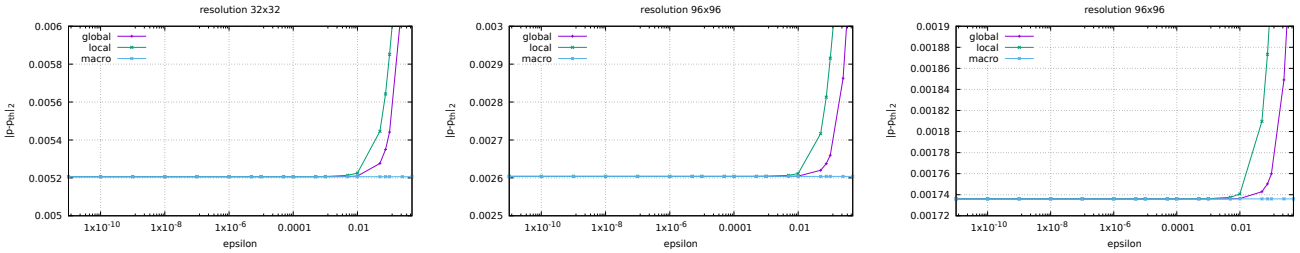




Velocity and pressure errors as a function of the element size for meshes 6x6 to 144x144

We find that the error convergence rate for velocity is always quadratic as expected, while the error convergence for pressure becomes monotonous and linear as expected too.

It also appears that the macro-element technique is the least sensitive to the value of ϵ, although it is rather surprising that even very low values of ϵ manage to stabilise the system so well. could it be because this benchmark is very smooth?



Influence of ϵ parameter on pressure error.

Conclusion: macro-element technique yields best results: lower velocity divergence, least sensitive to ϵ value, block matrix,

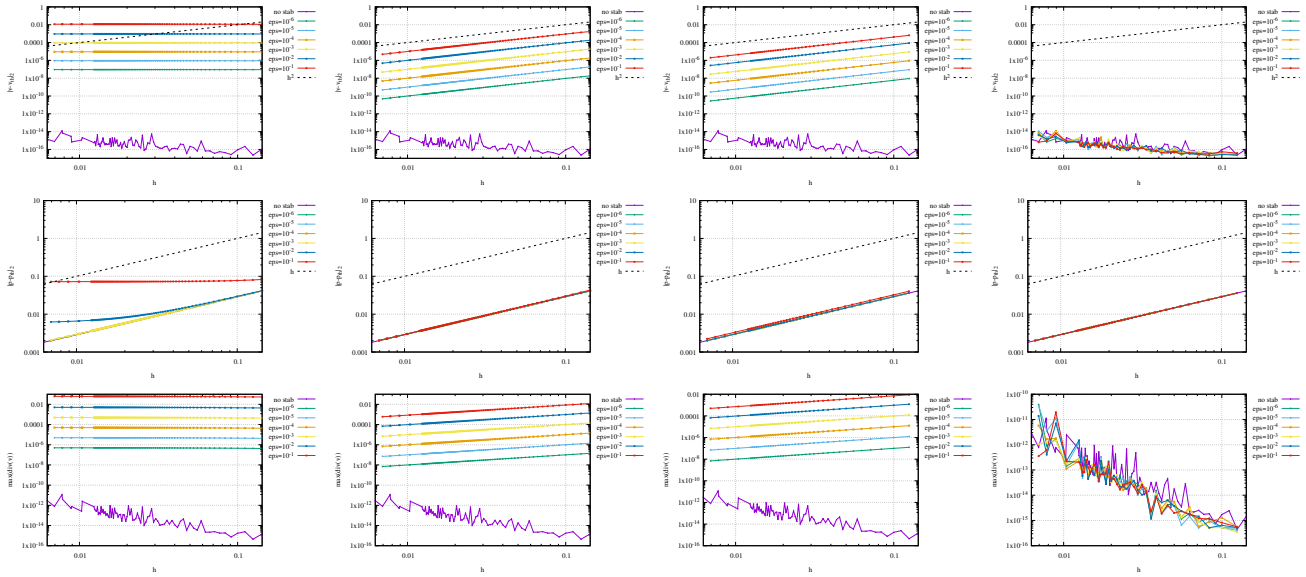
I should monitor the condition number of the matrix...

The aquarium

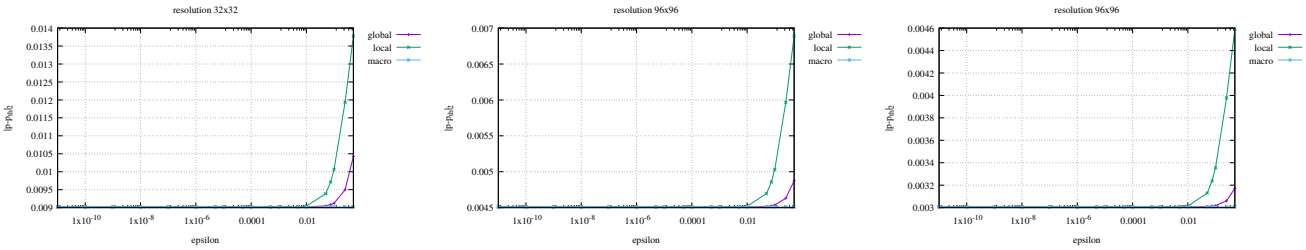
The domain is still a unit square. Boundary conditions are no slip on all sides. Density and viscosity are set to 1. Gravity is vertical with $\vec{g} = -\vec{e}_y$. The analytical velocity is obviously $\vec{v} = \vec{0}$ and the analytical pressure is $p(x, y) = 0.5 - y$ (which fulfils $\int_\Omega p dV = 0$).

Because the analytical velocity field is a constant, it can be represented exactly by bi-linear functions so we expect the velocity error to be at machine precision independently of the resolution. However, because of the presence of the stabilisation term, we find in the penalty case that it is proportional to ϵ and in the global and local cases that it converges quadratically.

The analytical pressure field, on the other hand is a linear polynomial which cannot be represented exactly by a set of discontinuous C^0 functions. We then expect and indeed recover a linear convergence.



Velocity and pressure errors as a function of the element size for meshes 6x6 to 144x144

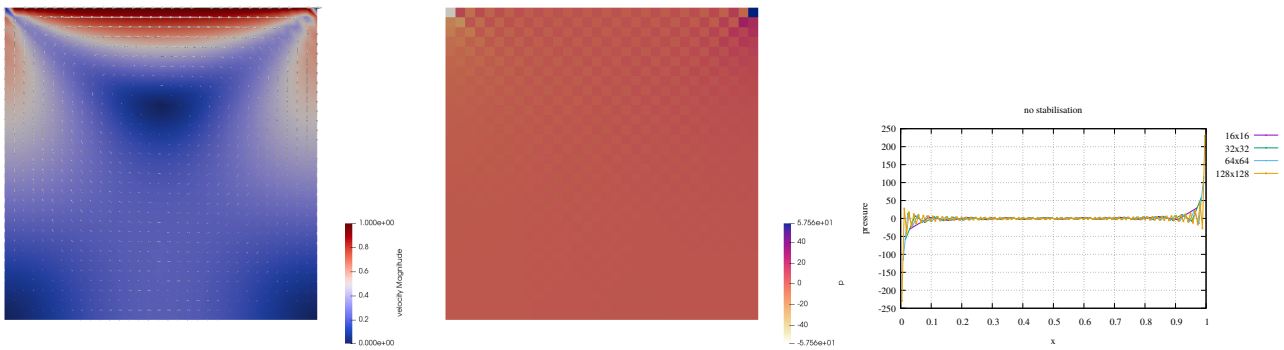


Influence of ϵ parameter on pressure error.

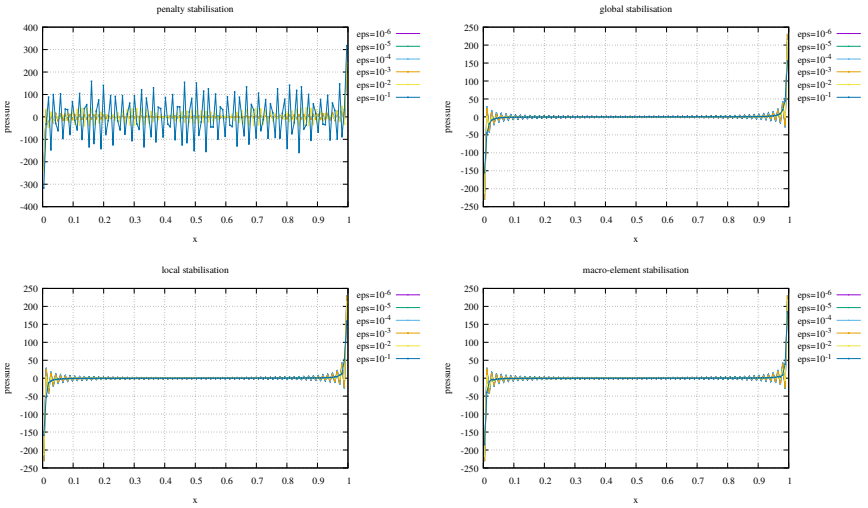
Lid driven cavity

This is not a benchmark but a simple setup which is not very physical. The most right and left points of the lid have zero velocity ('non-leaky cavity').

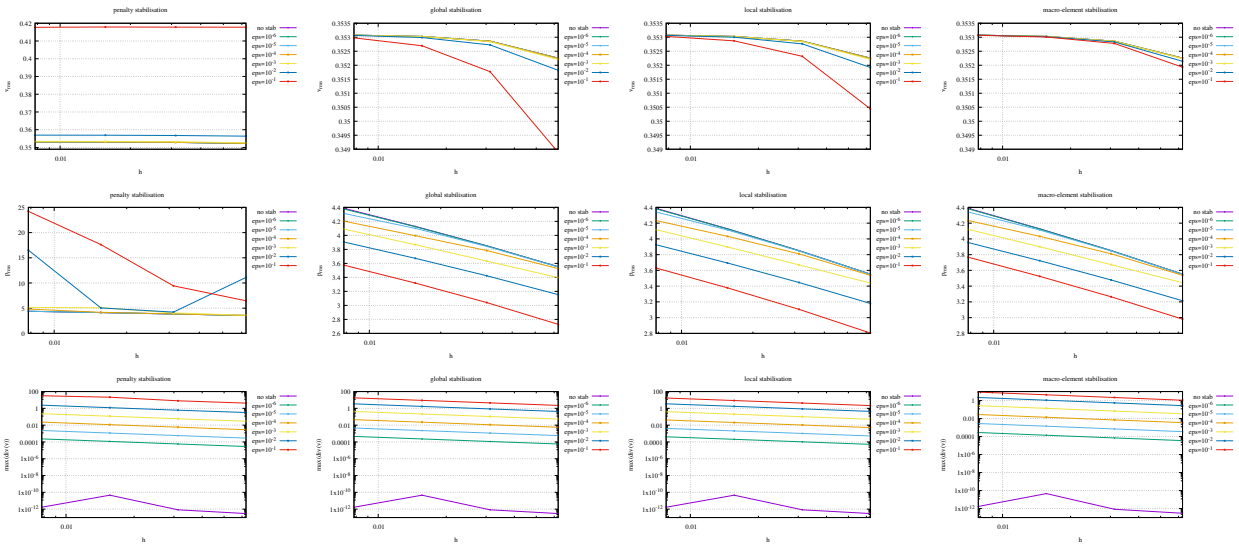
We find that in the absence of stabilisation the pressure chequerboard mode is very strong:



No stabilisation: velocity and pressure at 32x32. Right: pressure at the top at various resolutions.



Pressure at the top at 128 × 128 resolution.

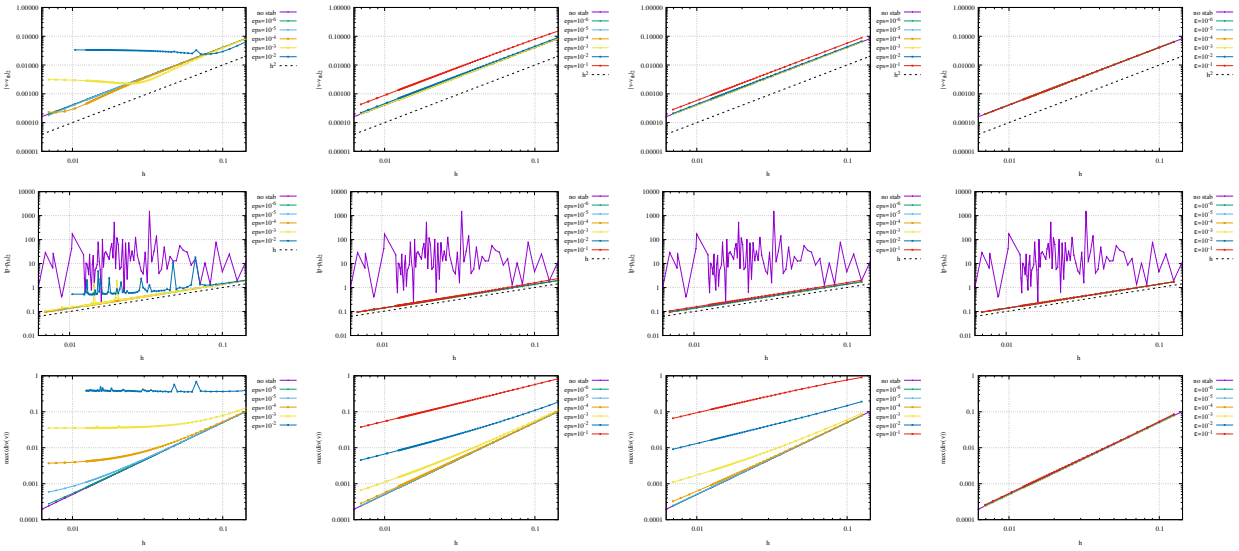


Left to right: penalty, global, local, macro-element

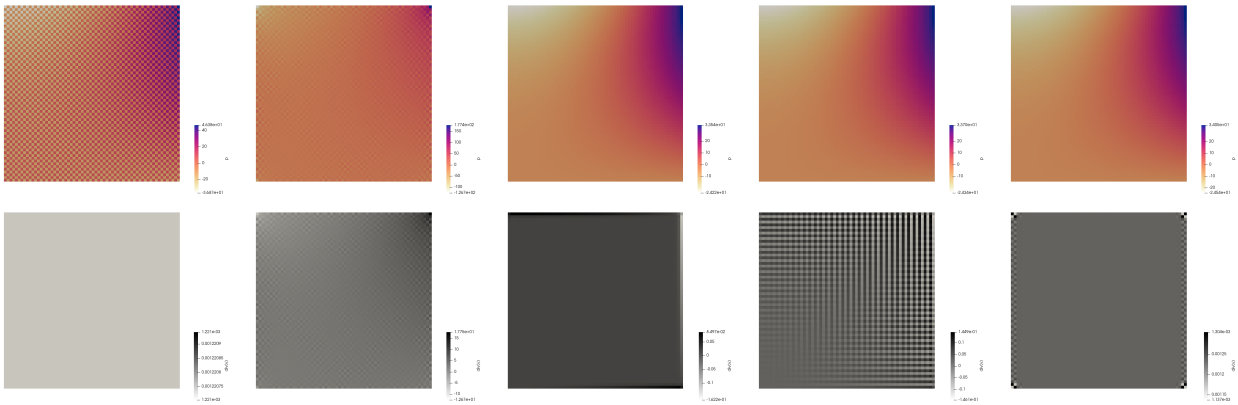
Manufactured solution of Burman and Hansbo [54]

See section **??**. The velocity and pressure fields are given in the unit square by

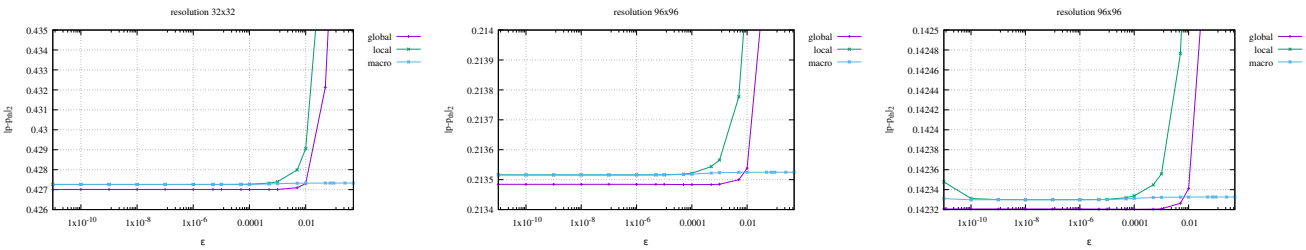
$$\begin{aligned} u(x,y) &= 20xy^3 \\ v(x,y) &= 5x^4 - 5y^4 \\ p(x,y) &= 60x^2y - 20y^3 - 5 \end{aligned} \tag{115.1}$$



Velocity and pressure errors as a function of the element size for meshes 6x6 to 144x144. Left to right: penalty, global, local, macro-element



64x64, epsi=0.1. Left to right: no stab, penalty, global, local, macro-elt. Top row is pressure, bottom row is elemental velocity divergence.



Influence of ϵ parameter on pressure error for global, local, macro-elt

Stone 116

$Q_1 \times P_0$-stab element with viscosity contrasts

python_codes/fieldstone_116/text.tex

```
2D
box
Q1P0
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
buoyancy_driven
paraview_output
pressure_normalisation
```

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_116`

As we have seen in Section **??** the formulation of the $\mathbb{C}$ matrix is substantially less straightforward when viscosity variations/contrasts are present in the domain.

This stone builds on the previous STONE 115 and explores the various approaches for mutiple typical manufactured solutions (SolCx, SolKz, SolVi, ...).

In the case of the macro element stabilisation, an effective viscosity is computed based on the elemental viscosity inside the 4 elements.

SolKz

we start with SolKz, which showcases 6 orders of magnitude viscosity variation in the domain albeit in a very smooth way:

$$\eta(x,y) = \exp(2By) \qquad \text{with} \qquad B = 13.8155$$

Stone 117

circular inclusion with bespoke mesh

python_codes/fieldstone_117/text.tex

2D
box
Q1P0
Q2Q1
Q2Pm1
Q3Q2
Q4Q3
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
use_mesher
vrms
paraview_output

P Codes at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_01`

This fieldstone was developed in collaboration with Lukas van de Wiel.

This is work in progress. There is still much to do. The idea here is to compare the accuracy of quads on a specific benchmark (SolVi) as obtained on a bespoke mesh.

The mesh

The mesh(es) used in this STONE are obtained by means of the quilt mesher program written by Lukas.
INSERT HERE MANUAL
In order to generate the meshes that the STONE needs, we need to modify two files:

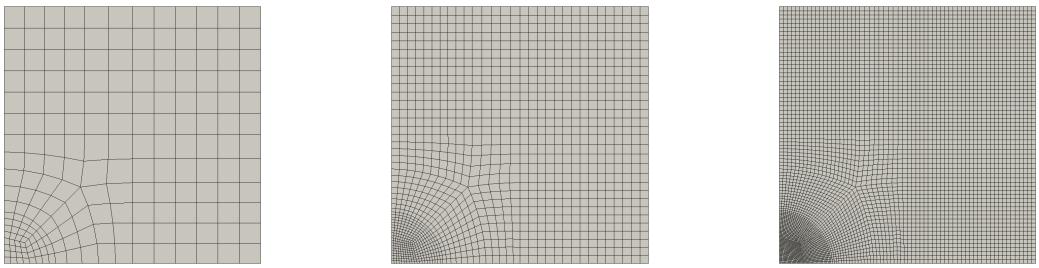
- `quilt.f`: elementType = Q0, Q1, Q2, Q3, or Q4
- `examples.f`: in subroutine meshThirteenPatches: nElemsBase=10

WARNING! As of now the edges of the elements are straight, so that the mid-egde nodes do not follow the curvature of the inclusion.

The benchmarks

I have implemented five finite element pairs: $Q_1 \times P_0$, $Q_2 \times Q_1$, $Q_2 \times P_{-1}$ (both mapped and unmapped flavours), $Q_3 \times Q_2$, $Q_4 \times Q_3$. Note that unfortunately the vtu format does not allow the representation of Q_3 or Q_4 fields. The basis functions for all these elements are to be found in Section **??**.

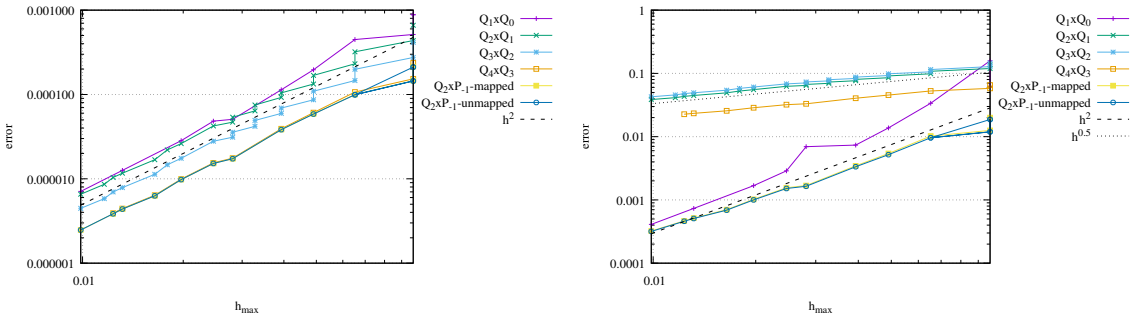
Two benchmarks are implemented in the code: Donea & Huerta (see Section **??**) and SolVi (see Section **??**). The first one is a manufactured solution which has a vertical and horizontal symmetry. As such the tailored mesh is of no use here. The second is of course why the mesher was developed in the first place: the viscosity contrast between inclusion and matrix aligns exactly with the mesh:



Mesh generated with nElemsBase=4,10,20.

`element`	FE pair
1	$Q_1 \times P_0$
2	$Q_2 \times Q_1$
3	$Q_3 \times Q_2$
4	$Q_4 \times Q_3$
5	$Q_2 \times P_{-1}$ (mapped)
6	$Q_2 \times P_{-1}$ (unmapped)

manifold for viscosity interface?
do Q2(8)Q1



SolVi benchmark: Velocity and pressure errors as a function of the largest element size in the mesh.

include figure of p at bottom

Stone 118

subduction initiation in the 80's ($Q_2 \times Q_1$)

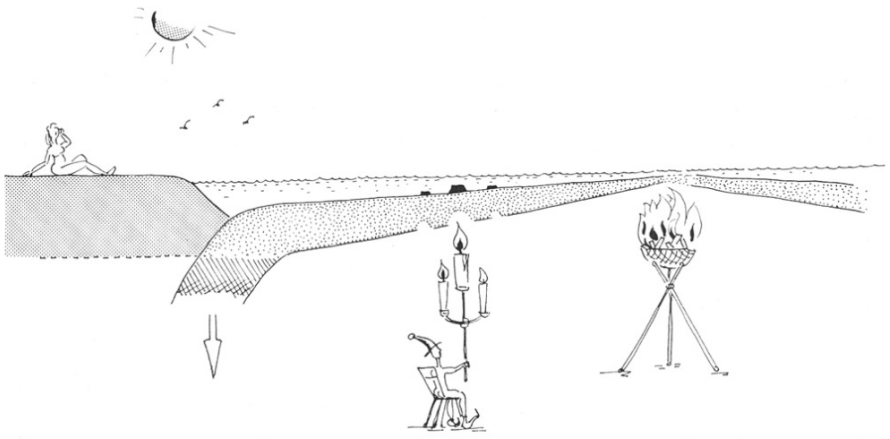


python_codes/fieldstone_118/text.tex

2D
box
Q2Q1
numerical_benchmark
stokes_equation
buoyancy_driven

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_118

The setup is taken from Matsumoto and Tomoda [224] (1983) and Fujimoto and Tomoda [119] (1985). The true motivation here is to replicate the 1985 study and have a reason to focus on the following most improbable figure in a peer reviewed article in geosciences:



Taken from Fujimoto and Tomoda [119] (1985).

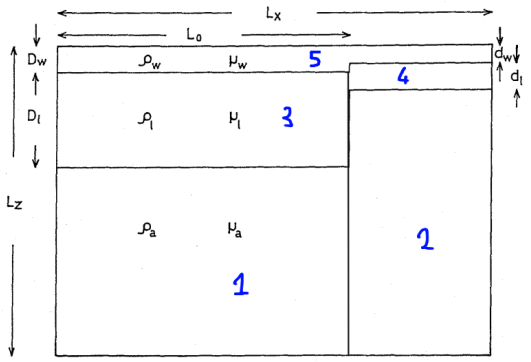
In the original paper (1983) the authors solve the incompressible isothermal Stokes equations with a stream function approach. The resulting equation is solved by means of the Finite Difference method and the Marker and Cell (MAC) method [149].

The domain is a 2D Cartesian box of size $L_x \times L_y = (400,180)$ km. There are three fluids in the domain: water ($\rho_w = 1030$ kg m^{-3}, $\eta_w = 10^{-3}$ Pa s), lithosphere ($\rho_l = 3300$ kg m^{-3}, η_l) and asthenosphere

($\rho_a = 3200$ kg m^{-3}, η_a). Boundary conditions are free-slip on all sides of the domain. Unfortunately the authors do not specify the mesh resolution that was used.

Six cases are considered[1]:

Case	η_l (Pa s)	η_a (Pa s)	domain size (km)
1	10^{22}	10^{21}	400×180
2	10^{22}	10^{20}	400×180
3	10^{22}	10^{19}	400×180
4	10^{23}	10^{21}	400×180
5	10^{22}	10^{20}	800×140
6	10^{22}	10^{19}	800×140



Taken from Matsumoto and Tomoda [224]. Dw=10km, Dl=50km, dw=8km, dl=10km, $L_0 = 3L_x/4$. Blue numbers indicate the numbering of each compositional field.

Interestingly the authors state: "The materials are initially placed so that uniform hydrostatic pressure holds at the bottom of the calculation area". On the left side, the pressure at the bottom is given by

$$p_b = (10 \cdot 1030 + 50 \cdot 3300 + 80 \cdot 3200)g_y = 431,300 g_y$$

while on the right side it is

$$p_b = (8 \cdot 1030 + 10 \cdot 3300 + 122 \cdot 3200)g_y = 431,640 g_y$$

which means that indeed the system is in isostatic equilibrium at the beginning of the simulation.

The authors use a CFL condition to determine the time step δt and use an arithmetic averaging scheme for both density and viscosity.

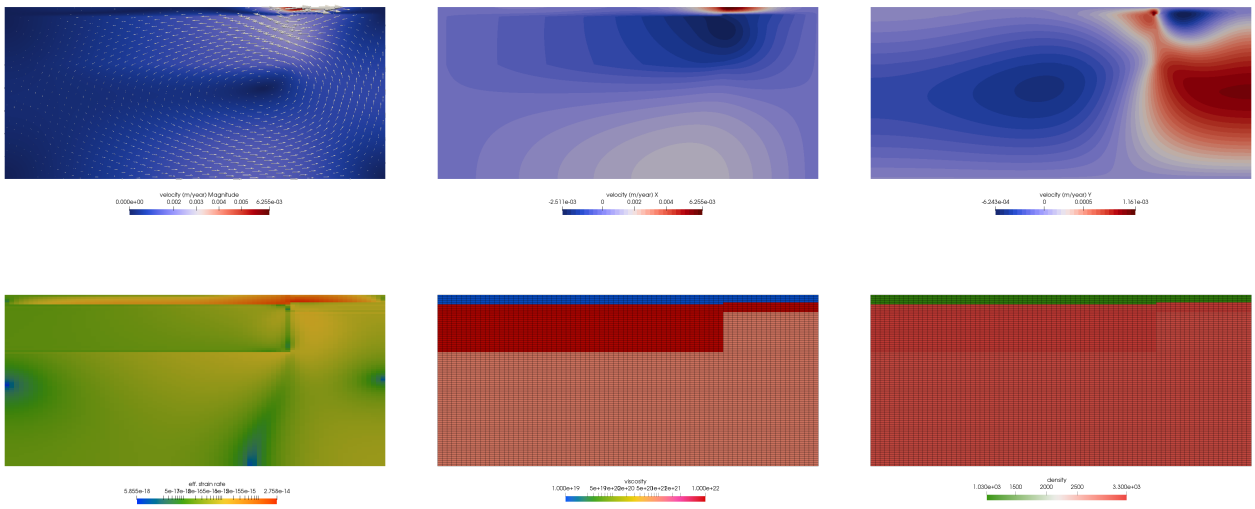
Using Aspect

I am using here ASPECT and the input files are included in the folder (in what follows I have used the one based on compositional fields but there is also a second prm file based on particles). It is straightforward since the flow is isothermal and incompressible and the domain a Cartesian box with free slip boundary conditions. A pressure normalisation is then necessary to remove the pressure nullspace and the pressure is then set to zero on average on the top boundary. A CFL condition is used with $C = 0.25$ and the models are run 50 Myr. Two repetitions are used in the horizontal directions so that elements are almost square. The mesh counts $128 \times 64 = 8192$ cells, i.e. a resolution of about 3 km which is barely enough to accurately follow the thin lithosphere.

This stone (instantaneous)

I have also written a $Q_2 \times Q_1$ code which is able to compute the solution of the problem at $t = 0$ (it stems from STONE 40). It does not feature any algorithm to advect interfaces or fluids and it is also quite naive and relies on full matrices for the blocks of the Stokes matrix. Nevertheless, it allows to compute the solution on a 80×90 mesh, i.e. a 5×2 km resolution with all fluid boundaries alligned with mesh boundaries:

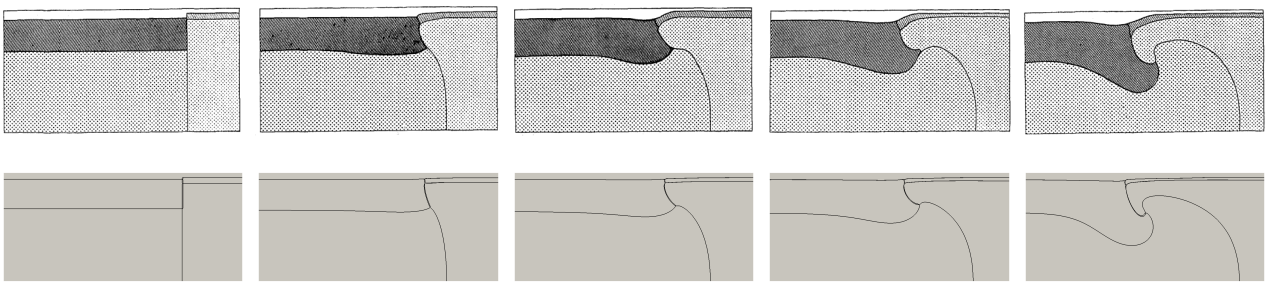
[1] Note that in the paper all viscosities are expressed in Poise, with 1 Poise=0.1 Pa s



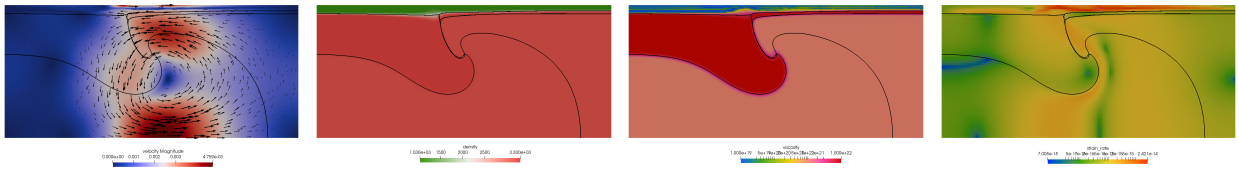
Results obtained with this STONE , resolution 80×90 for Case 1.

Results - Case 1

Results obtained with ASPECT agree nicely with those of the paper although the water layer and the thin lithosphere is much more deformed in their case. The run took approx. 6500 s to complete on a single core.



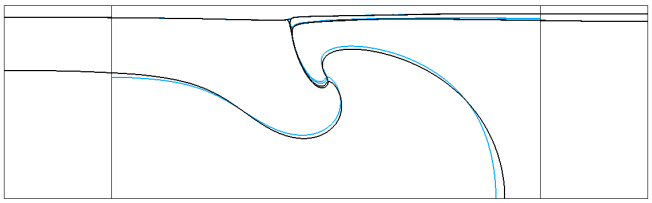
System at times 0, 10, 20, 30 and 50 Myr.



Various fields at $t = 50$ Myr.

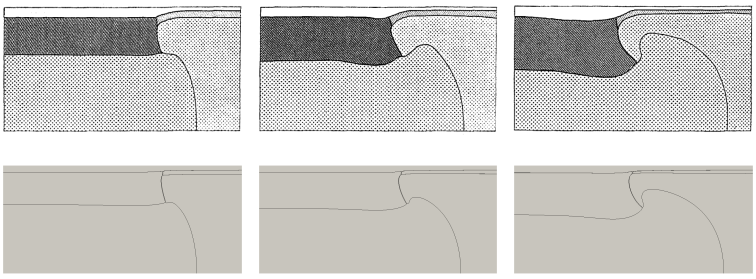
Results - Case 1*

This case is not in the original publication. It consists of Case 1 run with a 50% longer domain, all other things equal (with $L_0 = 2L_x/3$). We find that the domain size does not play a large role in the model evolution at all:

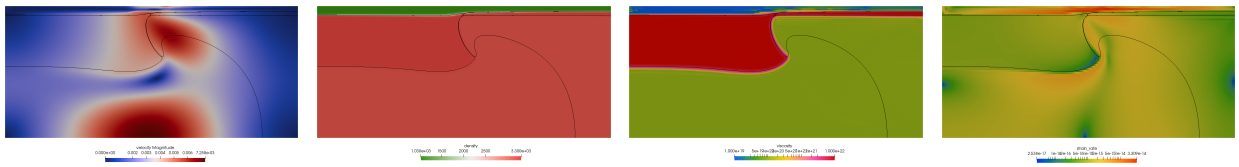


Compositional fields isolines at $t = 50$ Myr. Black lines correspond to 600x180 domain while blue lines correspond to 400x180km domain.

Results - Case 2



System at times 9.6, 16.6 and 25.1 Myr.



TODO: run stone 67 with pic!

Stone 119

on computing the lithostatic pressure in geodynamical models

python_codes/fieldstone_119/text.tex

2D
box
chunk
Q1
analytical_benchmark
numerical_benchmark
paraview_output

P Codes at https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_01

This stone was developed with input from Anthony Jourdon, Wolfgang Bangerth and Bob Myhill.

About lithostatic pressure

Let us look in section 2.2 of Turcotte and Schubert [335]: "The normal force per unit area on horizontal planes increases linearly with depth. The normal stress due to the weight of the overlying rock or overburden is known as the lithostatic stress or pressure."

Also, on wikipedia[1]: " Pressure is force magnitude applied over an area. Overburden pressure is a geology term that denotes the pressure caused by the weight of the overlying layers of material at a specific depth under the earth's surface. Overburden pressure is also called lithostatic pressure, or vertical stress. In a stratigraphic layer that is in hydrostatic equilibrium; the overburden pressure at a depth z, assuming the magnitude of the gravity acceleration is approximately constant, is given by:

$$P(z) = P_0 + g \int_0^z \rho(z) dz$$

where z is the depth in meters, $P(z)$ is the overburden pressure at depth d, P_0 is the pressure at the surface, $\rho(z)$ is the density of the material above the depth z, g is the gravity acceleration in $\mathrm{m\,s^{-2}}$.

In deep-earth geophysics/geodynamics, gravitational acceleration varies significantly over depth and g should not be assumed to be constant, and should be inside the integral. "

Finally, in Gerya's book: "In geosciences, pressure is often considered as corresponding to the hydrostatic (litho- static) condition everywhere and it is computed as a function of depth y and vertical density profile $\rho(y)$

$$P(y) = P_0 + g \int_0^y \rho(y) dy$$

[1] https://en.wikipedia.org/wiki/Overburden_pressure

where $P_0 = 0.1$ MPa is pressure on the Earth's surface and g is the gravitational acceleration."

On the AAPG wiki page[2]: "The geostatic pressure at a given depth is the vertical pressure due to the weight of a column of rock and the fluids contained in the rock above that depth. Lithostatic pressure is the vertical pressure due to the weight of the rock only. "

In conclusion there seems to be a widely accepted definition as to what lithostatic pressure is. At a given location it is solely given as a function of the column of material above it. Since $\rho > 0$ and $g > 0$ too, the (absolute value of) the lithostatic pressure can only increase with depth.

The numerical approach of Jourdon & May (2022)

What follows is based on Jourdon and May [174] (2022) in which the authors present an "efficient parallel method to compute lithostatic pressure in thermo-mechanical geodynamic models".

We then start from

$$-\vec{\nabla}p + \rho\vec{g} = \vec{0} \tag{119.1}$$

and we take the divergence of this expression:

$$\vec{\nabla}\cdot\vec{\nabla}p = \vec{\nabla}\cdot\rho\vec{g}$$

We multiply by a test function q and integrate over the domain:

$$\int_\Omega q\vec{\nabla}\cdot\vec{\nabla}p\,dV = \int_\Omega q\vec{\nabla}\cdot\rho\vec{g}\,dV$$

We integrate by parts both LHS and RHS :

$$-\int_\Omega \vec{\nabla}q\cdot\vec{\nabla}p\,dV + \int_\Gamma q\vec{\nabla}p\cdot\vec{n}\,dS = -\int_\Omega \vec{\nabla}q\cdot\rho\vec{g}\,dV + \int_\Gamma q\rho\vec{g}\ \cdot\vec{n}\,dS$$

We split the boundary Γ into a part which is at the surface Γ_s and the part in the interior Γ_i so that $\Gamma = \Gamma_s \cup \Gamma_i$.

If we require $\vec{\nabla}p\cdot\vec{n} = \rho\vec{g}\cdot\vec{n}$ on Γ_i then the surface integrals above simplify to an integral on Γ_s only. On the sides, we have $\vec{g}\propto\vec{e}_y$ and $\vec{n}\propto\vec{e}_x$, so $\vec{g}\cdot\vec{n}=0$ which means that $\vec{\nabla}p\cdot\vec{n}=0$, i.e. the pressure gradient has no horizontal component. At the bottom $\vec{g}\propto\vec{e}_y$ and $\vec{n}\propto\vec{e}_y$ so we are in effect prescribing a pressure gradient to be ρg in the vertical direction.

We have $p = 0$ on Γ_s so the test function q is zero there, so the integral over Γ_s is also taken care of.

After discretisation we arrive at a linear system

$$\boldsymbol{A}\cdot\vec{\mathcal{P}} = \vec{b} \tag{119.2}$$

where $\boldsymbol{A}$ is a sparse $N\times N$ matrix where N is the number of nodes in the mesh, and $\vec{\mathcal{P}}$ is the vector of lithostatic pressure unknowns.

Some discussion

What is interesting (puzzling?) is that Jourdon and May [174] are effectively solving a Poisson problem with pressure as unknown:

$$\Delta p = \vec{\nabla}\cdot\rho\vec{g}$$

but not the original equation Eq. (119.1).

The right hand side of the weak form is never entirely zero (unless there is no gravity or density is zero). Drawing from our physical intuition based on solving the temperature equation, we are in fact solving a purely conductive steady state problem where temperature is set to zero at the top, zero heat flux to the outside is prescribed on the other sides and there is a non-zero heat source inside the domain. If the source term is a function of x (experiments 3 and 4 below), then we expect (and indeed recover) a smooth temperature field which is not constant in the x direction. However, given the definitions above, at two

[2] `https://wiki.aapg.org/Geostatic_and_lithostatic_pressure`

locations x_1 and x_2 for which the column above these is identical we would expect to recover the same lithostatic pressure, which is not necessarily the case when the Poisson equation approach is used (see experiments 3 and 4).

... Which begs the question: *is the pressure computed via the Poisson equation the lithostatic pressure?* If not, what is it? and what does it mean to prescribe it on the sides of a model? Also, is there a single definition of the lithostatic pressure?

After exchanging a few emails with W. Bangerth and B. Myhill, the following remarks were made:

- As a separate consideration, essentiall all of the quotes from the literature above are actually wrong. They are correct if you consider a box geometry in which density only changes with y, and gravity is in a constant direction $\vec{e}_y$ and also only changes with y. All of this is also approximately true close to the surface. But it is not in general true for the whole earth. There, the "column" that sits above a certain area A is cone-shaped and widens as we get higher and higher. In that case, the relationship $p(y) = P_0 + \int_0^y \rho(z)g(z)dz$ is not in fact true. It misses a factor that describes the geometric widening of the cone. The equation with the Laplace of p is correct, however, because the right hand side has $div(\vec{g})$ (if you take $\rho = const$ for a moment), and $div(\vec{g})$ is going to be nonzero *even if the magnitude of $\vec{g}$ is constant* if $\vec{g}$ is a radial vector.

- what the experiments you show [see after] illustrate is that it isn't actually quite obvious what the "lithostatic pressure" should actually be. If you think of the lithosphere as a fluid, then lithospheric pressure = hydrostatic pressure, and the idea of that pressure is that that is what the pressure would be in the absence of dynamic effects (=in the absence of fluid flow). That would mean that the lithostatic pressure would satisfy the Stokes equations

$$-\vec{\nabla}p + \eta\Delta\vec{v} + \rho\vec{g} = \vec{0} \tag{119.3}$$

$$\vec{\nabla}\cdot\vec{v} = 0 \tag{119.4}$$

  if you set $\vec{v} = \vec{0}$ everywhere. But this concept only makes sense if you have an arrangement of $\rho\vec{g}$ that actually allows for a zero velocity. That's the case if you have a vertical layering of materials, for example. If the layering is so that the density increases with depth, then that's even a stable layering, but otherwise you have something that in the absence of perturbations will also stay as it is. But the arrangements in experiments 3, 4 & 5 don't have that, and so one can argue that there is not actually any well defined lithostatic pressure, and that the definition of $Deltap = \vec{\nabla}\cdot(\rho\vec{g})$ is as good as any other.

- If you consider the lithosphere a solid, then the pressure would be the trace of the stress tensor. For your piecewise constant *rhog* on the right hand side, the elasticity equation will have a continuously differentiable function as displacement, making the stress tensor continuous, and consequently the pressure is also continuous. That is again in contrast to the column view you show, which assumes that neighboring columns are mechanically decoupled (think about a steel column in air: Clearly, the pressure increases from top to bottom in the steel column, and it is independent of the surrounding air), but that just isn't the case for actual rocks – if you give the rocks just a little bit of compressibility, then the heavy rock column will settle a bit *and it will pull the neighboring light column down with it along their mutual contact*. Pulling the neighboring column down compresses it as well, increasing its pressure. In other words, the pressure in that neighboring column is not just determined by the weight of the overlying material, but also by the neighboring column.

- There is not always a solution to $\vec{\nabla}p + \rho\vec{g} = \vec{0}$. Take, for example, the case where $\rho\vec{g} = (x, x)^T$ in a 2d coordinate system. This means

$$dp/dx = x \qquad \Rightarrow p(x,y) = \frac{1}{2}x^2 + f(y) \tag{119.5}$$

$$dp/dy = x \qquad \Rightarrow p(x,y) = xy + f(x) \tag{119.6}$$

  This cannot be reconciled. There is only a solution if $\vec{g}$ is the gradient of a potential. On the other hand, $\vec{\nabla}\cdot\vec{\nabla}p = \vec{\nabla}\cdot(\rho\vec{g})$ always has a solution.

- B.M.: "1) what we have in ASPECT[3] follows the usual definition of lithostatic pressure. I don't think we should change it. 2) Jourdon and May's approach isn't the usual definition of lithostatic pressure. I don't know what I would call it. 3) Jourdon and May's approach may result in more stable boundary conditions than naive application of lithostatic pressure, but I have little intuition for whether it would be a reasonable approximation. 4) I don't have an intuitive feel for what they are modelling - their models do not satisfy the hydrostatic equation (their Eq. 5)."

- another WB comment: If the equation grad p = rho*g has a solution (that is, all 3 components of this equation are the same), then that means that rho*g is the gradient of a potential, and p is that potential. In that case, p also solves Delta p = div(rho*g) so the two formulations are the same.

  If rho*g is not the gradient of the potential, as in experiments 3,4,5, then the equation grad p = rho*g does not have a solution. But there will be a pressure that solves d/dz p(x,y,z) = rho*g for every x,y – that's the integration into depth direction. This pressure does not solve grad p = rho*g and does not solve Delta p = div(rho*p) either, and furthermore neither will not be a pressure that would guarantee a zero velocity.

  So my interpretation is that if there is a meaningful litho*static* pressure, then the two definitions are the same. If there is not a meaningful lithostatic pressure, then they have an equation that nevertheless provides them with a solvable problem that gives them something that can be used; whether that something is meaningful is a different question.

In order to investigate this further, I have written a short python code which solves the Poisson equation on a $nelx \times nely$ Q_1 elements. In what follows p_1 is the pressure computed with Eq. (119.2) while p_2 denotes the pressure obtained by integrating ρg over each column of nodes, having set $p_2 = 0$ at the surface and using a simple 1 point quadrature for each segment. The code also computes the pressure gradient $\vec{\nabla} p$ in the middle of each element for both pressure fields p_1 and p_2.

[3] vertical integration

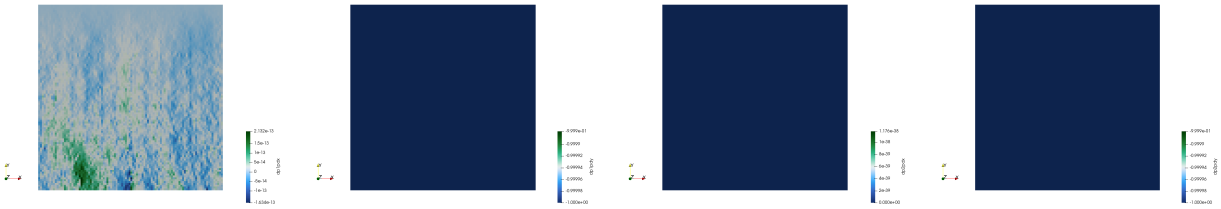
Experiment #1 - benchmark

The domain is a unit square. Density is constant with $\rho = 1$. Gravity vector is $\vec{g} = -\vec{e}_y$.



Left: p_1; Right: p_2. Resolution 100×100.

We find that pressures p_1 and p_2 are identical.



Pressure gradients from left to right: $\partial_x p_1$, $\partial_y p_1$, $\partial_x p_2$, $\partial_y p_2$. Resolution 100×100.

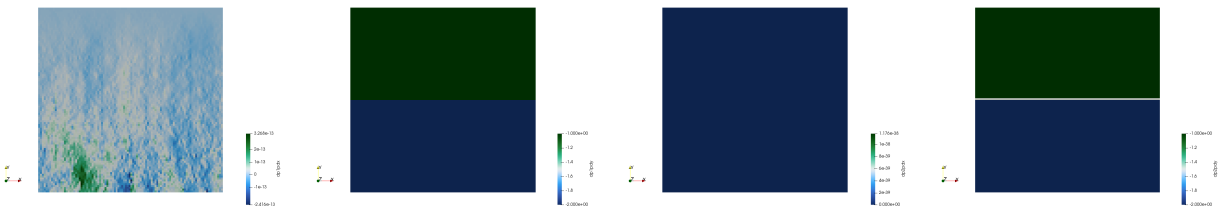
Experiment #2 - benchmark

Same setup as Experiment # 1, but now $\rho = 1$ in the top half of the domain and $\rho = 2$ in the bottom half.



Left: p_1; Right: p_2. Resolution 100×100.

We find that pressures p_1 and p_2 are identical.



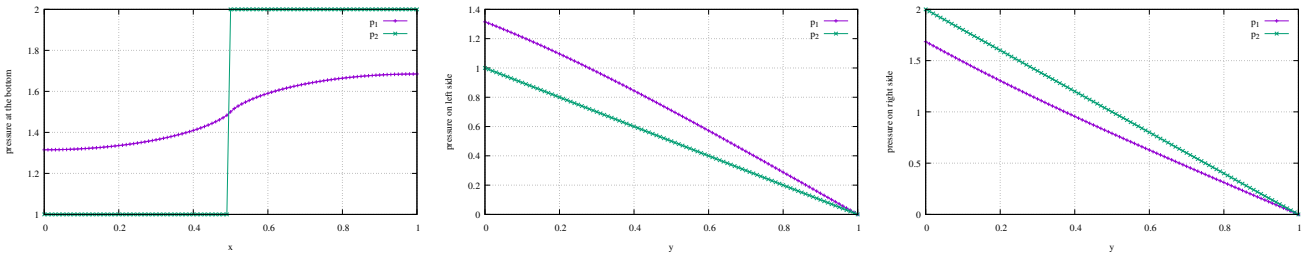
Pressure gradients from left to right: $\partial_x p_1$, $\partial_y p_1$, $\partial_x p_2$, $\partial_y p_2$. Resolution 100×100.

Experiment #3

Same setup as Experiment # 1, but now $\rho = 1$ in the left half of the domain and $\rho = 2$ in the right half.

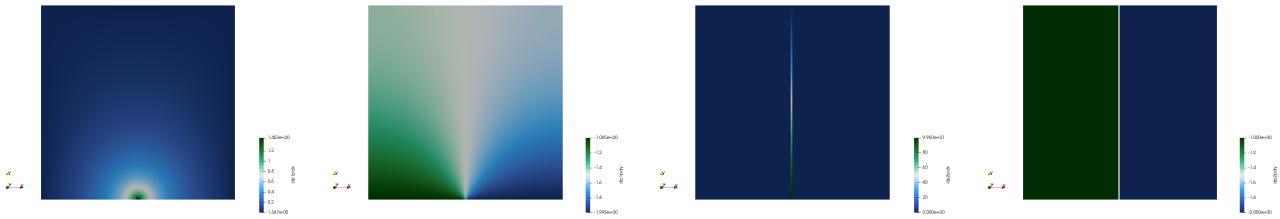


Left: density; middle: p_1; Right: p_2. Resolution 100×100.



Pressure profiles at the bottom, left and right boundaries

We find that pressures p_1 and p_2 are very different (pattern and amplitude).

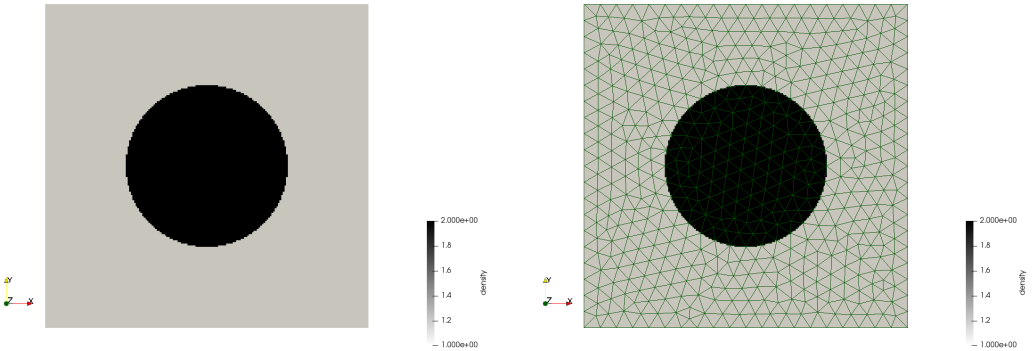


Pressure gradients from left to right: $\partial_x p_1$, $\partial_y p_1$, $\partial_x p_2$, $\partial_y p_2$. Resolution 100×100.

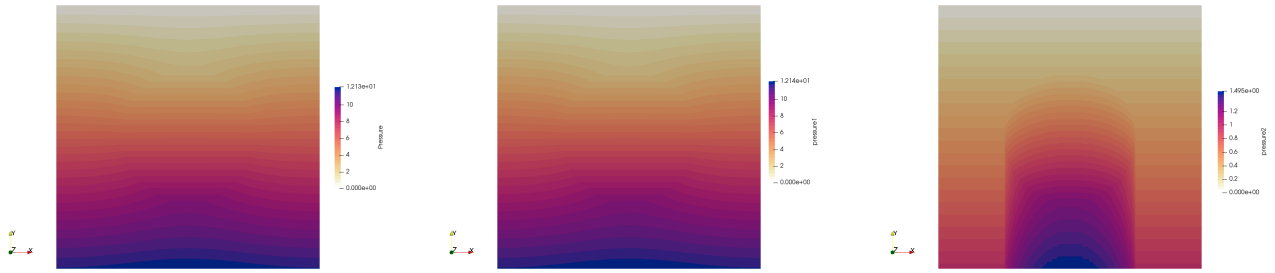
Obviously we do not have $\vec{\nabla} p = -\rho \vec{g}$ for the pressure obtained with the Poisson equation!

Experiment #4

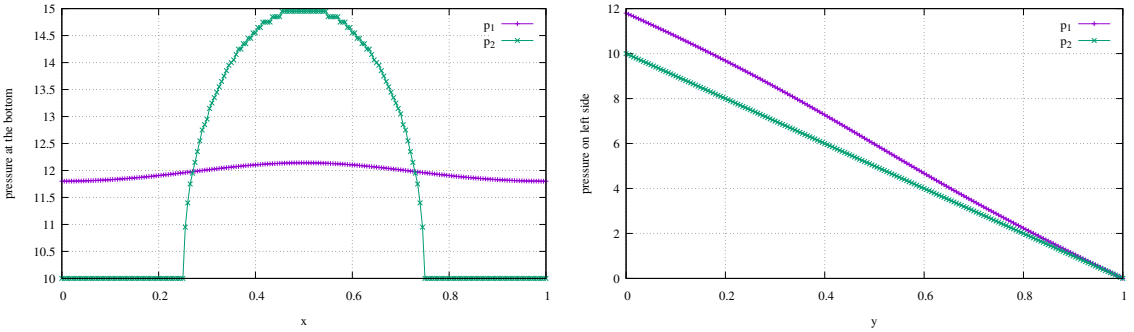
In this experiment the density is 1 except in a centered disc of radius 0.25 in which it is 2, and $g = 10$. I have approached A. Jourdon and asked him to carry out this experiment with the code used in his paper.



Left: density field on 200x200 mesh. right: mesh used by Jourdon with my density field superimposed.

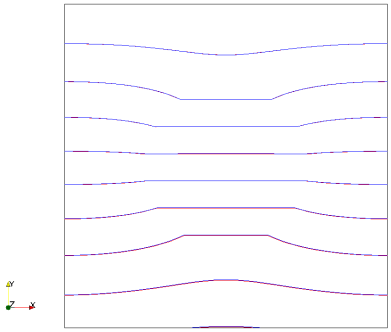


Left: pressure from Jourdon; Middle: p_1; Right: p_2. Resolution 200×200.

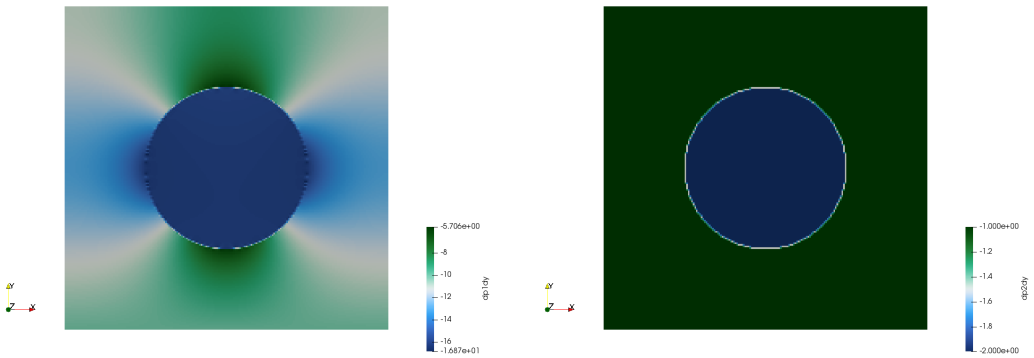


Pressures p_1 and p_2 at the bottom and left boundary.

We find that pressures p_1 and p_2 are again very different (pattern and amplitude), but we find that p_1 and the results sent by Jourdon are identical (which validates my implementation):



Pressure isocontours for both pressures obtained with this STONE and Jourdon's code.

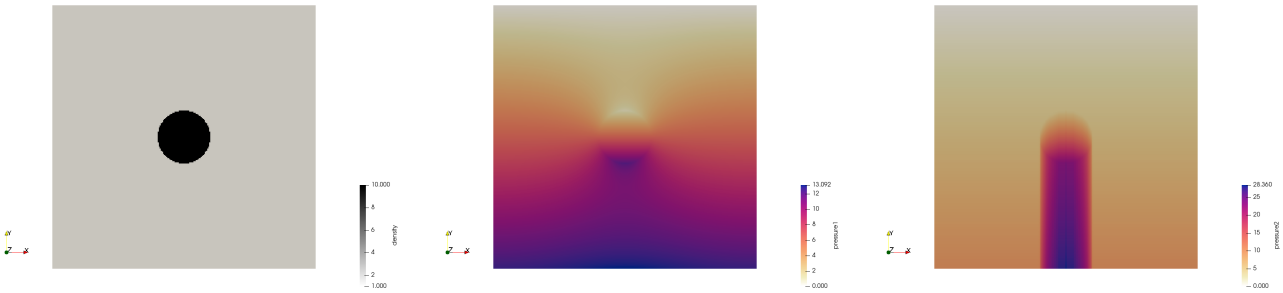


Pressure gradients. Left: $\partial_y p_1$; Right: $\partial_y p_2$. Resolution 200×200.

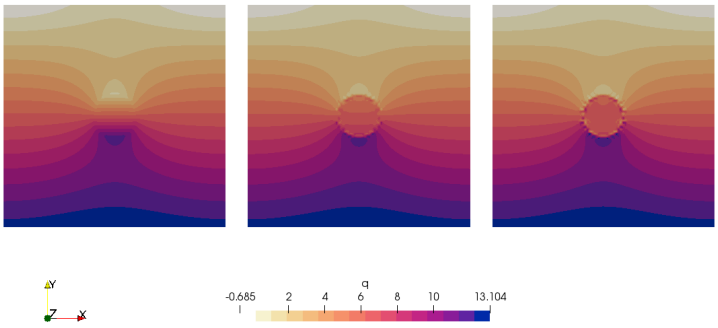
Obviously we here again do not have $\vec{\nabla} p = -\rho \vec{g}$ for the pressure obtained with the Poisson equation!

Experiment #5

In this experiment the density is 1 except in a centered disc of radius 0.1 in which it is 10, and $g = 10$. I have also written a $Q_2 \times Q_1$ FE Stokes solver so as to look at the computed full pressure. Viscosity of the ball is 100 while viscosity of the fluid is 1. Free slip boundary conditions are prescribed on the sides and on the bottom. Either the top surface is free, either free slip is also imposed in conjunction with a zero average surface pressure normalisation.

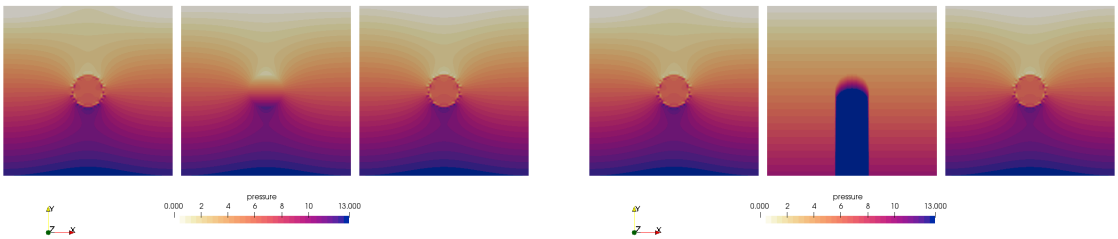


Left: density field on 100x100 mesh. Middle: p_1 pressure; Right: p_2 pressure.



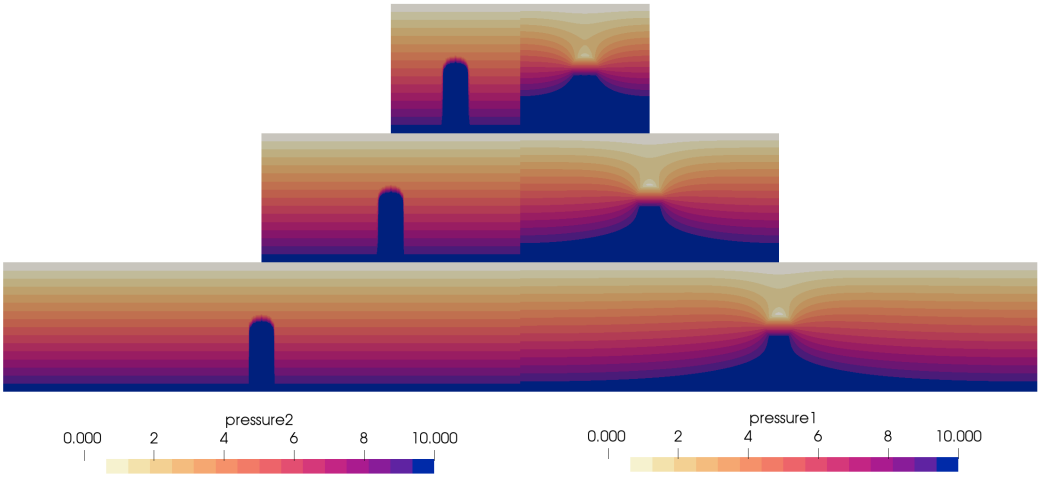
Full pressure computed with a sphere of viscosity 1 (left, isoviscous case), 10 (middle) and 100 (right). Free surface boundary conditions at the top. We find that this parameter does not significantly alter the pressure outside the sphere.

We find that for this example the pressure p_1 is closer to the full pressure than p_2:



Left panel: pressure p_1 surrounded by full pressure fields (free surface on the left, free slip on the right). Right panel: same with p_2 instead.

Finally one can explore the effect of the aspect ratio:



Left column: p_1 pressure; Right column: p_2 pressure. From top to bottom: aspect ratio 1:1, 2:1 and 4:1.

We not-so-surprisingly find that 'far away from the source of buoyancy' pressure p_1 slowly becomes close to p_2. On the one hand, this means that if a long enough domain is used then prescribing p_1 or p_2 on the sides should not have much consequences. On the other hand, the conclusion of Chertova, Geenen, Berg, and Spakman [67] (2012) (the 'first' paper documenting the use of open boundary conditions in subduction modelling) pertaining to the advantage of using such boundary conditions is then problematic: "Other advantages are the independence of the aspect ratio of the model domain, which allows for smaller models with increased resolution for modelling detail." Note that in this paper the authors do not specify how they compute the lithostatic pressure that they use for boundary conditions...

W.B.: "This is again a case where ρg is not the gradient of a potential, and as a consequence there is no function $p(x, y)$ that would satisfy $\nabla p = \rho g$. It *is* worth pointing out, I believe, that you don't actually solve $\vec{\nabla} p = \rho \vec{g}$ in your experiments. You are solving $dp/dz = \rho g_z$ but these are not the same. If you have a vertical gravity vector, then the first equation implies $dp/dx = 0$ (i.e., p has no horizontal variation) but that is clearly not the case in your experiments."

Stone 120

2D Stokes flow. Which FE pair is best?



python_codes/fieldstone_120/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_120`

This STONE was used to produce all the results in Thieulot and Bangerth [329] (Subm.).

Philosophy

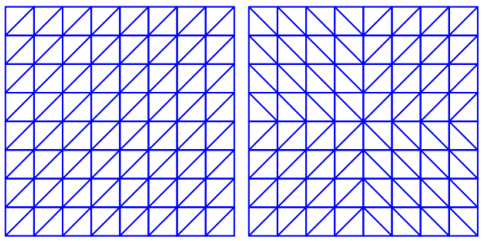
The rationale behind this stone is as follows:

- create a library of basis functions and quadrature rules, as well as other FE-related tools so as to be able to write much more compact FE codes. So far the boundary conditions of this exercise are:
  - Two-dimensional Cartesian domain
  - Continuous Galerkin method
  - Incompressible isothermal Stokes flow, but not isoviscous
  - Most of commonly used element pairs for Stokes equations + some more 'exotic' ones
  - Dirichlet boundary conditions
  - Isoparametric mapping
  - Interior nodes positions with added randomness (if desired)
  - Stokes matrix fully built and sequential direct solver is used (REVISIT)
  - No stabilised formulations, i.e. no $\boldsymbol{Q}_1 \times Q_1$-stab, $\boldsymbol{Q}_1 \times P_0$-stab, or $P_1 \times P_1$-stab
  - No $P_1 isoP_2$ or similar spaces
  - all triangles and quadrilaterals have straight edges.
- Run a suite of manufactured solution benchmarks with all these finite element pairs and assess their accuracy.
- show L_2 errors (v, p, div($\vec{\mathbf{v}}$)) as function of h and the total number of dofs (as in Carstensen, Köhler, Peterseim, and Schedensack [64] (2015))

Remark. 1) *For reasons explained in Boffi, Cavallini, Gardini, and Gastaldi [28] (2012) I use a symmetric grid ('SYMM') for triangular elements*[1] *:*

[1] See also remark of section 20.4.1 in Terrel, Scott, Knepley, Kirby, and Wells [321] (2012)

Fig. 1 Different mesh patterns involved in the numerical experiments: mesh SYMM satisfies our assumption, while REGULAR does not



(a) REGULAR, $h_x = 1/8$. (b) SYMM, $h_x = 1/8$.

Taken from Boffi, Cavallini, Gardini, and Gastaldi [28] (2012).

Remark. 2) *Unless specified I set nelx=nely, i.e. the number of (quadrilateral) elements in the x and y directions are equal.*

Finite element pairs for the Stokes equation

	pair	other name	information	remark
1	$\boldsymbol{Q}_1 \times Q_0$	$\boldsymbol{Q}_1 \times P_0$	Section 10.4.1	NOT LBB STABLE but usable
2	$\boldsymbol{Q}_2 \times Q_0$		Section 10.4.10	
3	$\boldsymbol{P}_2 \times P_0$		Section 10.5.5	
4	$\boldsymbol{Q}_2 \times Q_1$		Section 10.4.2	
5	$\boldsymbol{P}_2 \times P_1$		Section 10.5.8	
6	$\boldsymbol{Q}_3 \times Q_2$		Section 8.3.3	
7	$\boldsymbol{P}_3 \times P_2$		Section 10.5.12	
8	$\boldsymbol{Q}_1^+ \times Q_1$	Quad. MINI	Section 8.3.8	
9	$\boldsymbol{P}_1^+ \times P_1$	MINI	Section 10.5.7	
10	$RT_1 \times Q_0$		Section 10.4.9	
11	$RT_2 \times Q_0$		Section 10.4.9	
12	$DSSY_1 \times Q_0$		Section 10.4.13	
13	$DSSY_2 \times Q_0$		Section 10.4.13	
14	$Han \times Q_0$		Section 10.4.14	
15	$\boldsymbol{Q}_2 \times P_{-1}$		Section 10.4.3	
16	$\boldsymbol{Q}_2 \times P_{-1}(u)$		Section 10.4.3	
17	$\boldsymbol{Q}_2^{(8)} \times Q_1$	Serendipity	Section 8.3.5	
18	$\boldsymbol{P}_2^+ \times P_{-1}$	Crouzeix-Raviart	Section 10.5.9	
19	$\boldsymbol{P}_2^+ \times P_1$		Section 10.5.10	
20	$\boldsymbol{P}_2 \times (P_1 + P_0)$	Augm. $P_2 \times P_1$	Section 10.5.11	
21	$\boldsymbol{P}_1^{NC} \times P_0$	Crouzeix-Raviart	Section 10.5.17	
22	$\boldsymbol{P}_1 \times P_0$		Section 10.5.1	NOT LBB STABLE and unusable
23	$\boldsymbol{Q}_2 \times Q_{-1}$		Section 10.4.4	NOT LBB STABLE
24	$\boldsymbol{Q}_2 \times Q_1 + Q_0$	Augm. $\boldsymbol{Q}_2 \times Q_1$	Section 10.4.11	
25	$Chen \times Q_0$		Section 10.4.15	NOT SURE
26	$\boldsymbol{Q}_4 \times Q_3$		Section 8.3.7	
27	$\boldsymbol{P}_4 \times P_3$		Section 10.5.13	

Remark. 3) *I have indeed built support for 3 unstable elements but these should not be included in the publication.*

Remark. 4) *The augmented $\boldsymbol{P}_2 \times (P_1 + P_0)$ element pair is LBB stable but special care must be taken with respect to the rank-deficiency as explained in Boffi, Cavallini, Gardini, and Gastaldi [28] (2012). The same applies to $\boldsymbol{Q}_2 \times (Q_1 + Q_0)$. I propose a simple fix later.*

Remark. 5) *The nonconforming $\boldsymbol{P}_1^{NC} \times P_0$ gives me a headache and does not yield accurate results (esp. the pressure). Not sure why... Coercivity/Korn stuff? I have yet to find an article using the element 'as is'.*

Remark. 6) *Following John [171] book, I could also do* $\boldsymbol{P}_3^+ \times P_{-2}$, $\boldsymbol{Q}_3 \times P_{-2}$.

Remark. 7) *A modified* Q_2 *Serendipity element was proposed by Zhang and Xiang [363] (2020) (see Section 8.3.6) but it is only different from the standard serendipity element when the elements are not rectangles.*

Remark. 8) *Using high-order element spaces (*Q_4, P_4, *...) only makes sense in this context if the manufactured solution is also a high-order polynomial or contains non-polynomial terms.*

velocity-pressure pair spaces

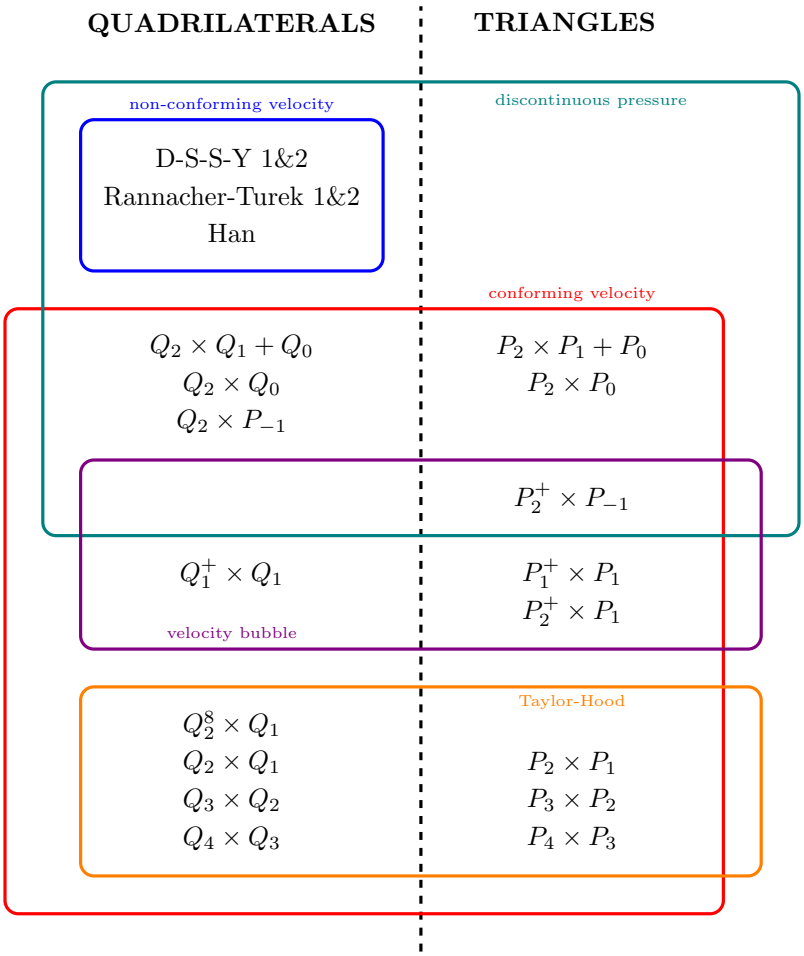
Quadrilateral elements

Vspace → / Pspace ↓	$\boldsymbol{Q}_1$	$\boldsymbol{Q}_2$	$\boldsymbol{Q}_2^{(8)}$	$\boldsymbol{Q}_3$	$\boldsymbol{Q}_4$	$\boldsymbol{Q}_1^+$
Q_0	√	√	?	?		?
Q_1	×	√	√	?		√
Q_2	×	×	×	√		×
Q_3	×	×	×	×	√	×
P_{-1}	?	√	?	?		?
$P_{-1}(u)$	?	√	?	?		?

Triangular elements

Vspace → / Pspace ↓	$\boldsymbol{P}_1$	$\boldsymbol{P}_2$	$\boldsymbol{P}_3$	$\boldsymbol{P}_4$	$\boldsymbol{P}_1^+$	$\boldsymbol{P}_2^+$	$\boldsymbol{P}_1^{NC}$
P_0	×	√	?	×	?	?	√
P_1	×	√	?	×	√	?	?
P_2	×	×	√	×	×	×	×
P_3	×	×	×	TODO			
P_{-1}	×	√	?		?	√	?

An attempt at classification



$P_1^{nc} \times P_0$ belongs to non-conforming but it does not work.

The libraries

There are three files which contain all the required tools to build most of a FE code:

- FEbasis2D.py contains
  - NNN(r,s,space): returns the basis functions $\mathcal{N}_i(i = 1, ...m)$ at position r, s.
  - dNNNdr(r,s,space): returns basis function derivative $\partial_r \mathcal{N}_i(i = 1, ...m)$ at position r, s.
  - dNNNds(r,s,space): returns basis function derivative $\partial_s \mathcal{N}_i(i = 1, ...m)$ at position r, s.
  - NNN_r(space): returns the $r_i(i = 1, ...m)$ coordinates of the support nodes.
  - NNN_s(space): returns the $s_i(i = 1, ...m)$ coordinates of the support nodes.
  - NNN_m(space): returns the number of support nodes.

```
if space=='Q0':      return 1
if space=='P0':      return 1
if space=='Q1':      return 4
if space=='Q-1':     return 4
if space=='Q1+':     return 5
if space=='P1':      return 3
if space=='P1NC':    return 3
if space=='P-1':     return 3
if space=='Pm1':     return 3
if space=='Pm1u':    return 3
if space=='P1+':     return 4
if space=='P1+P0':   return 4
if space=='Q1+Q0':   return 5
if space=='Q2':      return 9
if space=='Q2s':     return 8
if space=='P2':      return 6
if space=='P2+':     return 7
if space=='P3':      return 10
if space=='P4':      return 15
if space=='Q3':      return 16
if space=='Q4':      return 25
if space=='DSSY1':   return 4
if space=='DSSY2':   return 4
if space=='RT1':     return 4
if space=='RT2':     return 4
if space=='Han':     return 5
if space=='Chen':    return 5
```

  - mapping(space): returns the type of mapping, i.e. Q_1 or P_1.
  - visualise_nodes (space): generates a pdf file with the support nodes and the shape of the reference element.
  - visualise_basis_functions (space): is an attempt at making a colormap of the basis functions.
- FEquadrature.py contains
  - quadrature(space,nqpts): it returns the number of quadrature points in the element, their coordinates and associated weights. If the element is a quadrilateral then it contains nqpts×nqpts quadrature points. If it is a triangle then it contains nqpts quadrature points in total. In that case only 3,6,7,12,13,16 are authorized.
  - qcoords_1D(nqpts): returns the coordinates of the nqpts quadrature points between -1 and 1. Only used for quadrilateral elements.
  - qweights_1D(nqpts):returns the weights of the nqpts quadrature points. Only used for quadrilateral elements.
  - visualise_quadrature_points (space,nqpts): generates a pdf file.
- FEtools.py contains

- cartesian_mesh(Lx,Ly,nelx,nely,space): Creates a cartesian mesh of nelx $\times$nely elements in the domain $[0, L_x] \times [0, L_y]$. It returns the total number of nodes N, the number of elements nel, the coordinates of all the nodes in x,y arrays and the connectivity array icon. There is an internal parameter mtype which splits elements in 2 triangles if equal to zero, in 4 triangles if equal to 2, in 6 triangles in equal to 3.
  mtype=0 can be used in combination with P_0, P_1, P_{-1}, P_1^+, $P_1 + P_0$ or P_2.
  mtype=2 can be used in combination with P_0, P_1.
  mtype=3 can be used in combination with P_1.
- randomize_background_mesh(x1,y1,hx,hy,N1,Lx,Ly): this adds a random perturbation to the coordinates of all the nodes of the background mesh inside the domain (i.e. not on the boundary!). The amplitude of the perturbation is bounded by $0.1h_x$ and $0.1h_y$ in the x and y direction.
- adapt_FE_mesh(x1,y1,icon1,m1,space1,x,y,icon,nel,space): This makes sure that all the support nodes for a given space are displaced so as to follow the randomized background mesh, using a (bi-)linear mapping.
- export_swarm_to_ascii(x,y,filename):
- export_swarm_scalar_to_ascii(x,y,f,filename):
- export_swarm_vector_to_ascii(x,y,u,v,filename):
- export_connectivity_array_to_ascii (x,y,icon,filename):
- export_elements_to_vtu(x,y,icon,space,filename):
- export_swarm_to_vtu(x,y,filename):
- export_swarm_vector_to_vtu(x,y,vx,vy,filename):
- export_swarm_scalar_to_vtu(x,y,scalar,filename):
- bc_setup(x,y,Lx,Ly,ndof,left,right,bottom,top):
- J(m,dNdr,dNds,x,y): computes the Jacobian matrix and its determinant for the mapping.
- assemble_K(K_el,A_sparse,iconV,mV,ndofV,iel):
- assemble_G(G_el,A_sparse,iconV,iconP,NfemV,mV,mP,ndofV,ndofP,iel):
- assemble_f(f_el ,rhs,iconV,mV,ndofV,iel):
- apply_bc(K_el,G_el, f_el , h_el ,bc_val, bc_fix ,iconV,mV,ndofV,iel):
- visualise_with_tikz (x,y,space): generates a .tex file containing a tikz drawing of the nodes layout. Only works for 3×2 meshes.

tester #1

tester1.py: checks that each basis function is 1 on its support node and zero at all other nodes.

```
========================================
 tester 1: nodes
========================================
========================================
 Q0
========================================
m= 1
node 0 : [1.]
========================================
 Q1
========================================
m= 4
node 0 : [1. 0. 0. 0.]
node 1 : [0. 1. 0. 0.]
node 2 : [0. 0. 1. 0.]
node 3 : [0. 0. 0. 1.]
========================================
 Q1+
========================================
m= 5
node 0 : [1. 0. 0. 0. 0.]
node 1 : [0. 1. 0. 0. 0.]
node 2 : [0. 0. 1. 0. 0.]
node 3 : [0. 0. 0. 1. 0.]
node 4 : [0. 0. 0. 0. 1.]
========================================
 Q2
========================================
m= 9
node 0 : [ 1. -0.  0. -0.  0. -0. -0.  0.  0.]
node 1 : [ 0.  1. -0. -0.  0.  0. -0.  0.  0.]
node 2 : [0. 0. 1. 0. 0. 0. 0. 0. 0.]
node 3 : [ 0. -0. -0.  1.  0. -0.  0.  0.  0.]
node 4 : [-0.  0. -0.  0.  1.  0. -0. -0.  0.]
node 5 : [-0. -0.  0.  0. -0.  1.  0.  0.  0.]
node 6 : [-0.  0.  0. -0.  0.  0.  1. -0.  0.]
node 7 : [-0.  0. -0.  0. -0. -0.  0.  1.  0.]
node 8 : [ 0. -0.  0. -0. -0.  0.  0. -0.  1.]
========================================
 Q3
========================================
m= 16
node 0 : [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 1 : [ 0.00000000e+00  1.00000000e+00 -1.38777878e-17  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00]
node 2 : [ 0.00000000e+00 -1.38777878e-17  1.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00]
node 3 : [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 4 : [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -1.38777878e-17 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
node 5 : [ 0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  1.00000000e+00 -1.38777878e-17  0.00000000e+00
 -0.00000000e+00 -1.38777878e-17  1.92592994e-34 -0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00]
node 6 : [ 0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -1.38777878e-17  1.00000000e+00  0.00000000e+00
 -0.00000000e+00  1.92592994e-34 -1.38777878e-17 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00]
node 7 : [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -1.38777878e-17
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
node 8 : [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -1.38777878e-17 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  1.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

```
node 9 : [ 0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -1.38777878e-17  1.92592994e-34 -0.00000000e+00
  0.00000000e+00  1.00000000e+00 -1.38777878e-17  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00]
node 10 : [ 0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00  1.92592994e-34 -1.38777878e-17 -0.00000000e+00
  0.00000000e+00 -1.38777878e-17  1.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00]
node 11 : [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -1.38777878e-17
  0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]
node 12 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
node 13 : [ 0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  1.00000000e+00 -1.38777878e-17  0.00000000e+00]
node 14 : [ 0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00 -1.38777878e-17  1.00000000e+00  0.00000000e+00]
node 15 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
========================================
 Q2s
========================================
m= 8
node 0 : [ 1. -0. -0. -0.  0.  0.  0.  0.]
node 1 : [-0.  1. -0. -0.  0.  0.  0.  0.]
node 2 : [-0. -0.  1. -0.  0.  0.  0.  0.]
node 3 : [-0. -0. -0.  1.  0.  0.  0.  0.]
node 4 : [ 0.  0. -0. -0.  1.  0.  0.  0.]
node 5 : [-0.  0.  0. -0.  0.  1.  0.  0.]
node 6 : [-0. -0.  0.  0.  0.  0.  1.  0.]
node 7 : [ 0. -0. -0.  0.  0.  0.  0.  1.]
========================================
 P0
========================================
m= 1
node 0 : [1.]
========================================
 P1
========================================
m= 3
node 0 : [1. 0. 0.]
node 1 : [0. 1. 0.]
node 2 : [0. 0. 1.]
========================================
 P1+
========================================
m= 4
node 0 : [1. 0. 0. 0.]
node 1 : [0. 1. 0. 0.]
node 2 : [0. 0. 1. 0.]
node 3 : [ 5.55111512e-17 -5.55111512e-17 -5.55111512e-17  1.00000000e+00]
========================================
 P1NC
========================================
m= 3
node 0 : [1. 0. 0.]
node 1 : [0. 1. 0.]
node 2 : [0. 0. 1.]
========================================
 P2
========================================
m= 6
node 0 : [1. 0. 0. 0. 0. 0.]
node 1 : [0. 1. 0. 0. 0. 0.]
node 2 : [0. 0. 1. 0. 0. 0.]
node 3 : [0. 0. 0. 1. 0. 0.]
node 4 : [0. 0. 0. 0. 1. 0.]
node 5 : [0. 0. 0. 0. 0. 1.]
========================================
 P3
========================================
m= 10
node 0 : [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
node 1 : [ 1.11022302e-16  1.00000000e+00  1.11022302e-16 -5.55111512e-17
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00]
node 2 : [ 4.4408921e-16 -8.8817842e-16  1.0000000e+00 -4.4408921e-16
  0.0000000e+00  0.0000000e+00  0.0000000e+00  0.0000000e+00
  0.0000000e+00  0.0000000e+00]
node 3 : [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
node 4 : [ 1.11022302e-16  0.00000000e+00  0.00000000e+00  0.00000000e+00
  1.00000000e+00  0.00000000e+00  0.00000000e+00  1.11022302e-16
  0.00000000e+00 -5.55111512e-17]
node 5 : [ 3.33066907e-16  0.00000000e+00  1.11022302e-16 -5.55111512e-17
 -1.11022302e-16  1.00000000e+00  0.00000000e+00  1.11022302e-16
  0.00000000e+00 -5.55111512e-17]
node 6 : [ 3.33066907e-16 -8.88178420e-16  8.88178420e-16 -4.44089210e-16
 -1.11022302e-16  0.00000000e+00  1.00000000e+00  1.11022302e-16
  0.00000000e+00 -5.55111512e-17]
node 7 : [ 4.4408921e-16  0.0000000e+00  0.0000000e+00  0.0000000e+00
 -8.8817842e-16  0.0000000e+00  0.0000000e+00  1.0000000e+00
  0.0000000e+00 -4.4408921e-16]
node 8 : [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -5.55111512e-17
 -8.88178420e-16  0.00000000e+00  0.00000000e+00  8.88178420e-16
  1.00000000e+00 -4.44089210e-16]
node 9 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
========================================
 P4
========================================
m= 15
node 0 : [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 1 : [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 2 : [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 3 : [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 4 : [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 5 : [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
node 6 : [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
node 7 : [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
node 8 : [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
node 9 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
node 10 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
node 11 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
node 12 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
node 13 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
node 14 : [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
========================================
 P1+P0
========================================
m= 4
node 0 : [1. 0. 0. 1.]
node 1 : [0. 1. 0. 1.]
node 2 : [0. 0. 1. 1.]
node 3 : [0.33333333 0.33333333 0.33333333 1.        ]
========================================
 DSSY1
========================================
m= 4
node 0 : [1. 0. 0. 0.]
node 1 : [0. 1. 0. 0.]
node 2 : [0. 0. 1. 0.]
node 3 : [0. 0. 0. 1.]
========================================
 DSSY2
========================================
m= 4
node 0 : [1. 0. 0. 0.]
node 1 : [0. 1. 0. 0.]
node 2 : [0. 0. 1. 0.]
node 3 : [0. 0. 0. 1.]
========================================
 RT1
========================================
m= 4
node 0 : [1. 0. 0. 0.]
node 1 : [0. 1. 0. 0.]
node 2 : [0. 0. 1. 0.]
node 3 : [0. 0. 0. 1.]
========================================
 RT2
```

```
========================================
m= 4
node 0 : [ 1.125 -0.125  0.125 -0.125]
node 1 : [-0.125  1.125 -0.125  0.125]
node 2 : [ 0.125 -0.125  1.125 -0.125]
node 3 : [-0.125  0.125 -0.125  1.125]
========================================
 Han
========================================
m= 5
node 0 : [ 1.  0.  0. -0.  0.]
node 1 : [-0.  1.  0. -0.  0.]
node 2 : [-0.  0.  1. -0.  0.]
node 3 : [-0.  0.  0.  1.  0.]
node 4 : [-0.  0.  0. -0.  1.]
========================================
 Chen
========================================
m= 5
node 0 : [1.   0.   0.   0.   0.25]
node 1 : [0.   1.   0.   0.   0.25]
node 2 : [0.   0.   1.   0.   0.25]
node 3 : [0.   0.   0.   1.   0.25]
node 4 : [0.25 0.25 0.25 0.25 1.  ]
```

tester #2

tester2.py: computes the area of each element by means of numerical quadrature. The domain is 3×2 and there are 3×2 elements so that we expect 1 for quadrilateral elements and 0.5 for triangular elements.

```
========================================
 tester 2: areas
========================================
========================================
 Q1
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 Q1+
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
```

```
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 -0.00 -0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 Q2
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 Q3
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
```

```
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 -0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 Q2s
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 DSSY1
```

```
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 DSSY2
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
```

```
iel= 4 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 RT1
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 RT2
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
```

```
iel= 3 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 Han
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 0 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 0 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 1 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 1 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 0 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 1 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 2 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 2 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 1 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 3 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 3 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 0 | jcob= 2.00 -0.00 -0.00 2.00 0.25
iel= 4 | iq= 1 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 4 | iq= 3 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 6 | jcob= 2.00 0.00 -0.00 2.00 0.25
iel= 4 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 4 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 0 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 1 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 2 | jcob= 2.00 -0.00 0.00 2.00 0.25
iel= 5 | iq= 3 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 5 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 6 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 7 | jcob= 2.00 0.00 0.00 2.00 0.25
iel= 5 | iq= 8 | jcob= 2.00 0.00 0.00 2.00 0.25
areas= [1. 1. 1. 1. 1. 1.]
========================================
 Chen
========================================
nqel= 9
iel= 0 | iq= 0 | jcob= 1.30 -0.70 -0.70 1.30 0.83
iel= 0 | iq= 1 | jcob= 0.93 -1.07 0.00 2.00 0.54
iel= 0 | iq= 2 | jcob= -0.32 -2.32 2.32 4.32 0.25
iel= 0 | iq= 3 | jcob= 2.00 0.00 -1.07 0.93 0.54
iel= 0 | iq= 4 | jcob= 2.00 0.00 0.00 2.00 0.25
```

tester #3

tester3.py: computes the space derivatives of each coordinate minus one, so that we expect zero (down to machine precision). It returns 'passed' if the results is less than 10^{-12}.

```
========================================
 tester 3: linear fields
========================================
Q1 passed
Q1+ passed
Q2 passed
Q3 passed
Q2s passed
DSSY1 passed
DSSY2 passed
RT1 passed
RT2 passed
Han passed
jcob<0
```

tester #4

tester4.py: computes the gradient of $x^2/2$ and $y^2/2$ (i.e. resp. x and y) to which the coordinate of the quadrature point is subtracted so that one expect zero again. This test can only be passed by quadratic and higher order elements.

```
========================================
 tester 4: quadratic fields
========================================
Q2 passed
Q3 passed
Q2s passed
jcob<0
```

tester #5

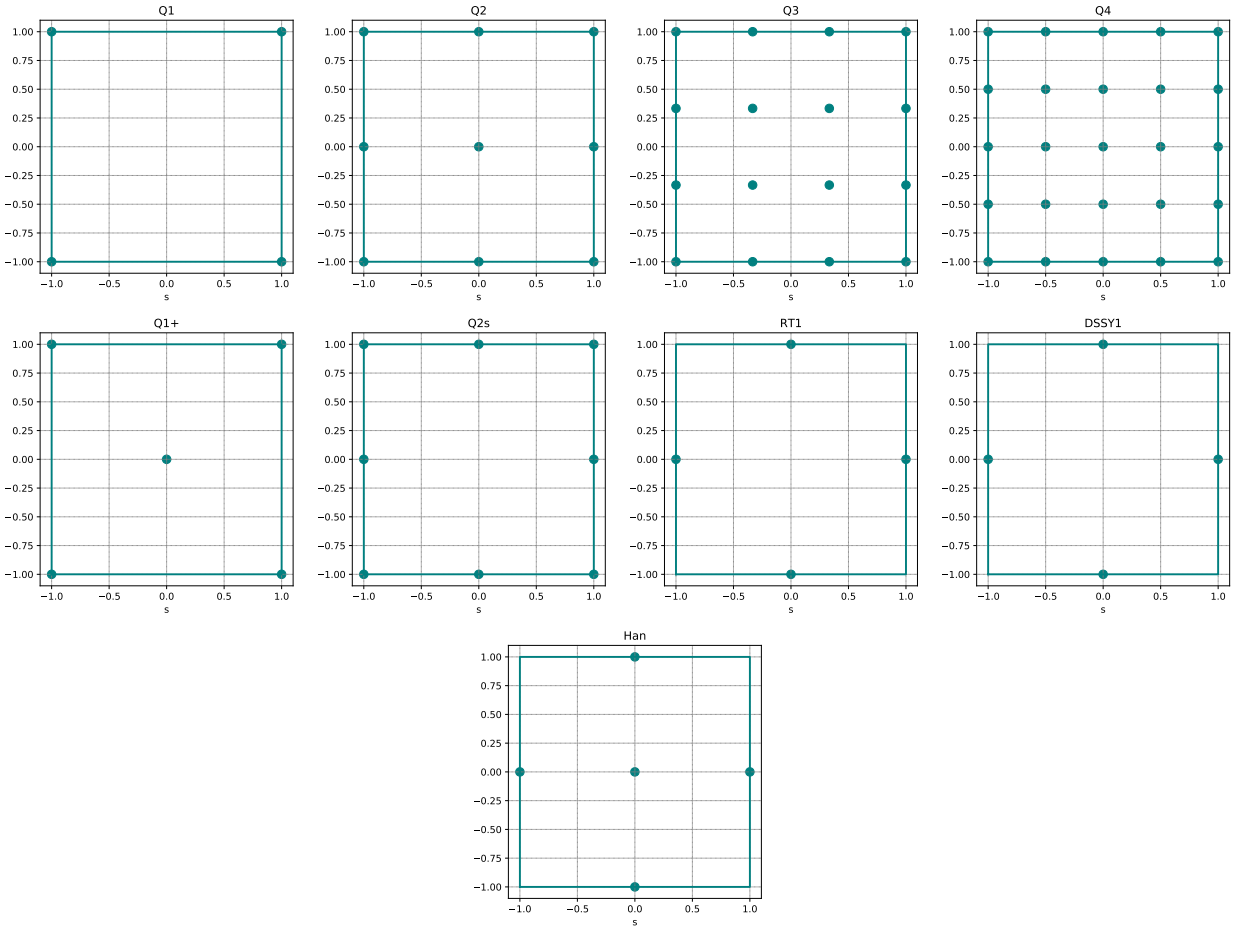
tester5.py: produces 3x2 tikz files and corresponding iconV files.

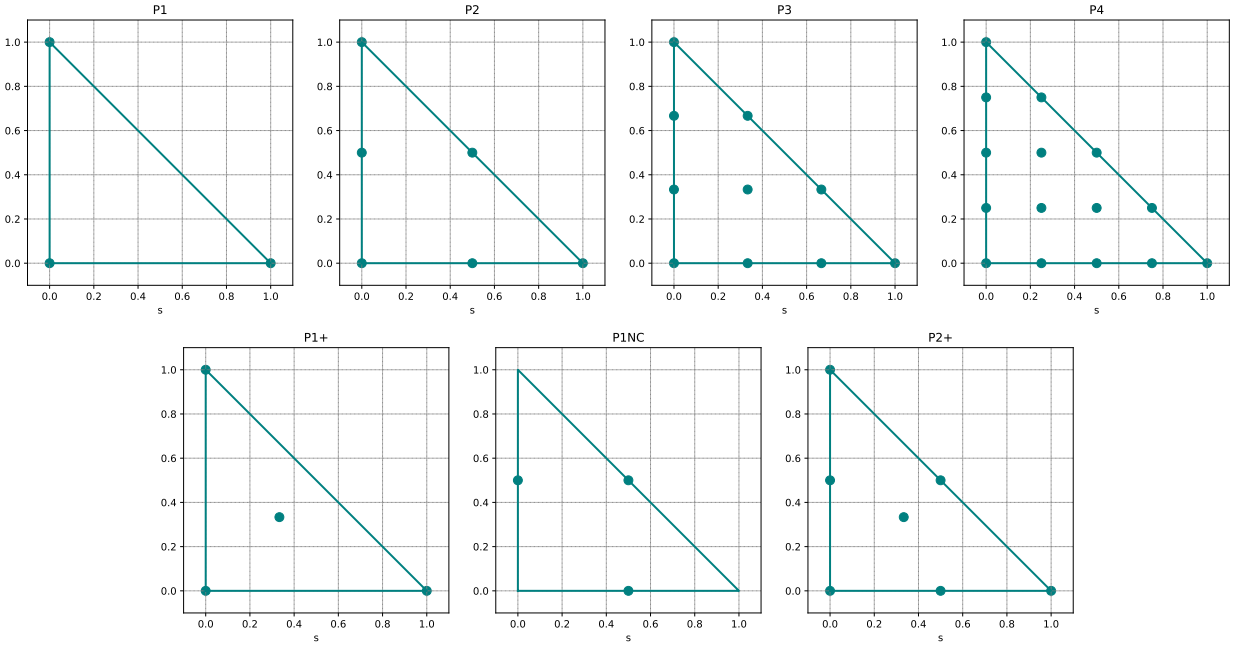
```
========================================
 Q1 : NV = 12  nel= 6
========================================
========================================
 Q1+ : NV = 18  nel= 6
========================================
========================================
 Q2 : NV = 35  nel= 6
========================================
========================================
 Q3 : NV = 70  nel= 6
========================================
========================================
 Q4 : NV = 117  nel= 6
========================================
========================================
 Q2s : NV = 29  nel= 6
========================================
========================================
 DSSY1 : NV = 17  nel= 6
========================================
========================================
 DSSY2 : NV = 17  nel= 6
========================================
========================================
 RT1 : NV = 17  nel= 6
========================================
========================================
 RT2 : NV = 17  nel= 6
========================================
========================================
 Han : NV = 23  nel= 6
========================================
========================================
 Chen : NV = 23  nel= 6
========================================
========================================
 P1 : NV = 12  nel= 12
========================================
========================================
 P1+ : NV = 24  nel= 12
========================================
========================================
 P1NC : NV = 23  nel= 12
========================================
========================================
 P2 : NV = 35  nel= 12
========================================
========================================
 P2+ : NV = 47  nel= 12
========================================
========================================
 P3 : NV = 70  nel= 12
========================================
========================================
 P1+P0 : NV = 24  nel= 12
========================================
========================================
 P4 : NV = 117  nel= 12
========================================
```

tester #6: Supported element spaces

tester6.py: produces pdf files of reference elements.

These figures are obtained by running tester6.py.





tester #7: zero-th order consistency

It computes the sume $\sum_i \mathcal{N}_i(x,y)$ at all quadrature points. We expect 1 (down to machine precision). If the value is not 1 it writes 'problem'.

```
========================================
 tester 7: 0th order consistency
========================================
========================================
 Q1
========================================
nqel= 49
========================================
 Q1+
========================================
nqel= 49
========================================
 Q2
========================================
nqel= 49
========================================
 Q3
========================================
nqel= 49
========================================
 Q2s
========================================
nqel= 49
========================================
 DSSY1
========================================
nqel= 49
========================================
 DSSY2
========================================
nqel= 49
========================================
 RT1
========================================
nqel= 49
========================================
 RT2
========================================
nqel= 49
========================================
 Han
========================================
nqel= 49
========================================
 Chen
========================================
nqel= 49
problem
problem
problem
problem
problem
problem
========================================
 P1
========================================
nqel= 7
========================================
 P1+
========================================
nqel= 7
========================================
 P2
========================================
nqel= 7
========================================
 P2+
========================================
nqel= 7
========================================
 P3
========================================
nqel= 7
========================================
 P4
========================================
nqel= 7
```

tester #8:

```
*********************************************************
QUADRILATERALS
*********************************************************
nqpts= 1 | sum(weights) 1D= 2.0
nqpts= 1 | sum(weights) 2D= 4.0
nqpts= 2 | sum(weights) 1D= 2.0
nqpts= 2 | sum(weights) 2D= 4.0
nqpts= 3 | sum(weights) 1D= 2.0
nqpts= 3 | sum(weights) 2D= 4.0
nqpts= 4 | sum(weights) 1D= 2.0
nqpts= 4 | sum(weights) 2D= 4.0
nqpts= 5 | sum(weights) 1D= 2.0
nqpts= 5 | sum(weights) 2D= 4.0
nqpts= 6 | sum(weights) 1D= 2.0
nqpts= 6 | sum(weights) 2D= 4.0
nqpts= 7 | sum(weights) 1D= 2.0
nqpts= 7 | sum(weights) 2D= 4.0
nqpts= 8 | sum(weights) 1D= 2.0
nqpts= 8 | sum(weights) 2D= 4.0
nqpts= 9 | sum(weights) 1D= 2.0
nqpts= 9 | sum(weights) 2D= 4.0
nqpts= 10 | sum(weights) 1D= 2.0
nqpts= 10 | sum(weights) 2D= 4.0
*********************************************************
TRIANGLES
*********************************************************
nqpts= 1 | sum(weights) 2D= 0.5
nqpts= 3 | sum(weights) 2D= 0.5
nqpts= 6 | sum(weights) 2D= 0.5
nqpts= 7 | sum(weights) 2D= 0.5
nqpts= 12 | sum(weights) 2D= 0.5
nqpts= 13 | sum(weights) 2D= 0.5
nqpts= 16 | sum(weights) 2D= 0.5
*********************************************************
```

List of scripts

The main code stone.py accepts either 5 or 7 arguments.

- 5 arguments: nelx,Vspace,Pspace,experiment,unstructured
- 7 arguments: nelx,Vspace,Pspace,experiment,unstructured,etastar,drho

The experiment is as follows:

```
if experiment=='dh'             : import mms_dh as mms
if experiment=='jolm17'         : import mms_jolm17 as mms
if experiment=='sinker'         : import mms_sinker as mms
if experiment=='sinker_reduced' : import mms_sinker_reduced as mms
if experiment=='sinker_open'    : import mms_sinker_open as mms
if experiment=='poiseuille'     : import mms_poiseuille as mms
if experiment=='johnbook'       : import mms_johnbook as mms
if experiment=='bocg12'         : import mms_bocg12 as mms
if experiment=='solcx'          : import mms_solcx as mms
if experiment=='solkz'          : import mms_solkz as mms
if experiment=='solvi'          : import mms_solvi as mms
if experiment=='RTwave'         : import mms_RTwave as mms
if experiment=='jokn16'         : import mms_jokn16 as mms
if experiment=='plin'           : import mms_plin as mms
if experiment=='lire19'         : import mms_lire19 as mms
```

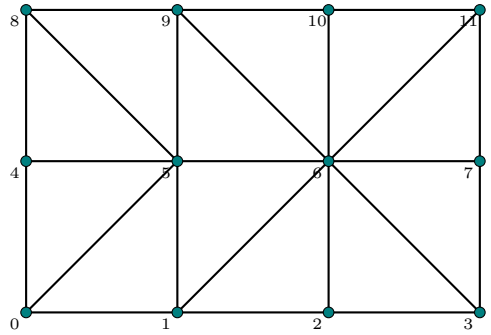
In the main folder there are many scripts that allow us to run the code for a wide range of parameter values (like resolution, element type, ...)

There are several bash scripts that run various series of benchmarks for many resolutions and/or number of quadrature points, boundary conditions, meshes, ...

```
-rwxrwxr-x 1 cedrict cedrict 1,5K nov 27  2023 script_paper_bocg12_structured
-rwxrwxr-x 1 cedrict cedrict 1,4K nov 27  2023 script_paper_bocg12_unstructured
-rwxrwxr-x 1 cedrict cedrict 1,5K dec  7  2023 script_paper_dh_structured
-rwxrwxr-x 1 cedrict cedrict 1,6K jan 14  2024 script_paper_dh_unstructured
-rwxrwxr-x 1 cedrict cedrict 1,4K nov 27  2023 script_paper_johnbook_structured
-rwxrwxr-x 1 cedrict cedrict 1,2K nov 27  2023 script_paper_johnbook_unstructured
-rwxrwxr-x 1 cedrict cedrict 1,4K dec  5  2023 script_paper_jokn16_structured
-rwxrwxr-x 1 cedrict cedrict 1,3K dec  6  2023 script_paper_jokn16_unstructured
-rwxrwxr-x 1 cedrict cedrict 1,4K nov 27  2023 script_paper_jolm17_structured
-rwxrwxr-x 1 cedrict cedrict  528 nov 27  2023 script_paper_master
-rwxrwxr-x 1 cedrict cedrict 1,6K nov 23  2023 script_paper_rtwave
-rwxrwxr-x 1 cedrict cedrict 1,7K nov 22  2023 script_paper_sinker
-rwxrwxr-x 1 cedrict cedrict 1,7K jul  2  2023 script_paper_sinker_open
-rwxrwxr-x 1 cedrict cedrict 1,7K jan 12  2024 script_paper_sinker_reduced
-rwxrwxr-x 1 cedrict cedrict 1,4K nov 27  2023 script_paper_solcx_structured
-rwxrwxr-x 1 cedrict cedrict 1,3K nov 27  2023 script_paper_solcx_unstructured
-rwxrwxr-x 1 cedrict cedrict 1,4K nov 27  2023 script_paper_solkz_structured
-rwxrwxr-x 1 cedrict cedrict 1,4K nov 27  2023 script_paper_solkz_unstructured
-rwxrwxr-x 1 cedrict cedrict 1,6K jan 14  2024 script_paper_solvi_structured
-rwxrwxr-x 1 cedrict cedrict 1,5K jan 14  2024 script_paper_solvi_unstructured
-rwxrwxr-x 1 cedrict cedrict 4,3K jul  2  2023 script_run_all
-rwxrwxr-x 1 cedrict cedrict 1,4K jan 14  2024 script_test_mini
```

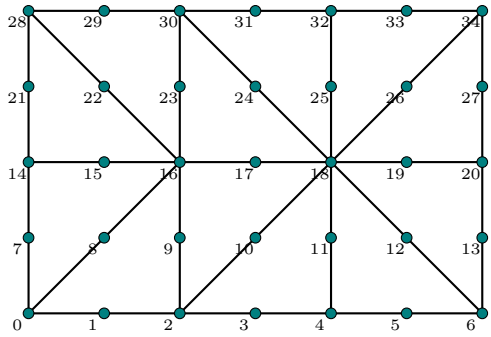
3×2 meshes with velocity nodes

P1



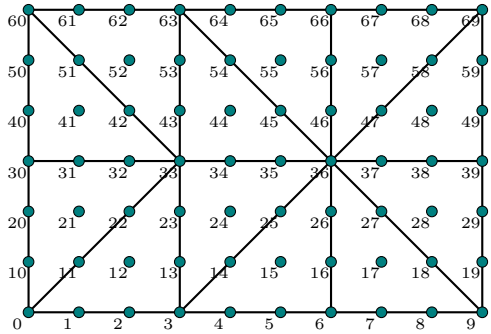

```
--------elt:0-------
node 0 | 1 at pos. 1.0,0.0
node 1 | 5 at pos. 1.0,1.0
node 2 | 0 at pos. 0.0,0.0
```

P2



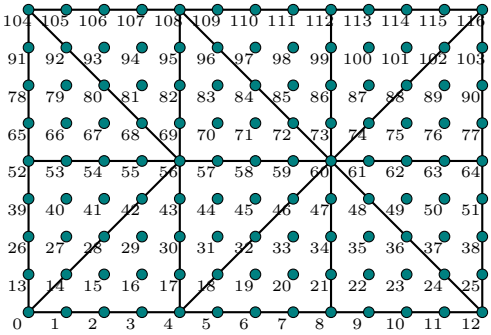

```
--------elt:0-------
node 0 | 2 at pos. 1.0,0.0
node 1 | 16 at pos. 1.0,1.0
node 2 | 0 at pos. 0.0,0.0
node 3 | 9 at pos. 1.0,0.5
node 4 | 8 at pos. 0.5,0.5
node 5 | 1 at pos. 0.5,0.0
```

P3



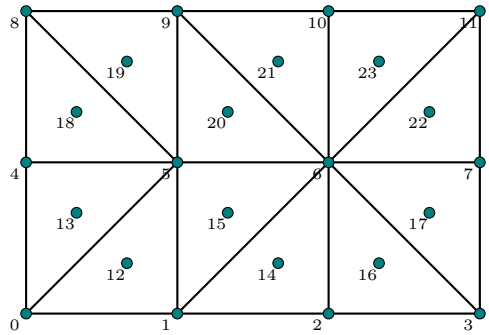

```
--------elt:0-------
node 0 | 3 at pos. 1.0,0.0
node 1 | 13 at pos. 1.0,0.3333333333333333
node 2 | 23 at pos. 1.0,0.6666666666666666
node 3 | 33 at pos. 1.0,1.0
node 4 | 2 at pos. 0.6666666666666666,0.0
node 5 | 12 at pos. 0.6666666666666666,0.3333333333333333
node 6 | 22 at pos. 0.6666666666666666,0.6666666666666666
node 7 | 1 at pos. 0.3333333333333333,0.0
node 8 | 11 at pos. 0.3333333333333333,0.3333333333333333
node 9 | 0 at pos. 0.0,0.0
```

P4



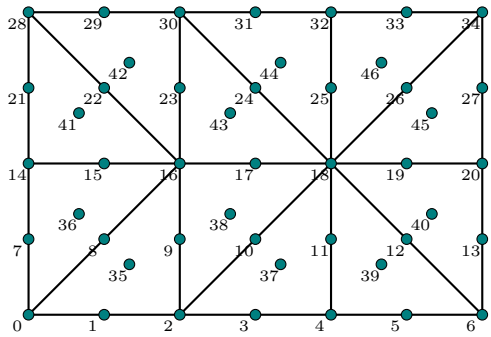

```
--------elt:0-------
node 0 | 4 at pos. 1.0,0.0
node 1 | 17 at pos. 1.0,0.25
node 2 | 30 at pos. 1.0,0.5
node 3 | 43 at pos. 1.0,0.75
node 4 | 56 at pos. 1.0,1.0
node 5 | 3 at pos. 0.75,0.0
node 6 | 16 at pos. 0.75,0.25
node 7 | 29 at pos. 0.75,0.5
node 8 | 42 at pos. 0.75,0.75
node 9 | 2 at pos. 0.5,0.0
node 10 | 15 at pos. 0.5,0.25
node 11 | 28 at pos. 0.5,0.5
node 12 | 1 at pos. 0.25,0.0
node 13 | 14 at pos. 0.25,0.25
node 14 | 0 at pos. 0.0,0.0
```

P1+



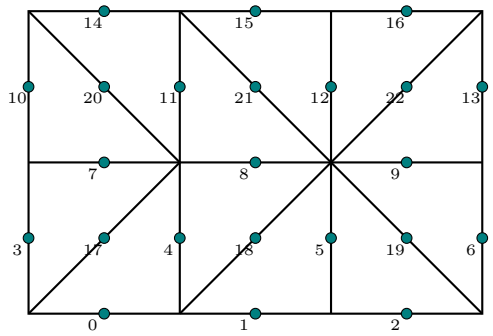

```
--------elt:0-------
node 0 | 1 at pos. 1.0,0.0
node 1 | 5 at pos. 1.0,1.0
node 2 | 0 at pos. 0.0,0.0
node 3 | 12 at pos. 0.6666666666666666,0.3333333333333333
```

P2+



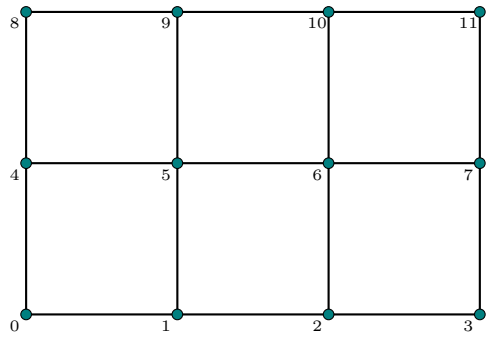

```
--------elt:0-------
node 0 | 2 at pos. 1.0,0.0
node 1 | 16 at pos. 1.0,1.0
node 2 | 0 at pos. 0.0,0.0
node 3 | 9 at pos. 1.0,0.5
node 4 | 8 at pos. 0.5,0.5
node 5 | 1 at pos. 0.5,0.0
node 6 | 35 at pos. 0.6666666666666666,0.3333333333333333
```

P1NC



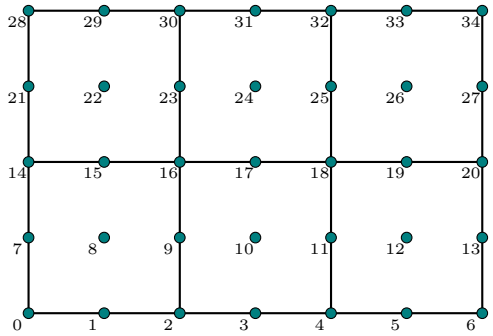

```
--------elt:0-------
node 0 | 4 at pos. 1.0,0.5
node 1 | 17 at pos. 0.5,0.5
node 2 | 0 at pos. 0.5,0.0
```

Q1



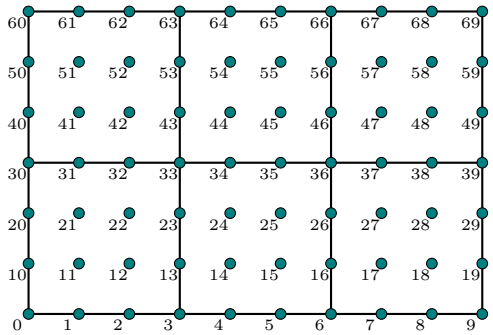

```
--------elt:0-------
node 0 | 0 at pos. 0.0,0.0
node 1 | 1 at pos. 1.0,0.0
node 2 | 5 at pos. 1.0,1.0
node 3 | 4 at pos. 0.0,1.0
```

Q2



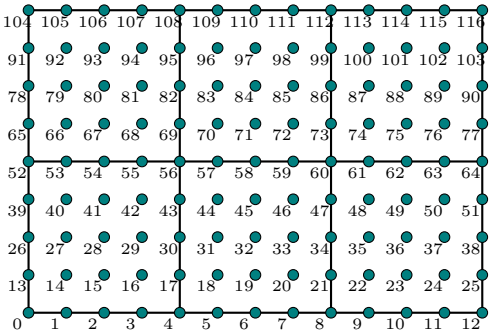

```
--------elt:0-------
node 0 | 0 at pos. 0.0,0.0
node 1 | 2 at pos. 1.0,0.0
node 2 | 16 at pos. 1.0,1.0
node 3 | 14 at pos. 0.0,1.0
node 4 | 1 at pos. 0.5,0.0
node 5 | 9 at pos. 1.0,0.5
node 6 | 15 at pos. 0.5,1.0
node 7 | 7 at pos. 0.0,0.5
node 8 | 8 at pos. 0.5,0.5
```

Q3




```
--------elt:0-------
node 0 | 0 at pos. 0.0,0.0
node 1 | 1 at pos. 0.3333333333333333,0.0
node 2 | 2 at pos. 0.6666666666666666,0.0
node 3 | 3 at pos. 1.0,0.0
node 4 | 10 at pos. 0.0,0.3333333333333333
node 5 | 11 at pos. 0.3333333333333333,0.3333333333333333
node 6 | 12 at pos. 0.6666666666666666,0.3333333333333333
node 7 | 13 at pos. 1.0,0.3333333333333333
node 8 | 20 at pos. 0.0,0.6666666666666666
node 9 | 21 at pos. 0.3333333333333333,0.6666666666666666
node 10 | 22 at pos. 0.6666666666666666,0.6666666666666666
node 11 | 23 at pos. 1.0,0.6666666666666666
node 12 | 30 at pos. 0.0,1.0
node 13 | 31 at pos. 0.3333333333333333,1.0
node 14 | 32 at pos. 0.6666666666666666,1.0
node 15 | 33 at pos. 1.0,1.0
```

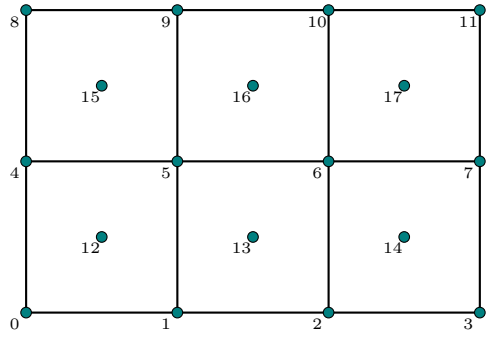
Q4




```
--------elt:0-------
node 0 | 0 at pos. 0.0,0.0
```

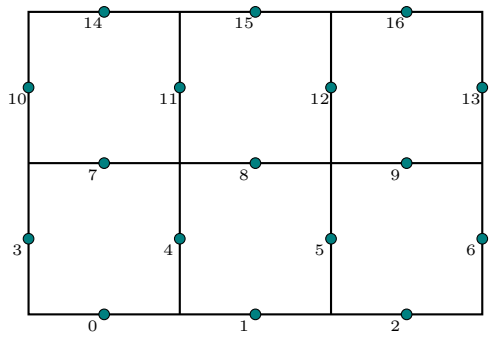
```
node 1 | 1 at pos. 0.25,0.0
node 2 | 2 at pos. 0.5,0.0
node 3 | 3 at pos. 0.75,0.0
node 4 | 4 at pos. 1.0,0.0
node 5 | 13 at pos. 0.0,0.25
node 6 | 14 at pos. 0.25,0.25
node 7 | 15 at pos. 0.5,0.25
node 8 | 16 at pos. 0.75,0.25
node 9 | 17 at pos. 1.0,0.25
node 10 | 26 at pos. 0.0,0.5
node 11 | 27 at pos. 0.25,0.5
node 12 | 28 at pos. 0.5,0.5
node 13 | 29 at pos. 0.75,0.5
node 14 | 30 at pos. 1.0,0.5
node 15 | 39 at pos. 0.0,0.75
node 16 | 40 at pos. 0.25,0.75
node 17 | 41 at pos. 0.5,0.75
node 18 | 42 at pos. 0.75,0.75
node 19 | 43 at pos. 1.0,0.75
node 20 | 52 at pos. 0.0,1.0
node 21 | 53 at pos. 0.25,1.0
node 22 | 54 at pos. 0.5,1.0
node 23 | 55 at pos. 0.75,1.0
node 24 | 56 at pos. 1.0,1.0
```

Q1+



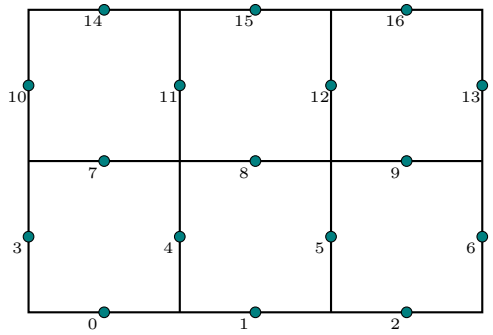

```
--------elt:0-------
node 0 | 0 at pos. 0.0,0.0
node 1 | 1 at pos. 1.0,0.0
node 2 | 5 at pos. 1.0,1.0
node 3 | 4 at pos. 0.0,1.0
node 4 | 12 at pos. 0.5,0.5
```

DSSY1



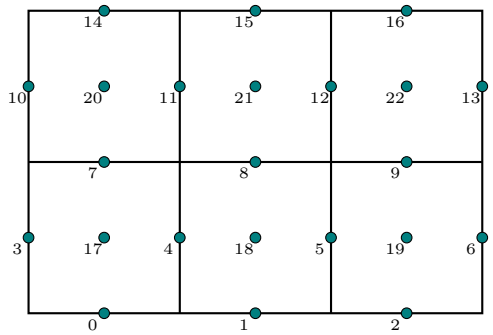

```
--------elt:0-------
node 0 | 0 at pos. 0.5,0.0
node 1 | 4 at pos. 1.0,0.5
node 2 | 7 at pos. 0.5,1.0
node 3 | 3 at pos. 0.0,0.5
```

RT1



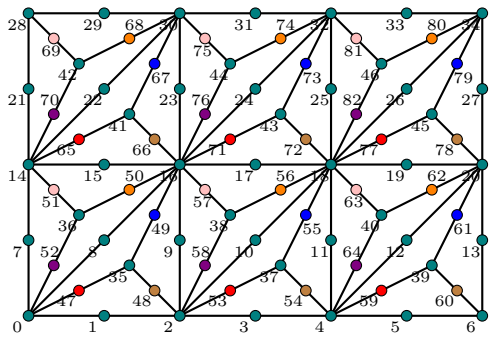

```
--------elt:0-------
node 0 | 0 at pos. 0.5,0.0
node 1 | 4 at pos. 1.0,0.5
node 2 | 7 at pos. 0.5,1.0
node 3 | 3 at pos. 0.0,0.5
```

Han

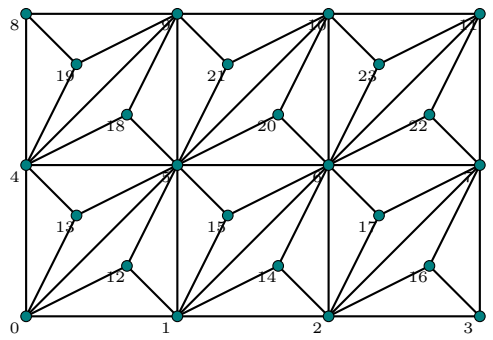



```
--------elt:0-------
node 0 | 0 at pos. 0.5,0.0
node 1 | 4 at pos. 1.0,0.5
node 2 | 7 at pos. 0.5,1.0
node 3 | 3 at pos. 0.0,0.5
node 4 | 17 at pos. 0.5,0.5
```

P2(sv)

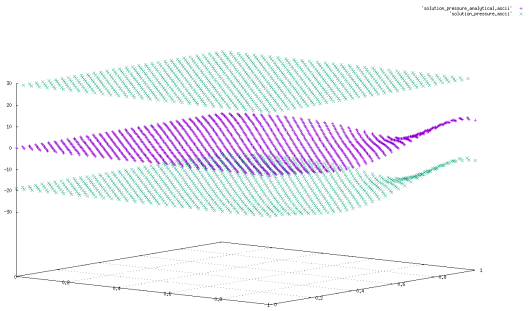


P1(sv)



Pressure normalisation for augmented Taylor-Hood elements

I tried both $\boldsymbol{P}_2 \times (P_1 + P_0)$ and $\boldsymbol{Q}_2 \times (Q_1 + Q_0)$ on different manufactured solutions and I observed for both that the pressure I obtained visually consisted of 2 fields: (for quads) the continuous Q_1 which looked similar to the expected analytical field although offset by what seemed a constant (bottom green points), and the Q_0 field (top green points) which was very different than the Q_1 pressure:



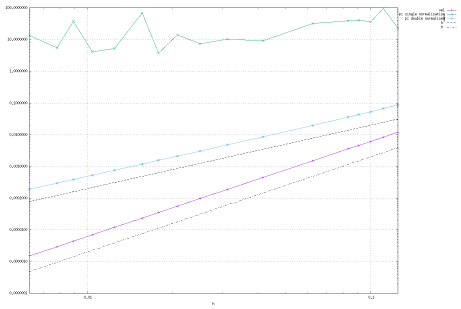
I of course make sure in my code that the pressure fulfills $\int p dV = 0$ but since the global constant function $p = constant$ is both in the Q_1 and Q_0 spaces the resulting normalised pressure was never good (see green line on plot below). This got me thinking about the '2 hydrostatic modes' of Gresho & Sani and I ended up looking at pressure normalisation in the following way (for quads again):

$$\begin{aligned} 0 &= \int_\Omega p dV \\ &= \sum_e \int_{\Omega_e} p dV \\ &= \sum_e \int_{\Omega_e} \left(\sum_{i=1}^{5} \mathcal{N}_i(x,y) p_i \right) dV \\ &= \sum_e \int_{\Omega_e} \left(\sum_{i=1}^{4} \mathcal{N}_i(x,y) p_i + \mathcal{N}_5 p_5 \right) dV \end{aligned} \tag{120.1}$$

with $\mathcal{N}_{1,2,3,4}(r,s) = \frac{1}{4}(1 \pm r)(1 \pm s)$ being the Q_1 basis functions, $p_{1,2,3,4}$ the pressures at the corners of the quad and $\mathcal{N}_5(r,s) = 1$ the Q_0 basis functions with p_5 the elemental pressure. I then obtain

$$0 = \underbrace{\sum_e \int_{\Omega_e} \left(\sum_{i=1}^{4} \mathcal{N}_i(x,y) p_i \right) dV}_{<p>_{Q_1}} + \underbrace{\sum_e \int_{\Omega_e} p_5 dV}_{<p>_{Q_0}} \tag{120.2}$$

and proceed to normalise the pressure by imposing both $< p >_{Q_1} = 0$ and $< p >_{Q_0} = 0$. In this case the 'doubly normalised' pressure agrees with the analytical solution and I obtain the following error convergence plot:



'doubly normalised' pressure error convergence is quadratic (blue line) and velocity error convergence is cubic (purple line). The green line is the 'wrong/raw' pressure.

Breakdown of the code

One starts by loading the required finite element functions for the basis functions, the numerical quadrature and various tools:

```
import FEbasis2D as FE
import FEquadrature as Q
import FEtools as Tools
```

Then the desired setup/manufacture solution is imported, say:

```
import mms_jolm17 as mms
```

The domain is a unit square:

```
Lx=1
Ly=1
```

It is discretised by means of a $nelx \times nely$ cells. If quadrilateral elements are to be used then there are $nelx \times nely$ elements. If triangular elements are to be used then the cells are cut into two triangles and there are then $2 \times nelx \times nely$ elements.

```
nelx=16
```

In practice nely is always set to the nelx value.

In two dimensions there are two velocity degrees of freedom per velocity node but only one pressure degree of freedom per pressure node:

```
ndofV=2
ndofP=1
```

A finite element space must be assigned to both velocity and pressure. For example:

```
Vspace='Q2'
Pspace='Q1'
```

Whether a (bi-)linear mapping or isoparametric mapping is used is set with this parameter:

```
isoparametric=False
```

Whether the mesh is randomized or not is set with

```
randomize_mesh=True
```

Whether the mesh structured or unstructured is set with

```
unstructured=0
```

A quadrature order must be assigned. If quadrilateral elements are used this parameter is the number of quadrature points per dimension. If triangular elements are used it is the total number of quadrature points inside the element. For example:

```
nqpts=Q.nqpts_default(Vpsace)
```

For the chosen velocity and pressure spaces we retrieve the number of nodes ('support points') inside an element.

```
mV=FE.NNN_m(Vspace)
mP=FE.NNN_m(Pspace)
```

We then setup the quadrature rule for an element. This function returns the number of quadrature points inside the element, their coordinates in the r, s space and their weights:

```
nqel,qcoords_r,qcoords_s,qweights=Q.quadrature(Vspace,nqpts)
```

The mesh is then created, or rather the meshes: one for the velocity nodes, one for the pressure nodes. They both count the same number of elements. There are NV velocity nodes and their coordinates are stored in the xV and yV arrays. Likewise there are NP pressure nodes and their coordinates are stored in the xP and yP arrays.

```
if not unstructured:
    NV,nel,xV,yV,iconV=Tools.cartesian_mesh(Lx,Ly,nelx,nely,Vspace,mtype)
    NP,nel,xP,yP,iconP=Tools.cartesian_mesh(Lx,Ly,nelx,nely,Pspace,mtype)
else:
    nel,NV,NP,xV,yV,iconV,xP,yP,iconP=Tools.generate_random_mesh(Lx,nelx,\
                                                    Vspace,Pspace,experiment)
```

Now that we know the number of elements and nodes we can compute the total number of quadrature points nq, the total number of velocity dofs NfemV, the total number of pressure dofs NfemP, and the total number of dofs:

```
nq=nqel*nel
NfemV=NV*ndofV
NfemP=NP*ndofP
Nfem=NfemV+NfemP
```

We will later need two arrays of size NfemV (we are only imposing velocity boundary conditions). bc_fix is a boolean array. We set bc_fix[i]=True if the value of a given velocity dof i is set. and the value of bc_val[i] is the value of the desired boundary condition.

```
bc_fix,bc_val=Tools.bc_setup(xV,yV,uth,vth,Lx,Ly,ndofV,mms.left_bc,mms.right_bc,mms.bottom_bc,mms.
    top_bc)
```

We then build the background mesh used for the mapping:

```
space1=FE.mapping(Vspace)
m1=FE.NNN_m(space1)
N1,nel1,x1,y1,icon1=Tools.cartesian_mesh(Lx,Ly,nelx,nely,space1,mtype)
if randomize_mesh:
    hx=Lx/nelx
    hy=Ly/nely
    Tools.adapt_FE_mesh(x1,y1,icon1,m1,space1,xV,yV,iconV,nel,Vspace)
    Tools.adapt_FE_mesh(x1,y1,icon1,m1,space1,xP,yP,iconP,nel,Pspace)
```

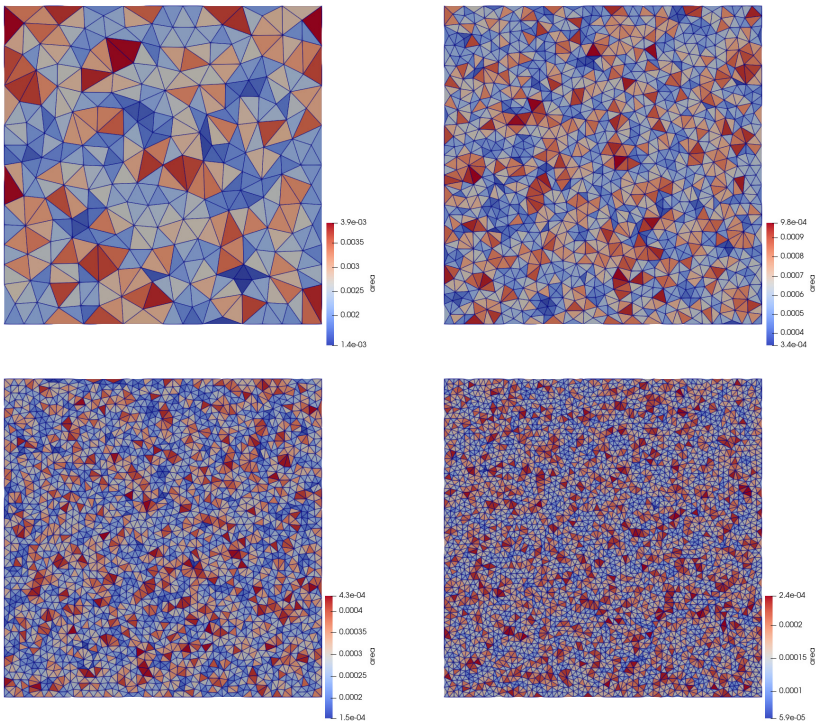
Then we proceed to compute the volume of each element, i.e.

$$V_e = \int_{\Omega_e} dV = \sum_{iq=1}^{n_q} \omega_{i_q} |J_{i_q}|$$

which translates as follows:

```
area=np.zeros(nel,dtype=np.float64)
for iel in range(0,nel):
    for iq in range(0,nqel):
        rq=qcoords_r[iq]
        sq=qcoords_s[iq]
        weightq=qweights[iq]
        NNNV=FE.NNN(rq,sq,Vspace)
        xq=NNNV.dot(xV[iconV[:,iel]])
        yq=NNNV.dot(yV[iconV[:,iel]])
        dNNNVdr=FE.dNNNdr(rq,sq,Vspace)
        dNNNVds=FE.dNNNds(rq,sq,Vspace)
        if isoparametric:
            jcob,jcbi=Tools.J(mV,dNNNVdr,dNNNVds,xV[iconV[0:mV,iel]],yV[iconV[0:mV,iel]])
        else:
            dNNN1dr=FE.dNNNdr(rq,sq,space1)
            dNNN1ds=FE.dNNNds(rq,sq,space1)
            jcob,jcbi=Tools.J(m1,dNNN1dr,dNNN1ds,x1[icon1[0:m1,iel]],y1[icon1[0:m1,iel]])
        area[iel]+=jcob*weightq
```

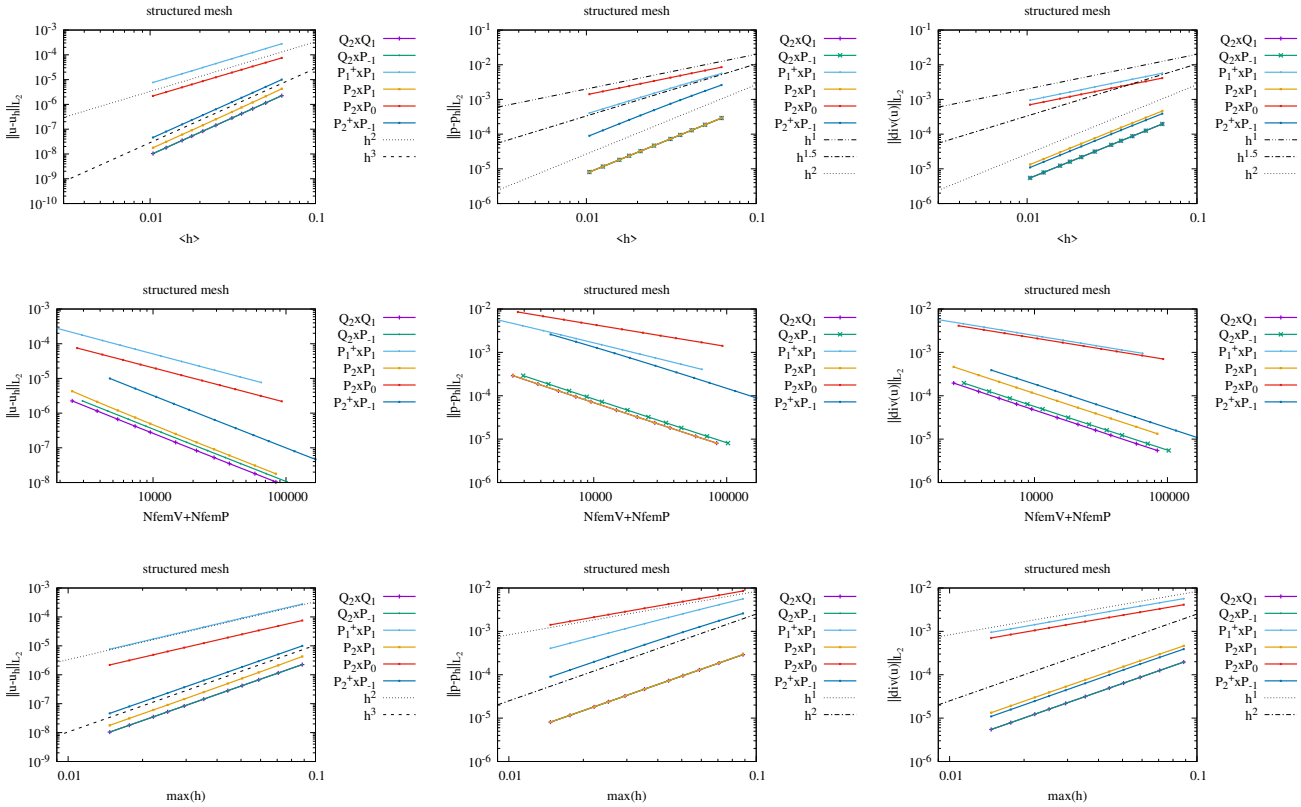
Examples of unstructured meshes



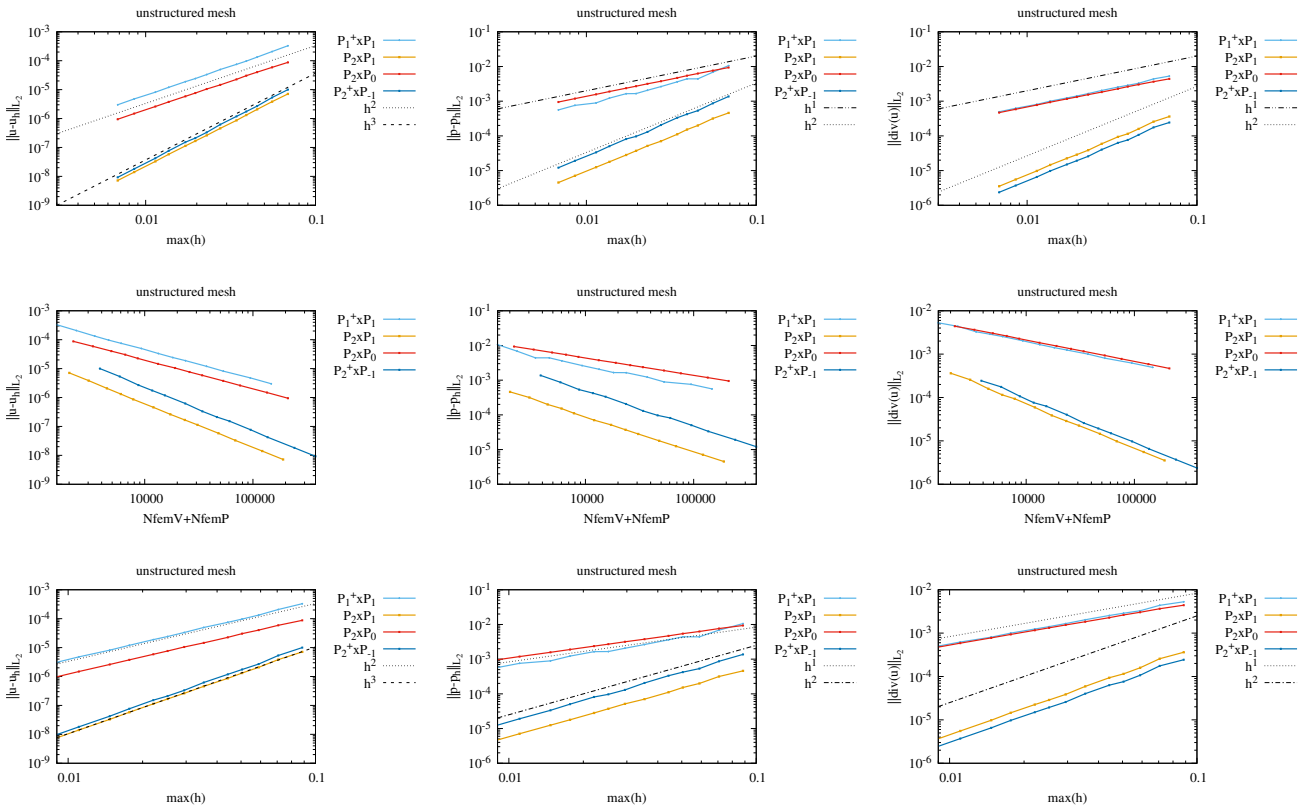
Results - Donea & Huerta (nov. 2024)

`script_paper_dh_structured` and `script_paper_dh_unstructured`

Structured mesh

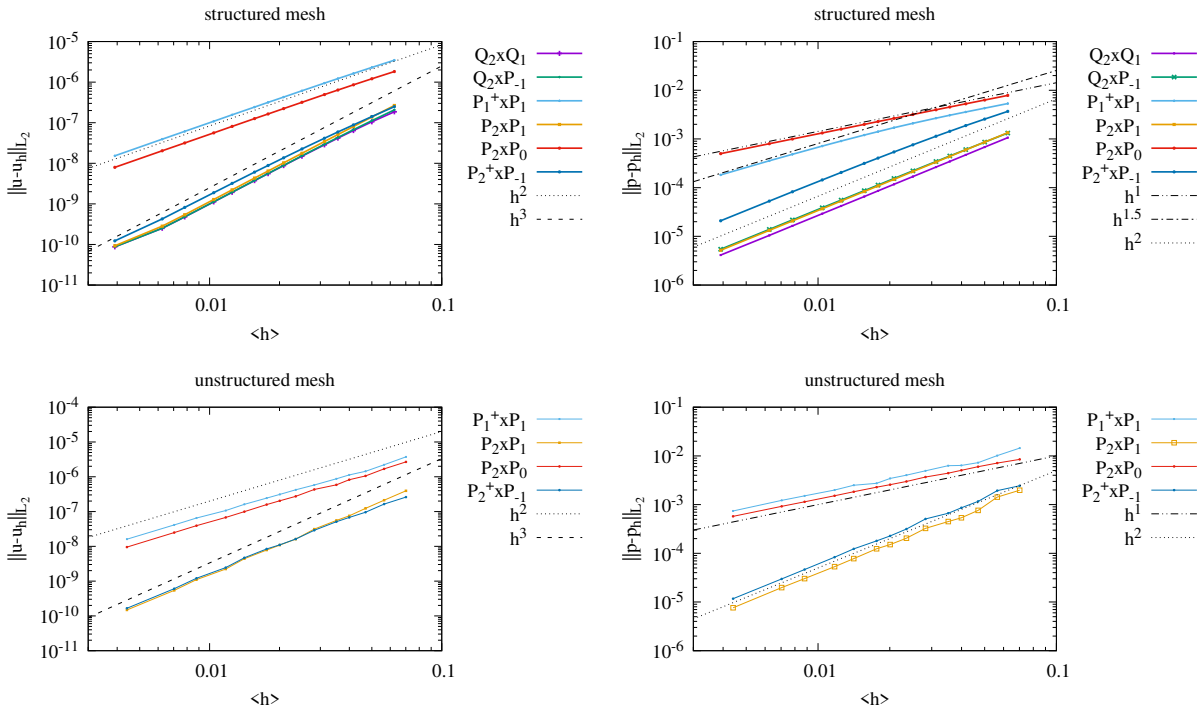


Unstructured mesh



NEW Results - SolKz

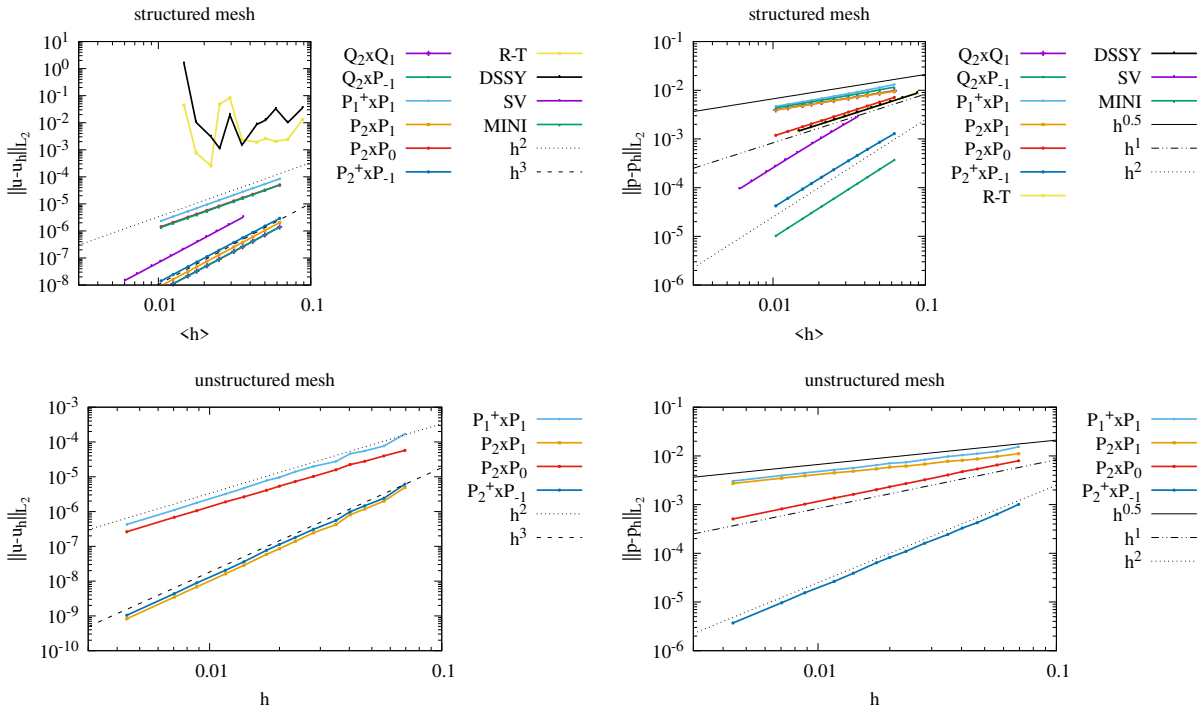
`script_paper_solkz_structured` and `script_paper_solkz_unstructured`



de Montserrat, Morgan, and Hasenclever [87] uses CR elements and report cubic convergence for velocity and quadratic for pressure.

NEW Results - SolCx

`script_paper_solcx_structured` and `script_paper_solcx_unstructured`



de Montserrat, Morgan, and Hasenclever [87] uses CR elements and also reports cubic convergence for velocity and quadratic for pressure for both even and odd meshes

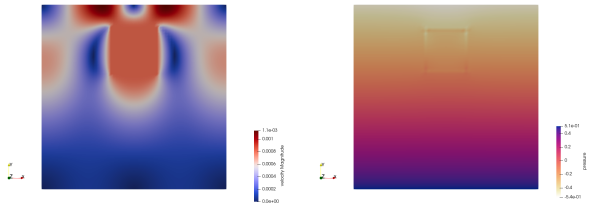
NEW Results - square sinker

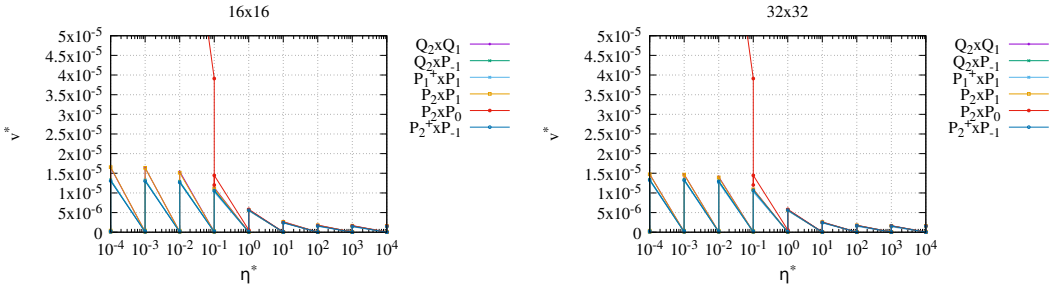
The domain is the unit square. Free-slip boundary conditions are applied on all four sides. The domain is filled with fluid with $\eta_f = 1$ and $\rho_f = 1$. There is a square sinker in the middle of the domain of size 0.25×0.25 with $\eta_s = 100$ and $\rho_s = 1.001$. Gravity is set to $\vec{g} = -\vec{e}_y$. Resolutions nelx=16,32,64,128 are chosen so that element boundaries align with sinker boundaries (material averaging is then irrelevant, an element is either 100% fluid or 100% sinker material).

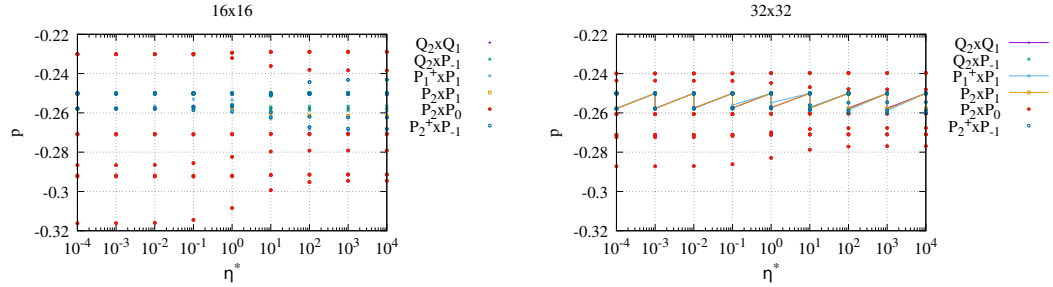
Reported velocity is the root mean square velocity inside the whole block. Reported pressure is the pressure(s) on the middle node of the block.

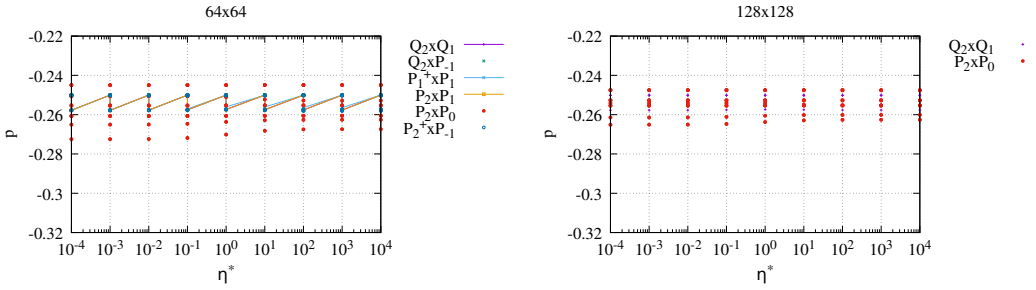
structured mesh

`script_paper_sinker`



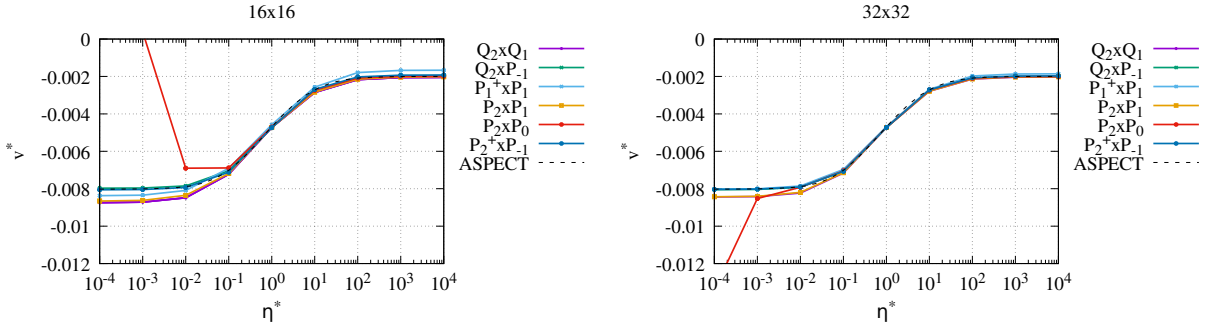


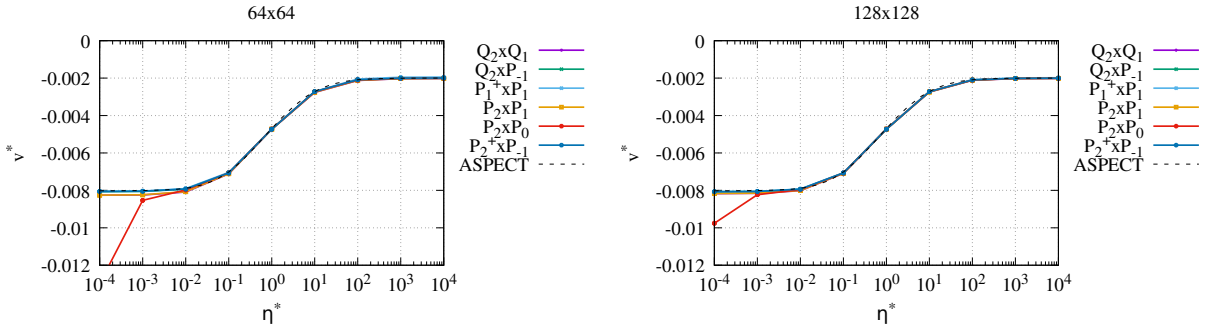


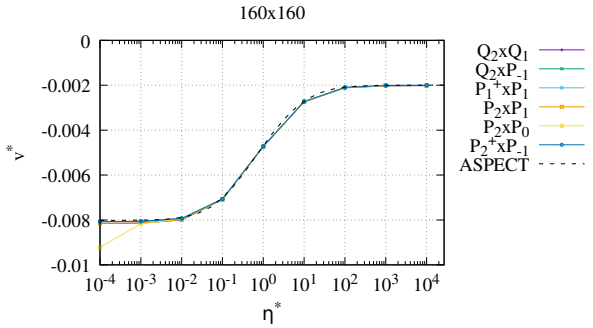

64x64
128x128
p
η*
Q2xQ1
Q2xP-1
P1+xP1
P2xP1
P2xP0
P2+xP-1
-0.22
-0.24
-0.26
-0.28
-0.3
-0.32
10-4 10-3 10-2 10-1 100 101 102 103 104

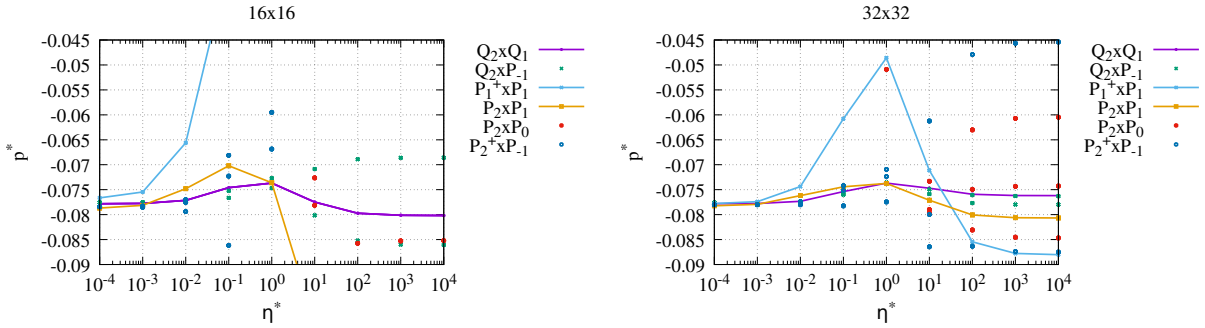
NEW Results - square sinker (reduced density)

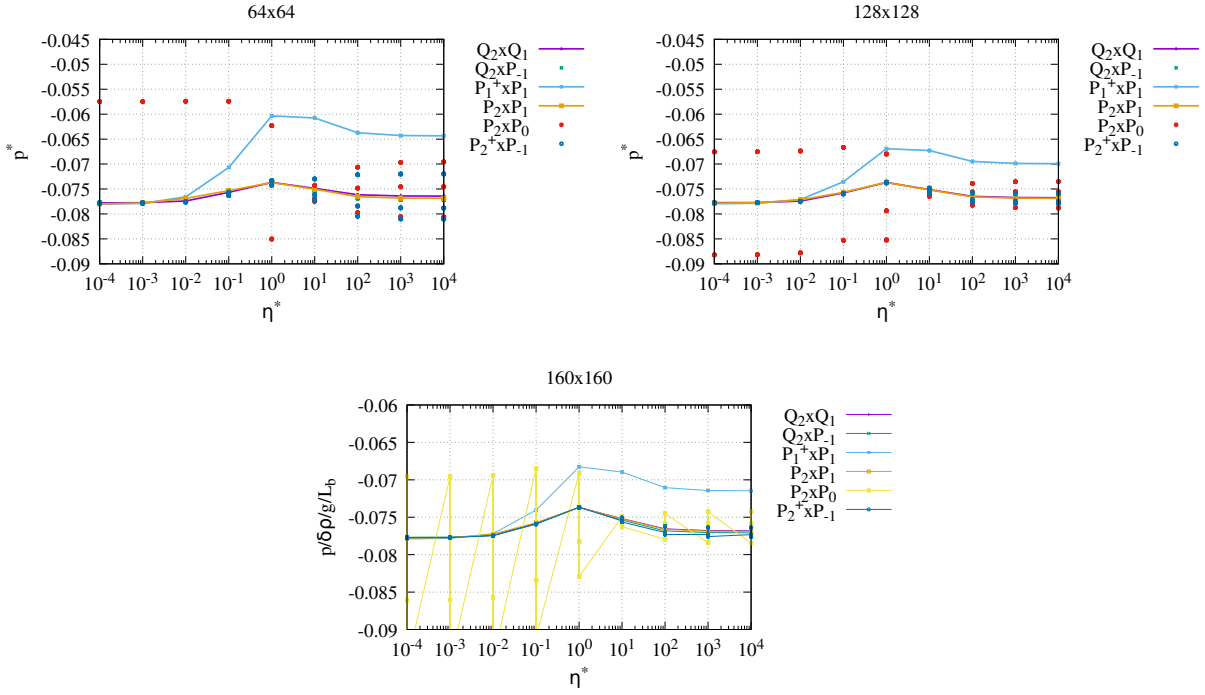
Structured mesh





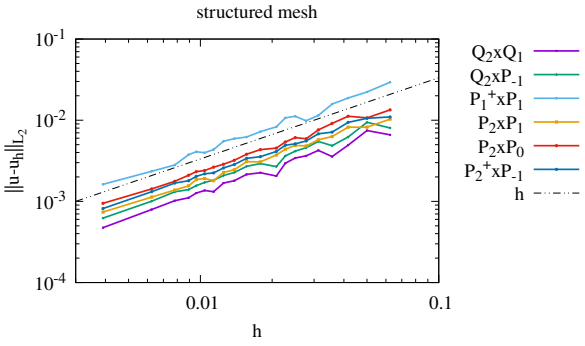


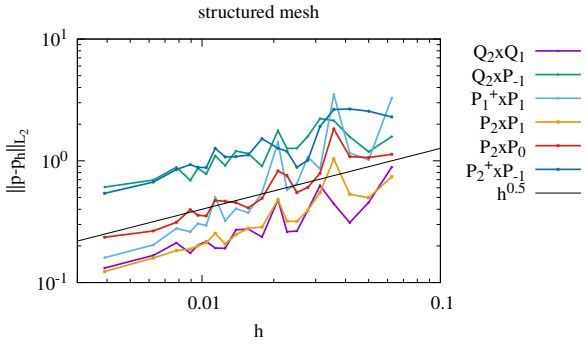


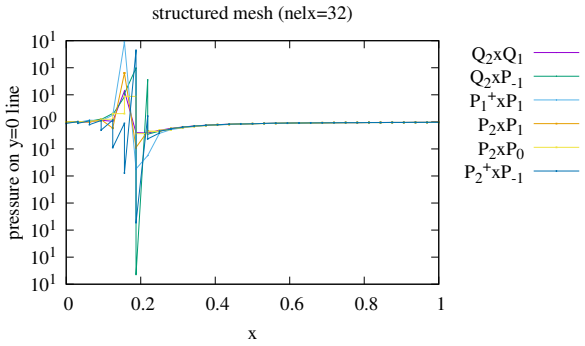

64x64
128x128
160x160
p*
p/δρ/g/L_b
η*
Q2xQ1
Q2xP-1
P1+xP1
P2xP1
P2xP0
P2+xP-1

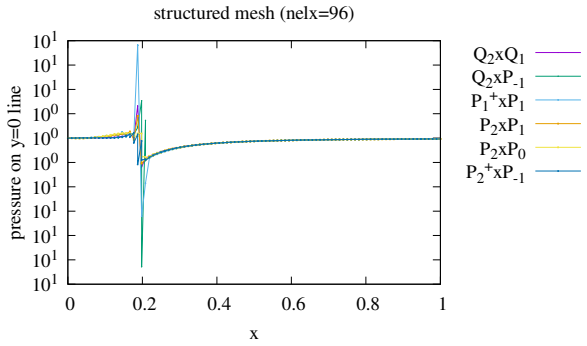
NEW Results - SolVi

structured mesh

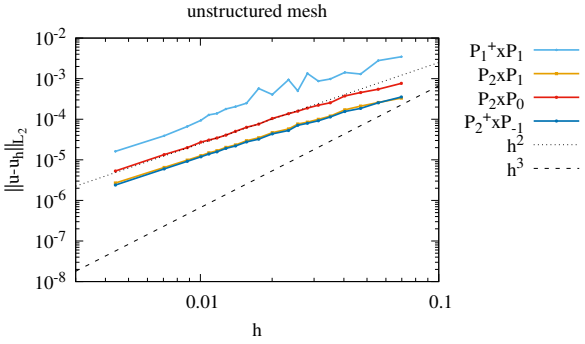


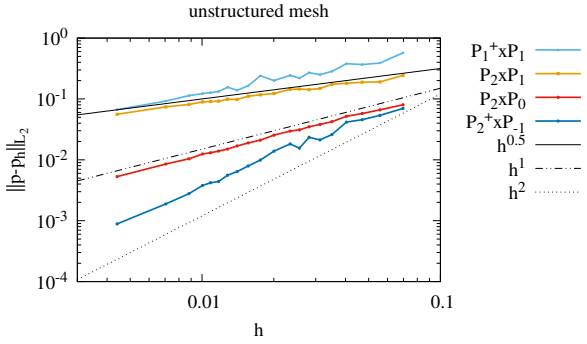


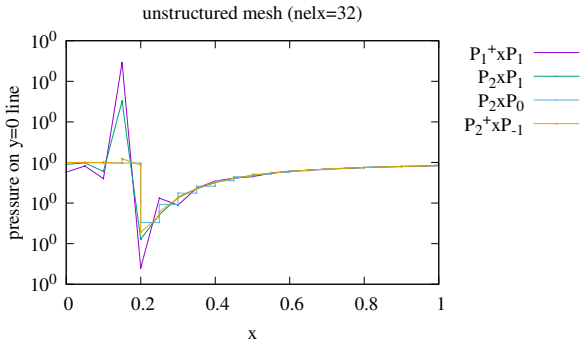


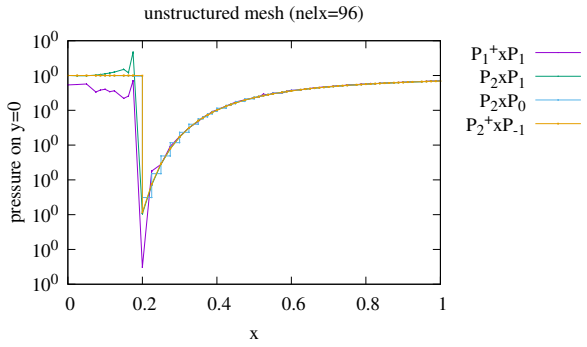


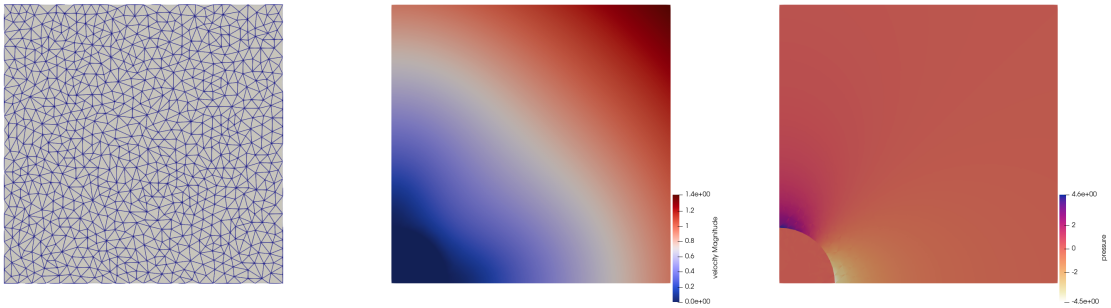
unstructured mesh



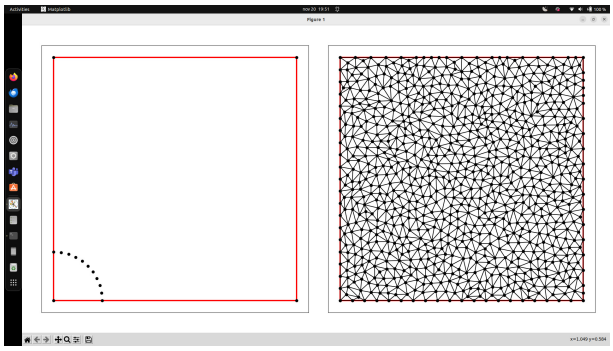






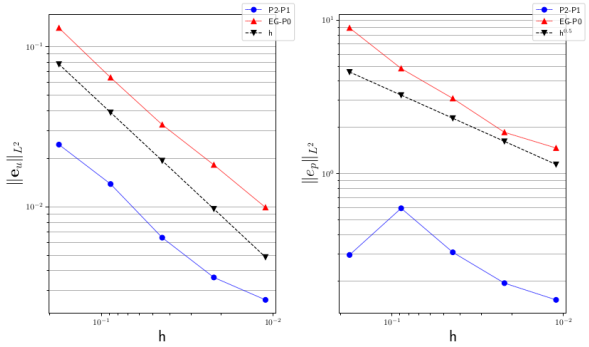


P2+P-1, nelx=32



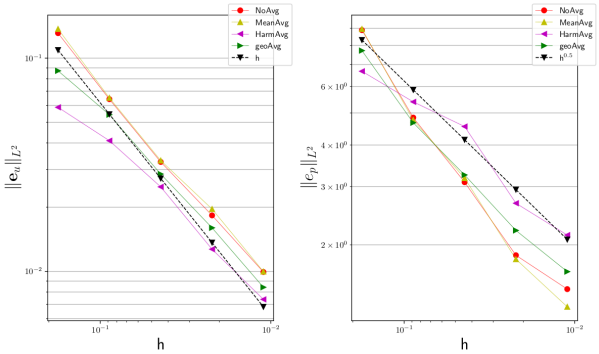
How the unstructured mesh is built.

In Böhm [29] (2023) the author recovers rates 1-0.5 for $P_2 \times P_1$.



Note that this is obtained with the Enriched Galerkin method.

He also explores the influence of averaging inside each element:



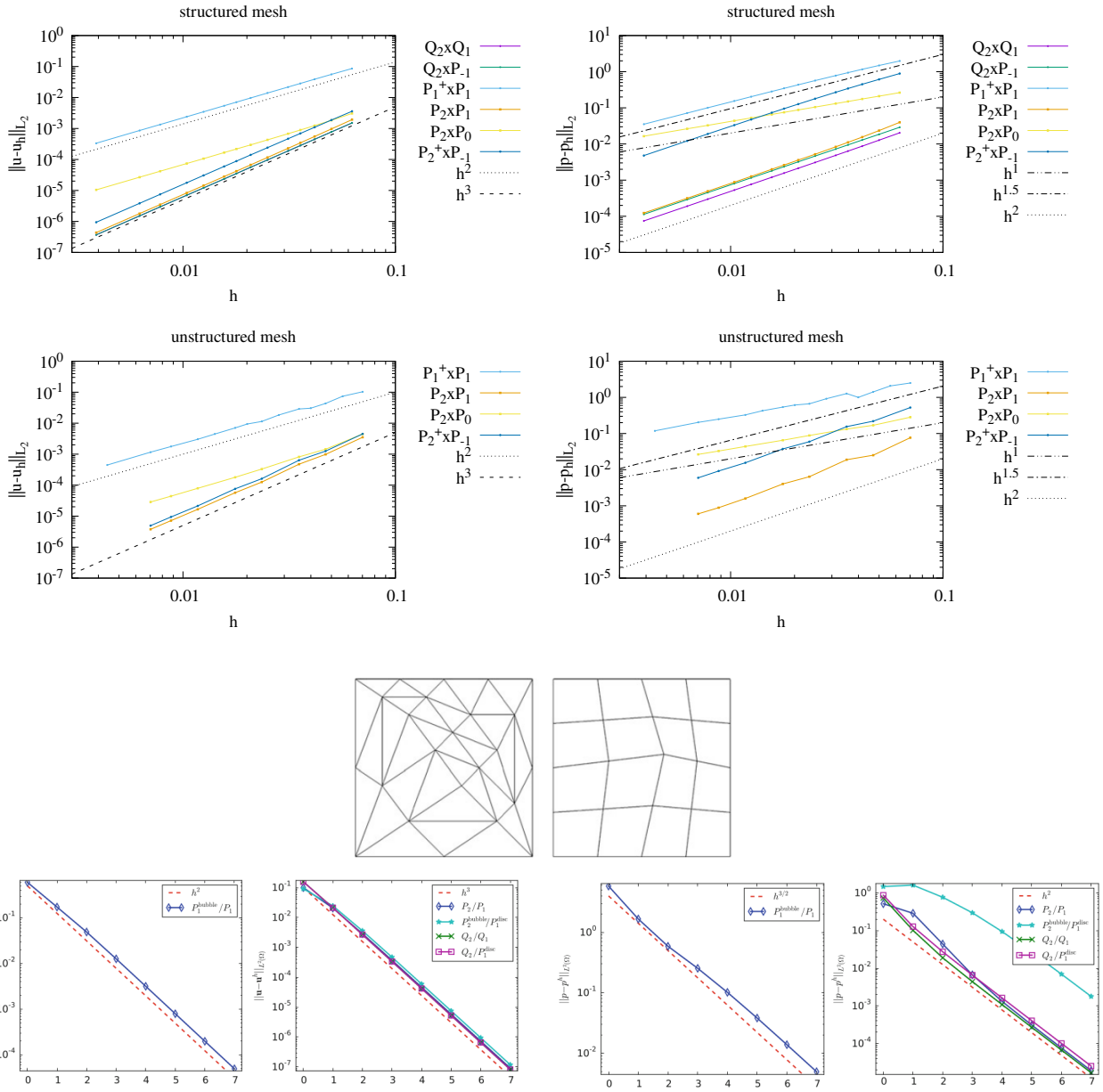
Note that this is obtained with the Enriched Galerkin method.

NEW Results - mms from John's book

It is derived in Section 18.17. The velocity and pressure fields are:

$$u(x,y) = 1000x^2(1-x)^4y^2(3-5y)(1-y) \quad (120.3)$$
$$v(x,y) = -10002x(1-3x)(1-x)^3y^3(1-y)^2 \quad (120.4)$$
$$p(x,y) = \pi^2[xy^3\cos(2\pi x^2y) - x^2y\sin(2\pi xy)] + 1/8 \quad (120.5)$$

In this case the viscosity is constant $\eta = 1$. The same fields with non-constant viscosity is explored in the jokn16 benchmark here after.



Taken from John [171] (book). Top are the initial grids (level 0).

check the type of grid used in the book

Comparison between my results and John's. Between parenthesis are the rates obtained by John. I get identical rates: happy! Rather interestingly he and I (on structured grid) find that CR pressure errors are much higher (more than 2 orders of magnitude!) than all other elements that converge quadratically.

	vel	p
$\boldsymbol{Q}_2 \times Q_1$	3(3)	2(2)
$\boldsymbol{Q}_2 \times P_{-1}$	3(3)	2(2)
$\boldsymbol{P}_1^+ \times P_1$	2(2)	1.5 (1.5)
$\boldsymbol{P}_2 \times P_1$	3(3)	2(2)
$\boldsymbol{P}_2 \times P_0$	2(x)	1 (x)
$\boldsymbol{P}_2^+ \times P_{-1}$	3(3)	2(2)

check rate of mini that seems to have decreased!

Rayleigh-Taylor wave (Nov. 2024)

This benchmark is described in Section 19.2, although here the domain is the unit square and all quantities are effectively dimensionless. Free slip boundary conditions are prescribed on left and right sides, no-slip on bottom and top. The implementation in `mms_RTwave.py` is as follows:

```
amplitude=0.01
llambda=0.5
def eta(x,y,param):
    if y>0.5+amplitude*np.cos(2*np.pi*x/llambda):
        return 1
    else:
        return param
def by(x,y,drho):
    if y>0.5+amplitude*np.cos(2*np.pi*x/llambda):
        return -(1+drho)
    else:
        return -1
```

The bash script `script_paper_rtwave` loops over $\eta_2 = \eta^\star \in [10^{-2} : 10^2]$ ('param' above) for drho=0.1, while λ is set to 0.5 and the amplitude is set to $\Delta = 0.01$.

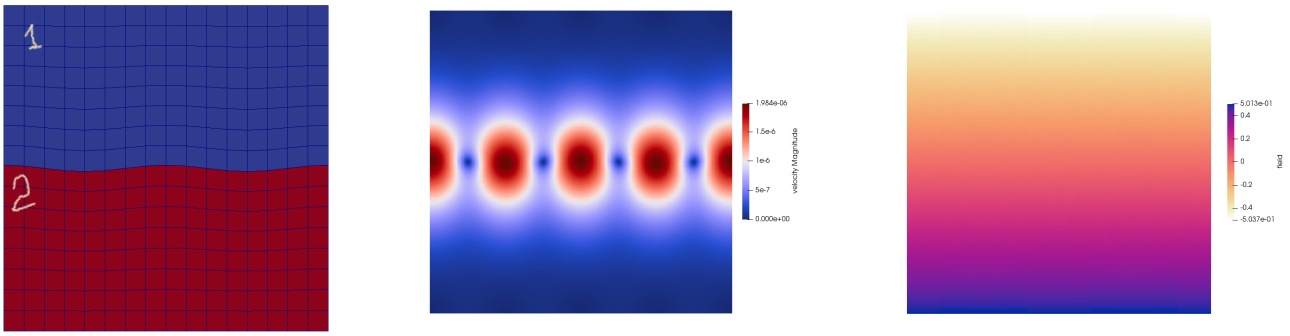
Since this experiment requires a bespoke mesh, we find this in the code:

```
if experiment=='RTwave':
    Tools.deform_mesh_RTwave(x1,y1,N1,Lx,Ly,nelx,nely)
    Tools.adapt_FE_mesh(x1,y1,icon1,m1,space1,xV,yV,iconV,nel,Vspace)
    Tools.adapt_FE_mesh(x1,y1,icon1,m1,space1,xP,yP,iconP,nel,Pspace)
```

which ensures that mid edge nodes are placed so that all elements are straight edged. The results for the benchmark are exported as follows:

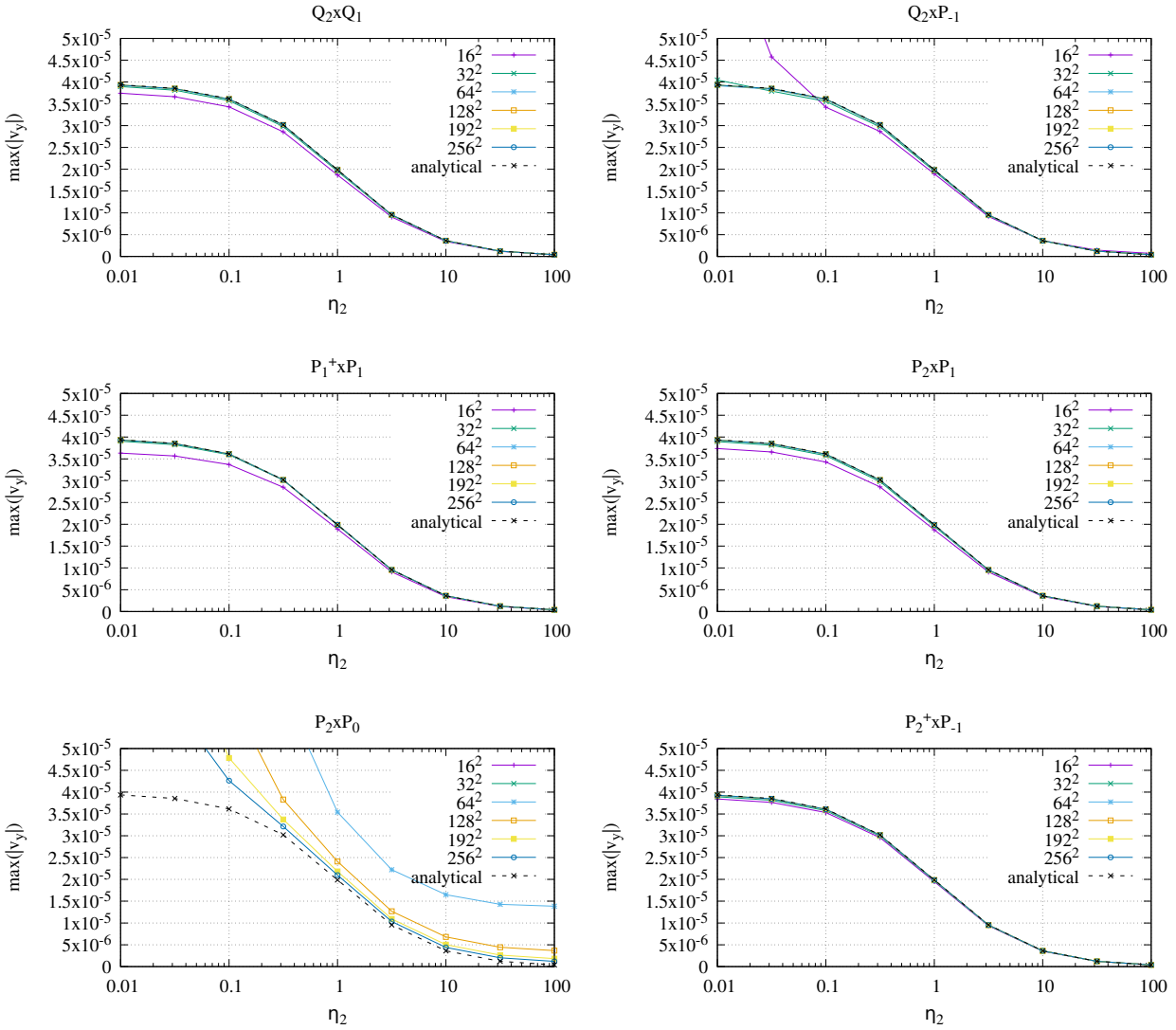
```
print('-----> rt_wave',np.max(abs(v)),phi1,etastar,drho,nelx,val)
```

where val is the analytical value.

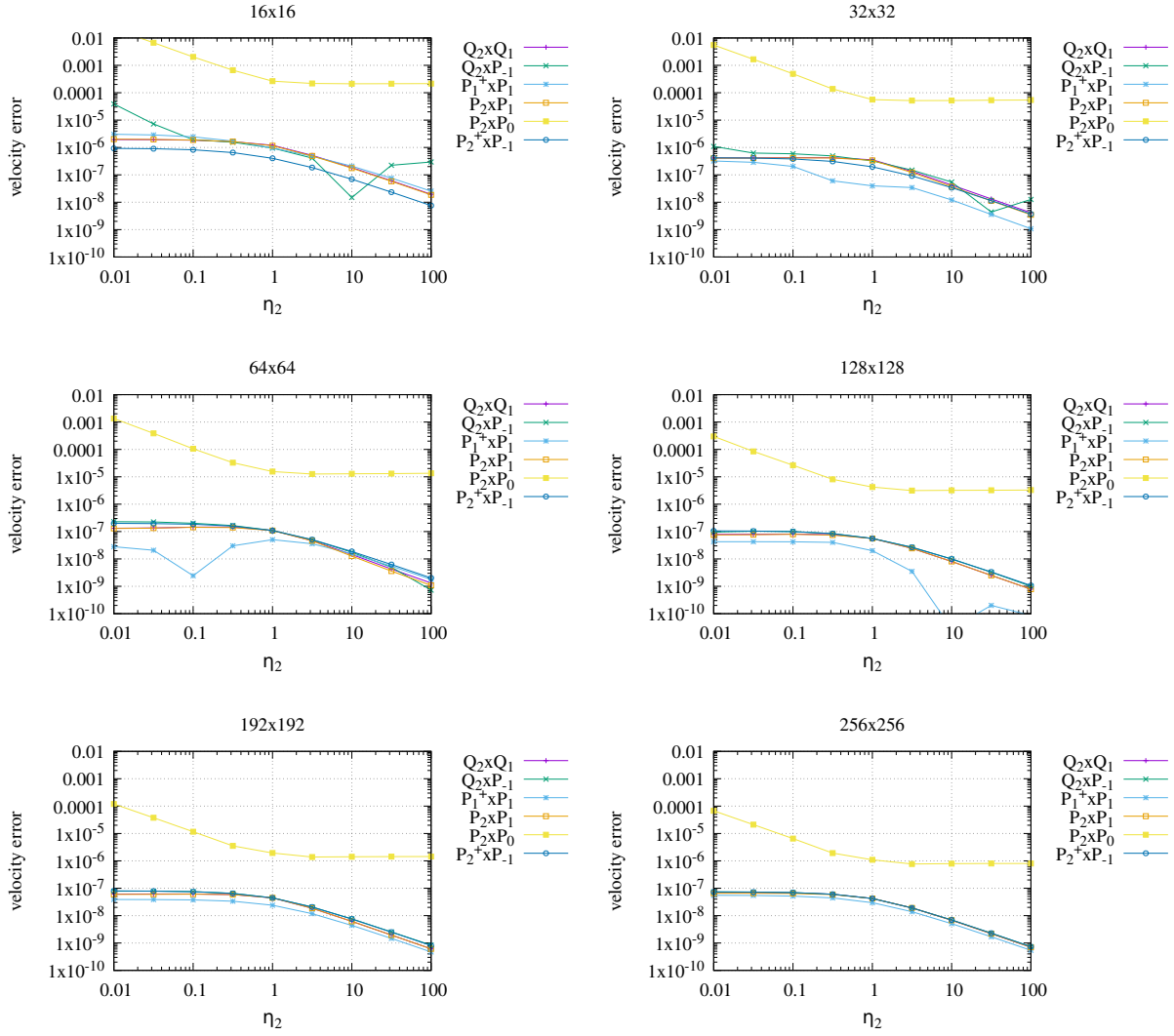


Left: mesh for 16x16; middle: velocity; Right: pressure field. Obtained for isoviscous case $\eta^\star = \eta_2/\eta_1 = 1$.

The figures below show $max(v_y)$ for the whole domain (which is in fact the vertical velocity at the bellies of the perturbation) against the expected analytical values as a function of the viscosity η_2 ($\eta_1 = 1$) for all 6 element pairs.



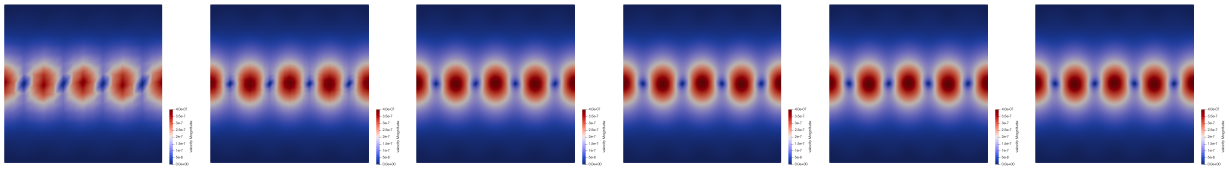
The figures below now show the velocity error $|max(v_y) - v_{theory}|$ as a function of viscosity again, but this time for different resolutions, comparing element pairs.



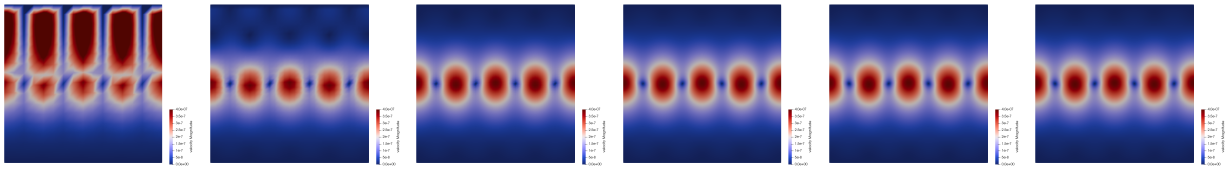
$P_2 \times P_0$ is surprisingly bad but error does decrease with resolution. Otherwise all other elements yield about the same error, except $P_1^+ \times P_1$ who seems to be marginally better. Rather worryingly the error does not much improve much with increasing resolution.

Below are the velocity fields obtained for $\eta_2 = 100$ (i.e. $\eta^\star = 100$). From left to right: resolution $16^2, 32^2, 64^2, 128^2, 192^2, 256^2$. Continuous pressure elements seem to fare better at low resolutions. Only $P_2 \times P_0$ fails to recover the correct field at high resolution. Colorscale barely readable, but identical for all plots.

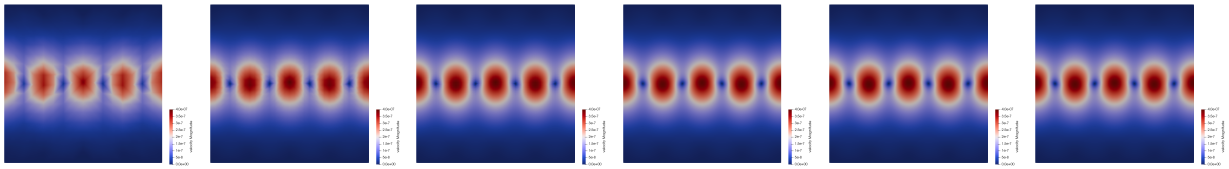
- $Q_2 \times Q_1$



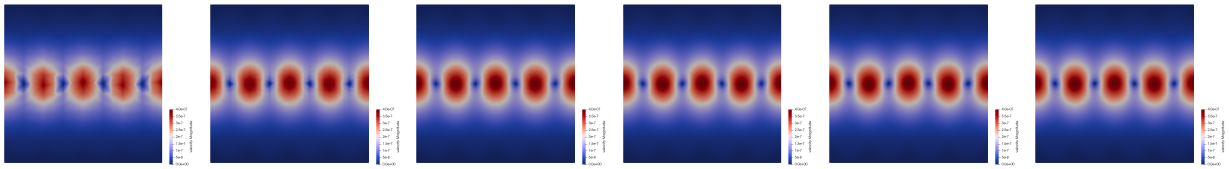
- $Q_2 \times P_{-1}$



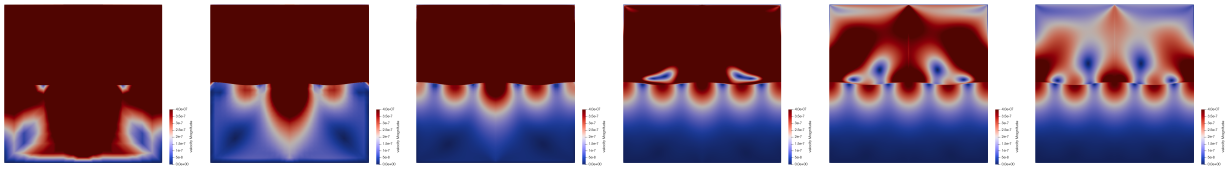
- $P_1^+ \times P_1$



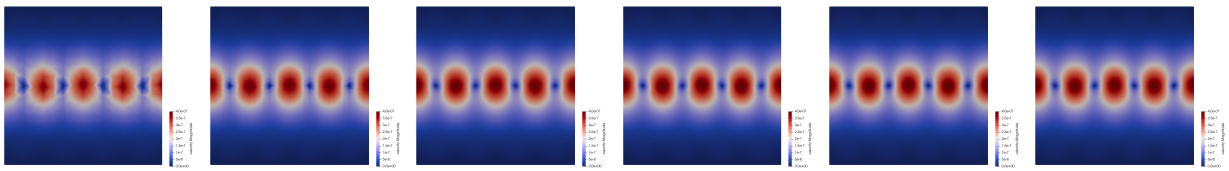
- $P_2 \times P_1$



- $P_2 \times P_0$



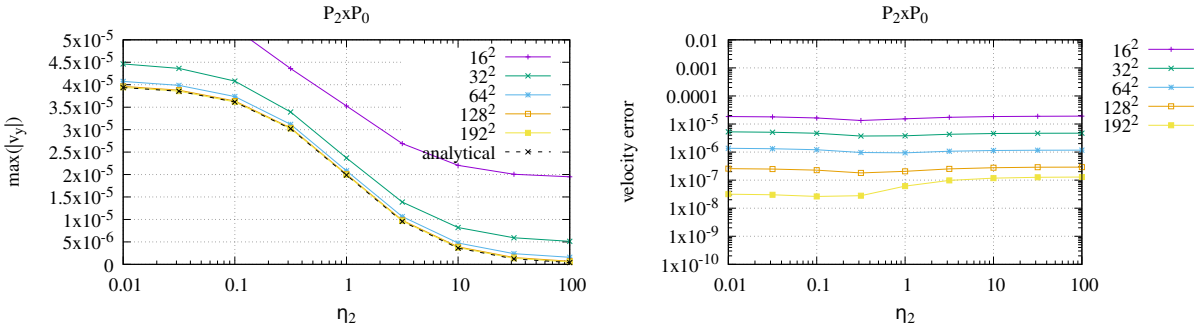
- $P_2^+ \times P_{-1}$



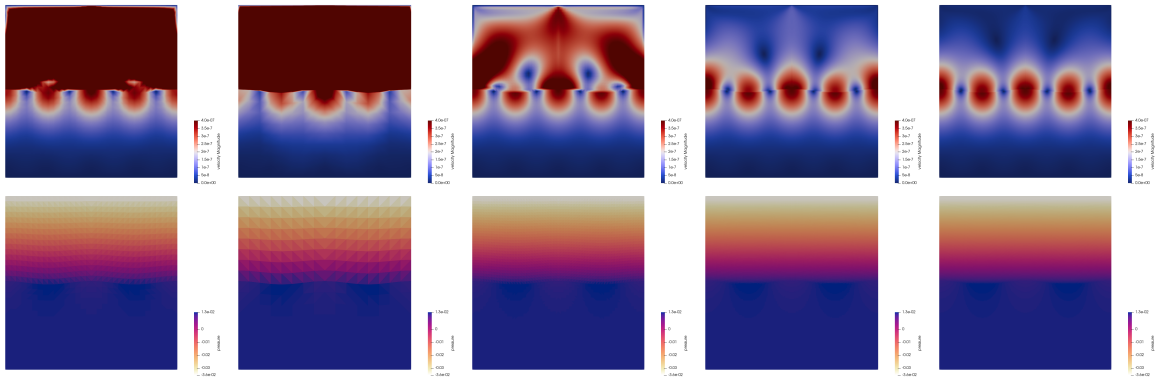
I am quite puzzled by the abysmal results obtained with the $P_2 \times P_0$ so I have re-run the experiment run with reduced densities, i.e. top layer has density 0.1 and bottom layer has zero density (while the lower layer is still 100 times more viscous than the top layer), i.e. $\eta_\star = 100$.

We find that results are substantially improved, which means that similarly to the $Q_1 \times P_0$ element the background lithostatic pressure is a source of large errors in the context of buoyancy driven flow. In this case it could also be that the mesh topology/anisotropy plays a role (as opposed to the $Q_1 \times P_0$ one?).

The recovered values are getting closer to the expected analytical ones with increasing resolution and the error decreases in a predictable manner.



Left to right: velocity and pressure fields obtained with $16^2, 32^2, 64^2, 128^2, 192^2$ resolutions:



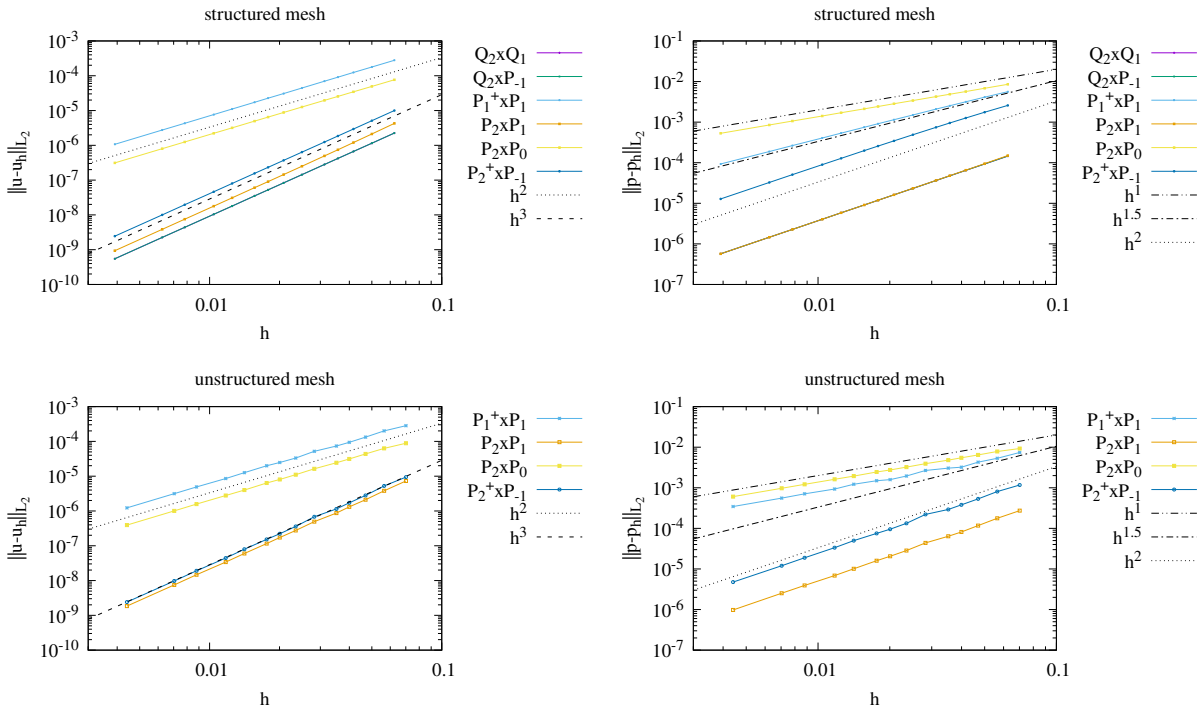
NEW Results - bocg12 mms

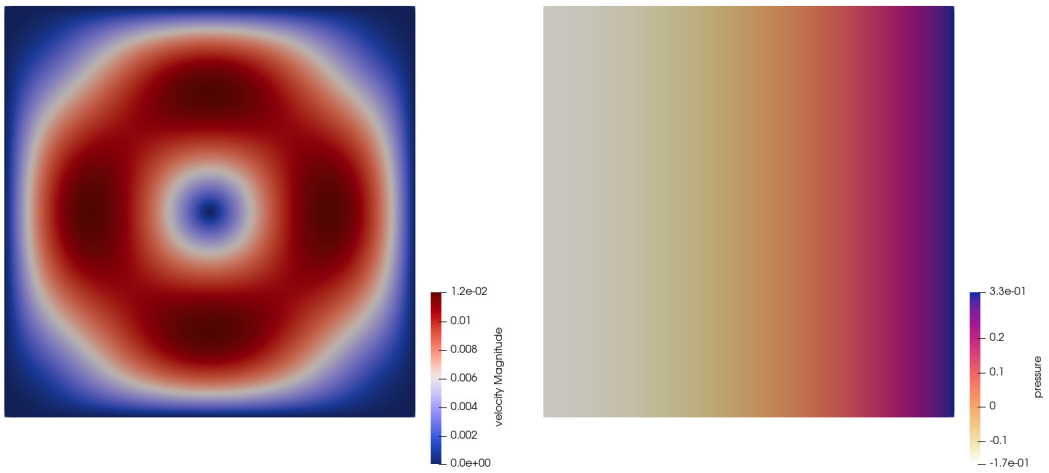
It originates in page 392 of Boffi, Cavallini, Gardini, and Gastaldi [28], see Section 18.16. I recall here the solution:

$$
\begin{aligned}
u(x,y) &= x^2(x-1)^2 2y(y-1)(2y-1) \\
v(x,y) &= -2x(x-1)(2x-1)y^2(y-1)^2 \\
p(x,y) &= \frac{1}{2}x^2 - \frac{1}{6}
\end{aligned}
$$

P_2/P_1								
h_x	$\|p-p_h\|_{L^2}$		$\|\mathbf{u}-\mathbf{u}_h\|_{L^2}$		$\|\nabla(\mathbf{u}-\mathbf{u}_h)\|_{L^2}$		$\|\operatorname{div}\mathbf{u}_h\|_{L^2}$	
	Error	Rate	Error	Rate	Error	Rate	Error	Rate
1/4	2.713e–03	–	2.862e–04	–	8.270e–03	–	6.373e–03	–
1/8	6.134e–04	2.1	3.750e–05	2.9	2.296e–03	1.8	1.797e–03	1.8
1/16	1.468e–04	2.1	4.640e–06	3.0	5.934e–04	2.0	4.710e–04	1.9
1/32	3.643e–05	2.0	5.785e–07	3.0	1.499e–04	2.0	1.197e–04	2.0
1/64	9.099e–06	2.0	7.232e–08	3.0	3.758e–05	2.0	3.006e–05	2.0
P_2/P_1+P_0								
h_x	$\|p-p_h\|_{L^2}$		$\|\mathbf{u}-\mathbf{u}_h\|_{L^2}$		$\|\nabla(\mathbf{u}-\mathbf{u}_h)\|_{L^2}$		$\|\operatorname{div}\mathbf{u}_h\|_{L^2}$	
	Error	Rate	Error	Rate	Error	Rate	Error	Rate
1/4	3.335e–03	–	2.939e–04	–	8.372e–03	–	6.385e–03	–
1/8	6.927e–04	2.3	3.765e–05	3.0	2.311e–03	1.9	1.809e–03	1.8
1/16	1.542e–04	2.2	4.649e–06	3.0	5.949e–04	2.0	4.724e–04	1.9
1/32	3.699e–05	2.1	5.790e–07	3.0	1.500e–04	2.0	1.198e–04	2.0
1/64	9.140e–06	2.0	7.235e–08	3.0	3.759e–05	2.0	3.007e–05	2.0

`script_paper_bocg12_structured` and `script_paper_bocg12_unstructured`

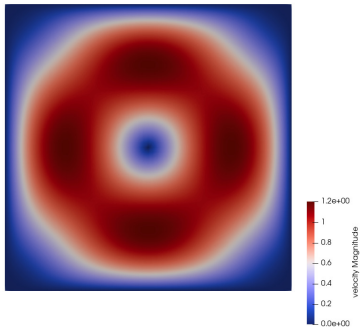


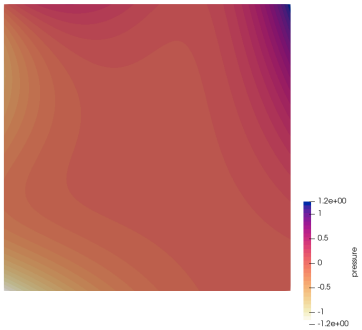

1.2e-02
0.01
0.008
0.006
0.004
0.002
0.0e+00
velocity Magnitude
3.3e-01
0.2
0.1
0
-0.1
-1.7e-01
pressure

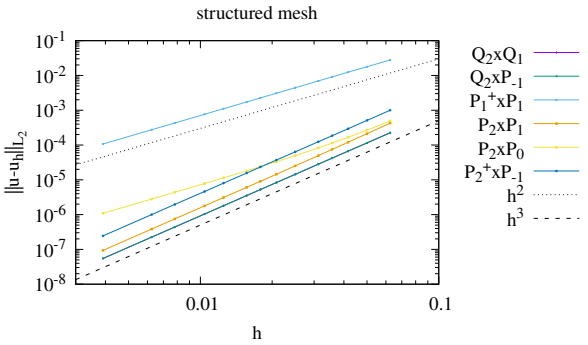
NEW Results - jolm17 mms

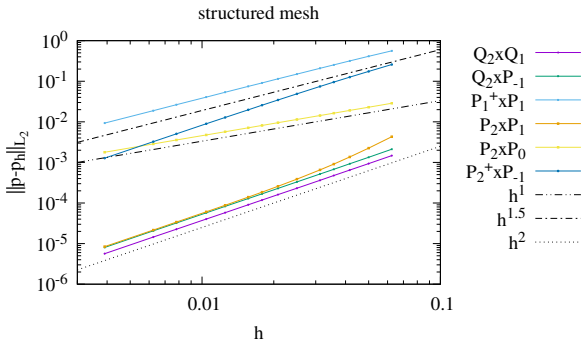
It originates in John, Linke, Merdon, Neilan, and Rebholz [172] (2017), see Section 18.36. I recall here the solution:

$$u(x,y) = 200x^2(1-x)^2y(1-y)(1-2y) \tag{120.6}$$
$$v(x,y) = -200x(1-x)(1-2x)y^2(1-y)^2 \tag{120.7}$$
$$p(x,y) = 10\left[(x-1/2)^3y^2 + (1-x)^3(y-1/2)^3\right] \tag{120.8}$$









NEW Results - jokn16 mms

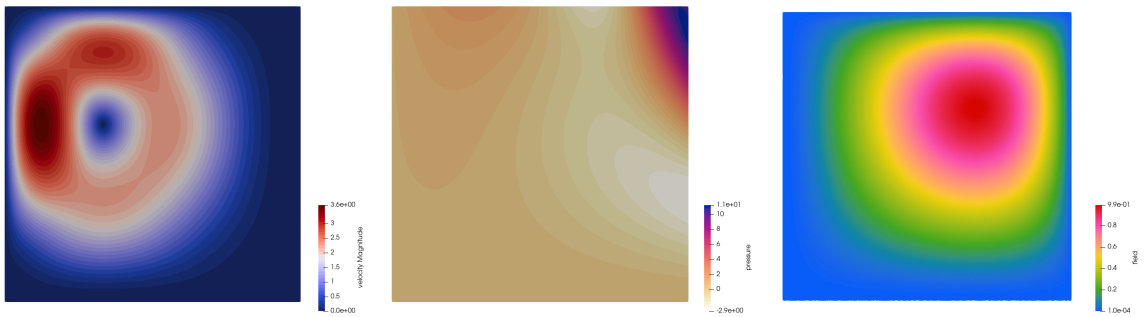
`script_paper_jokn16_structured` and `script_paper_jokn16_unstructured`

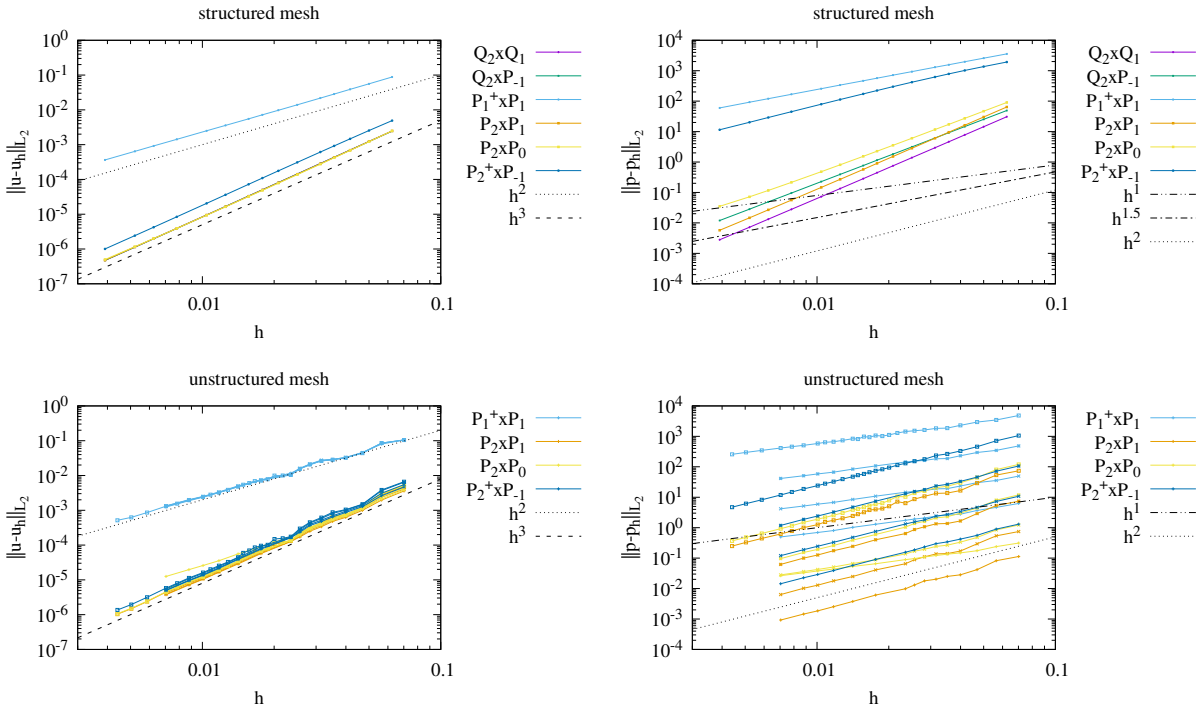
This benchmark is presented in Section 18.18.

$$
\begin{aligned}
u(x,y) &= 1000x^2(1-x)^4y^2(3-5y)(1-y) \\
v(x,y) &= -10002x(1-3x)(1-x)^3y^3(1-y)^2 \\
p(x,y) &= \pi^2[xy^3\cos(2\pi x^2y) - x^2y\sin(2\pi xy)] + 1/8 \\
\eta_1(x,y) &= \eta_{min} + (\eta_{max}-\eta_{min})x^2(1-x)y^2(1-y)\frac{721}{16}
\end{aligned}
$$

with $\eta_{min} = 1$ and $\eta_{max} = 10^p$.

Note that this is the 1st benchmark with viscosity contrast where the rhs (forcing term) actually contains the derivative of the viscosity.





obtained with $p = 1, 2, 3, 4$

Results - linear pressure 'plin' mms

This manufactured solution is presented in Section 18.6.

The domain is the unit square. The velocity field is the same as the Donea & Huerta benchmark:

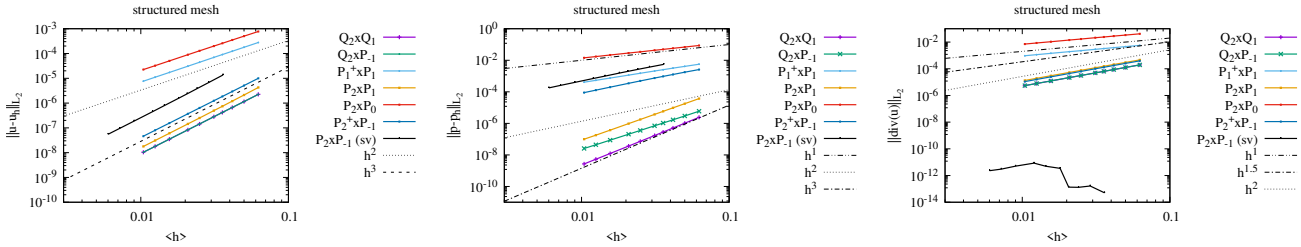
$$\begin{aligned} u(x,y) &= x^2(1-x)^2(2y-6y^2+4y^3) \\ v(x,y) &= -y^2(1-y)^2(2x-6x^2+4x^3) \end{aligned}$$

but the pressure is different and given by a linear field with zero average over the domain:

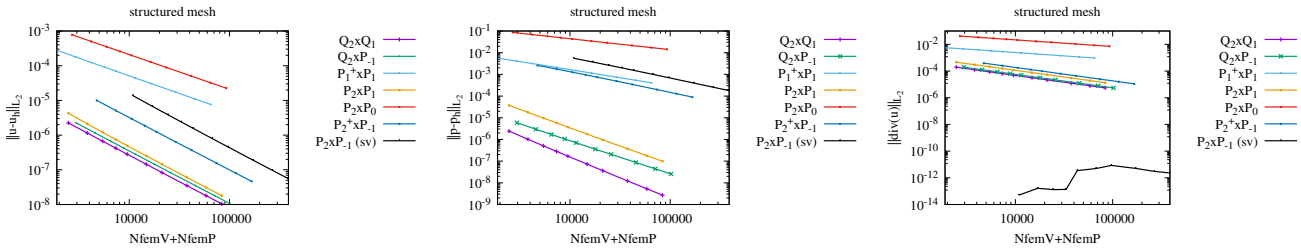
$$p(x,y) = a\left(x-\frac{1}{2}\right) + b\left(y-\frac{1}{2}\right)$$

with $a=3$ and $b=5$.

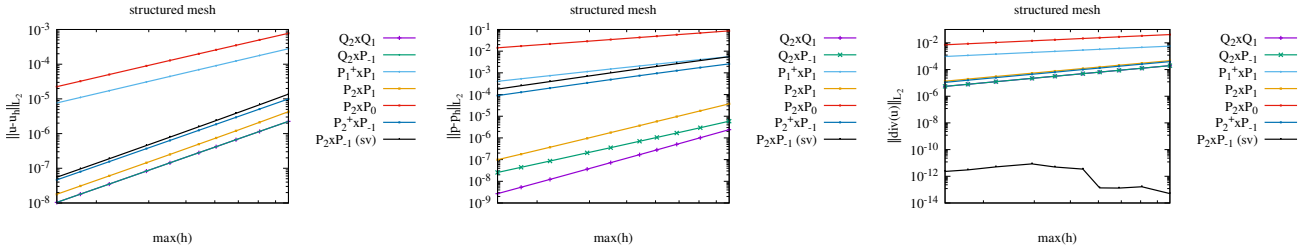
One can plot the errors as a function of the average element size, where the size of an element is computed using the square root of the element area (If the triangles are close to be equilateral this is a reasonable assumption, but we know that in the case of the barycentric refinement for the Scott-Vogelius we obtain flat triangles).



We can plot the same errors as a function of the total number of dofs in the system, which puts the S-V element at a disadvantage:



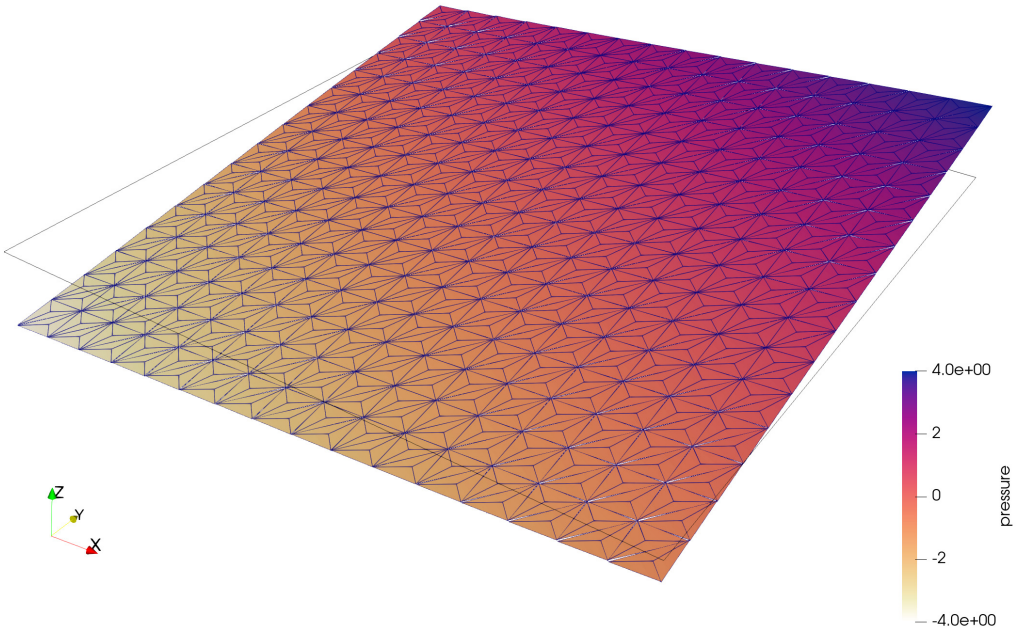
Finally we can plot the errors as a function of the maximum element size computed as the longest edge of each triangle (In practice, since we start from a $nelx \times nely$ mesh of square elements subdivided in two, the used mesh size in the plot is simply $Lx/nelx * \sqrt{2}$, i.e. the diagonal of a quadrilateral). In this case the S-V element pair is very close to the Crouzeix-Raviart pair and the difference can be explained by the triangle 'aspect ratios' (W wrote: " There is of course the issue that the triangles you use have fairly large aspect ratios (or, alternatively, rather large and small angles). That goes into the error constant").



If my implementation is correct, this makes the S-V element pair rather useless: more costly than C-R, mesh is a pain to build, and accuracy is less than C-R. I need to replicate all this on unstructured meshes but I do not hold my breath.

Wolfgang adds: " so if you can reproduce the exact solution, you've got a pretty good case that your implementation is correct. The fact that you get comparable accuracy when you measure 'h' as the longest edge length also suggests that the issue is simply the use of poorly shaped elements. I think you've found the reason for the poor performance.

I will note that the longest edge is not a silly measure. If you try to approximate a curved function in 1d with piecewise linears, the size of the element clearly matters – that's what determines the accuracy. In the mesh you use, if the exact solution varied quadratically in the direction bottom left – top right, the long edges in this direction are going to be the dominant cause of the error. The fact that the elements are smaller than $h^2/2$ "on average" (with h=longest edge length), or that the mesh size is smaller in the direction of the other diagonal is entirely immaterial in that case. That mesh is just pretty terrible, and you will probably get poor errors from any other element on this mesh as well. "



Pressure field obtained with S-V element based on 16x16 quads.

Finally, it was also proposed that the additional node inside each triangle be the incenter and not the barycenter. In our case this yields a mesh with triangles of better quality (according to q1 and q2 measurements) but the errors are marginally better for incenter than barycenter.

Results - Linke & Rebholz (2019) mms (nov. 2024)

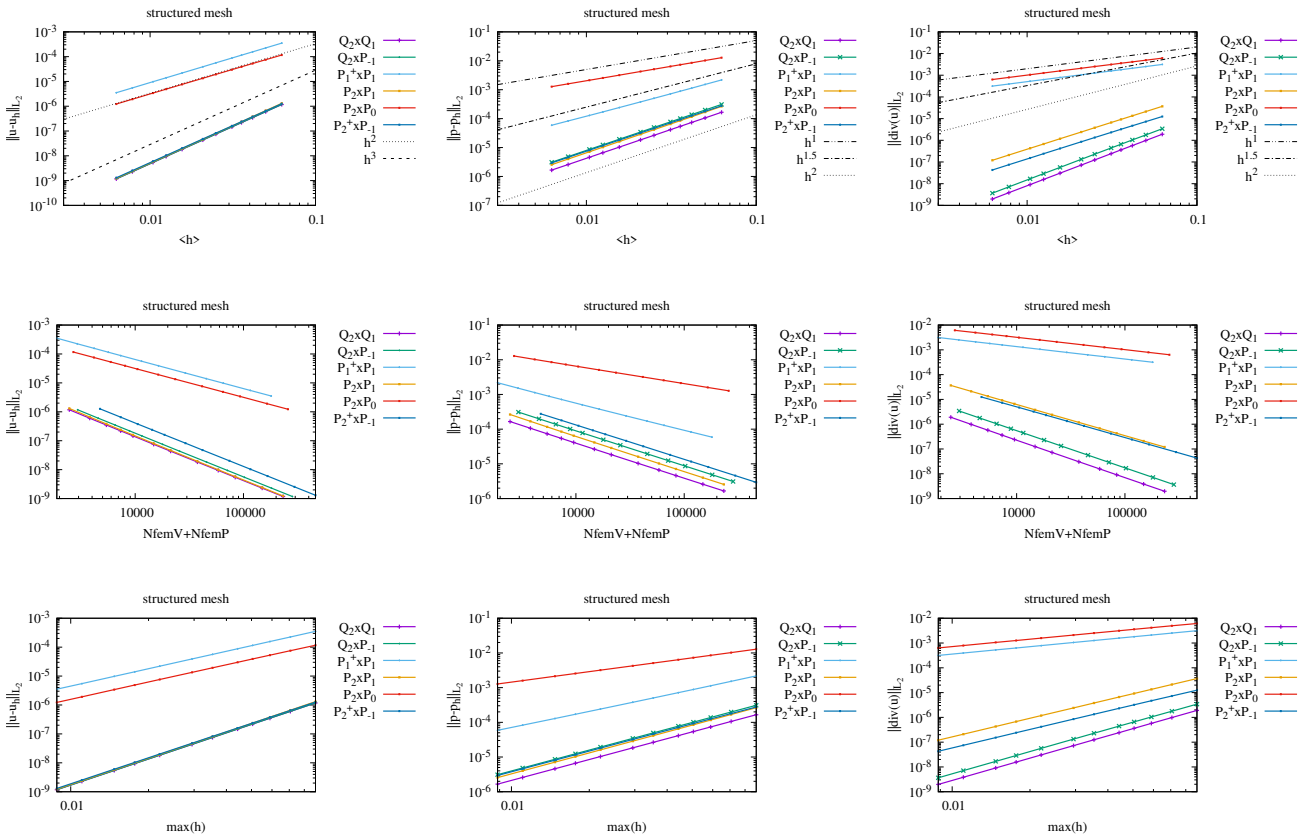
In the original paper [218] the authors consider the time-dependent Stokes equations:

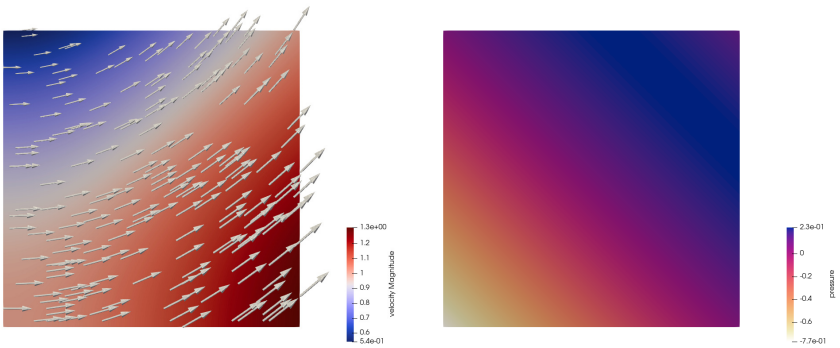
$$\vec{v}_t - \nu\Delta\vec{v} + \vec{\nabla}p = \vec{f}$$

$$\vec{\nabla}\cdot\vec{v} = 0$$

Since I do not wish to deal with the time derivative I set it to zero (steady state) and I also set $t = 0$ in their proposed analytical velocity field so that we get

$$\vec{v}(x,y) = \begin{pmatrix} \cos y \\ \sin x \end{pmatrix} \qquad p(x,y) = \sin(x+y) \qquad \begin{pmatrix} \cos(y) + \cos(x+y) \\ \sin(x) + \cos(x+y) \end{pmatrix}$$

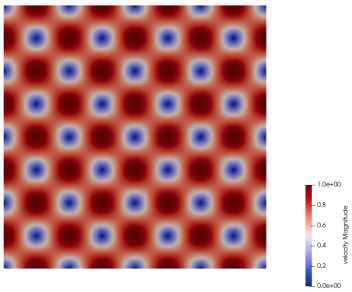


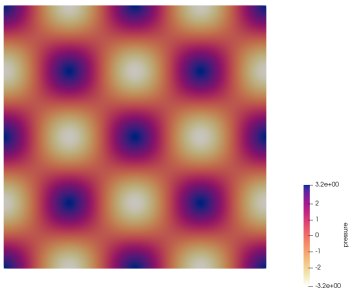


Results - Terrel et al. (2012) mms (nov. 2024)

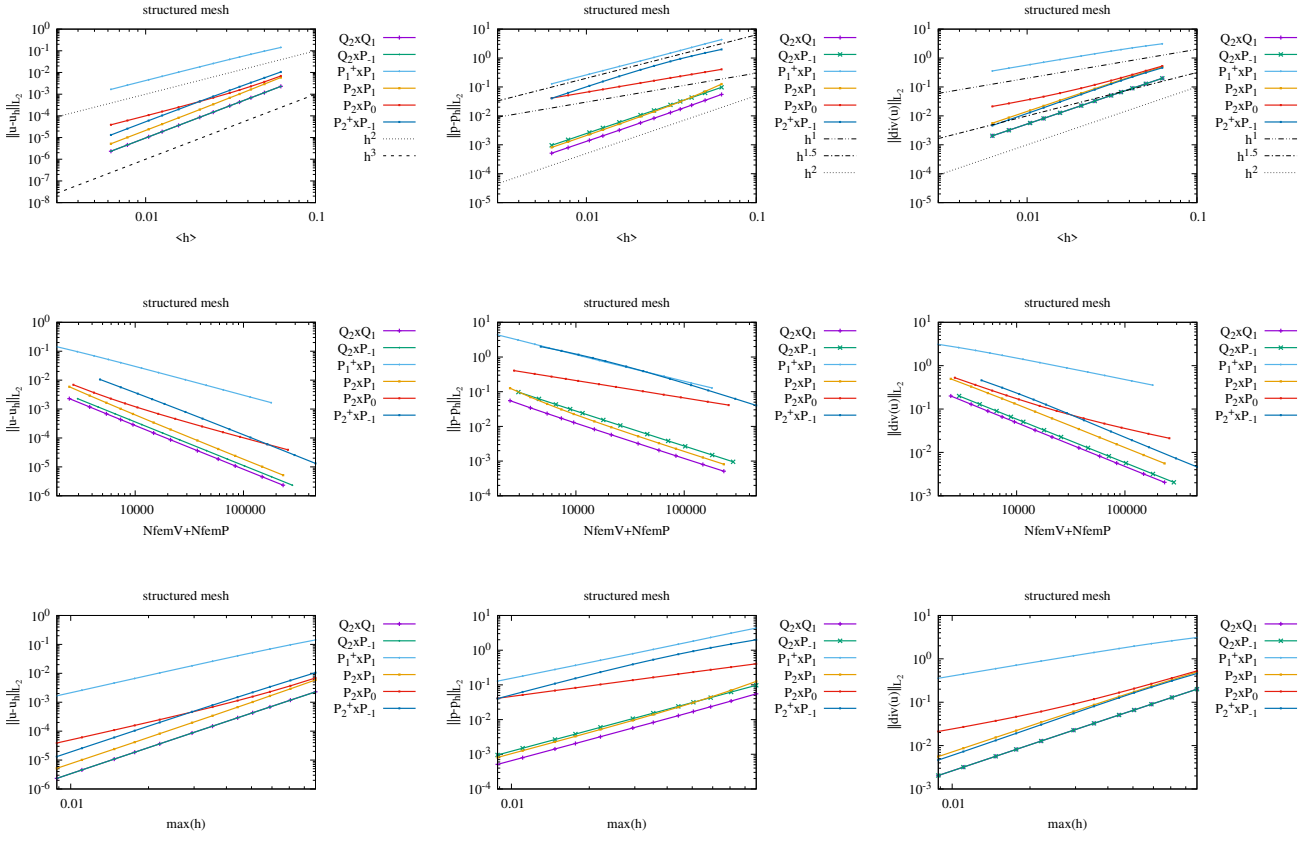
This manufactured solution originates in Terrel, Scott, Knepley, Kirby, and Wells [321] (2012). It is implemented in `mms_tesk12.py`

$$
\begin{aligned}
u(x,y) &= \sin(4\pi x)\cos(4\pi y) \\
v(x,y) &= -\cos(4\pi x)\sin(4\pi y) \\
p(x,y) &= \pi\cos(4\pi x)\cos(4\pi y) \\
b_x(x,y) &= 28\pi^2\sin(4\pi x)\cos(4\pi y) \\
b_y(x,y) &= -36\pi^2\cos(4\pi x)\sin(4\pi y)
\end{aligned} \tag{120.9}
$$

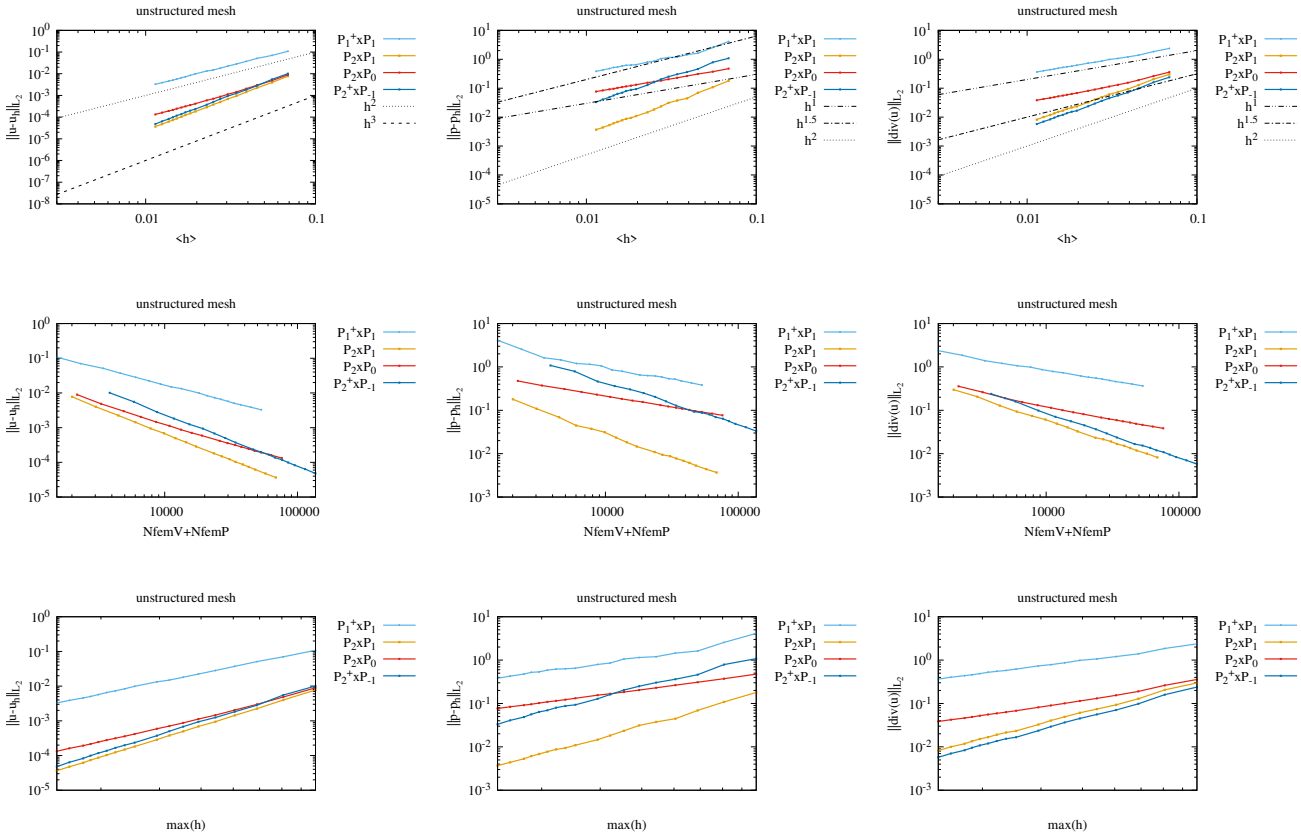




Structured mesh



Unstructured mesh



Summary of mms results

We have

$$\begin{aligned} e_{\boldsymbol{u}} &= ||u - u_h||_{L_2} \propto h^m \\ e_p &= ||p - p_h||_{L_2} \propto h^n \end{aligned}$$

In what follows I report the rates m and n of the velocity and pressure errors. All values in this color are the expected ones.

Structured mesh

I need to verify all of these!

element	D&H m,n	SolKz m,n	johnbook m,n	SolCx m,n	SolVi m,n	bocg12 m,n	jokn16 m,n	jolm17 m,n	tesk12 m,n	lire19 m,n
$Q_2 \times Q_1$	3,2	3,2	3,2	3, 0.5	1, 0.5	3,2	3,2	3,2	3,2	3,2
$Q_2 \times P_{-1}$	3,2	3,2	3,2	3,2	1, 0.5	3,2	3,2	3,2	3,2	3,2
$P_1^+ \times P_1$	2,1.5	2,1.5	2,1.5	2, 0.5	1, 0.5	2,1.5	2,1.5	2,1.5	2,1.5	2,1.5
$P_2 \times P_1$	3,2	3,2	3,2	3, 0.5	1, 0.5	3,2	3,2	3,2	3,2	3,2
$P_2 \times P_0$	2,1	2,1	2,1	2,1	1, 0.5	2,1	2,1	2,1	2,1	2,1
$P_2^+ \times P_{-1}$	3,2	3,2	3,2	3,2	1, 0.5	3,2	3,2	3,2	3,2	3,2

Unstructured mesh

element	D&H m,n	SolKz m,n	johnbook m,n	SolCx m,n	SolVi m,n	bocg12 m,n	jokn16 m,n	jolm17 m,n	tesk12 m,n	lire19 m,n
$P_1^+ \times P_1$	2,1	2,1	2,?	2,0.5	2,0.5	2,1	2,1			
$P_2 \times P_1$	3,2	3,2	3,2	3,0.5	3,0.5	3,2	3,2			
$P_2 \times P_0$	2,1	2,1	2,1	2,1	2,1	2,1	3,2 then 1?			
$P_2^+ \times P_{-1}$	3,2	3,2	3,2	3,2	3,2	3,2	3,2	?,?		

Conclusions:

- $Q_1 \times Q_0$ pressure is plagued by checkeboard mode (nothing new, won't be included in paper anyways)
- P1+P1 and Q1+Q1 exhibit the same convergence order: 2 for velocity and 1.5 for pressure.
- compare Q2Q1 and Q2P-1
- compare Q2Q1 and Q28Q1
- P2P0 vel convergence not cubic ?

TODO:
Write code which computes null space of G
randomize mesh
write function in FEtools with interpolates v and p on point ?
mms with nonzero dirichlet bc?
Korn inequality something ?
Remaining questions/ideas:
- no flow benchmark ? i.e bent downwards 2D domain.
- compute jcb, jci, jcob for 3x2 with all elements.
- should I implement edge stab - dG stuff for R-T elements ? if not they should be removed from study. Probably same with Han and Chen.

Remark:
- P1NC-P0, MINI, BR and P2P0 compared on mms in [64]

Mention Teseo Schneider, Yixin Hu, Xifeng Gao, Jérémie Dumas, Denis Zorin, and Daniele Panozzo. "A Large-Scale Comparison of Tetrahedral and Hexahedral Elements for Solving Elliptic PDEs with the Finite Element Method". In: *ACM Transactions on Graphics (TOG)* 41.3 (2022), pp. 1–14. DOI: 10.1145/3508372

Results - square sinker

Unit square. Free-slip boundary conditions. domain filled with fluid with $\eta_f = 1$ and $\rho_f = 1$. Square sinker in the middle of the domain of size 0.25×0.25 with $\eta_s = 100$ and $\rho_s = 1.001$. Gravity is $\vec{g} = -\vec{e}_y$.

There is no analytical solution so by setting $\vec{v}^{th} = \vec{0}$ and $p^{th} = 0$ the computed errors are in fact the vrms and prms shown hereunder.

conclusion:

- DSSY, RT and Han elements are not usable for buoyancy-driven flows in the presence of a hydrostatic background. Their velocity is all over the place, but their pressure seems to very slowly converge at high resolution to the pressure of the other elements. RETRIEVE email correspondance with Rannacher! show velocity field modes. Has smthg to do with Korn stuff.

Stone 121

Olivine deformation maps (Gueydan & Précigout, 2014)



python_codes/fieldstone_121/text.tex

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_121`

This stone was largely developed by Prof. Gueydan and later improved/debugged by Jan Veenhof.

This stone is based on Gueydan and Précigout [145] (2014). The strain rate is to be decomposed into its various contributions coming from the different deformation mechanisms, dislocation creep, diffusion creep, disGBS and low-temperature plasticity:

$$\dot{\varepsilon} = \dot{\varepsilon}_{dsl} + \dot{\varepsilon}_{dif} + \dot{\varepsilon}_{gbs} + \dot{\varepsilon}_{exp}$$

with

$$\dot{\varepsilon}_{dsl} = A_{dsl} \exp\left(-\frac{Q_{dsl}}{RT}\right) \tau^{n_{dsl}} \tag{121.1}$$

$$\dot{\varepsilon}_{dif} = A_{dif} \exp\left(-\frac{Q_{dif}}{RT}\right) \tau^{n_{dif}} d^{-m_{dif}} \tag{121.2}$$

$$\dot{\varepsilon}_{gbs} = A_{gbs} \exp\left(-\frac{Q_{gbs}}{RT}\right) \tau^{n_{gbs}} d^{-m_{gbs}} \tag{121.3}$$

$$\dot{\varepsilon}_{exp} = A_{exp} \exp\left[-\frac{Q_{exp}}{RT}\left(1-\frac{\tau}{\tau_p}\right)^{n_{exp}}\right] \tag{121.4}$$

where d is the grain size, m is the grain size exponent, τ_p is the Peierls stress defined for low-temperature plasticity.

Looking at Gueydan and Précigout [145] (2014), we have the following material parameters:

	A	Q	n	m	τ_p	ref.
	MPa^{-n}/s	$kJ\,mol^{-1}$			MPa?	
Mantle (olivine)						
Dislocation	$1.1 \cdot 10^5$	530	3.5	-	-	[155]
Diffusion	$3.98 \cdot 10^7$	370	1	3	-	
Dry GBS	$6.5 \cdot 10^3$	400	3.5	2	-	
Dry GBS	$6.31 \cdot 10^4$	445	2.9	0.7	-	[148]
Exponential	$5.7 \cdot 10^{11}$	535	2	-	8500	[135]
Crust (quartz)						
Dislocation (weak)	$3.2 \cdot 10^{-4}$	154	2.3	-	-	[195]
Dislocation (strong)	$6.31 \cdot 10^{-12}$	135	4	-	-	[156]

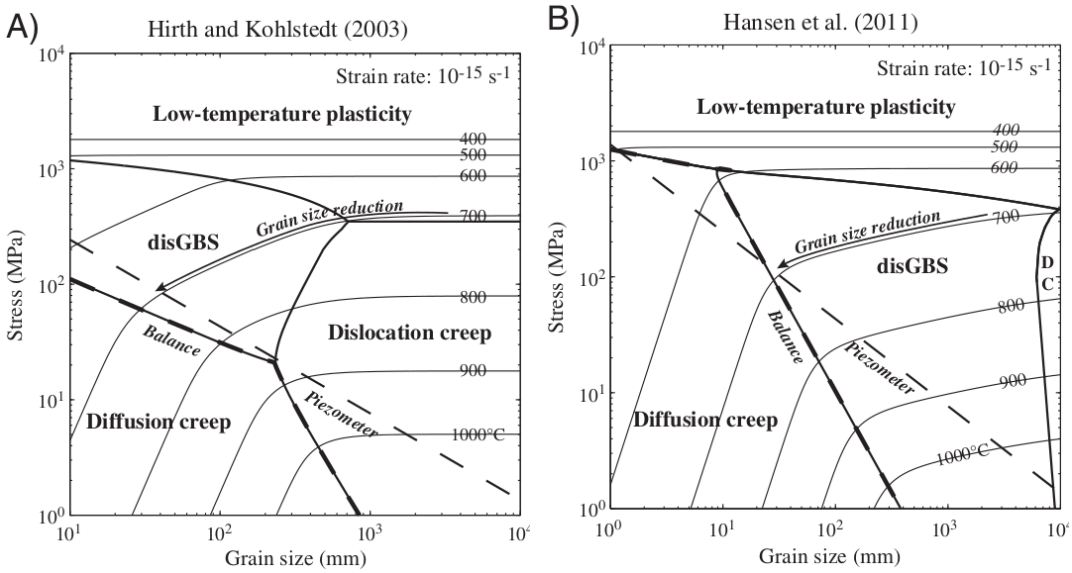
The composite olivine rheology thus involves several mechanisms that compete with each other; the mechanism with the highest strain rate/ lowest stress dominates the deforming aggregate depending on stress, grain size, temperature and overall strain rate [145].

As explained in Section 5.23, there is one major problem with the equations above: Assuming $\dot{\varepsilon}$ and temperature T known (as well as all the material parameters A, Q, n, ...), and that the deformation mechanisms are in series and subjected to the same deviatoric stress τ, we must find τ such that

$$\mathcal{F}(\tau) = \dot{\varepsilon} - \dot{\varepsilon}_{dsl}(\tau) - \dot{\varepsilon}_{dif}(\tau) - \dot{\varepsilon}_{gbs}(\tau) - \dot{\varepsilon}_{exp}(\tau) = 0$$

Unfortunately, this equation is non-linear in τ so that finding its zero(es) is not straightforward. A Newton-Raphson[1] algorithm is then used. How to build such an algorithm is presented in Section 5.23 but we will here use an existing python function. We load `scipy.optimize` module and use the `newton` function[2] which finds a zero of a real or complex function using the Newton-Raphson (or secant or Halley's) method. Once τ (`tau_NR`) has been found, it can then be inserted in the strain rate equations above and the strain rate partitioning is then complete.

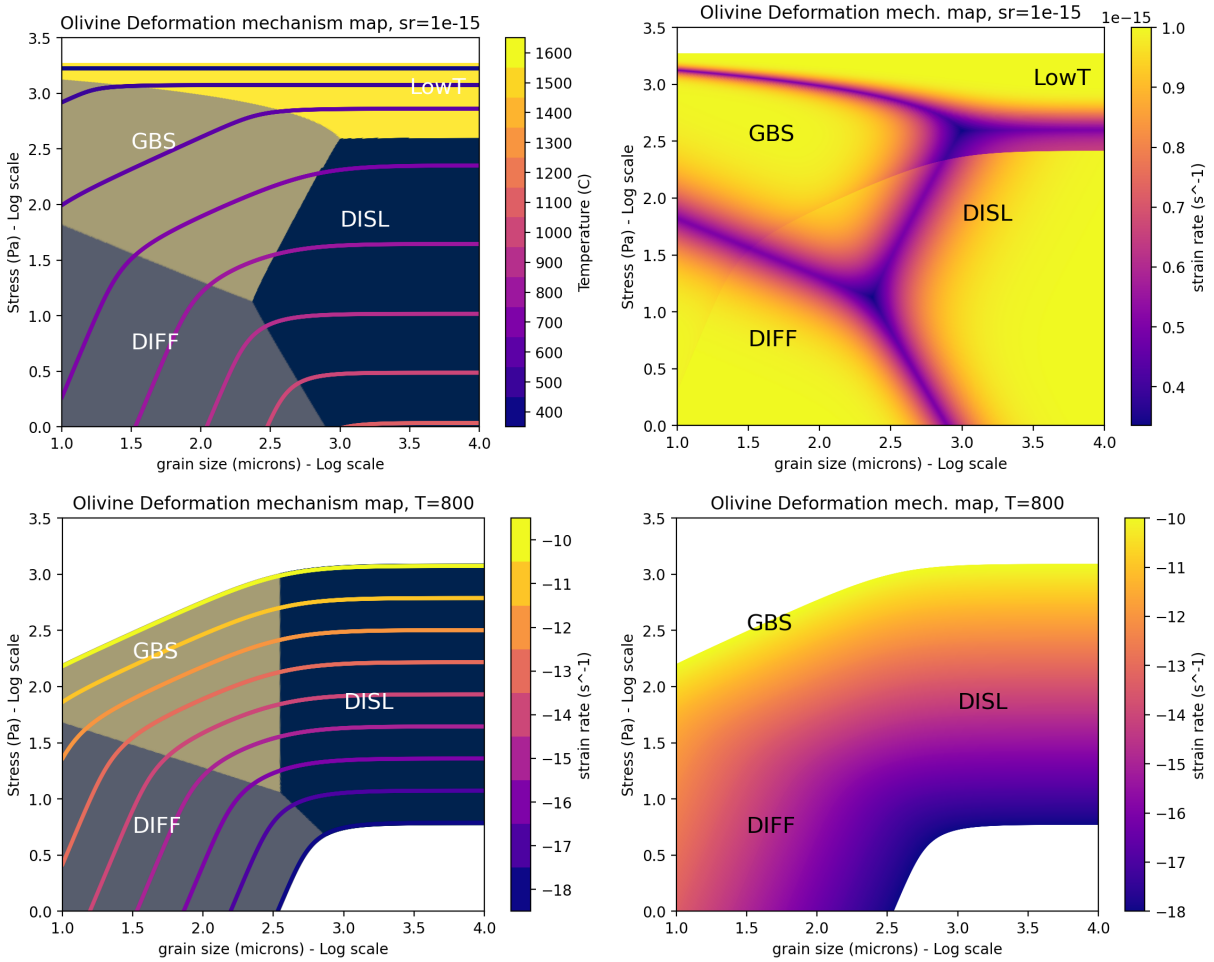
Quoting Gueydan and Précigout [145] (2014):"The conditions for which each mechanism dominates are displayed through a so-called deformation map, i.e., a stress-grain size log–log plot for a range of temperature at constant overall strain rate. Based on the iso-temperature curves, this graph highlights four stress-grain size fields, respectively for dislocation creep at large grain size and low stress, for exponential creep at high stress, for diffusion creep at low grain size and low stress, and for disGBS at intermediate conditions."



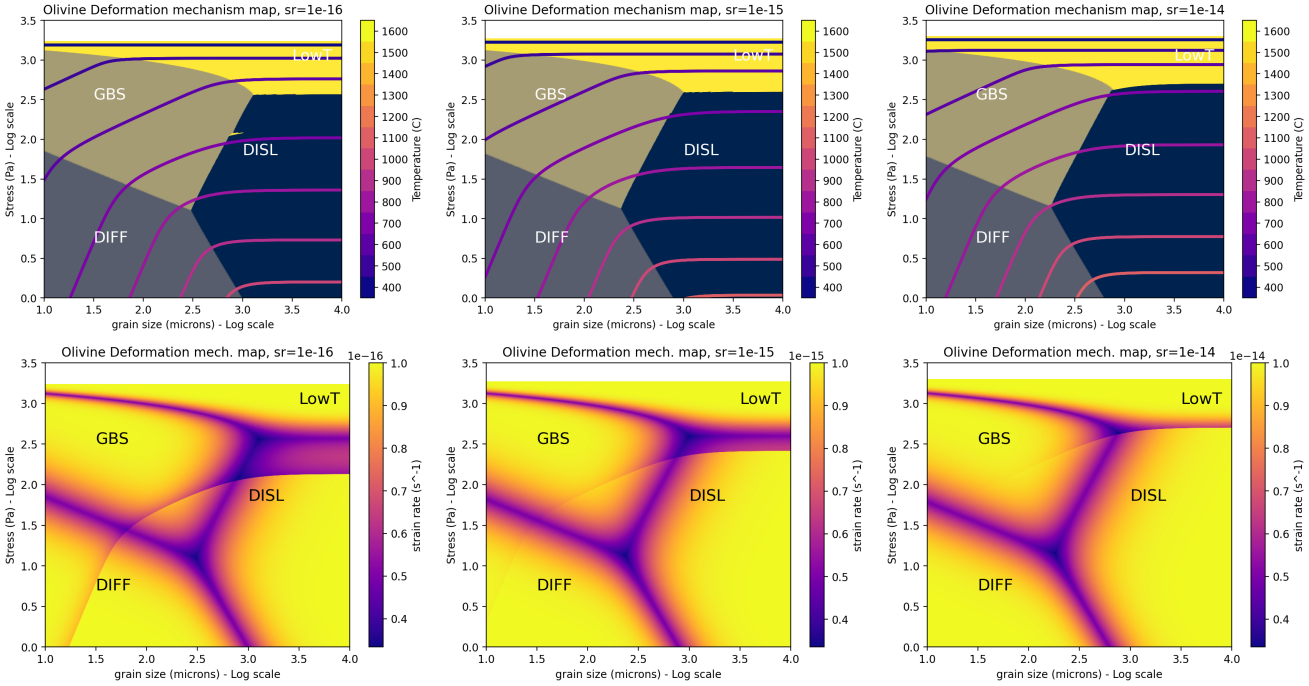
Taken from Gueydan and Précigout [145] (2014). Olivine deformation maps (shear stress vs. mean grain size) at a constant strain rate (10^{-15} s^{-1}) showing the four deformation mechanisms that compete to control the mantle rheology: low-temperature plasticity (exponential creep), dislocation creep, disGBS and diffusion creep. Iso-temperature curves are provided to show stress/grain sizes for temperatures ranging from 1000 to 400 °C. A) deformation map including

[1] `https://en.wikipedia.org/wiki/Newton's_method`
[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html`

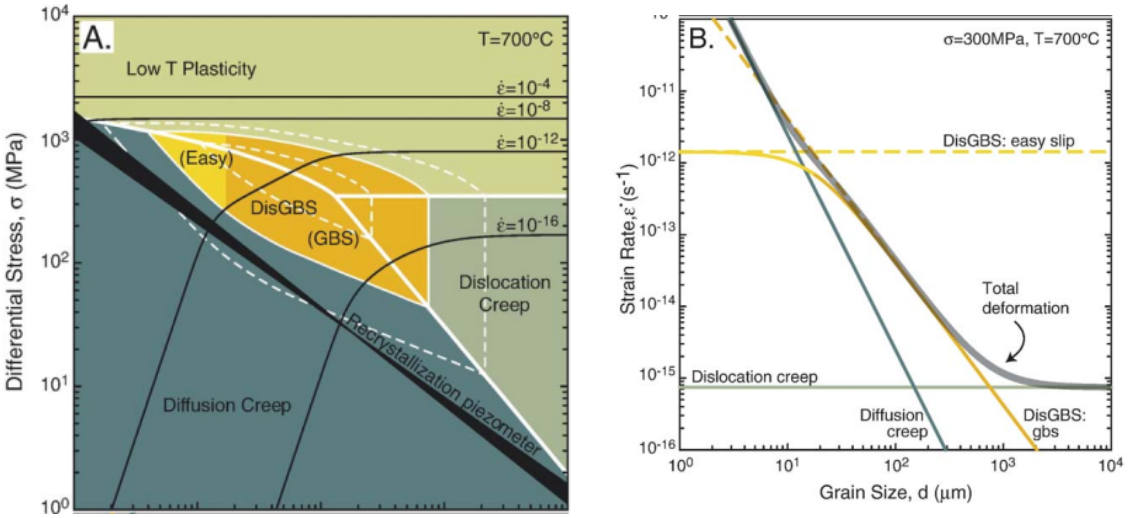
the disGBS flow law from Hirth and Kohlstedt [155] (2003); B) deformation map including the disGBS flow law from Hansen, Zimmerman, and Kohlstedt [148] (2011). Hypotheses for the recrystallized grain size are either equilibrium (at the boundary between grain size sensitive and grain size insensitive creeps, thick dashed line; de Bresser et al., 1998) or experimentally constrained paleo- piezometer (thin dashed line; Van der Wal et al., 1993). Values of the ductile flow laws are given in the Table above



Deformation map as generated by the code at constant $\dot{\varepsilon} = 10^{-15}$ s^{-1} (top row), and at constant temperature $T = 800^{\circ}$C (bottom row).



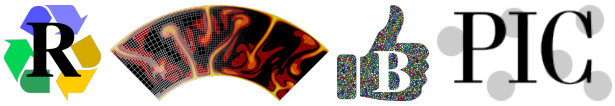
Deformation map as generated by the code at constant strain rates $\dot{\varepsilon} = 10^{-16,15,14}$ s^{-1}.



Taken from Warren and Hirth [349] (2006). (A) An olivine deformation mechanism map, on axes of differential stress versus grain size, contoured for strain rate, plotted using the flow laws for dry, melt free olivine at 700C and 12 km depth. The black recrystallization piezometer field delimits the grain size predicted by the empirical piezometric relationship for olivine deformed under dry conditions. The thick white boundaries between deformation fields indicate field boundaries in the absence of DisGBS. The thin white boundaries delineate the DisGBS field and the dashed boundaries on either side represent uncertainty in the location of the DisGBS boundaries due to uncertainty in its activation energy. The change in shading within the DisGBS field indicates the transition from the regime where creep on the easy slip system limits the strain rate to the regime where GBS limits the strain rate. (B) The variation of total and individual strain rates with grain size, at a differential stress of 300 MPa, and at 700C and 12 km depth. The dashed lines represent the individual components of DisGBS.

Stone 122

Runge-Kutta methods & particle advection

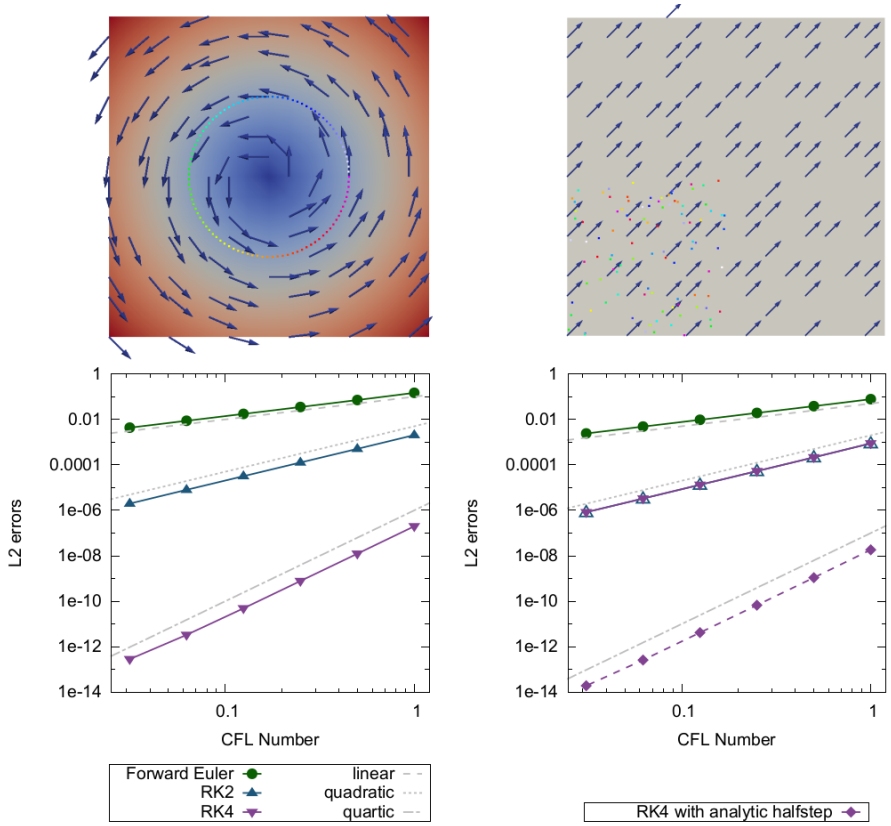


python_codes/fieldstone_122/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_122`

This STONE is a replication of the benchmarks found in the supplementary material of Gassmöller, Lokavarapu, Heien, Puckett, and Bangerth [124] (2018) in which we read:

> We tested the accuracy of the implemented advection schemes with two example models that separately show the convergence of the integration error in spatially and temporally variable flow. Both models use an analytically prescribed velocity function that is evaluated and transferred to the finite-element solution vector. By this procedure we ensure that the particle algorithm behaves as in a normal coupled computation, but retain full control over the velocity field that allows us to choose benchmarks with analytical solutions, and are independent of the accuracy of the finite element solver.



Taken from the supplementary material of Gassmöller, Lokavarapu, Heien, Puckett, and Bangerth [124] (2018). Left is benchmark #1, with a rotational time-independent velocity field, Right is benchmark #2 with a linear but time-dependent velocity field.

Benchmark 1

This benchmark is described as follows:

> The first benchmark model consists of a circular flow around the center of a two-dimensional box. The model end time and velocity of the flow is chosen to match one turn-around time around the center. The model contains 100 particles that are distributed in a ring with a constant radius around the model center. The error is measured as distance between final and initial position that should approach zero in an analytical solution. To ensure that the error is independent of the particle's starting position we plot the maximal error of all particles.

The code is quite simple. First a $nelx \times nely$ mesh of Q_2 elements is created with its connectivity array `icon`. Then 100 particles are placed in an equidistant manner on a circle or radius 1, placed in the middle of the 2×2 domain. A velocity field $\vec{\mathbf{v}} = (y-2, -x+2)$ is used to advect these particles (in a clockwise fashion). Various Runge-Kutta methods are implemented, i.e. RK1, RK2, RK3, RK4, RK4(3/8), RKF and ODE87 (see Section 14.10).

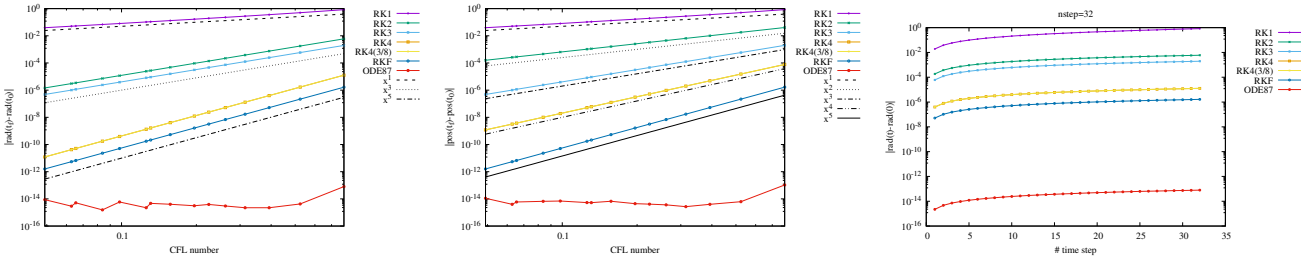
When a number of timesteps is chosen, then since the maximum velocity is 1 on the particle circle, we have:

```
vel_max=1
dt=(2*np.pi/vel_max)/nstep
CFL=dt*vel_max/hx
tfinal=nstep*dt
```

Note that when resolution is increased, then the same chosen number of time steps yields higher CFL numbers.

When the option `interpolate` is False, the velocity is computed on each particle and at any sub-timestep using the analytical velocity and here are the results[1]:

[1] These results are obtained on a 16x16 mesh but that is not so relevant since it is not used anywhere, except than for computing dt. Also, I only consider 1 particle out of 100 for simplicity.



Left: radius error (i.e. distance to center error) as a function of the CFL number and for various RK methods for one particle; Middle: distance between starting point position and ending point position after a 2π rotation as a function of CFL number; Right: radius error as a function of time for a given CFL number of 0.5.

I find that even order RK methods over-perform for radius (distance to center of the circle) error, not sure I can explain this[2]. When considering the distance between particles after 2π rotation, $r(t_{final}) - r(t_0)$, I recover the expected rates and match the results found in Gassmöller, Lokavarapu, Heien, Puckett, and Bangerth [124]. Also, I find virtually no difference between RK4 and RK4(3/8) results.

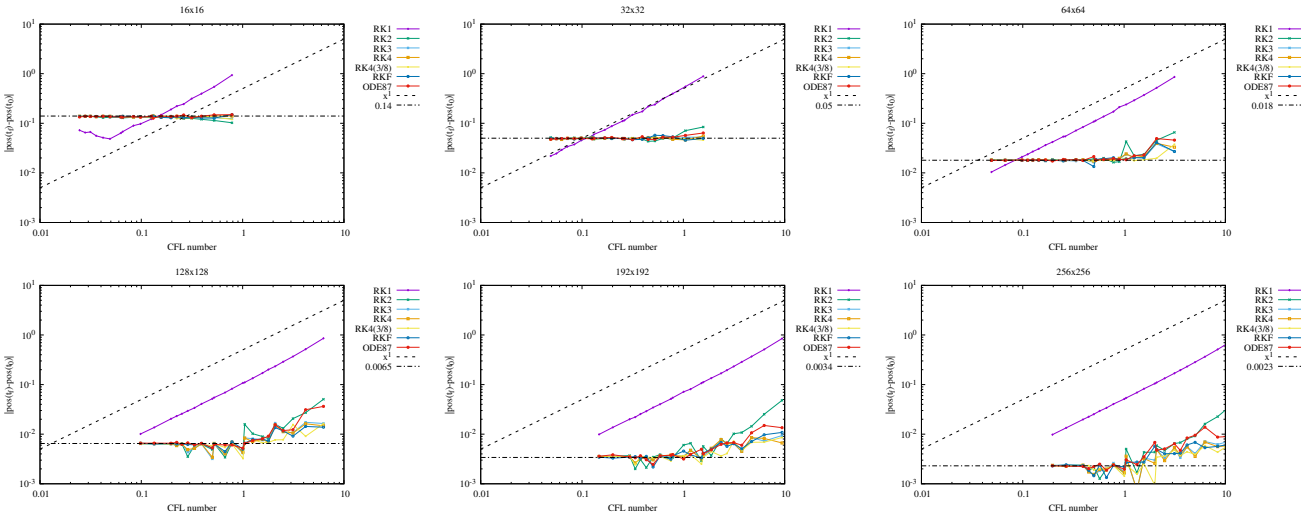
The performance of ODE87 is remarkable (error at machine precision), but at a conservative CFL number of 0.5, RK4 is near 10^{-8}. Given that a typical lengthscale for mantle modelling is $\sim 6000km$, then this is already an error of $\sim 10^{-5}km$, or $\sim 10^{-2}$m. In essence such a level of accuracy is not really needed in practice.

If I now first compute the velocity on the mesh and use the Q_2 basis functions to interpolate the velocity on the particles (interpolate=True), the code takes longer to run, as justified by this code excerpt for RK1:

```
for istep in range(0,nstep):
    for im in range(nmarker):
        if interpolate:
            ielx=int(swarm_x[im]/hx)
            iely=int(swarm_y[im]/hy)
            iel=iely*nelx+ielx
            rm=(swarm_x[im]-x[iconV[0,iel]])/hx-0.5
            sm=(swarm_y[im]-y[iconV[0,iel]])/hy-0.5
            NNNV=NNV(rm,sm)
            um=NNNV.dot(u[iconV[:,iel]])
            vm=NNNV.dot(v[iconV[:,iel]])
        else:
            um,vm=velocity(swarm_x[im],swarm_y[im])
        swarm_x[im]+=um*dt
        swarm_y[im]+=vm*dt
```

[2] René says 'Not sure either why the radius error seems to show higher than expected convergence, maybe the x and y components of the error somehow compensate each other?'

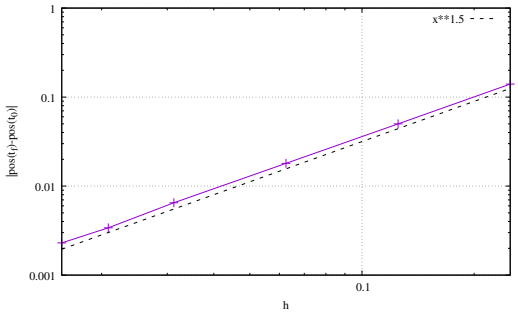
Lookin at the results for various mesh resolutions, we get a totally different picture:



From top to bottom: $nel = 16^2, 32^2, 64^2, 128^2, 192^2, 256^2$.

We find that the error is then 100% controlled by the mesh size, and high order methods do make any difference *at all*. I am somewhat surprised, but it makes sense since the velocity on the particle is now only second-order (Q_2 basis functions are used) so that only a high(er) resolution can help. The conclusion is that any method higher order than RK2 is probably pointless.

I have also measured how the error (measured at the smallest CFL and found to be independent of the RK method) varies as a function of the resolution:



I find that its convergence is not quite quadratic... also not sure what to think of this.

Benchmark 2

This benchmark is described as follows:

> The second benchmark model measures the convergence in time-variable flow, which is simulated by a model domain with spatially constant but over time exponentially- increasing velocity. All particles are generated at random positions in the lower-left quadrant of the model domain and are advected in a flow of the form
>
> $$u(t) = 1 + e^t \qquad v(t) = 1 + e^t.$$
>
> The model end time of 1 then leads to an expected final position $\vec{x}(1) = \vec{x}(0) + \int_0^1 \vec{\mathbf{v}}(t)\ dt$ with a distance between the initial and end position of $d = ||\vec{x}_1 - \vec{x}_0||_2 = 2e$.

Stone 123

Boussinesq & Love elastic problems

python_codes/fieldstone_123/text.tex

3D
box
Q1
sparse_matrix
compressible
analytical_benchmark
error_convergence
elastic_rheology
vrms
paraview_output
displacement_code

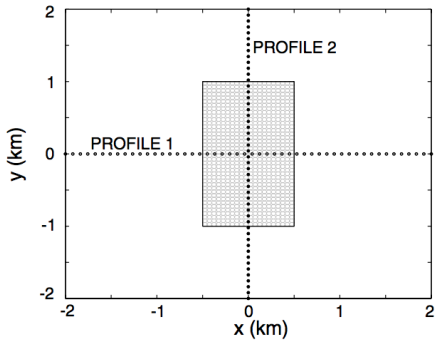
Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_123`

This fieldstone was partially co-developed in collaboration with Lukas van de Wiel.

Love's problem (experiment=1)

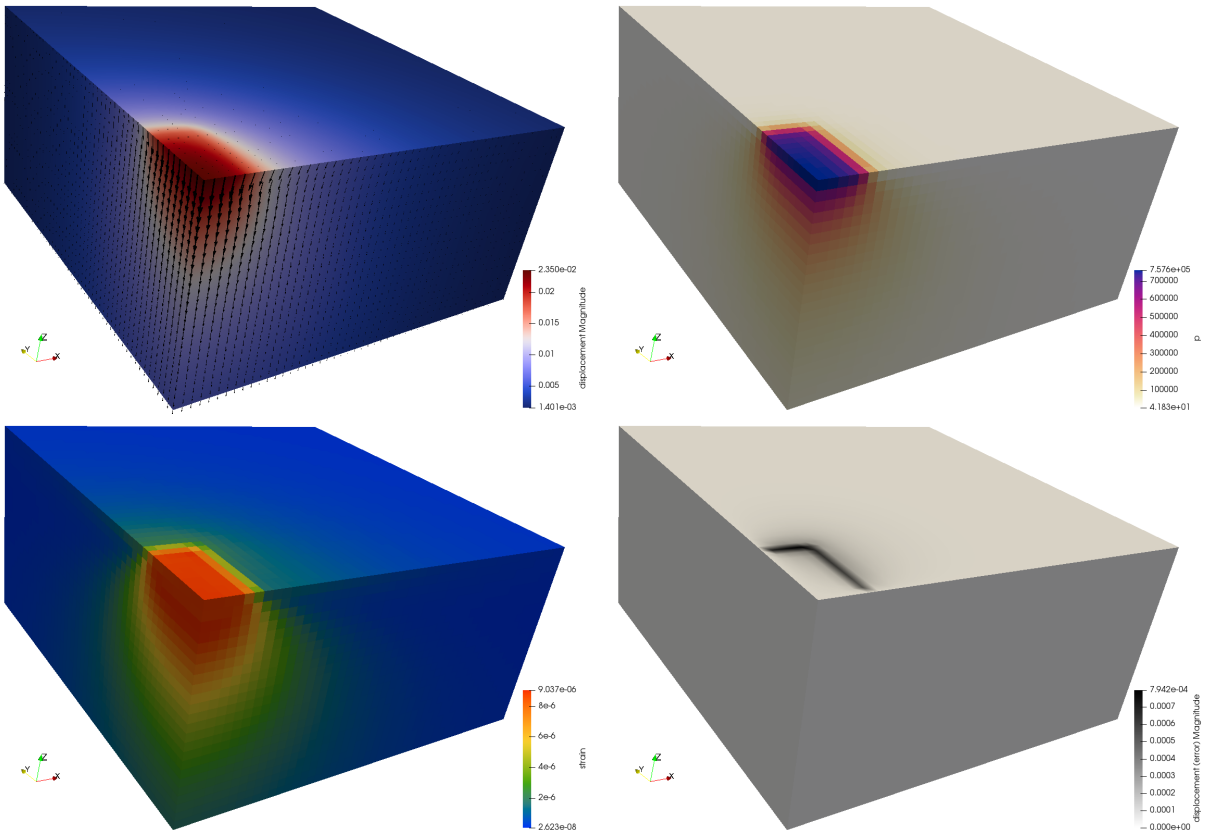
This problem was first published by Love [221] (1929) but we here rely on Becker and Bevis [14] (2004). The domain is a cuboid of size $L_x \times L_y \times L_z$ with $L_x = L_y = 5\,\text{km}$ and $L_z = 2.5\,\text{km}$. It is filled with a single elastic material characterised by $E = 0.6 \times 10^{11}\,\text{Pa}$ and $\nu = 0.25$. Gravity is set to zero. This corresponds to $\lambda = 24\,\text{GPa}$ and $\mu = 24\,\text{GPa}$.

We consider the surface displacements generated by an applied pressure equivalent to 100 m water depth in a rectangular area $2a \times 2b = 2 \times 1$ km.



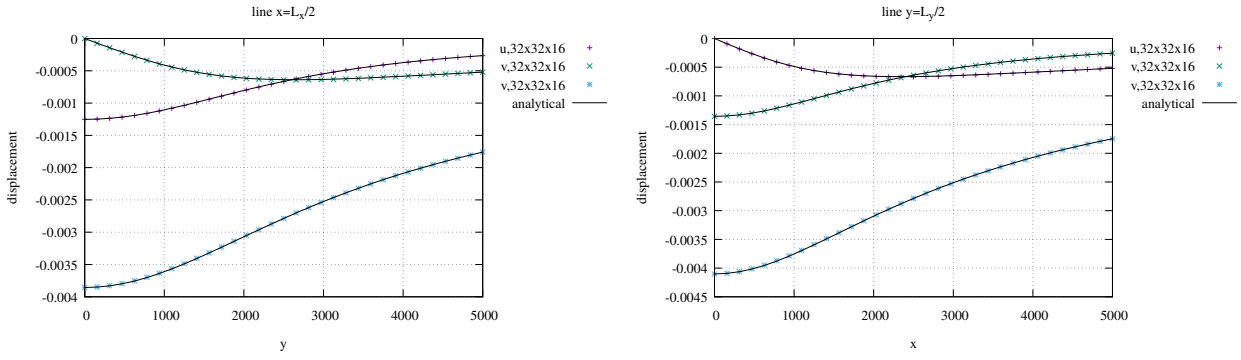
Top view of the problem. The gray area represents where pressure is applied. Note that we use a domain of different size.

Because of the inherent symmetry of the problem we model only one quarter ('the top right') of the problem. Boundary conditions are as follows: The analytical solution, which is valid for a semi-infinite half-space, is prescribed on all sides and bottom while traction boundary conditions are prescribed on a rectangle of size $a \times b$ (the rest of the surface is left free). This traction b.c. finds its way in the rhs. It concerns the z-component of nodes 4,5,6,7, i.e. dofs 14,17,20,23. Note that it is not applied on the sides where the displacement is prescribed. The analytical solution of this problem is is the love.py file. Q_1 elements are used.



Displacement field, pressure, effective strain rate and displacement error on a $50 \times 50 \times 25$ mesh.

As in Becker and Bevis [14] the displacement components are plotted on surface profiles and are found to agree very well with the analytical solution:



Three components of the displacement vector $\vec{v}$ on $x = L_x/2$ and $y = L_y/2$ lines.

Boussinesq problem (experiment=2)

We now focus on the classical Boussinesq problem (also sometimes called Boussinesq–Cerruti solution[1]). See Selvadurai [290] (2001) or Nwoji, Onah, Mama, and Ike [243] (2017) for a procedure for obtaining the

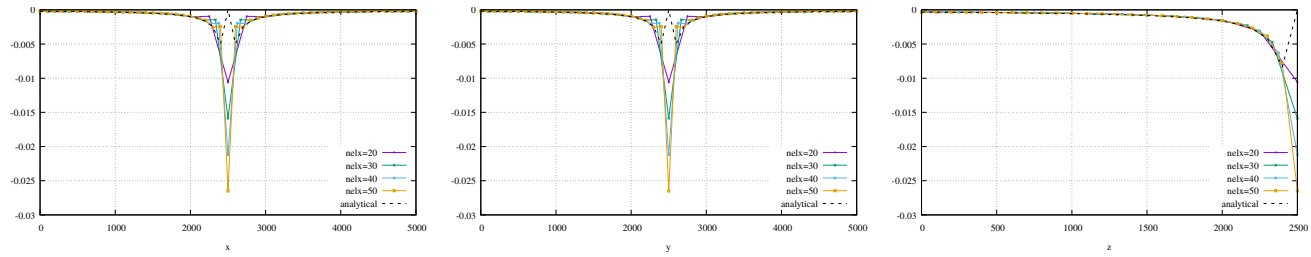
[1] `https://en.wikipedia.org/wiki/Linear_elasticity`

solution to this problem.

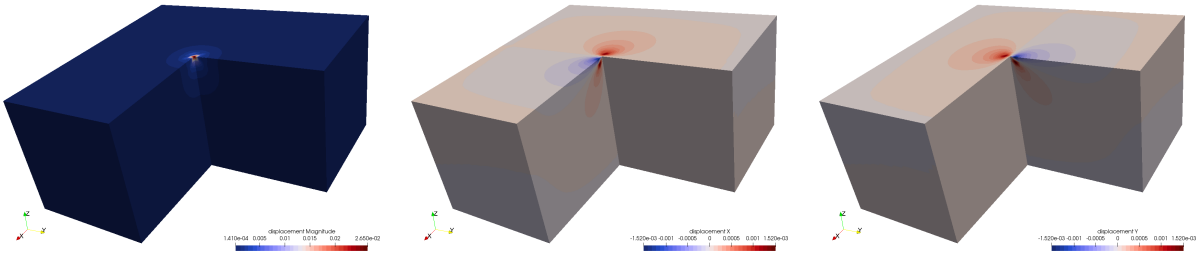
It consists of a point force at the origin of an infinite isotropic half-space. In the literature the z-axis is taken pointing downwards so that the force load is then $P\delta(x)\delta(y)\vec{e}_z$. The analytical solution in Cartesian coordinates is as follows:

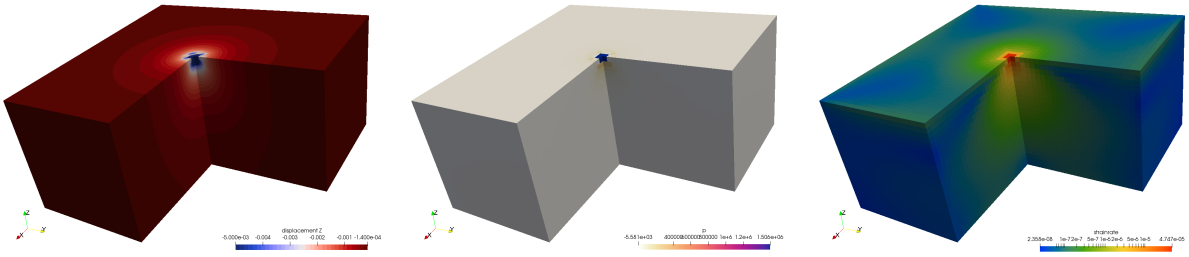
$$
\begin{aligned}
R &= \sqrt{x^2+y^2+z^2} \\
\upsilon_x &= \frac{Px}{4\pi\mu R}\left(\frac{z}{R^2}-\frac{1-2\nu}{R+z}\right) \\
\upsilon_y &= \frac{Py}{4\pi\mu R}\left(\frac{z}{R^2}-\frac{1-2\nu}{R+z}\right) \\
\upsilon_z &= \frac{P}{4\pi\mu R}\left(2(1-\nu)+\frac{z^2}{R^2}\right) \\
\sigma_{xx} &= -\frac{P}{2\pi R^2}\left[\frac{3x^2z}{R^3}-(1-2\nu)\left(\frac{z}{R}-\frac{R}{R+z}+\frac{x^2(2R+z)}{R(R+z)^2}\right)\right] \\
\sigma_{yy} &= -\frac{P}{2\pi R^2}\left[\frac{3y^2z}{R^3}-(1-2\nu)\left(\frac{z}{R}-\frac{R}{R+z}+\frac{y^2(2R+z)}{R(R+z)^2}\right)\right] \\
\sigma_{zz} &= -\frac{3Pz^3}{2\pi R^5} \\
\sigma_{xy} &= -\frac{P}{2\pi R^2}\left(\frac{3xyz}{R^3}-(1-2\nu)\frac{xy(2R+z)}{R(R+z)^2}\right) \\
\sigma_{xz} &= -\frac{3Pxz^2}{2\pi R^5} \\
\sigma_{yz} &= -\frac{3Pyz^2}{2\pi R^5}
\end{aligned}
$$

The domain and material parameters are identical to those of the previous benchmark. The load is set to 100 GPa and set at $x = L_x/2, y = L_y/2$. The solution is unfortunately singular where the load is applied, and this translates numerically in displacements that become larger and larger just below it. However, a few elements away from the singularity the analytical solution is recovered (Note that the analytical solution is set to zero at the point load - see boussinesq.py):



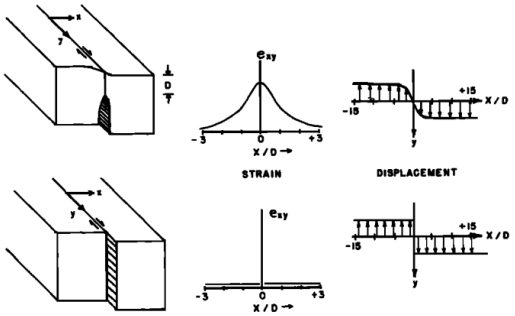
Vertical component of the displacement as measured on horizontal and vertical lines passing through the point load.





Various fields as obtained on a $50 \times 50 \times 25$ mesh which seems to be about the maximum resolution. It takes over 10min to run and uses about 16Gb of RAM.

Fault motion - Savage and Burford [280] (1973) (experiment=3)

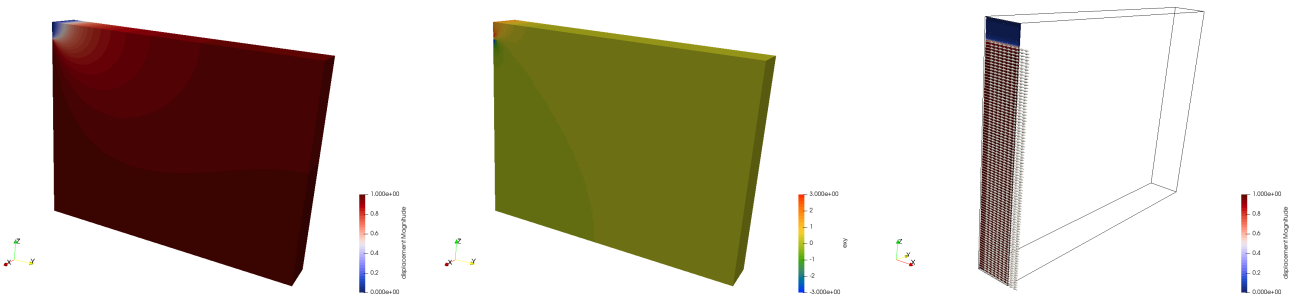


Simple models for fault motion showing the distribution of shear strain and strike slip displacement on the surface. The upper sketches refer to a locked-fault model, in which slip on the fault occurs only at depths greater than D. The lower sketches refer to the rigid-block model, in which slip is approximately uniform with depth. Taken from Savage and Burford [280].

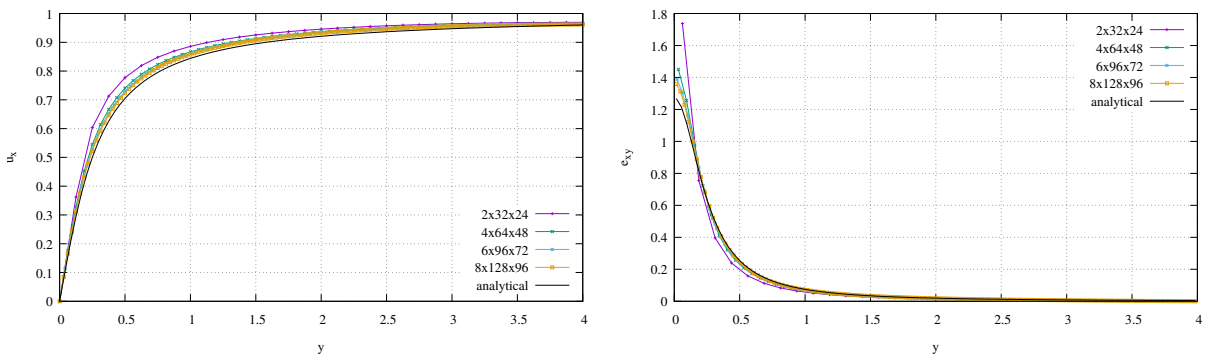
Both υ_y and υ_z components are entirely prescribed to be zero in the domain. Only υ_x is being solved for. It is prescribed to be zero over a length $D = 0.25$ on the left and $b = 1$ underneath. The bottom boundary can be left free or set to $b = 1$ as well. In what follows the domain is $0.5 \times 4 \times 3$. Savage and Burford [280] show that the displacement υ_x and strain component ε_{xy} at the surface $z = L_z$ are given by:

$$\upsilon_x = \frac{u_0}{\pi} \arctan\left(\frac{y}{D}\right) \qquad \varepsilon_{xy} = \frac{bD}{2\pi} \frac{1}{y^2 + D^2}$$

Material properties are $\nu = 0.25$, $\mu = 2\,\text{Pa}$, $\lambda = 1.25\,\text{Pa}$, $E = 3.2\,\text{Pa}$.



Obtained on 8x128x96 mesh. Displacement prescribed at the bottom of the domain.



Note that in their publication b is the fault displacement and what I prescribe is only half of it so factor 2 is needed to recover the theoretical curves.

Stone 124

Plate with a hole elastic problems

python_codes/fieldstone_124/text.tex

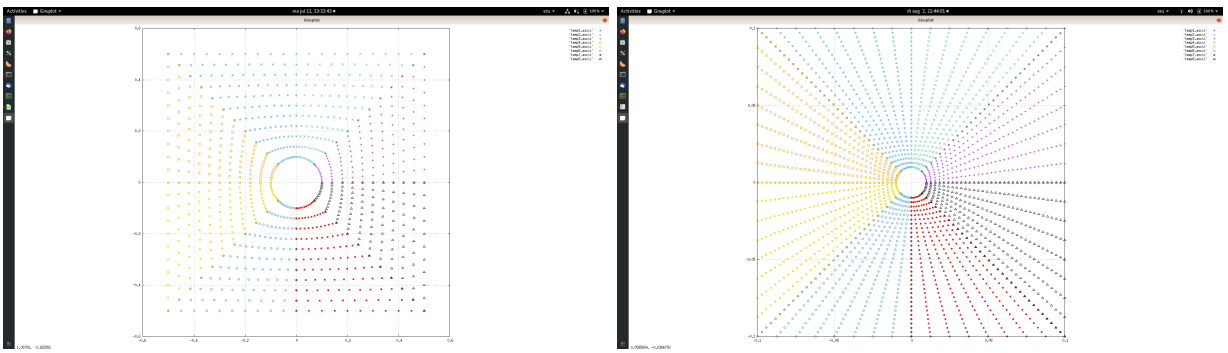
2D
box
Q1
sparse_matrix
compressible
analytical_benchmark
elastic_rheology
paraview_output
displacement_code

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_124`

In what follows we rely on 2D plane strain calculations and the discrete displacement solution lives in the Q_1 finite element space.

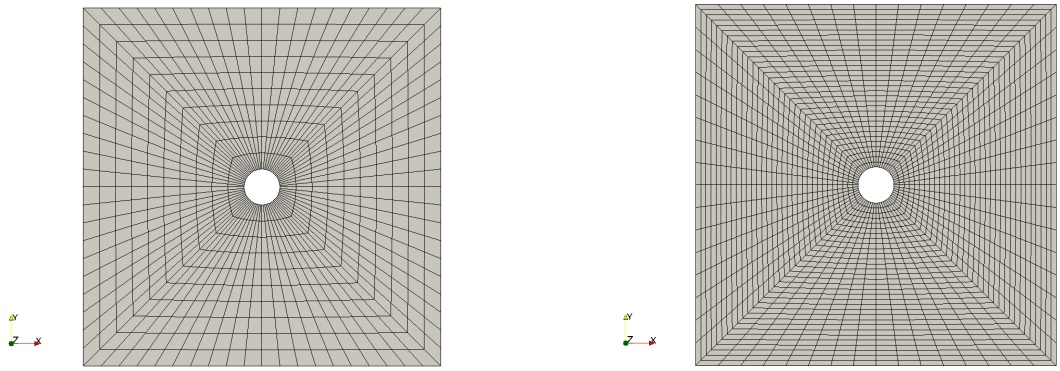
Mesh generation

The mesh is composed of 8 blocks, as shown here:



Left: nelx=nely=10; Right: nelx=32, nely=8.

Blocks 1 and 2 (purple and green respectively) are made from a regular $nelx \times nely$ mesh that is mapped to conform to the desired shape. Blocks 3 to 8 are rotations of either block 1 or 2. When all 8 blocks are obtained nodes which appear twice are removed and a global connectivity array is made, yielding the following mesh:

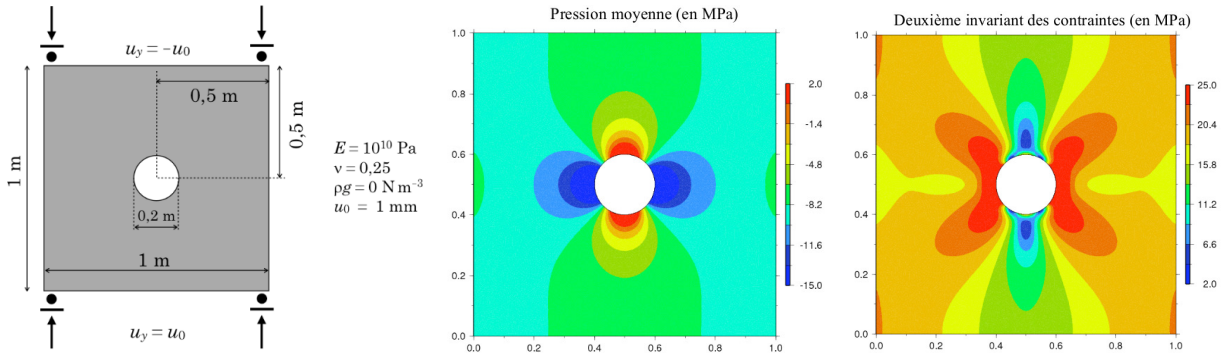


Left: nelx=nely=10; Right: nelx=32, nely=8.

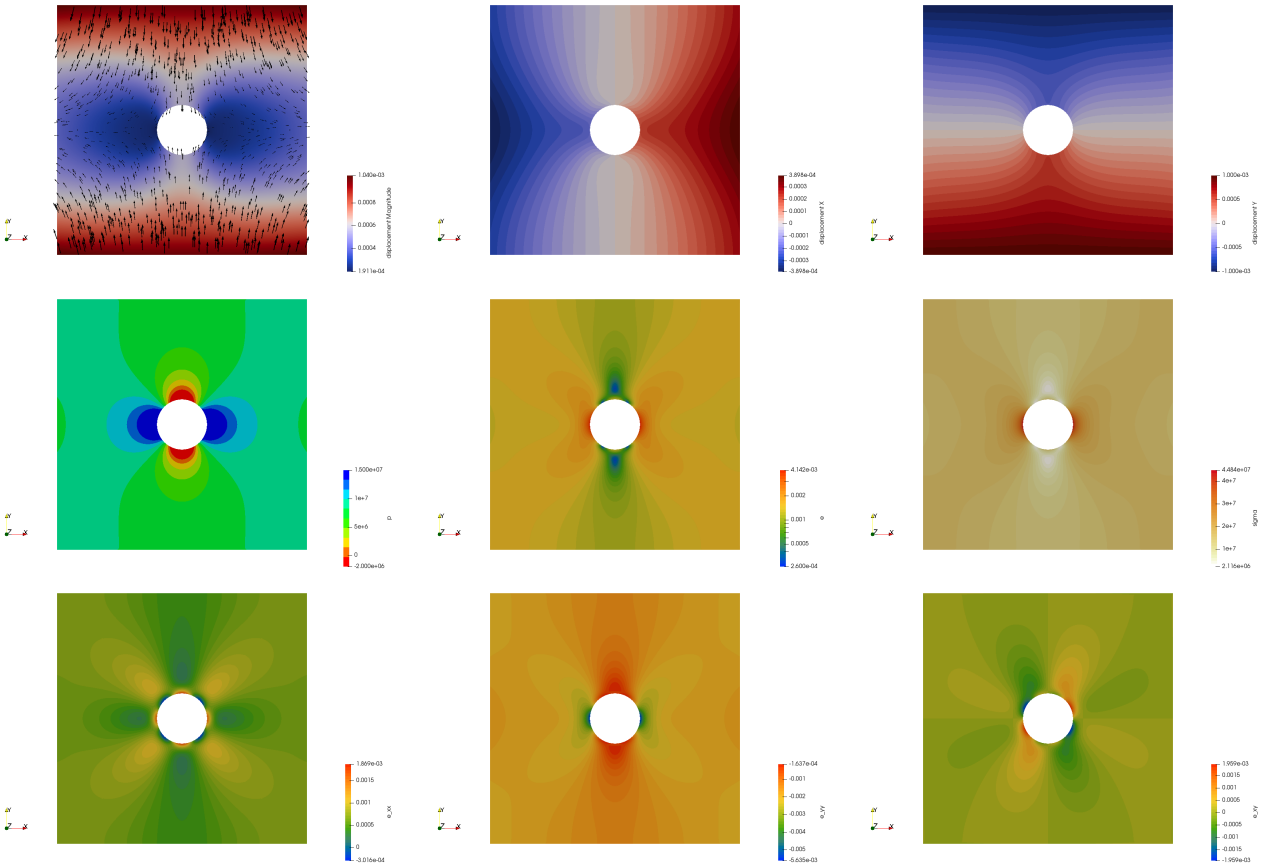
This is unfortunately a rather slow algorithm and it will take several minutes to generate a mesh for nelx=nely values above 60.

Hassani experiment (experiment=1)

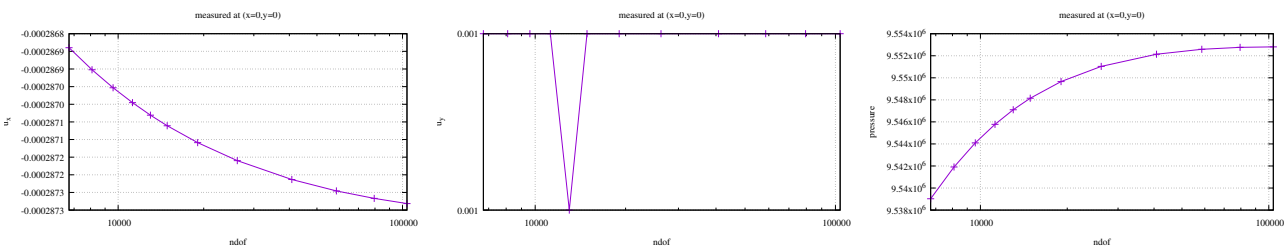
This is taken from Prof. Hassani's syllabus 'Mise en oeuvre de la méthode des éléments finis'. The domain is a plate of 1×1 m with a 0.2 m diameter hole in its center. Vertical displacements $\pm\upsilon_0$ are prescribed on the top and bottom surface while the horizontal displacements are left free. No boundary condition is prescribed on the hole (i.e. $\boldsymbol{\sigma} \cdot \vec{n} = \vec{0}$). In order to remove the translational nullspace, we impose $\upsilon_x = 0$ at $(x = 0.5, y = 0)$. Material properties are indicated on the following figure.

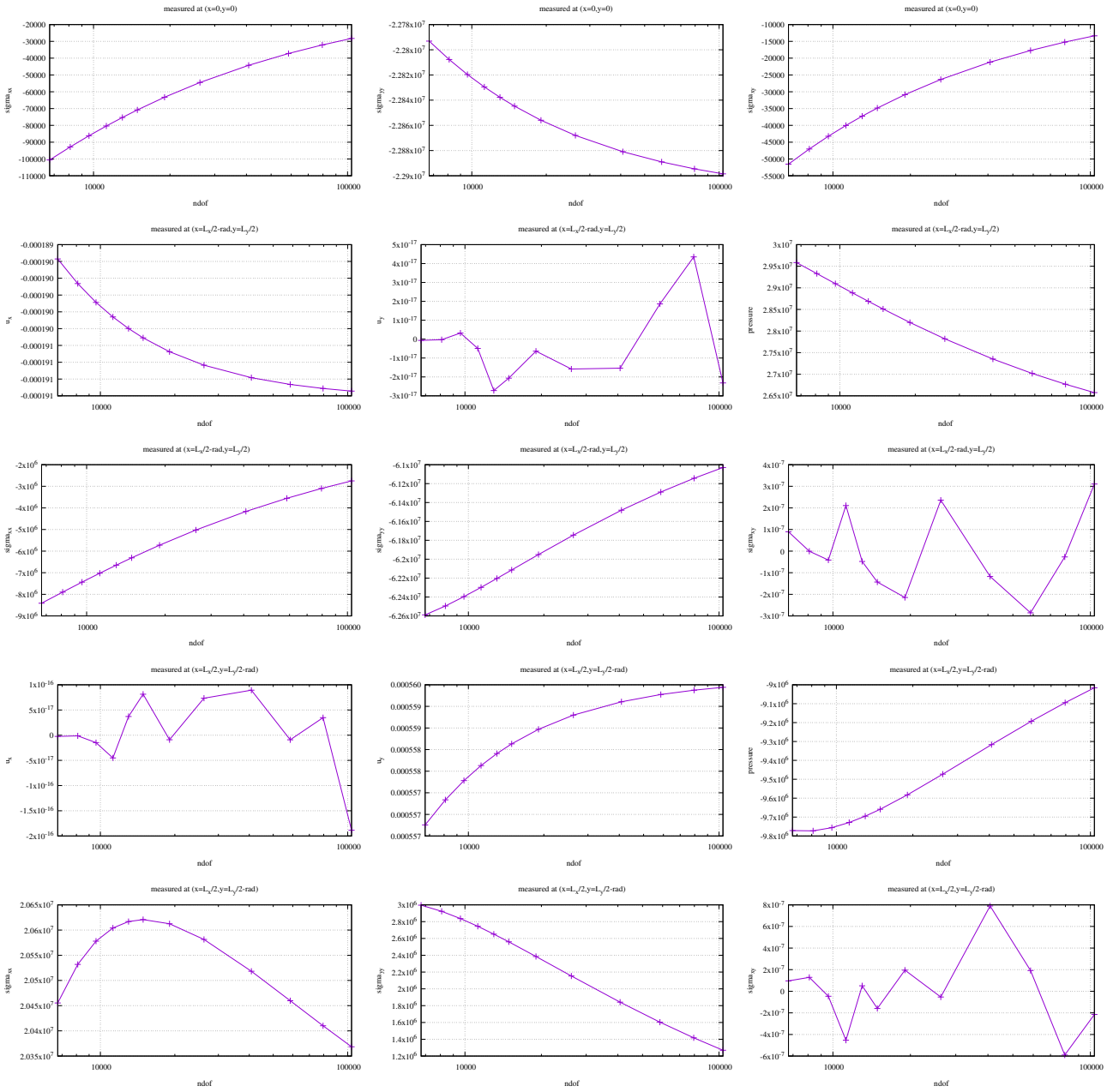


Taken from Prof. Hassani's syllabus. Note that pressure in this plot is the opposite of our pressure (different convention). Also, it is not exactly clear which exact invariant was used in the right plot (private comm.).



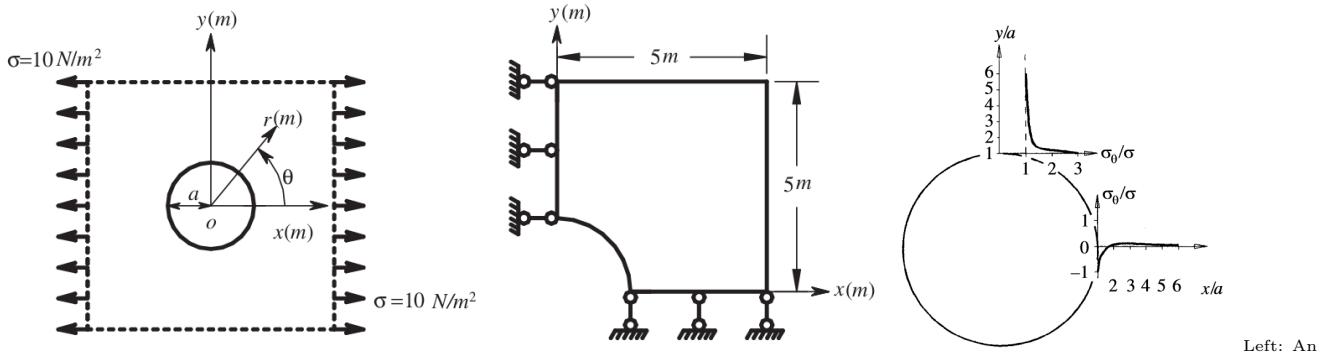
Each block counts 64×64 elements. Exceptionally a rainbow color scale is used for the pressure so as to allow for a comparison with the figure from Hassani.




measured at (x=0,y=0)
sigma_xx
ndof
measured at (x=0,y=0)
sigma_xy
ndof
measured at (x=0,y=0)
sigma_yy
ndof
measured at (x=L_x/2-rad,y=L_y/2)
u_x
ndof
measured at (x=L_x/2-rad,y=L_y/2)
u_y
ndof
measured at (x=L_x/2-rad,y=L_y/2)
pressure
ndof
measured at (x=L_x/2-rad,y=L_y/2)
sigma_xx
ndof
measured at (x=L_x/2-rad,y=L_y/2)
sigma_yy
ndof
measured at (x=L_x/2-rad,y=L_y/2)
sigma_xy
ndof
measured at (x=L_x/2,y=L_y/2-rad)
u_x
ndof
measured at (x=L_x/2,y=L_y/2-rad)
u_y
ndof
measured at (x=L_x/2,y=L_y/2-rad)
pressure
ndof
measured at (x=L_x/2,y=L_y/2-rad)
sigma_xx
ndof
measured at (x=L_x/2,y=L_y/2-rad)
sigma_yy
ndof
measured at (x=L_x/2,y=L_y/2-rad)
sigma_xy
ndof

Ramachandran [256] (experiment=2)

An infinite plate with a circular hole of radius a is subjected to a unidirectional tensile load of σ in the x direction as shown in the figure. In this case, only one quarter of the domain could be analysed due to symmetry along x and y axis (see figure below) but we nevertheless model the full plate.



Left: An infinite plate with a circular hole subjected to unidirectional tension and its quarter model with symmetric conditions imposed on the left and bottom edges. Right: Tangential stress distribution for $\theta = 0$ and $\theta = \pi/2$. [358]

The inner boundary of the hole is traction free. On the left and right edges tractions based on the analytical solutions are prescribed. The plane stress condition is considered and the parameters are: Young modulus $E = 3 \times 10^7$ MPa, Poisson Ratio $\nu = 0.3$, Load $\sigma = 10\ \mathrm{N\,m^{-2}}$, $a = 1$ m.

The analytical stress components for this problem are

$$\sigma_{xx}(x,y) = \sigma\left(1 - \frac{a^2}{r^2}\left(\frac{3}{2}\cos 2\theta + \cos 4\theta\right) + \frac{3a^4}{2r^4}\cos 4\theta\right) \tag{124.1}$$

$$\sigma_{yy}(x,y) = -\sigma\left(\frac{a^2}{r^2}\left(\frac{1}{2}\cos 2\theta - \cos 4\theta\right) - \frac{3a^4}{2r^4}\cos 4\theta\right) \tag{124.2}$$

$$\sigma_{xy}(x,y) = -\sigma\left(\frac{a^2}{r^2}\left(\frac{1}{2}\sin 2\theta + \sin 4\theta\right) + \frac{3a^4}{2r^4}\sin 4\theta\right) \tag{124.3}$$

Note that on page 31 of Ramachandran [256] (2016) cites Chen, Nguyen-Xuan, Nguyen-Thoi, Zeng, and Wu [66] (2010) which cites the book [358, p772] which cites Budynas [46] for the solution!

there are discrepancies between [256] and [66]. because of course.

Following [358], it can be shown, from linear elasticity, that the tangential stress throughout the plate is given by

$$\sigma_\theta = \frac{\sigma}{2}\left[1 + \frac{a^2}{r^2} - \left(1 + 3\frac{a^4}{r^4}\right)\cos 2\theta\right]$$

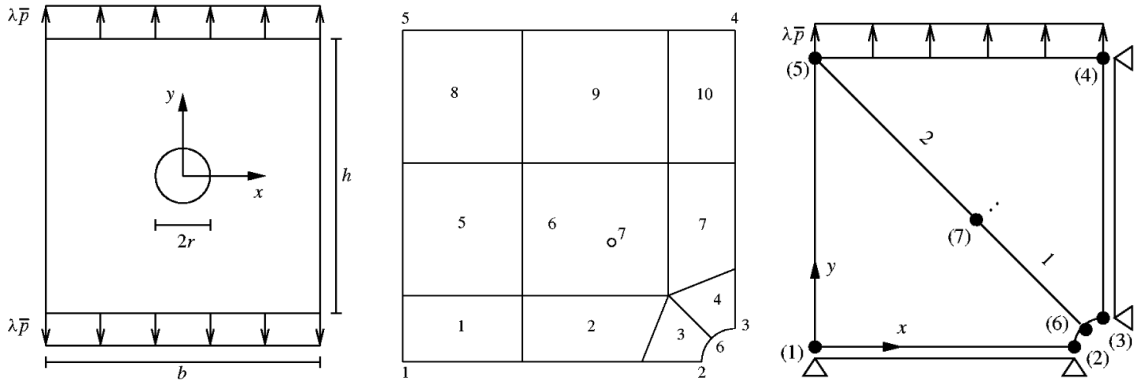
The maximum stress is $\sigma_\theta = 3\sigma$ at $r = a$ and $\theta = \pm\pi/2$. Along the surface of the hole, the tangential stress is $-\sigma$ at y $\theta = 0$ and $\theta = \pi$, and increases, as θ increases, to 3σ at $\theta = \pi/2$ and $\theta = 3\pi/2$.

Ramm et al. [258] (experiment=3)

This is section 11.2.2 of the book. The benchmark consists of a stretched plate with a circular central hole under plane strain condition. The width b of the plate is equal to its height $h = 100\,\mathrm{mm}$; the radius r of the hole is 10 mm. The following figure shows the system with its boundary conditions. The applied tractions $\bar{p} = 100\,\mathrm{N\,mm^{-2}}$ are scaled with the load factor[1] $\lambda = 4.5$. We use Young's modulus $E = 206\,900\,\mathrm{kN\,mm^{-2}}$, and Poisson's ratio $\nu = 0.29$.

It is not clear what the boundary conditions are at the hole but we assume that no boundary condition is prescribed, which is confirmed by the results that follow. Because we model the whole plate we have $L_x = L_y = 2h$.

There are two translational nullspaces and these are removed by prescribing horizontal or vertical displacements at key locations where they have to be zero because of the symmetry of the problem.



Taken from Ramm et al. [258]. In our domain Point (2) has coordinates $(L_x/2 - rad, Ly/2)$, Point (4) has coordinates $(L_x/2, Ly)$, Point (5) has coordinates $(0, Ly)$.

The following measurements are carried out:

- displacement υ_x at point (2).
- stress σ_{yy} at point (2);
- displacement υ_y at point (4);
- $\int_{(4)}^{(5)} \upsilon_y$ (currently not implemented)
- displacement υ_x at point (5);

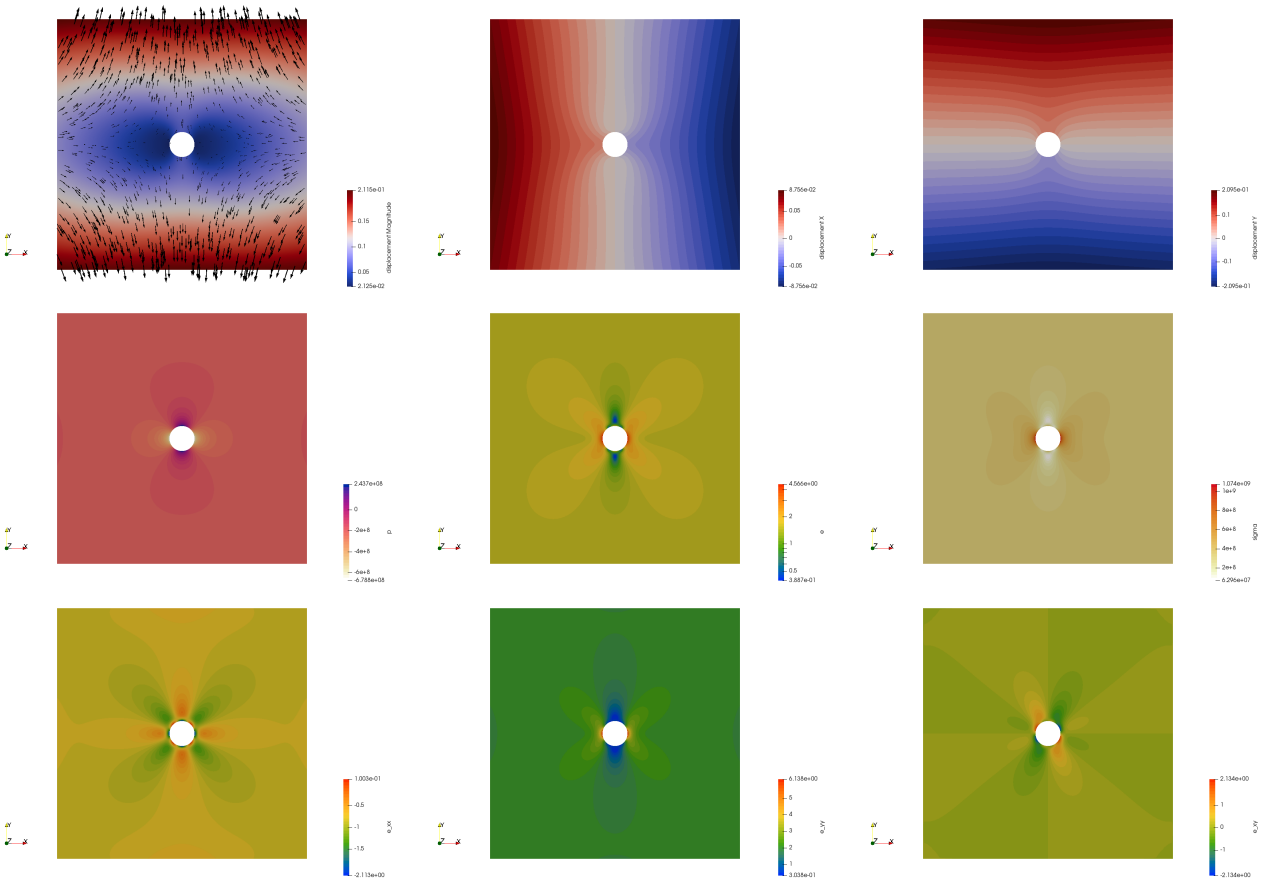
Note that there seems to be a problem with the scaling of the results when the above parameters are used. Using instead $E = 206\,900\,\mathrm{kN\,m^{-2}} = 2.069e8$ we recover the values of the following table:

[1] This has nothing to do with the real elastic parameter λ!

Participant	u_x at (2) in mm	σ_{yy} at (2) in N/mm^2	u_y at (4) in mm	type of element
Reference	0.021290	1388.732343	0.20951	Q19
Rank	0.021290	1388.858414	0.20951	Q10
Rannacher	0.021238	1363.9	0.20950	Q1
Stein	0.021175	1409.69999	0.20944	Q1
Stein	0.021287	1392.32958	0.20951	Q2
Stein	0.021133	1426.3608	0.20946	Q1–P0
Stein	0.021253	1420.5582	0.20947	Q1–E5
Wendland	0.021292	1467.93	0.20952	BEM
Wittum	0.021291	1389.0	0.02095	Q2
Wunderlich	0.021290	1382.00336	0.02095	Q2
Wriggers	0.021177	1339.16	0.02094	T1

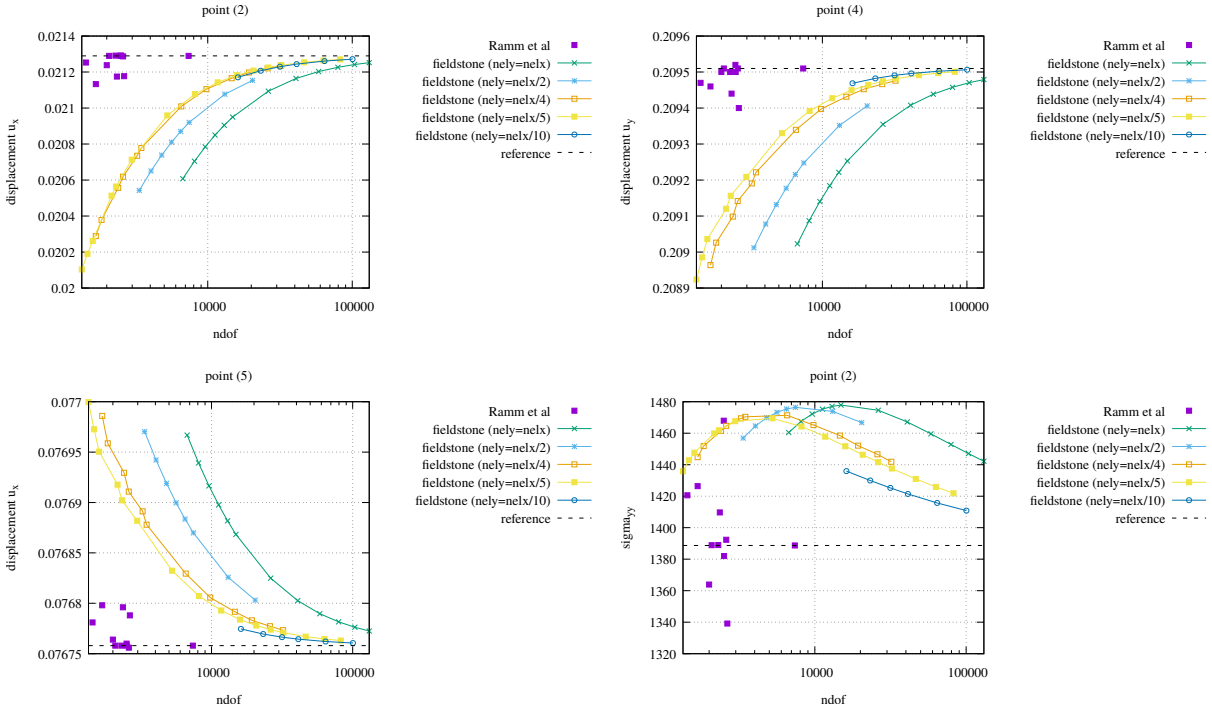
Participant	u_x at (5) in mm	$\int_{(4)}^{(5)} u_y$ in mm^2	# of d.o.f.	type of element
Reference	0.076758	20.40344	7372	Q19
Rank	0.076758	20.40344	2080	Q10
Rannacher	0.076764	20.403	2000	Q1
Stein	0.076796	20.40156	2353	Q1
Stein	0.076756	20.40338	2596	Q2
Stein	0.076798	20.40080	1680	Q1–P0
Stein	0.076781	20.40123	1436	Q1–E5
Wendland	0.076760	20.40321	2500	BEM
Wittum	0.076758	20.4034	2300	Q2
Wunderlich	0.076758	20.4034	2512	Q2
Wriggers	0.076788	20.4016	2639	T1

Taken from Ramm et al. [258]. Note that there probably is a problem with 3 values of the reported v_y for point (4).



Each block counts 90 × 90 elements.

The values in the above tables are now plotted (purple squares) against our results. We find a good agreement but it looks like we need a *much higher* high resolution to recover the published results (although ajdusting the ratio of nely to nelx does help).


point (2)
displacement u_x
ndof
0.0214
0.0212
0.021
0.0208
0.0206
0.0204
0.0202
0.02
10000
100000
Ramm et al
fieldstone (nely=nelx)
fieldstone (nely=nelx/2)
fieldstone (nely=nelx/4)
fieldstone (nely=nelx/5)
fieldstone (nely=nelx/10)
reference
point (4)
displacement u_y
ndof
0.2096
0.2095
0.2094
0.2093
0.2092
0.2091
0.209
0.2089
10000
100000
Ramm et al
fieldstone (nely=nelx)
fieldstone (nely=nelx/2)
fieldstone (nely=nelx/4)
fieldstone (nely=nelx/5)
fieldstone (nely=nelx/10)
reference
point (5)
displacement u_x
ndof
0.077
0.07695
0.0769
0.07685
0.0768
0.07675
10000
100000
Ramm et al
fieldstone (nely=nelx)
fieldstone (nely=nelx/2)
fieldstone (nely=nelx/4)
fieldstone (nely=nelx/5)
fieldstone (nely=nelx/10)
reference
point (2)
sigma$_{yy}$
ndof
1480
1460
1440
1420
1400
1380
1360
1340
1320
10000
100000
Ramm et al
fieldstone (nely=nelx)
fieldstone (nely=nelx/2)
fieldstone (nely=nelx/4)
fieldstone (nely=nelx/5)
fieldstone (nely=nelx/10)
reference

Stone 125

Building Voronoi cells

python_codes/fieldstone_125/text.tex

P Code at https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_125

---

eneral]Voronoi Diagram

To quote Wikipedia[1]: "In mathematics, a Voronoi diagram is a partition of a plane into regions close to each of a given set of objects. In the simplest case, these objects are just finitely many points in the plane (called seeds, sites, or generators). For each seed there is a corresponding region, called a Voronoi cell, consisting of all points of the plane closer to that seed than to any other. The Voronoi diagram of a set of points is dual to its Delaunay triangulation. "

Relevant Literature: May [226] (2012), Hüttig and Stemmer [164] (2008).

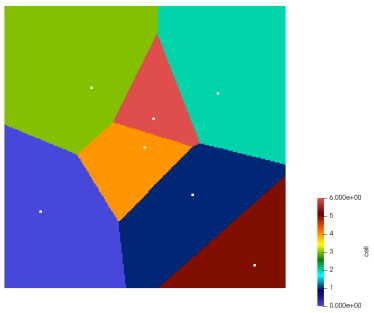
The domain is a 2D (ndim=2) or 3D (ndim=3) Cartesian box. A regular grid spans this domain with resolution $nelx \times nely$ or $nelx \times nely \times nelz$. In each cell $nmarker_per_dim^{ndim}$ markers are regularly placed. Finally nvo seeds are randomly placed in the domain $[0.01 : 0.99] \times [0.01 : 0.99]$ and wish to compute the Voronoi cell attached to each seed. There are three test cases in the code, differing only by the number of seeds (7,27,111 respectively). The volume of each cell is also computed (based on the area assigned to each marker).

Note that this code is by no means fast. It simply does the job. If large number of points and/or cells are used or one wishes to generate hundreds of diagrams then optimised algorithms should be used. Also, the algorithm does not return the identity of the neighbouring cells.

TODO: write a faster version which assigns a V cell to a marker using the cell number of its neighbour. Credit Henry and link to his thesis appendix.

See Furstoss [121] (2020) phd thesis, chapter 2.

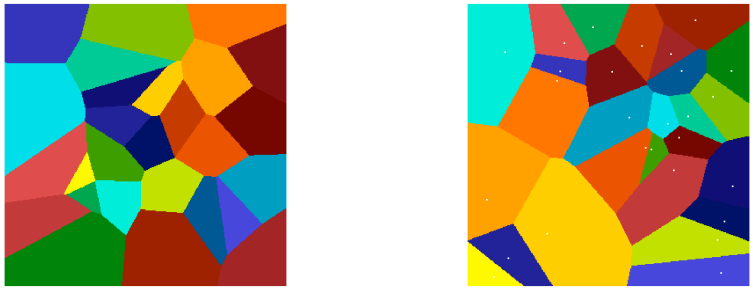
Test # 1



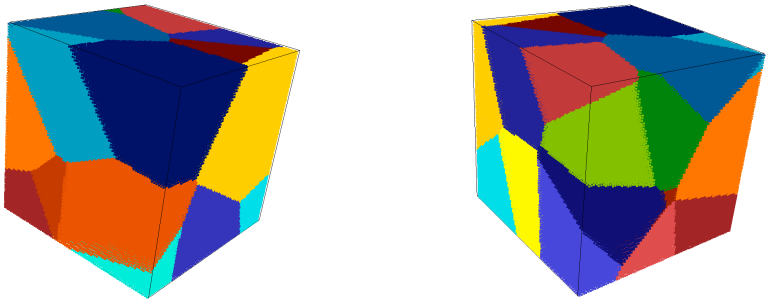
Example with 7 seeds.

[1] https://en.wikipedia.org/wiki/Voronoi_diagram

Test # 2

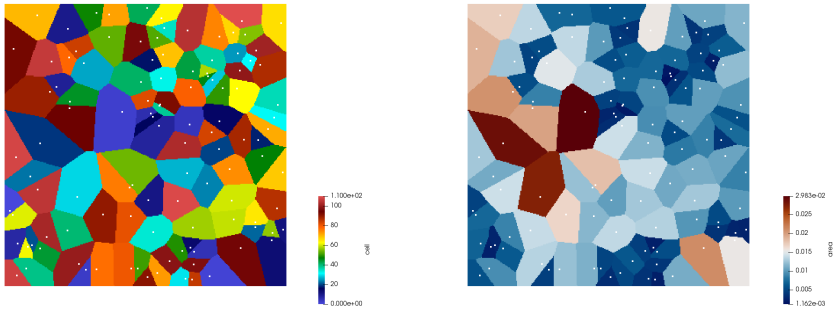


Examples with two different sets of 27 random seeds in 2D.



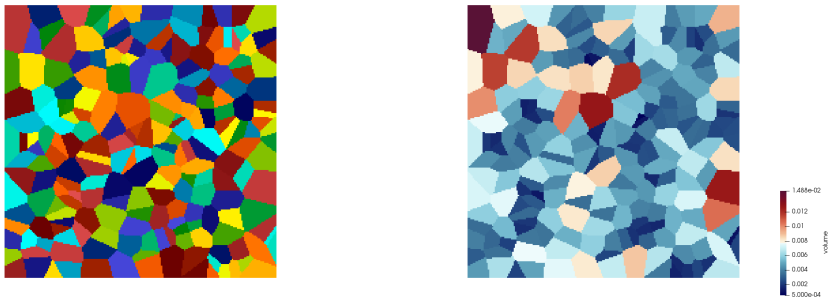
Example with 27 random seeds in 2D (different views).

Test # 3



Examples with 111 random seeds. Approx 310s.

Test # 4



Examples with 40,000 markers and 213 random seeds. Approx 275s.

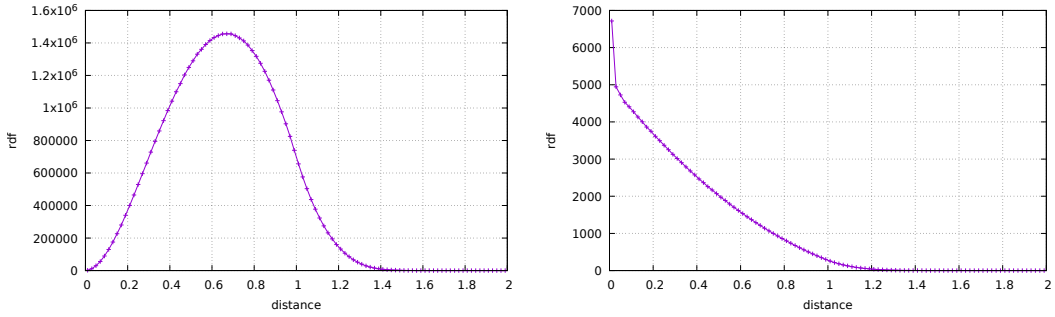
---

eneral]Radial Distribution Function

This is a completely different exercise. A swarm of points is generated in a 3D Cartesian domain using a random number generator for the coordinates. The code computes the so-called Radial Distribution

Function [2], which is describes how density varies as a function of distance from a reference particle. In simplest terms it is a measure of the probability of finding a particle at a distance of r away from a given reference particle, relative to that for an ideal gas. The general algorithm involves determining how many particles are within a distance of r and $r + dr$ away from a particle.

The implemented algorithm is rather simple. Loop over all particles, and for each of them loop over the rest of the particles, compute the distance, find out in which bin of size dr it belongs to and increase the count in that bin. We end up with the raw distribution. Note that in order to avoid to count the same pair twice the second loop runs over 0 to i, and not 0 to nmarker (It also saves a lot of time!). As explained on Wikipedia, one can also normalise it by $\rho 4\pi r^2 dr$, where ρ is the average density of points (number of points / volume). This yields the normalised rdf. Also note that points close to the sides of the domain are not 'seeing' the ones on the other side (no periodic conditions) so this is likely to alter the results.



a

[2] https://en.wikipedia.org/wiki/Radial_distribution_function

Stone 126

Darcy stuff

python_codes/fieldstone_126/text.tex

P Code at https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_126

What follows is borrowed from Gratier, Favreau, and Renard [139] (2003). Two main relations are used, expressing fluid mass conservation

$$\frac{\partial(\rho\phi)}{\partial t}+\vec{\nabla}\cdot(\rho\vec{v})=f$$

and Darcy's law

$$\vec{v}=-\boldsymbol{K}\cdot(\vec{\nabla}P+\rho\vec{g})$$

where ρ is the density of the fluid, ϕ is the connected mean porosity of the rock, $\vec{v}$ is the Darcy flow rate (fluid particle velocity times porosity)[1], $\boldsymbol{K}$ is the permeability tensor, P is the total pore pressure and $\vec{g}$ is the gravity.

First, during the evolution, we consider that the density $\rho=\rho'+\rho_m$ will evolve in time of an amount ρ' around its local mean value ρ_m , and finally, we rewrite the fluid mass conservation by neglecting the terms ρ' compared to ρ_m.

Then,

$$\frac{\partial((\rho'+\rho_m)\phi)}{\partial t}+\vec{\nabla}\cdot((\rho'+\rho_m)\vec{v})=f$$

$$(\rho'+\rho_m)\frac{\partial\phi}{\partial t}+\phi\frac{\partial(\rho'+\rho_m)}{\partial t}+\vec{\nabla}\cdot((\rho'+\rho_m)\vec{v})=f$$

Since ρ_m is constant then $\partial_t\rho_m=0$ so we obtain

$$\rho_m\frac{\partial\phi}{\partial t}+\phi\frac{\partial\rho'}{\partial t}+\vec{\nabla}\cdot(\rho_m\vec{v})=f$$

Second, we consider that the total pore pressure P can be decomposed as $P=p+P_{hydro}$ such that P_{hydro} is the hydrostatic pressure verifying $\vec{\nabla}P_{hydro}+\rho_m\vec{g}=\vec{0}$. Then

$$\begin{aligned}\vec{v}&=\boldsymbol{K}\cdot(\vec{\nabla}(p+P_{hydro})+(\rho'+\rho_m)\vec{g})\\ \Rightarrow\vec{v}&=\boldsymbol{K}\cdot(\vec{\nabla}p+\rho'\vec{g})\end{aligned}$$

Furthermore, JUSTIFY!![2]

$$\phi\frac{\partial\rho'}{\partial t}=\rho_m C_f\phi\frac{\partial p}{\partial t}$$

so in the end

$$\rho_m C_f\phi\frac{\partial p}{\partial t}+\rho_m\frac{\partial\phi}{\partial t}-\vec{\nabla}\cdot(\rho_m\boldsymbol{K}\cdot\vec{\nabla}p))=f \tag{126.1}$$

[1] Whaaaat?

[2] Why does grad rho' g disappear ?!? is it because it is negligible in our context compared to pressure gradients?

We assume an isotropic permeability so that the tensor now simply writes:

$$K = K_0 \phi^3$$

and then

$$\rho_m C_f \phi \frac{\partial p}{\partial t} + \rho_m \frac{\partial \phi}{\partial t} - \vec{\nabla} \cdot (\rho_m K \cdot \vec{\nabla} p)) = f \tag{126.2}$$

The discretised weak form takes the form

$$\underbrace{\left(\int_\Omega \vec{\mathcal{N}}^T \rho_m C_f \phi \vec{\mathcal{N}} dV \right)}_{\boldsymbol{M}} \cdot \frac{\partial \vec{\mathcal{P}}}{\partial t} + \underbrace{\left(\int_\Omega \boldsymbol{B}^T \cdot \rho_m \boldsymbol{K} \cdot \boldsymbol{B} dV \right)}_{\boldsymbol{K}_d} \cdot \vec{\mathcal{P}} = \underbrace{\int_\Omega \vec{\mathcal{N}}^T f dV - \int_\Omega \vec{\mathcal{N}}^T \rho_m \frac{\partial \phi}{\partial t} dV}_{\vec{b}}$$

The porosity ϕ evolution equation is given by:

$$\frac{\partial \phi}{\partial t} = \alpha \dot{\varepsilon}_e \qquad \Rightarrow \qquad \phi^{t+\delta t} = \phi^t + \alpha \dot{\varepsilon}_e \delta t$$

while the source term f equation is:

$$\frac{\partial f}{\partial t} = \varphi f \dot{\varepsilon}_e \qquad \Rightarrow \qquad f^{t+\delta t} = f^t (1 + \varphi \dot{\varepsilon}_e \delta t)$$

Likewise the time derivative is discretised by means of a simple explict scheme:

$$\frac{\partial \vec{\mathcal{P}}}{\partial t} \simeq \frac{\vec{\mathcal{P}}^{t+\delta t} - \vec{\mathcal{P}}^t}{\delta t}$$

We arrive at

$$(\boldsymbol{M} + \boldsymbol{K}_d) \cdot \vec{\mathcal{P}}^{t+\delta t} = \boldsymbol{M} \cdot \vec{\mathcal{P}}^t + \vec{b}$$

The matrix is symmetric positive definite which is good news.

The PDE above contains two parameters ϕ (and therefore also K) and f which depend on the strainrate. The strainrate field will ultimately come from the other code which solves the Stokes equations in the shear band. The timestep of that code is of the order of at least a few thousands of years. However the characteristic time step for Darcy is about a year. This means that in the span of 1 tectonic timestep the Darcy equations will be solved with ϕ and f constant. This simplifies things considerably since this renders the Darcy equation a linear diffusion equation with a heat source.

In the end, when we incorporate the Darcy solver in the Stokes code we may even choose to solve the steady-state Darcy equation once per tectonic time step!

Given the nature of the equation describing the evolution of the porosity it can and will ultimately exceed physically meaningful values so we implement a cutoff value ϕ_{max}.

characteristic time

$$t_c = \frac{\phi C_f h^2}{K} = \frac{\phi^{-2} C_f h^2}{K_0}$$

TODO: - f constant, derive analytical soltuion, compare with ss results

dilatancy a la chpe15 ?

Stone 127

circular inclusion in simple shear ($P_2^+ \times P_{-1}$)

python_codes/fieldstone_01/text.tex

2D
box
P2pPm1
sparse_matrix
numerical_benchmark
stokes_equation
use_mesher
nonlinear
viscous_rheology
paraview_output
pressure_scaling
pressure_normalisation

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_127`

This stone was developed in collaboration with Fanny Garel and Catherine Thoraval.

TODO: remove call to triangle code and use Delaunay triangulator of f96,f140 instead

The domain is $L_x \times L_y = 200 \times 50$ km. Pure shear boundary conditions are imposed by applying a velocity $\vec{\mathbf{v}} = (+u_{bc}, 0)$ at the top and $\vec{\mathbf{v}} = (-u_{bc}, 0)$ at the bottom so that $\dot{\varepsilon}_0 = \frac{1}{2}(2u_{bc}/L_y = u_{bc}/L_y)$. The temperature is constant and set to T_0 in the domain. Two materials are present: the background matrix (material 1) and a circular inclusion (material 2) centered in the domain and of 10 km radius with a constant viscosity $\eta_i = 1 \times 10^{25}$ Pa s. Buoyancy forces are neglected so that density values are irrelevant.

Meshing is based on STONE 93. Crouzeix-Raviart elements are used. Pressure is normalised to be on average zero over the domain. Standard Picard iterations are carried out and they stop when both components of the velocity do not change by more than tol in relative norm.

The rheology of the matrix can take several forms:

- rheology=0: Constant viscosity $\eta = 1 \times 10^{22}$ Pa s. This renders the physics linear (and independent of temperature). Changing the velocity boundary conditions only changes the magnitude of the computed fields.

- rheology=1: Diffusion creep only, with $Q_{\rm df} = 410 \times 10^3\,{\rm J\,mol^{-1}}$, $A_{\rm df} = 1e-7$, no activation volume.

$$\eta_{\rm df} = A_{\rm df}^{-1} \exp \frac{Q_{\rm df}}{RT}$$

- rheology=2: Diffusion creep of rheology 1 + tanh formulation from Garel, Thoraval, Tommasi, Demouchy, and Davies [122]

$$\begin{aligned} \sigma &= (a_0 + b_0 T)\left(1 + \tanh((a_1 + b_1 T)(\log_{10}(\dot{\varepsilon}_e) - (a_2 + b_2 T + c_2 T^2)))\right) \\ \eta_{\text{tanh}} &= \frac{\sigma}{2\dot{\varepsilon}_e} \end{aligned}$$

- rheology=3: Diffusion creep of rheology 1 + erf formulation from Garel, Thoraval, Tommasi, Demouchy, and Davies [122]

$$\begin{aligned} \sigma &= (a_0 + b_0 T)\left(1 + \text{erf}((a_1 + b_1 T)(\log_{10}(\dot{\varepsilon}_e) - (a_2 + b_2 T + c_2 T^2)))\right) \\ \eta_{\text{erf}} &= \frac{\sigma}{2\dot{\varepsilon}_e} \end{aligned}$$

- rheology=4: Diffusion creep of rheology 1 + Dislocation creep from Gouriet et al. [137] (2019):

$$\eta_{\text{ds}} = A_{\text{ds}}^{-1/n} \dot{\varepsilon}_e^{-1+1/n_{\text{ds}}} \exp\left(\frac{Q_{\text{df}}}{n_{\text{ds}} RT}\right)$$

  with $A_{\text{ds}} = 5.27e-29$, $n_{\text{ds}} = 4.5$, $Q_{\text{ds}} = 443e3$.

- rheology=5: Diffusion creep of rheology 1 + Dislocation creep from Garel, Thoraval, Tommasi, Demouchy, and Davies [122] (2020):

$$\eta_{\text{ds}} = A_{\text{ds}}^{-1/n} \dot{\varepsilon}_e^{-1+1/n_{\text{ds}}} \exp\left(\frac{Q_{\text{df}}}{n_{\text{ds}} RT}\right)$$

  with $A_{\text{ds}} = 5e-16$, $n_{\text{ds}} = 3.5$, $Q_{\text{ds}} = 540e3$.

WRITE about multivalued fields on nodes.

Stone 128

fluid flow in porous media - Simpson book chapter 10 (Q_1)

python_codes/fieldstone_128/text.tex



P Code at https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_128

Last revision: Oct. 1st, 2024.

Following chapter 10 of Guy Simpson. *Practical Finite Element Modeling in Earth Science Using Matlab.* Wiley-Blackwell, 2017. ISBN: 978-1-119-24862-0 the equations governing the evolution of the fluid pressure and fluid flow velocities are[1]

$$\varphi\beta\frac{\partial p}{\partial t} = -\vec{\nabla}\cdot\vec{\mathrm{v}} + H \tag{128.1}$$

$$\vec{\mathrm{v}} = -\frac{K}{\eta_f}\left(\vec{\nabla}p - \rho_f g\vec{e}_z\right) \tag{128.2}$$

where

- p is the fluid pressure,
- $\vec{\mathrm{v}}$ is the fluid velocity vector,
- φ is the porosity (-),
- β is the bulk compressibility (Pa^{-1}),
- K is the permeability (m^2),
- η_f is the fluid viscosity ($\mathrm{Pa\,s}$),
- ρ_f is the water density ($\mathrm{kg\,m}^{-3}$),
- g is acceleration due to gravity ($\mathrm{m\,s}^{-2}$),
- $\vec{e}_z$ is a unit vector oriented in the vertical direction,
- H accounts for any fluid pressure sources or sinks (e.g., due to devolatilization reactions)

[1] I have slightly changed the notations with respect to the book.

Introducing the excess pore fluid pressure defined as the pressure in excess of the hydrostatic pressure (i.e., $p_e = p - p_h$ where $p_h = \rho_f g(L_z - z)$) and substituting (128.2) into (128.1) lead to a single parabolic equation for the excess fluid pressure as follows:

$$\varphi\beta\frac{\partial p_e}{\partial t} = \vec{\nabla}\cdot\left(\frac{K}{\eta_f}\vec{\nabla}p_e\right) + H \tag{128.3}$$

Note that once the excess pressure is computed one can recover the fluid velocity via $\vec{\mathrm{v}} = -\frac{K}{\eta_f}\vec{\nabla}p_e$. Note that $H = 0$ in all three experiments carried out in this STONE .

Applying standard FE methodology to what is essentially a diffusion equation, we arrive at:

$$\mathbb{M}\cdot\frac{\partial}{\partial t}\vec{\mathcal{P}}_e + \mathbb{K}_d\cdot\vec{\mathcal{P}}_e = \vec{b}$$

with

$$\mathbb{M} = \int_\Omega \varphi\beta\vec{\mathcal{N}}^T\vec{\mathcal{N}}$$

$$\mathbb{K}_d = \int_\Omega \boldsymbol{B}^T\boldsymbol{B}\frac{K}{\eta_f}$$

$$\vec{b} = \int_\Omega \vec{\mathcal{N}}^T H$$

Using a simple first order time discretisation yields

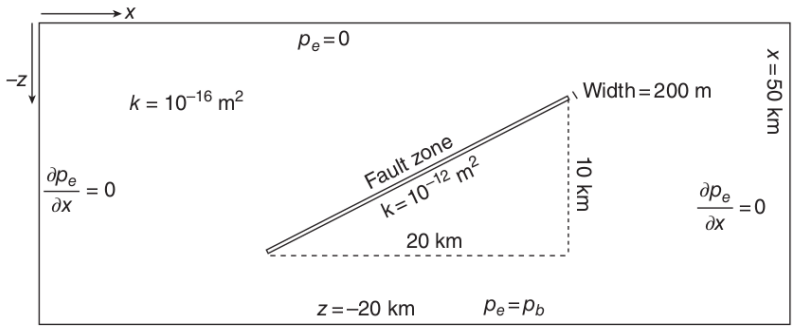
$$(\mathbb{M} + \mathbb{K}_d\delta t)\cdot\vec{\mathcal{P}}_e^n = \mathbb{M}\cdot\vec{\mathcal{P}}_e^{n-1} + \vec{b}\delta t$$

Fluid Flow Around a Fault (`experiment=2`)

A porous crust with an initially uniform and isotropic permeability has an overpressure that increases linearly with depth. At $t = 0$ a dipping, high permeability fault zone is suddenly introduced into the crust. The transient evolution of the excess pore pressure is governed by

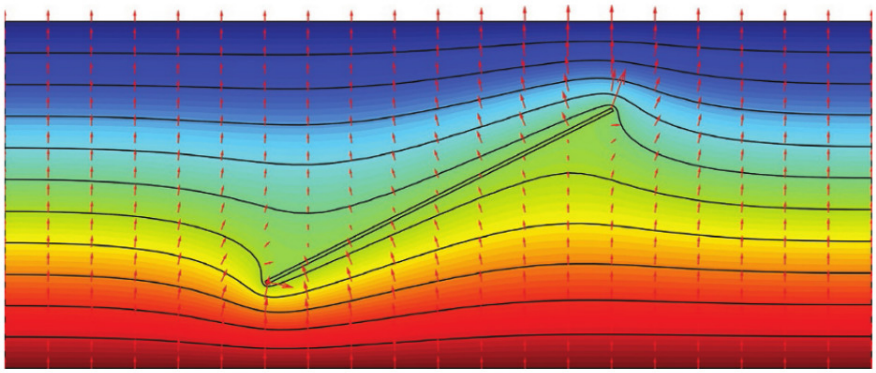
$$\varphi\beta\frac{\partial p_e}{\partial t} = \vec{\nabla}\cdot\left(\frac{K(x,y)}{\eta_f}\vec{\nabla}p_e\right) \tag{128.4}$$

subject to the boundary conditions as specified in the following figure:



Taken from [298]. Setup for model of fluid flow around a high permeability fault zone.

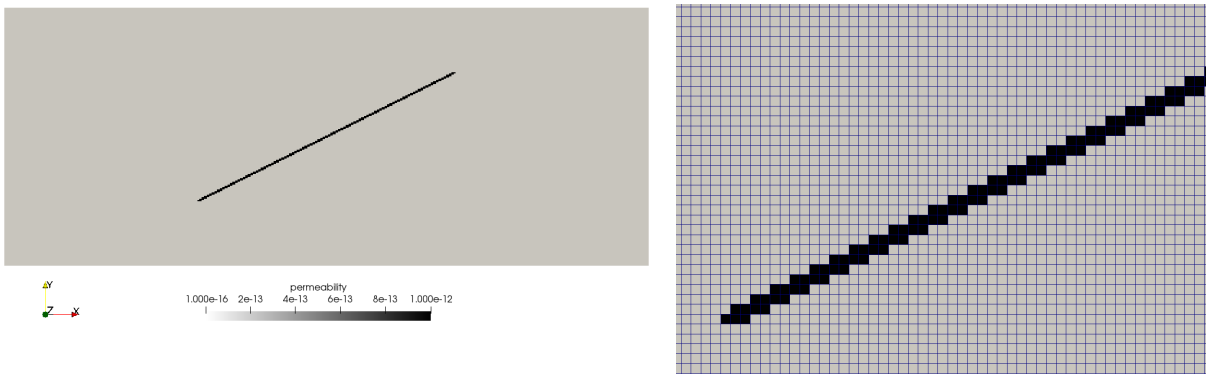
In the book the problem was solved with a total of 5248 triangles, a time step of 0.1 years, and the following physical parameters: $K_{crust} = 10^{-16}$ m^2 , $K_{fault} = 10^{-12}$ m^2 , $\eta_f = 1.33\times10^{-4}$ Pa s, $\beta = 1\times10^{-10}$ Pa^{-1}, and $\varphi = 0.1$. The results, after an approximately steady situation has been achieved (after 10 years), show how the fluid overpressure and fluid flow vectors in the crust are strongly perturbed by the presence of the high permeability fault zone. Fluid is focused toward the base of the fault, while it diverges away from its upper tip.



Taken from [298]. Excess fluid pressure and flow vectors in the crust (with a uniform permeability of $1 \times 10^{-16}\,\mathrm{m}^{-2}$) 10 years after introduction of a high permeability ($1 \times 10^{-12}\,\mathrm{m}^{-2}$) fault zone.

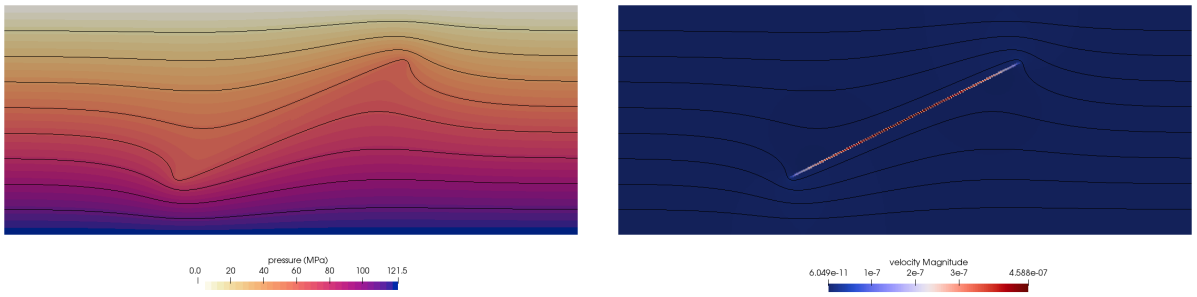
The matlab code in the book relies on a bespoke mesh of linear triangles. Unfortunately the mesh is stored in a `.mat` format which is very inconvenient. In this STONE we rely on linear quadrilaterals (Q_1) instead.

In what follows we use a 500×200 mesh. Because of the geometry of the fault and the fact that we rely on a regular mesh, the fault is then represented as follows in the model:

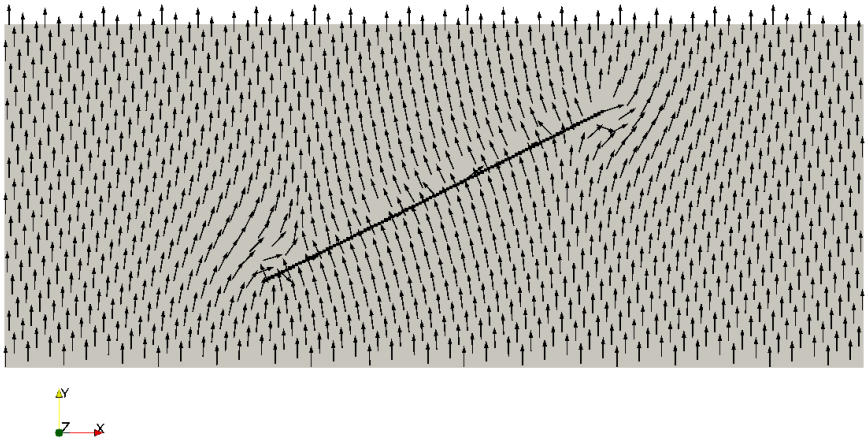


Permeability on 500×200 mesh.

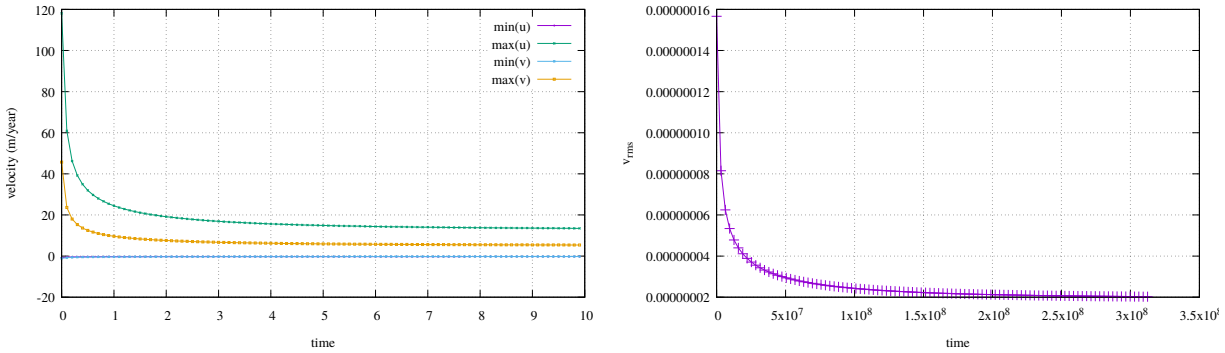
The rest of the code is rather straight forward since it is a diffusion equation (see for instance STONE 03). We carry out 100 timesteps of $\delta t = 0.1$ yr each, which is indeed enough time to reach a steady state. The recovered pressure and velocity fields are similar to those in the book but since the book figure does not showcase a colorscale we cannot quantitatively compare[2]. The velocity field is essentially a scaled pressure gradient and we compute it in the middle of each element for simplicity. The code computes the root mean square velocity and the average pressure over the domain.



[2]Run code with Matlab?



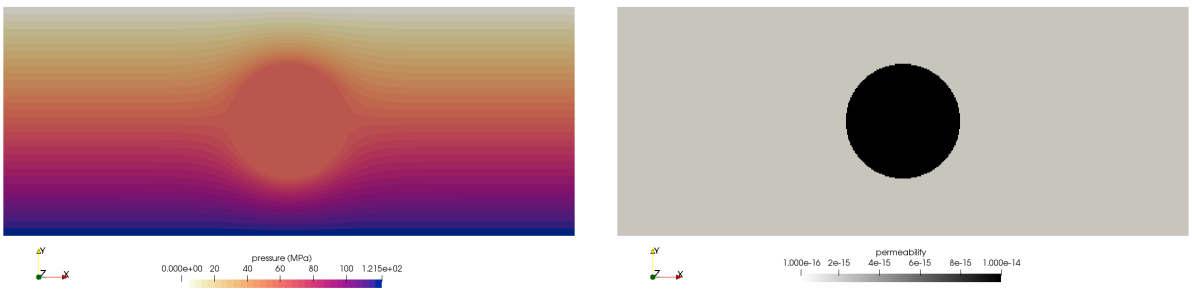
Pressure and velocity field at $t = 10$ yr. Note that there is a very large difference in velocity magnitude between the crust and the fault. The velocity arrows are actually obtained by normalising the velocity vector to one in each element.

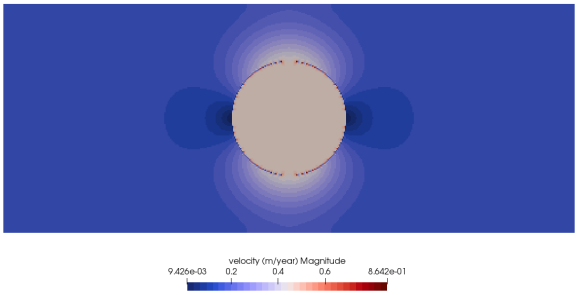


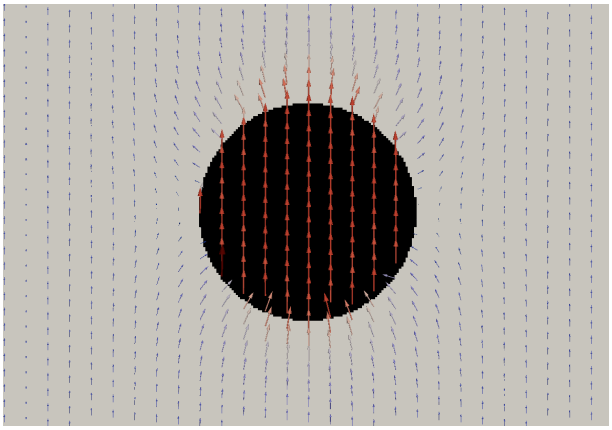
Velocity statistics. Pressure statistics and volume average plots are not interesting so not shown.

Fluid Flow Around a circular inclusion (`experiment=1`)

Same setup as experiment 2, but the fault has now been replaced by a circular inclusion placed in the middle of the domain and of radius 5 km.

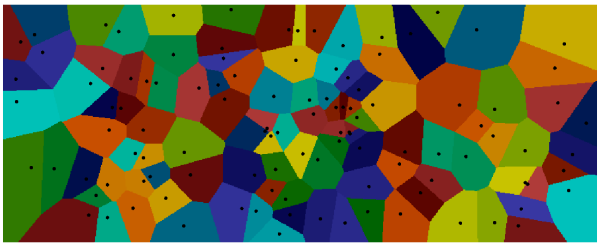


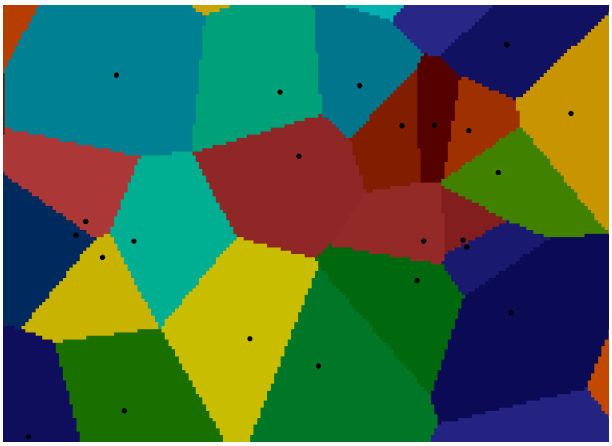




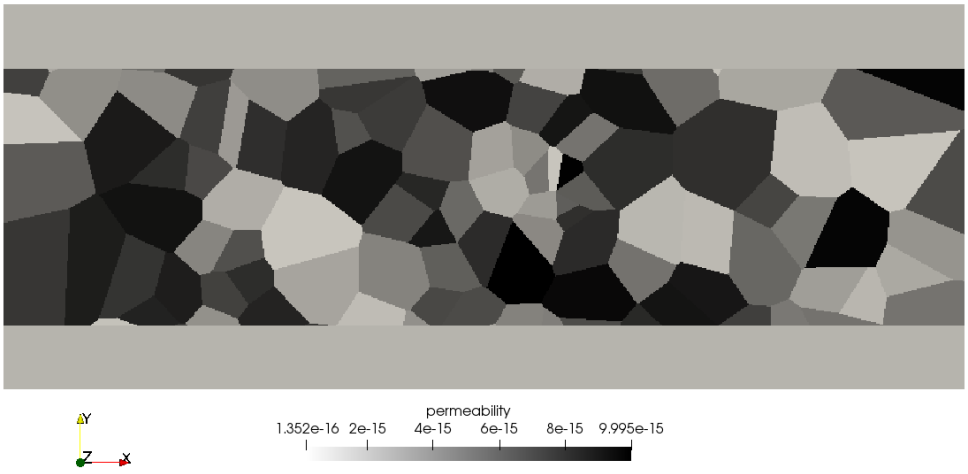
Fluid Flow in a (`experiment=3`)

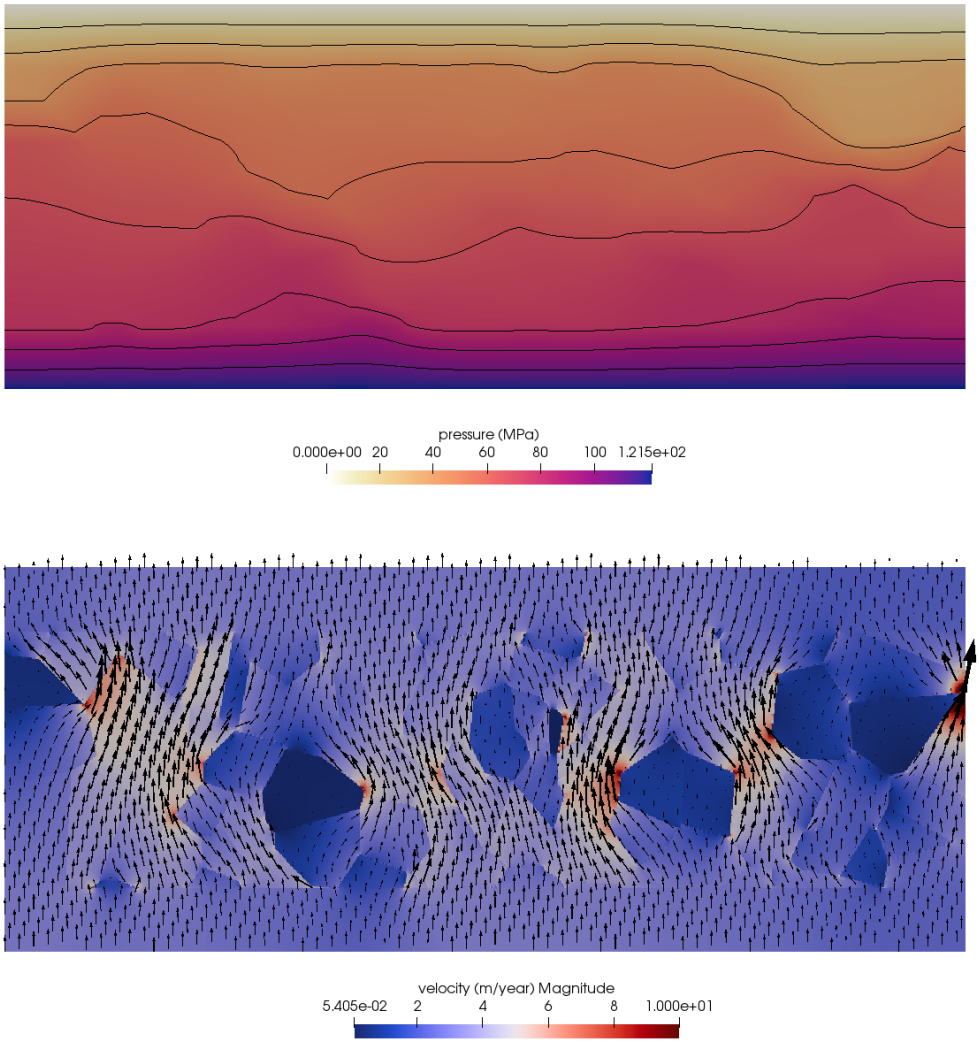
All parameters are identical to experiment 2. However I have placed 111 random seeds in the domain and computed the voronoi diagram of these as projected onto the elements:





Each Voronoi cell is then attributed a random permeability between 10^{-14} and 10^{-16}, using the algorithm documented in STONE 125. Two bands of thickness $L_y/6$ with constant permeability 10^{-15} are prescribed at the top and bottom of the domain. Simulations are run until steady state is reached.





Run on 800x320 mesh. This could easily be increased.

Remark: this setup as it is now makes no sense (there is no reason why the spatial distribution of the permeability would follow a Voronoi cell distribution). However, it could be scaled down to a few meters or centimeters. Pressure boundary conditions and permeability values could also be adapted. Three-dimensional simulations could equally easily be run, albeit with a much higher computational cost. Obviously the number of 'grains', their position, their permeability, etc ... can be changed too.

Stone 129

Visco-elastic materials - Simpson book chapter 12 (Q_2)

python_codes/fieldstone_129/text.tex

2D
box
Q2
sparse_matrix
compressible
analytical_benchmark
numerical_benchmark
stokes_equation
time_stepping
elastic_rheology
viscous_rheology
paraview_output

P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_129`

This stone was developed in collaboration with Taco Broerse.

The flow is compressible, but we start from Eq. (12.40) in the book of Simpson (I here use my notations and not those of the book), which only considers the deviatoric part of the strain rate tensor:

$$\begin{aligned}
\dot{\boldsymbol{\varepsilon}}^d &= \frac{1}{2\mu}\frac{\partial \boldsymbol{\tau}}{\partial t} + \frac{1}{2\eta}\boldsymbol{\tau} \\
&\simeq \frac{1}{2\mu}\frac{\boldsymbol{\tau}-\boldsymbol{\tau}^0}{\delta t} + \frac{1}{2\eta}\boldsymbol{\tau} \\
&= \frac{1}{2}\left(\frac{1}{\mu\delta t}+\frac{1}{\eta}\right)\boldsymbol{\tau} - \frac{1}{2\mu\delta t}\boldsymbol{\tau}^0 \\
&= \frac{1}{2}\frac{\mu\delta t+\eta}{\mu\delta t\eta}\boldsymbol{\tau} - \frac{1}{2\mu\delta t}\boldsymbol{\tau}^0
\end{aligned} \tag{129.1}$$

where $\boldsymbol{\tau}^0$ is the already known/previous deviatoric stress tensor. This can be rewritten as

$$\begin{aligned}
\boldsymbol{\tau} &= 2\frac{\mu\delta t\,\eta}{\mu\delta t+\eta}\dot{\boldsymbol{\varepsilon}}^d + 2\frac{\mu\delta t\,\eta}{\mu\delta t+\eta}\frac{1}{2\mu\delta t}\boldsymbol{\tau}^0 \\
&= \frac{2\mu\delta t\eta}{\mu\delta t+\eta}\dot{\boldsymbol{\varepsilon}}^d + \frac{\eta}{\mu\delta t+\eta}\boldsymbol{\tau}^0
\end{aligned} \tag{129.2}$$

Defining the mean stress σ_m as

$$\sigma_m = \frac{1}{3}(\sigma_{xx} + \sigma_{yy} + \sigma_{zz})$$

then by definition of the deviatoric stress $\boldsymbol{\tau} = \boldsymbol{\sigma} - \sigma_m \mathbf{1}$. Also, the viscous deformation is deviatoric so the volume changes are only due to the elastic deformation and we have[1]

$$\sigma_m = K \mathrm{tr}(\boldsymbol{\varepsilon})$$

Then

$$\frac{\partial \sigma_m}{\partial t} \simeq \frac{\sigma_m - \sigma_m^0}{\delta t} = K \mathrm{tr}(\dot{\boldsymbol{\varepsilon}}) \qquad \Rightarrow \qquad \sigma_m \simeq K \mathrm{tr}(\dot{\boldsymbol{\varepsilon}}) \delta t + \sigma_m^0$$

and then

$$\boldsymbol{\tau} = \boldsymbol{\sigma} - \sigma_m \mathbf{1} = \boldsymbol{\sigma} - [K \mathrm{tr}(\dot{\boldsymbol{\varepsilon}}) \delta t + \sigma_m^0] \mathbf{1}$$

Inserting this in Eq. (129.2) we obtain

$$\boldsymbol{\sigma} = K \mathrm{tr}(\dot{\boldsymbol{\varepsilon}}) \delta t \mathbf{1} + \frac{2\mu\eta\delta t}{\mu\delta t + \eta} \dot{\boldsymbol{\varepsilon}}^d + \frac{\eta}{\mu\delta t + \eta} \boldsymbol{\tau}^0 + \sigma_m^0 \mathbf{1} \tag{129.3}$$

We can now switch to using a vector representation of the tensors[2] as it is necessary for implementation in a FEM code:

$$\begin{aligned} \boldsymbol{\sigma} &\rightarrow \vec{\sigma} \\ \boldsymbol{\varepsilon} &\rightarrow \boldsymbol{C} \cdot \vec{\varepsilon} \\ \mathbf{1} &\rightarrow \vec{m} \end{aligned}$$

with

$$\vec{m} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \qquad \boldsymbol{C} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{pmatrix}, \qquad \text{and} \qquad \dot{\vec{\varepsilon}} = \begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix}.$$

Also, as we have seen in Eq. (**??**) we have $\vec{\tau} = \boldsymbol{I}_d \cdot \vec{\sigma}$ with $\boldsymbol{I}_d$ the deviatoric operator:

$$\boldsymbol{I}_d = \begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

[1] In the book it is written as $\sigma_m = 3K\varepsilon_v$

[2] We need to be extra careful about the strain rate vector $\dot{\vec{\varepsilon}}$ since it contains a factor 2 in front of the off diagonal tensor terms, hence the $\boldsymbol{C}$ tensor.

so that now

$$
\begin{aligned}
\vec{\sigma} &= K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t\vec{m} + \frac{2\mu\eta\delta t}{\mu\delta t+\eta}\boldsymbol{I}_d\cdot\boldsymbol{C}\cdot\dot{\vec{\boldsymbol{\varepsilon}}} + \frac{\eta}{\mu\delta t+\eta}\boldsymbol{I}_d\cdot\boldsymbol{\sigma}^0 + \sigma^0_m\vec{m} \\
&= \begin{pmatrix} K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t \\ K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t \\ K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{2\mu\eta\delta t}{\mu\delta t+\eta}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} \\
&+ \frac{\eta}{\mu\delta t+\eta}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} \sigma^0_{xx} \\ \sigma^0_{yy} \\ \sigma^0_{zz} \\ \sigma^0_{xy} \\ \sigma^0_{xz} \\ \sigma^0_{yz} \end{pmatrix} + \sigma^0_m\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \delta t\begin{pmatrix} K & K & K & 0 & 0 & 0 \\ K & K & K & 0 & 0 & 0 \\ K & K & K & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} + \frac{2\mu\eta\delta t}{\mu\delta t+\eta}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} \\
&+ \frac{\eta}{\mu\delta t+\eta}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} \sigma^0_{xx} \\ \sigma^0_{yy} \\ \sigma^0_{zz} \\ \sigma^0_{xy} \\ \sigma^0_{xz} \\ \sigma^0_{yz} \end{pmatrix} + \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}\cdot\begin{pmatrix} \sigma^0_{xx} \\ \sigma^0_{yy} \\ \sigma^0_{zz} \\ \sigma^0_{xy} \\ \sigma^0_{xz} \\ \sigma^0_{yz} \end{pmatrix} \\
&= \delta t\left[\begin{pmatrix} K & K & K & 0 & 0 & 0 \\ K & K & K & 0 & 0 & 0 \\ K & K & K & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} + \frac{2\mu\eta}{\mu\delta t+\eta}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{pmatrix}\right]\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} \\
&+ \left[\frac{\eta}{\mu\delta t+\eta}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}\right]\cdot\begin{pmatrix} \sigma^0_{xx} \\ \sigma^0_{yy} \\ \sigma^0_{zz} \\ \sigma^0_{xy} \\ \sigma^0_{xz} \\ \sigma^0_{yz} \end{pmatrix} \\
&= \delta t\begin{pmatrix} K+\frac{4\mu\eta/3}{\mu\delta t+\eta} & K-\frac{2\mu\eta/3}{\mu\delta t+\eta} & K-\frac{2\mu\eta/3}{\mu\delta t+\eta} & 0 & 0 & 0 \\ K-\frac{2\mu\eta/3}{\mu\delta t+\eta} & K+\frac{4\mu\eta/3}{\mu\delta t+\eta} & K-\frac{2\mu\eta/3}{\mu\delta t+\eta} & 0 & 0 & 0 \\ K-\frac{2\mu\eta/3}{\mu\delta t+\eta} & K-\frac{2\mu\eta/3}{\mu\delta t+\eta} & K+\frac{4\mu\eta/3}{\mu\delta t+\eta} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\mu\eta}{\mu\delta t+\eta} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\mu\eta}{\mu\delta t+\eta} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\mu\eta}{\mu\delta t+\eta} \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} \\
&+ \begin{pmatrix} \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & 0 & 0 & 0 \\ \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & 0 & 0 & 0 \\ \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\eta}{\mu\delta t+\eta} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\eta}{\mu\delta t+\eta} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\eta}{\mu\delta t+\eta} \end{pmatrix}\cdot\begin{pmatrix} \sigma^0_{xx} \\ \sigma^0_{yy} \\ \sigma^0_{zz} \\ \sigma^0_{xy} \\ \sigma^0_{xz} \\ \sigma^0_{yz} \end{pmatrix} \qquad (129.4)
\end{aligned}
$$

In Plane strain

We start again from

$$\boldsymbol{\sigma} \quad = \quad K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t\mathbf{1} + \frac{2\mu\eta\delta t}{\mu\delta t+\eta}\dot{\boldsymbol{\varepsilon}}^d + \frac{\eta}{\mu\delta t+\eta}\boldsymbol{\tau}^0 + \sigma_m^0\mathbf{1} \tag{129.5}$$

In plane strain we have $\dot{\varepsilon}_{zz} = \dot{\varepsilon}_{xz} = \dot{\varepsilon}_{yz} = 0$ and $\sigma_{xz} = \sigma_{yz} = 0$ (Note that $\sigma_{zz} \neq 0$!). Furthermore we have[3] the viscous strain rate given by:

$$\dot{\boldsymbol{\varepsilon}}_v = \frac{1}{2\eta}\boldsymbol{\tau} = \frac{1}{2\eta}\left(\boldsymbol{\sigma} - \frac{1}{3}\mathrm{tr}(\boldsymbol{\sigma})\mathbf{1}\right)$$

Looking at the zz component yields

$$\dot{\varepsilon}_{v,zz} = \frac{1}{2\eta}\frac{1}{3}(2\sigma_{zz} - \sigma_{xx} - \sigma_{yy})$$

Assuming $\dot{\varepsilon}_{v,zz} = 0$ (stricly speaking only the total strain $\dot{\varepsilon}_{zz} = 0$ but it is then logical to assume that the zz-strain rate for both mechanisms is zero simultaneously), we obtain $\sigma_{zz} = \frac{1}{2}(\sigma_{xx} + \sigma_{yy})$. In that sense we can discard the σ_{zz} equation since it is not unknown anymore in this case.

Note that then

$$\sigma_m = \frac{1}{3}(\sigma_{xx} + \sigma_{yy} + \sigma_{zz}) = \frac{1}{3}(\sigma_{xx} + \sigma_{yy} + \frac{1}{2}(\sigma_{xx} + \sigma_{yy})) = \frac{1}{2}(\sigma_{xx} + \sigma_{yy})$$

We are left with the σ_{xx}, σ_{yy} and σ_{xy} equations. All we have to do is select the 1st, 2nd and 4th lines and columns of the vectors/matrices in Eq. (129.4):

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = \delta t \begin{pmatrix} K + \frac{4\mu\eta/3}{\mu\delta t+\eta} & K - \frac{2\mu\eta/3}{\mu\delta t+\eta} & 0 \\ K - \frac{2\mu\eta/3}{\mu\delta t+\eta} & K + \frac{4\mu\eta/3}{\mu\delta t+\eta} & 0 \\ 0 & 0 & \frac{\mu\eta}{\mu\delta t+\eta} \end{pmatrix} \cdot \begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ 2\dot{\varepsilon}_{xy} \end{pmatrix} + \begin{pmatrix} \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & 0 \\ \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & 0 \\ 0 & 0 & \frac{\eta}{\mu\delta t+\eta} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx}^0 \\ \sigma_{yy}^0 \\ \sigma_{xy}^0 \end{pmatrix}$$

or,

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = \begin{pmatrix} \frac{\delta t(4\mu\eta+3K\eta+3K\mu\delta t)}{3(\mu\delta t+\eta)} & \frac{\delta t(3K\eta+3K\mu\delta t-2\mu\eta)}{3(\mu\delta t+\eta)} & 0 \\ \frac{\delta t(3K\eta+3K\mu\delta t-2\mu\eta)}{3(\mu\delta t+\eta)} & \frac{\delta t(4\mu\eta+3K\eta+3K\mu\delta t)}{3(\mu\delta t+\eta)} & 0 \\ 0 & 0 & \frac{\mu\eta\delta t}{\mu\delta t+\eta} \end{pmatrix} \cdot \begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ 2\dot{\varepsilon}_{xy} \end{pmatrix} + \begin{pmatrix} \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & 0 \\ \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & 0 \\ 0 & 0 & \frac{\eta}{\mu\delta t+\eta} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{xx}^0 \\ \sigma_{yy}^0 \\ \sigma_{xy}^0 \end{pmatrix} \tag{129.6}$$

In order to compare this with the book's equations, I rewrite the above equation with the book's notations ($\mu \to G$, $\eta \to \mu$):

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{xy} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\delta t(4G\mu+3K\mu+3KG\delta t)}{3(G\delta t+\mu)} & \frac{\delta t(3K\mu+3KG\delta t-2G\mu)}{3(G\delta t+\mu)} & 0 \\ \frac{\delta t(3K\mu+3KG\delta t-2G\mu)}{3(G\delta t+\mu)} & \frac{\delta t(4G\mu+3K\mu+3KG\delta t)}{3(G\delta t+\mu)} & 0 \\ 0 & 0 & \frac{G\mu\delta t}{G\delta t+\mu} \end{pmatrix}}_{\tilde{\boldsymbol{D}}} \cdot \begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ 2\dot{\varepsilon}_{xy} \end{pmatrix} + \underbrace{\begin{pmatrix} \frac{3\mu+G\delta t}{3(G\delta t+\mu)} & \frac{G\delta t}{3(G\delta t+\mu)} & 0 \\ \frac{G\delta t}{3(G\delta t+\mu)} & \frac{3\mu+G\delta t}{3(G\delta t+\mu)} & 0 \\ 0 & 0 & \frac{\mu}{G\delta t+\eta} \end{pmatrix}}_{\tilde{\boldsymbol{D}}_s} \cdot \begin{pmatrix} \sigma \\ \sigma \\ \sigma \end{pmatrix}$$

[3] Since the spring and dashpot are in series, they are under the same stress

as Equation 12.19. Collecting coefficients in front of strain rates ($\dot{\epsilon}$) yields the viscoelastic material matrix

$$\tilde{\mathbf{D}} = \begin{bmatrix} \frac{\Delta t(4G\mu+3\mu K+3\Delta tGK)}{3(\mu+\Delta tG)} & \frac{\Delta t(3\mu K+3\Delta tGK-2G\mu)}{3(\mu+\Delta tG)} & 0 \\ \frac{\Delta t(3\mu K+3\Delta tGK-2G\mu)}{3(\mu+\Delta tG)} & \frac{\Delta t(4G\mu+3\mu K+3\Delta tGK)}{3(\mu+\Delta tG)} & 0 \\ 0 & 0 & \frac{2\mu\Delta tG}{2\mu+\Delta tG} \end{bmatrix} \tag{12.45}$$

while collecting coefficients in front of the old stresses ($\boldsymbol{\sigma}^0$) leads to

$$\mathbf{D}_s = \begin{bmatrix} \frac{3\mu+\Delta tG}{3(\mu+\Delta tG)} & \frac{\Delta tG}{3(\mu+\Delta tG)} & 0 \\ \frac{\Delta tG}{3(\mu+\Delta tG)} & \frac{3\mu+\Delta tG}{3(\mu+\Delta tG)} & 0 \\ 0 & 0 & \frac{2\mu}{(2\mu+\Delta tG)} \end{bmatrix} \tag{12.46}$$

Taken from the book.

A few remarks:

- Good news, I agree with the book's equations except for the (3,3) term which in my opinion is missing a factor 2 in front of the ΔtG of the denominator. I will need an analytical benchmark to sort this out.
- Also I obtain $\vec{\sigma} = \tilde{\boldsymbol{D}} \cdot \dot{\vec{\varepsilon}} + \boldsymbol{D}_s \cdot \vec{\sigma}^0$ which is coherent with Eq. (12.19) of the book but the book has a minus sign in front of $\boldsymbol{D}_s$ on page 188.
- In any case the book is missing a discussion about the 'disappearance' of σ_{yy} after Eq. (12.33). Also Eq. (12.39) should probably showcase a factor 2 in front of the xz term.
- The time derivative in Eq. (12.40) is in fact the co-rotational (Jaumann) time derivative of the deviatoric stress tensor. What is presented here is a simplification, which is ok because a Lagrangian approach is taken?

Using an effective viscosity

Let us recall the following matrices

$$\mathbf{\Lambda} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{\Lambda}^d = \begin{pmatrix} \frac{4}{3} & -\frac{2}{3} & -\frac{2}{3} & 0 & 0 & 0 \\ -\frac{2}{3} & \frac{4}{3} & -\frac{2}{3} & 0 & 0 & 0 \\ -\frac{2}{3} & -\frac{2}{3} & \frac{4}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\tilde{\mathbf{\Lambda}}^d = \begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{\Xi} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

It is common to find in the literature the so-called effective viscosity η_{eff} defined as

$$\eta_{eff} = \frac{\eta\delta t}{\delta t + \eta/\mu} = \frac{\eta}{1 + t_M/\delta t} = \frac{\mu\eta\delta t}{\mu\delta t + \eta}$$

Then

$$\boldsymbol{\sigma} \;=\; K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t\mathbf{1} + 2\eta_{eff}\dot{\boldsymbol{\varepsilon}}^d + \frac{\eta_{eff}}{\mu\delta t}\boldsymbol{\tau}^0 + \sigma_m^0\mathbf{1} \tag{129.7}$$

$$
\begin{aligned}
\vec{\sigma} &= K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t\vec{m} + 2\eta_{eff}\boldsymbol{I}_d\cdot\boldsymbol{C}\cdot\dot{\vec{\varepsilon}} + \frac{\eta_{eff}}{\mu\delta t}\boldsymbol{I}_d\cdot\vec{\sigma}^0 + \sigma_m^0\vec{m} \\
&= \begin{pmatrix} K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t \\ K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t \\ K\mathrm{tr}(\dot{\boldsymbol{\varepsilon}})\delta t \\ 0 \\ 0 \\ 0 \end{pmatrix} + 2\eta_{eff}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} \\
&+ \frac{\eta_{eff}}{\mu\delta t}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} \sigma_{xx}^0 \\ \sigma_{yy}^0 \\ \sigma_{zz}^0 \\ \sigma_{xy}^0 \\ \sigma_{xz}^0 \\ \sigma_{yz}^0 \end{pmatrix} + \sigma_m^0\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \delta t\begin{pmatrix} K & K & K & 0 & 0 & 0 \\ K & K & K & 0 & 0 & 0 \\ K & K & K & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} + 2\eta_{eff}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} \\
&+ \frac{\eta_{eff}}{\mu\delta t}\begin{pmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{3} & 0 & 0 & 0 \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}\cdot\begin{pmatrix} \sigma_{xx}^0 \\ \sigma_{yy}^0 \\ \sigma_{zz}^0 \\ \sigma_{xy}^0 \\ \sigma_{xz}^0 \\ \sigma_{yz}^0 \end{pmatrix} + \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}\cdot\begin{pmatrix} \sigma_{xx}^0 \\ \sigma_{yy}^0 \\ \sigma_{zz}^0 \\ \sigma_{xy}^0 \\ \sigma_{xz}^0 \\ \sigma_{yz}^0 \end{pmatrix} \\
&= \left[K\delta t\boldsymbol{\Xi} + \eta_{eff}\boldsymbol{\Lambda}^d\right]\cdot\begin{pmatrix} \dot{\varepsilon}_{xx} \\ \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{zz} \\ 2\dot{\varepsilon}_{xy} \\ 2\dot{\varepsilon}_{xz} \\ 2\dot{\varepsilon}_{yz} \end{pmatrix} + \left[\frac{\eta_{eff}}{\mu\delta t}\tilde{\boldsymbol{\Lambda}}^d + \frac{1}{3}\boldsymbol{\Xi}\right]\cdot\begin{pmatrix} \sigma_{xx}^0 \\ \sigma_{yy}^0 \\ \sigma_{zz}^0 \\ \sigma_{xy}^0 \\ \sigma_{xz}^0 \\ \sigma_{yz}^0 \end{pmatrix} \qquad (129.8)
\end{aligned}
$$

$$
\boxed{\vec{\sigma} = \left(K\delta t\boldsymbol{\Xi} + \eta_{eff}\boldsymbol{\Lambda}^d\right)\cdot\dot{\vec{\varepsilon}} + \left[\frac{\eta_{eff}}{\mu\delta t}\tilde{\boldsymbol{\Lambda}}^d + \frac{1}{3}\boldsymbol{\Xi}\right]\cdot\vec{\sigma}_0}
$$

We see that this yields much more compact notations than the section above.

End member: infinite viscosity In this case the Maxwell time $t_M = \eta/\mu$ is very large. We have $\eta/\mu >> \delta t$ so that $\eta_{eff} \to \mu\delta t$. This makes sense, infinite viscosity means that the dashpot behaves as a rigid block and its viscosity will not play a role in the deformation and it'll propagate the stress to the spring.

Then the rhs of the equation just above becomes:

$$
\frac{\eta_{eff}}{\mu\delta t}\tilde{\boldsymbol{\Lambda}}^d + \frac{1}{3}\boldsymbol{\Xi} \to \tilde{\boldsymbol{\Lambda}}^d + \frac{1}{3}\boldsymbol{\Xi} = \boldsymbol{I}
$$

Then

$$
\begin{aligned}
\vec{\sigma} &= \left(K\delta t\boldsymbol{\Xi} + \mu\delta t\boldsymbol{\Lambda}^d\right)\cdot\dot{\vec{\varepsilon}} + \vec{\sigma}_0 \\
&= \delta t(K\boldsymbol{\Xi} + \mu\boldsymbol{\Lambda}^d)\cdot\dot{\vec{\varepsilon}} + \vec{\sigma}_0 \qquad (129.9)
\end{aligned}
$$

or,

$$\frac{\vec{\sigma}-\vec{\sigma}_0}{\delta t} = (K\boldsymbol{\Xi} + \mu\boldsymbol{\Lambda}^d)\cdot\dot{\vec{\varepsilon}}$$

which is a discretised form of the time derivative of Eq. (XYZ). It all makes sense, we recover a purely elastic formulation!

End member: infinite shear modulus In this case the Maxwell time is very small and the effective viscosity tends to the real viscosity and also

$$\frac{\eta_{eff}}{\mu\delta t} \to \frac{\eta}{\mu\delta t} \to 0$$

Then

$$\vec{\sigma} = \left(\underbrace{K\delta t}_{\lambda}\boldsymbol{\Xi} + \eta\boldsymbol{\Lambda}^d\right)\cdot\dot{\vec{\varepsilon}} + \vec{\sigma}_0 = \lambda\begin{pmatrix}1&1&1&0&0&0\\1&1&1&0&0&0\\1&1&1&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{pmatrix} + \eta\begin{pmatrix}\frac{4}{3}&-\frac{2}{3}&-\frac{2}{3}&0&0&0\\-\frac{2}{3}&\frac{4}{3}&-\frac{2}{3}&0&0&0\\-\frac{2}{3}&-\frac{2}{3}&\frac{4}{3}&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{pmatrix}\dot{} \quad (129.10)$$

We notice that $\boldsymbol{\Lambda}^d$ is identical to the matrix $\boldsymbol{C}^d$ in (**??**). Also, this formulation looks identical to the standard penalty formulation (see Section **??**), except for the $\vec{\sigma}_0$ term which accounts for the initial stress.

This also probably means that reduced integration can/should be done ?

Implementation details

Following chapter 12 of Guy Simpson's book[4] [298] or following the derivations above we have Eq. (129.6) which I here recall:

$$\begin{pmatrix}\sigma_{xx}\\\sigma_{yy}\\\sigma_{xy}\end{pmatrix} = \begin{pmatrix}\frac{\delta t(4\mu\eta+3K\eta+3K\mu\delta t)}{3(\mu\delta t+\eta)} & \frac{\delta t(3K\eta+3K\mu\delta t-2\mu\eta)}{3(\mu\delta t+\eta)} & 0\\ \frac{\delta t(3K\eta+3K\mu\delta t-2\mu\eta)}{3(\mu\delta t+\eta)} & \frac{\delta t(4\mu\eta+3K\eta+3K\mu\delta t)}{3(\mu\delta t+\eta)} & 0\\ 0 & 0 & \frac{\mu\eta\delta t}{\mu\delta t+\eta}\end{pmatrix}\cdot\begin{pmatrix}\dot{\varepsilon}_{xx}\\\dot{\varepsilon}_{yy}\\2\dot{\varepsilon}_{xy}\end{pmatrix} + \begin{pmatrix}\frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & 0\\ \frac{1}{3}\frac{\mu\delta t}{\mu\delta t+\eta} & \frac{1}{3}\frac{3\eta+\mu\delta t}{\mu\delta t+\eta} & 0\\ 0 & 0 & \frac{\eta}{\mu\delta t+\eta}\end{pmatrix}\cdot\begin{pmatrix}\sigma^0_{xx}\\\sigma^0_{yy}\\\sigma^0_{xy}\end{pmatrix}$$

or,

$$\vec{\sigma} = \tilde{\boldsymbol{D}}\cdot\dot{\vec{\varepsilon}} + \boldsymbol{D}_s\cdot\vec{\sigma}_0$$

Note the green '2' in the equation above: in Simpson's book it is in the lower right term of the matrix. Also, Simpson uses μ for the viscosity while we here use η.

We see that pressure is not a degree of freedom so the finite element formulation and discretisation will not yield a saddle point problem, which is a good thing in terms of complexity and solving procedure.

In the end the following weak form is obtained:

$$\int \boldsymbol{B}^T\cdot\tilde{\boldsymbol{D}}\cdot\boldsymbol{B}dV\cdot\vec{\mathcal{V}} = \int\vec{\mathcal{N}}\rho\vec{g}dV - \int\boldsymbol{B}^T\cdot\boldsymbol{D}_s\cdot\vec{\sigma}_0 dV$$

We need to keep track of the stress $\vec{\sigma}_0$ and we do so directly at the quadrature points. The strain rate vector is recomputed at every time step but it is also directly stored on the quadrature points.

The current stress vector $\vec{\sigma}$ is also on the quadrature points and is computed by means of the equation above.

In the end we declare the following arrays:

```
nqperdim=3
nq=nel*nqperdim**ndim # total number of quadrature points
stress0_vector     = np.zeros((3,nq),dtype=np.float64) # stress vector memory
stress_vector      = np.zeros((3,nq),dtype=np.float64) # stress vector
strainrate_vector = np.zeros((3,nq),dtype=np.float64) # strain rate vector
```

[4] I have slightly changed the notations.

The code relies on quadratic elements (Q_2). Each element is assigned a 'phase' (a material) and each material can have different values of μ, η, ν, K, E.

As is common when modelling elasto-viscous materials we define an effective viscosity:

$$\eta_{eff} = \frac{\eta \delta t}{\delta t + \eta/\mu} = \frac{\eta}{1 + \eta/(\mu \delta t)}$$

Note that if $\mu \to \infty$ (or rather $\mu \delta t >> \eta$) then $\eta_{eff} \to \eta$.

The 'stress memory' rhs is computed as follows:

```
f_el-=b_mat.T.dot(stress_vector[:,counterq])*weightq*jcob
```

The mesh deforms with the material flow (Lagrangian approach) and this is simply implemented as follows

```
xV[:]+=u[:]*dt
yV[:]+=v[:]*dt
```

The mean stress can also be computed and visualised:

$$\sigma_m = \frac{1}{3}(\sigma_{xx} + \sigma_{yy} + \sigma_{zz}) = \frac{1}{3}(\sigma_{xx} + \sigma_{yy} + \frac{1}{2}(\sigma_{xx} + \sigma_{yy})) = \frac{1}{2}(\sigma_{xx} + \sigma_{yy})$$

The matrix $\tilde{\boldsymbol{D}}$ is implemented as follows:

```
di=dt*(3*eta[iel]*K[iel]+3*dt*mu[iel]*K[iel]+4*mu[iel]*eta[iel])
od=dt*(-2*mu[iel]*eta[iel]+3*eta[iel]*K[iel]+3*dt*mu[iel]*K[iel])
d=3*(eta[iel]+dt*mu[iel])
ed=eta[iel]*dt*mu[iel]/(eta[iel]+dt*mu[iel])
Dee = np.array([[di/d, od/d,  0],\
                [od/d, di/d,  0],\
                [0,       0, ed]],dtype=np.float64)
```

while the matrix $\boldsymbol{D}_s$ is implemented as follows

```
di=3*eta[iel]+dt*mu[iel]
od=dt*mu[iel]
ed=eta[iel]/(eta[iel]+dt*mu[iel])
Dees = np.array([[di/d, od/d,  0],\
                 [od/d, di/d,  0],\
                 [0,       0, ed]],dtype=np.float64)
```

In the case of a power-law viscous rheology, we have $\dot{\varepsilon} = A\tau^n$. If we now write $\tau = 2\eta_{eff}\dot{\varepsilon}$, or

$$\eta_{eff} = \frac{1}{2}\frac{\tau}{\dot{\varepsilon}} = \frac{1}{2}\frac{\tau}{A\tau^n} = \frac{1}{2}A^{-1}\tau^{1-n}$$

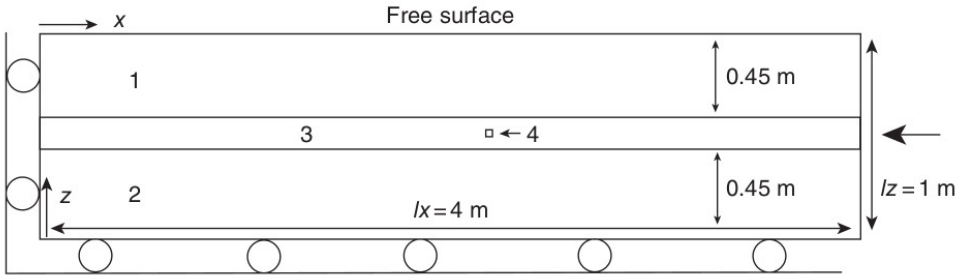
Are there visco-elastic benchmarks ? bepo10 ?

Folding (experiment=1)

Boundary conditions are free slip on the left, right and bottom boundaries, while a horizontal velocity of $5\,\mathrm{mm\,yr^{-1}}$ is prescribed on the right boundary[5].

In the book, the time step is fixed to

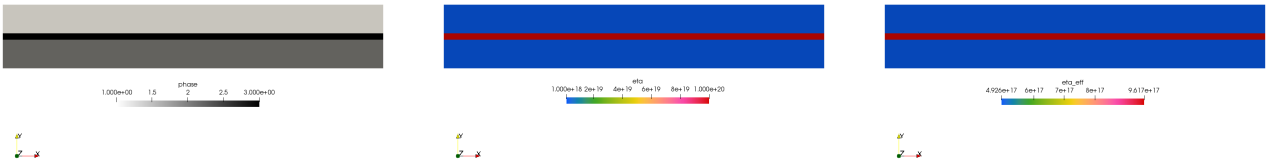
$$\delta t = 0.01/edot = 0.01 * Lx/u_{bc} = 0.01 * 4/5e-3 = 0.8\,\mathrm{yr}$$



Numbers indicate the lithological unit (termed the "phase" in the Matlab script). The folding experiment was performed with a viscoelastic material where the central layer (i.e., phases 3 and 4) has a shear viscosity ($\eta = 1 \times 10^{20}$ Pa s) that is 100 times greater than that of the surrounding matrix (1×10^{18} Pa s). Other parameters are as follows: density = $2700\,\mathrm{kg\,m^{-3}}$, gravity = $9.8\,\mathrm{m\,s^{-2}}$, Young's modulus = 1×10^{11} Pa, Poisson's ratio = 0.3 (all considered uniform throughout), and a 40x40 elements.

The position of the nodes on the interface between the layer and the matrix is perturbed with a random signal:

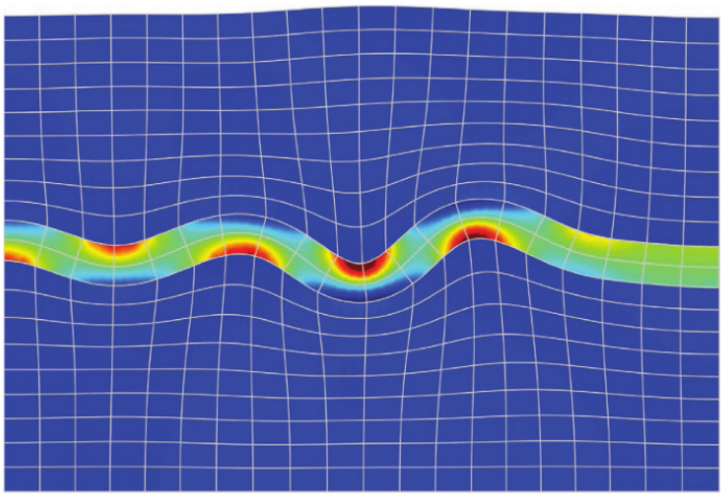
```
for i in range(0,NV):
    if abs(yV[i]-0.45)/Ly<eps:
        yV[i]+=hy*0.05*random.uniform(-1,1)
    if abs(yV[i]-0.55)/Ly<eps:
        yV[i]+=hy*0.05*random.uniform(-1,1)
```



Initial setup of the experiment.

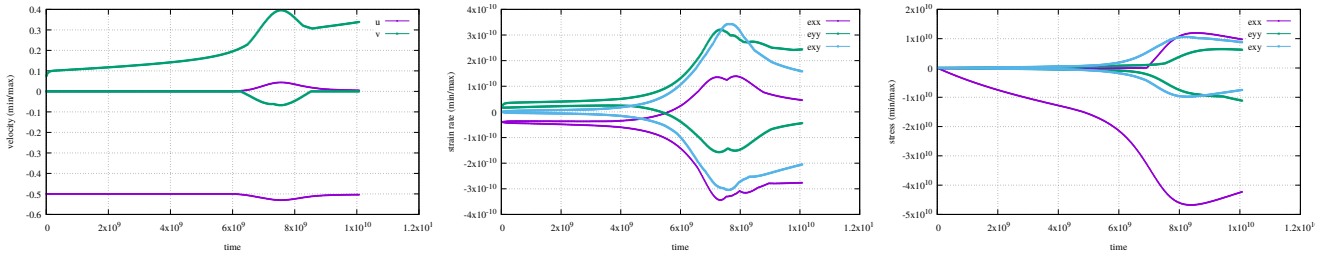
```
viscosity eta    = [1.e+18 1.e+18 1.e+20]
Young modulus E  = [1.e+11 1.e+11 1.e+11]
shear modulus mu = [3.84615385e+10 3.84615385e+10 3.84615385e+10]
bulk modulus K   = [8.33333333e+10 8.33333333e+10 8.33333333e+10]
poisson ratio nu = [0.3 0.3 0.3]
```

[5]The vertical component of the velocity on the right wall is not clear, but looking at the figure we see that the thickening is the same on the left and right, so that the vertical velocity on the right is unconstrained.

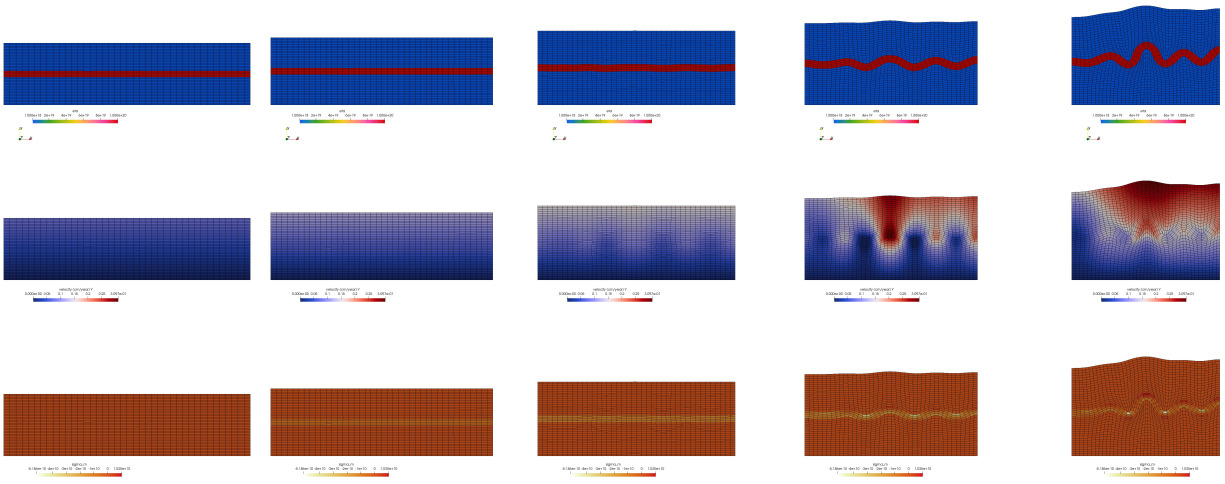


Folding in a layered viscoelastic material after 40% shortening. The central layer has a viscosity 100 higher than the surrounding matrix. The shaded colors represent mean stress, while the grid shows finite deformation (which is not the computational mesh). Unfortunately no color legend.

what about pre-stress, initial compaction?
Remark: - p191, viscosity values in 1) are wrong ? different?



a



this experiment works fine, elastic properties constant in space, only viscous props vary

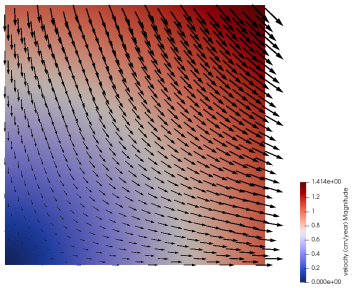
Analytical benchmark - pure shear (experiment=2)

The domain is 50 km × 50 km. There is a single material in the domain characterised by $\eta = 1 \times 10^{21}$ Pa s, $\mu = 1 \times 10^{10}$ Pa. The time step is $\delta t = 100$ yr. Boundary conditions are free slip on all sides, with $u = +1\,\mathrm{cm\,yr^{-1}}$ on the right and $v = -1\,\mathrm{cm\,yr^{-1}}$ on the top.

The unstressed, *incompressible* viscoelastic Maxwell medium is subjected to a velocity field resulting in pure shear. The increase of the accumulated stress with time is given by an analytical solution:

$$\boldsymbol{\tau} = 2\eta\ \dot{\boldsymbol{\varepsilon}} \left(1 - \exp\left(-\frac{\mu}{\eta} t\right)\right) = 2\eta\ \dot{\boldsymbol{\varepsilon}} \left(1 - \exp\left(-\frac{t}{t_M}\right)\right) \tag{129.11}$$

where the Maxwell time is $t_M = \frac{\eta}{\mu} = 1 \times 10^{11}$ s $\simeq 3171$ yr.

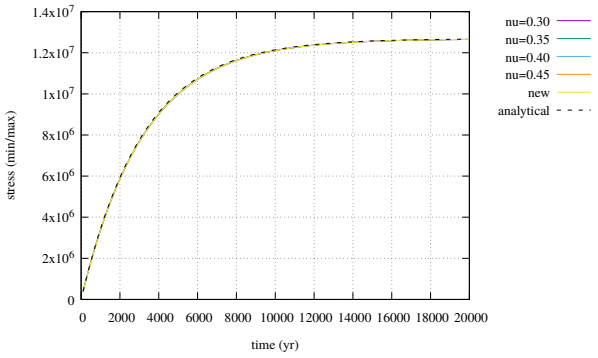


Velocity field.

We have

$$\eta_{eff} = \frac{\eta \delta t}{\delta t + \eta/\mu} = \frac{10^{21} \cdot 3.154 \times 10^{9}}{3.154 \times 10^{9} + 10^{21}/10^{10}} \simeq 3.0592 \times 10^{19}\,\mathrm{Pa\,s}$$

The resolution is set to 10 × 10 elements. 200 time steps are carried out.



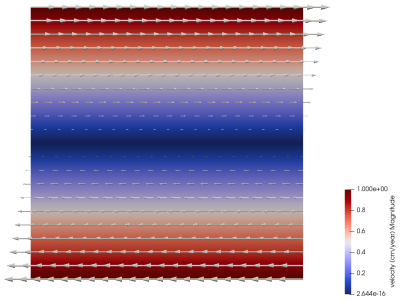
Evolution or σ_{xx} statistics over 20 kyr for various values of ν.

This benchmark is also carried out in Gerya [129]. The solution is also used in Kaus and Becker [179].

Analytical benchmark (experiment=7)

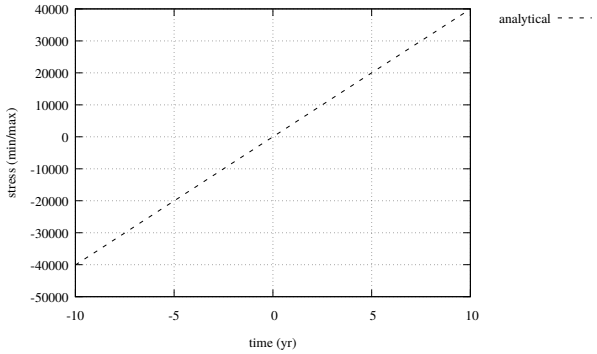
This is also a very simple setup. Gravity is set to zero. The domain is 50x50km, $\eta = 10^{21}$, $\mu = 10^{10}$.

Boundary conditions are $v = 0$ on the left and right sides, $\vec{\mathbf{v}} = (-1, 0)$ cm yr^{-1} at the bottom and $\vec{\mathbf{v}} = (+1, 0)$ cm yr^{-1} at the top.



In that case the strain rate tensor is given by

$$\dot{\boldsymbol{\varepsilon}} = \begin{pmatrix} \dot{\varepsilon}_{xx} & \dot{\varepsilon}_{xy} \\ \dot{\varepsilon}_{yx} & \dot{\varepsilon}_{yy} \end{pmatrix} = \begin{pmatrix} \partial_x u & \frac{1}{2}(\partial_x v + \partial_y u) \\ \frac{1}{2}(\partial_x v + \partial_y u) & \partial_y v \end{pmatrix} = \begin{pmatrix} 0 & \frac{v_0}{L_y} \\ \frac{v_0}{L_y} & 0 \end{pmatrix} = \begin{pmatrix} 0 & \dot{\varepsilon}_0 \\ \dot{\varepsilon}_0 & 0 \end{pmatrix}$$



Evolution or σ_{xx} statistics over 20 kyr for various values of ν.

Analytical benchmark (experiment=6) "compacting aquarium"

This is an extremely simple setup. The domain contains a single material. Boundary conditions are free slip on left, right and bottom, stress-free boundary conditions at the top.

The domain is 50x50km, gravity is $\vec{g} = -9.81\ \mathrm{m\,s^{-2}}$, $\eta = 10^{21}$, $\rho = 3300$, $\mu = 10^{10}$.

We expect the system to compact under its own weight and reach a steady state. FIND OUT !!

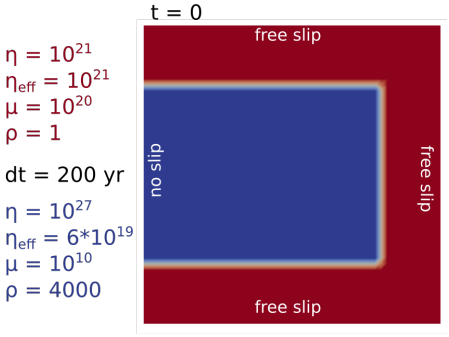
Given the setup, this is actually a 1D problem. Indeed we expect $u = 0$, so that $\partial_x u = \partial_y u = 0$ and also we expect v to be independent of x, so that $\partial_x v = 0$, so

$$\dot{\boldsymbol{\varepsilon}} = \begin{pmatrix} \dot{\varepsilon}_{xx} & \dot{\varepsilon}_{xy} \\ \dot{\varepsilon}_{yx} & \dot{\varepsilon}_{yy} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & \partial_y v \end{pmatrix}$$

Numerical benchmark - Bending of elastic slab (experiment=3)

This benchmark is a numerical benchmark, but not an analytical one. It originates in Taras Gerya's book [129]. It is originally for incompressible visco-elastic fluids so we will run it at different values of the Poisson ratio parameter. It is also carried out in STONE 64.

The sinking slab benchmark consists of a beam of elastic material which is placed in a weak and viscous surrounding medium. The initially unstressed beam is attached to the left domain boundary through boundary conditions. A stress is then applied to the beam in the form of gravity. The applied gravity force results in the deformation of the beam through bending. After 20 kyr, the gravity field is turned off and the elastic properties of the beam will then force itself to its original position. The set-up of the benchmark is given in the following figure:

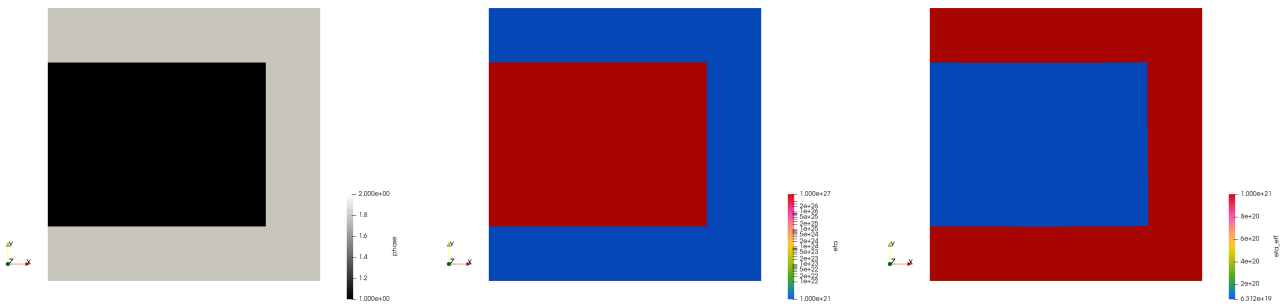


Set-up of the benchmark from [128]. The properties of the two materials are given on the left, together with the initial configuration of the benchmark.

The 800×600 km beam is surrounded by a low-density, low-viscosity and high shear modulus medium of which the specifications are given in the following table. The boundary conditions of the domain consist of a no slip condition at the left boundary where the slab is attached and free slip boundary conditions along all other sides. The results are calculated on a grid with a resolution of 50x50 elements. The time step is set to $\delta t = 200$ yr (i.e. gravity is switched off after 100 time steps).

As explained in the book: " The slab deformation is purely elastic due to the large Maxwell time (3 170 000 kyr) of slab material compared to the total deformation time (20 000 kyr). In contrast, the low viscosity medium is subjected to irreversible, purely viscous deformation since its Maxwell time (3.17×10^{-10} kyr) is negligible compared to the deformation time."

Material properties	*Elastic slab (fluid 1)*	*Surrounding medium (fluid 2)*
Density ρ [kg/m^3]	4000	1
Viscosity η [Pa s]	10^{27}	10^{21}
Shear modulus μ [Pa]	10^{10}	10^{20}
Maxwell time t_M [yr]	3.17Gyr	3.17×10^{-7}yr
eff. visc. η_{eff} [Pa s]	6.307199602192306e+19	9.999999984145105e+20





Initial setup of the experiment.

Unfortunately this simulation explodes no matter the poisson ratio value. It kinda looks like things go mostly wrong in the purely viscous part?

Here is a tentative analysis of what is going on. Let us look at the $\tilde{\boldsymbol{D}}$ for both phases (phase 1 is the slab -purely elastic, phase 2 is the fluid -purely viscous):

$$\tilde{\boldsymbol{D}}_{phase=1} = \begin{pmatrix} 2.003e+20 & 7.409e+19 & 0 \\ 7.409e+19 & 2.003e+20 & 0 \\ 0 & 0 & 6.311e+19 \end{pmatrix} \qquad \tilde{\boldsymbol{D}}_{phase=2} = \begin{pmatrix} 1.161e+30 & 1.161e+30 & 0 \\ 1.161e+30 & 1.161e+30 & 0 \\ 0 & 0 & 9.999e+20 \end{pmatrix}$$

$$\boldsymbol{D}_{s,phase=1} = \begin{pmatrix} 0.999 & 2.103e-08 & 0 \\ 2.103e-08 & 0.999 & 0 \\ 0 & 0 & 0.999 \end{pmatrix} \qquad \boldsymbol{D}_{s,phase=2} = \begin{pmatrix} 0.333 & 0.333 & 0 \\ 0.333 & 0.333 & 0 \\ 0 & 0 & 1.584e-09 \end{pmatrix}$$

In this case we see that we have more than 10 orders of magnitude of difference between the coefficients of the $\tilde{\boldsymbol{D}}_s$ matrices. This is obviously bad news and is likely the cause of the problems experienced.

In order to insure that the fluid is behaving viscously, all we need to have is $\mu\delta t >> \eta$. In this case, we have $\mu\delta t = 10^{20} \cdot 200 \cdot 3600 \cdot 365 \cdot 24 \simeq 6.3 \times 10^{29}$ while $\eta = 10^{21}$ and we have seen that the Maxwell time is then 3.17×10^{-7} yr.

However one could argue that $\mu\delta t/\eta \simeq 6.3 \times 10^{8}$ is overkill. Let us then impose $\mu\delta t/\eta \simeq 6.3 \times 10^{3}$, i.e. let us take $\mu = 10^{15}$.

The stress fields look smoother, but there are still rather wild oscillations!!!

is there a bug or is this a feature of this formulation? I NEED BENCHMARKS!!!

Numerical benchmark - Flexure of elastic plate (experiment=4)

It is described in Section **??**.

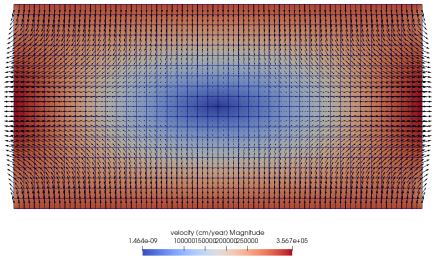


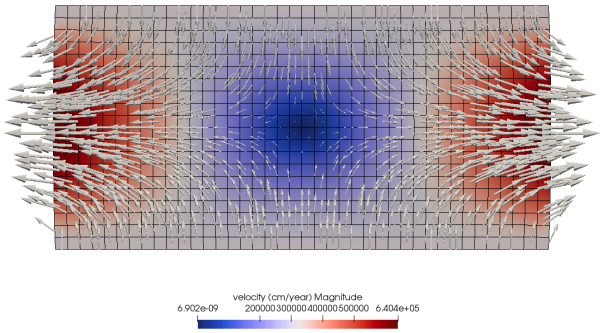
Initial setup of the experiment.

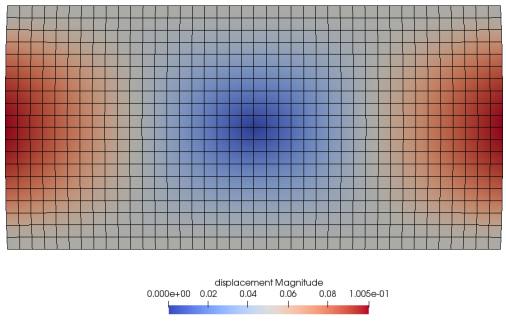
This explodes when shear modulus contrast becomes too big, i.e. when a material essentially becomes purely viscous by having its μ very large. It showcases oscillations at the beginning too!

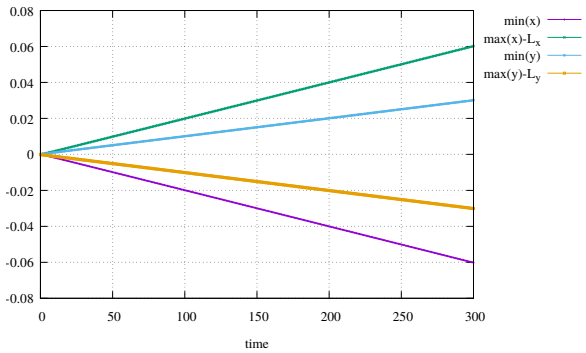
Numerical benchmark - Parallel-plate viscosimeter (experiment=5)

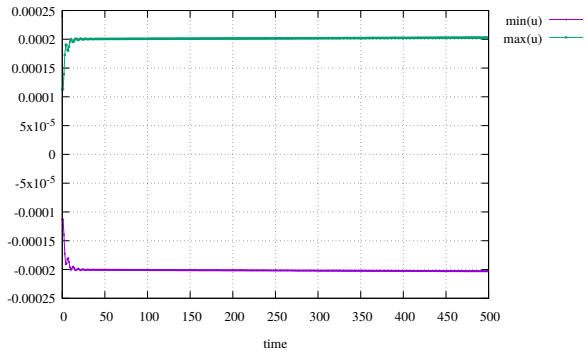
It is described in Section **??**.











Note the oscillations!!

TODO: Look at Seok-Hyeon Do and Byung-Dal So. "Finite element modeling for 2D viscoelastic buckling formation with mechanical strength of mantle and lithosphere". In: *J. Geol. Soc. Korea* 55.3 (2019), pp. 315–332. DOI: 10.14770/jgsk.2019.55.3.315 and redo it ?

Stone 130

Reaction-diffusion (Q_1) - Simpson book chapter 7



python_codes/fieldstone_130/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_130`

This is based on Simpson's book [298], chapter 7.3.

As an introduction to the topic of how the FEM can be performed on systems of equations, we consider the following equations

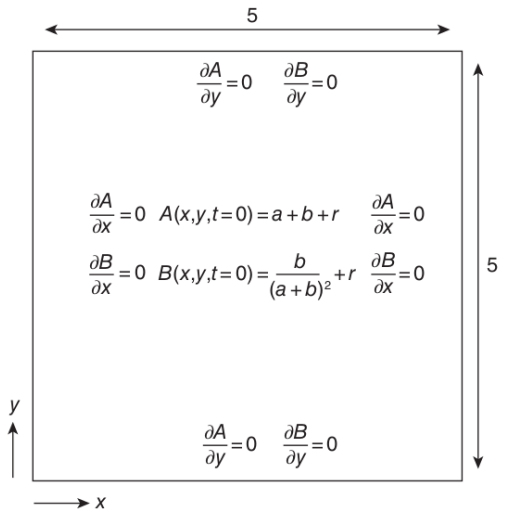
$$\frac{\partial A}{\partial t} = \Delta A + \gamma(a - A + A^2 B) \tag{130.1}$$

$$\frac{\partial B}{\partial t} = d\Delta B + \gamma(b - A^2 B) \tag{130.2}$$

with the two unknowns, A and B. This system of equations is used to model the interaction between two interacting chemicals A and B [222].

B represents the concentration of a "substrate" (inhibitor) chemical that is consumed in a reaction by some chemical "activator", with concentration A. The reaction produces the activator, which explains the A^2B terms in both equations. Both substances are also produced at some background rate (γa for A and γb for B, respectively) and A decays with first-order kinetics at a rate γ.

This "substrate depletion" reaction model is known as Schnakenberg kinetics [131, 284]. This model is most widely applied in biology, but it might also be relevant in Earth science to explain the formation of self-organizing patterns, for example, related to mineral growth, stromatolites and corals, concretions, and so on.



The domain is a square of size 5. At $t = 0$ the fields are given by

$$\begin{aligned} A(x,y) &= a+b+r \\ B(x,y) &= \frac{b}{(a+b)^2}+r \end{aligned} \tag{130.3}$$

which is the steady-state solution with a small random perturbation r (that has a maximum amplitude of 1/100). No flux (zero gradient) conditions are considered across all lateral boundaries.

In the book we find:

```
lx = 5 ; % length of x domain
ly = 5 ; % length of y domain
d = 20 ; % diffusivity of species B
a = 0.05 ; % growth rate of species A
b = 1 ; % growth rate of species B
gamma = 600 ; % kinetics
amp = 0.01 ; % max. amplitude of random noise
ntime = 200 ; % total number of time steps to compute
nxe = 50 ; % number of elements in x-direction
nye = 50 ; % number of elements in y-direction
dt = 5e-4 ; % time step
```

The coupled PDEs are nonlinear: they both contain the term A^2B. As such a specific algorithm must be employed so as to make sure that the equations are adequately solved. Picard iterations are then used. Since the matrix does not change (only the rhs) we will be making use of this fact.

$$\begin{aligned} \int \vec{\mathcal{N}}^T\vec{\mathcal{N}}dV\cdot\frac{\partial\vec{\mathcal{A}}}{\partial t}+\int \boldsymbol{B}^T\cdot\boldsymbol{B}dV\cdot\vec{\mathcal{A}} &= \int \vec{\mathcal{N}}^T\gamma a dV-\gamma\int \vec{\mathcal{N}}^T\vec{\mathcal{N}}dV\cdot\vec{\mathcal{A}}+\int \vec{\mathcal{N}}^T\gamma A^2BdV \\ \int \vec{\mathcal{N}}^T\vec{\mathcal{N}}dV\cdot\frac{\partial\vec{\mathcal{B}}}{\partial t}+\int d\boldsymbol{B}^T\cdot\boldsymbol{B}dV\cdot\vec{\mathcal{B}} &= \int \vec{\mathcal{N}}^T\gamma b dV-\int \vec{\mathcal{N}}^T\gamma A^2BdV \end{aligned} \tag{130.4}$$

We define the mass matrix $\mathbb{M}$ and the diffusion matrix $\mathbb{K}_D$ as follows:

$$\mathbb{M}=\int \vec{\mathcal{N}}^T\vec{\mathcal{N}}\,dV \qquad \mathbb{K}_D=\int \boldsymbol{B}^T\cdot\boldsymbol{B}\,dV$$

so that we have (note that the a, b, d and γ coefficients are constant in space so we can take them out of the integrals)

$$\begin{aligned} \mathbb{M}\cdot\frac{\partial\vec{\mathcal{A}}}{\partial t}+\mathbb{K}_D\cdot\vec{\mathcal{A}} &= \int \vec{\mathcal{N}}^T\gamma a dV-\gamma\int \vec{\mathcal{N}}^T\vec{\mathcal{N}}dV\cdot\vec{\mathcal{A}}+\int \vec{\mathcal{N}}^TA^2BdV \\ \mathbb{M}\cdot\frac{\partial\vec{\mathcal{B}}}{\partial t}+d\,\mathbb{K}_D\cdot\vec{\mathcal{B}} &= \int \vec{\mathcal{N}}^T\gamma b dV-\gamma\int \vec{\mathcal{N}}^TA^2BdV \end{aligned} \tag{130.5}$$

We approximate the time derivative by a simple forward step:

$$\frac{\partial \vec{\mathcal{A}}}{\partial t} \simeq \frac{\vec{\mathcal{A}}^n - \vec{\mathcal{A}}^{n-1}}{\delta t}$$

so that we have

$$\begin{aligned}
\mathbb{M} \cdot \frac{\vec{\mathcal{A}}^n - \vec{\mathcal{A}}^{n-1}}{\delta t} + \mathbb{K}_D \cdot \vec{\mathcal{A}}^n &= \gamma a \int \vec{\mathcal{N}}^T \, dV - \gamma \underbrace{\int \vec{\mathcal{N}}^T \vec{\mathcal{N}} \, dV}_{\mathbb{M}} \cdot \vec{\mathcal{A}}^n + \gamma \int \vec{\mathcal{N}}^T (A^n)^2 B^n \, dV \\
\mathbb{M} \cdot \frac{\vec{\mathcal{B}}^n - \vec{\mathcal{B}}^{n-1}}{\delta t} + d\mathbb{K}_D \cdot \vec{\mathcal{B}}^n &= \gamma b \int \vec{\mathcal{N}}^T \, dV - \gamma \int \vec{\mathcal{N}}^T (A^n)^2 B^n \, dV
\end{aligned} \tag{130.6}$$

or, after multiplying both equations by δt,

$$\begin{aligned}
\underbrace{((1+\gamma\delta t)\mathbb{M} + \mathbb{K}_D \delta t)}_{\mathbb{K}_A} \cdot \vec{\mathcal{A}}^n &= \underbrace{\delta t \gamma a \int \vec{\mathcal{N}}^T dV + \delta t \int \vec{\mathcal{N}}^T (A^n)^2 B^n dV + \mathbb{M} \cdot \vec{\mathcal{A}}^{n-1}}_{\vec{f}_A} \\
\underbrace{(\mathbb{M} + d\mathbb{K}_D \delta t)}_{\mathbb{K}_B} \cdot \vec{\mathcal{B}}^n &= \underbrace{\delta t \gamma b \int \vec{\mathcal{N}}^T dV - \delta t \int \vec{\mathcal{N}}^T (A^n)^2 B^n dV + \mathbb{M} \cdot \vec{\mathcal{B}}^{n-1}}_{\vec{f}_B}
\end{aligned} \tag{130.7}$$

Note that the rhs vectors depend on An, Bn and ...

We assume $\vec{\mathcal{A}}^{n-1}$ and $\vec{\mathcal{B}}^{n-1}$ known.

Inside the integrals, we have

$$A^n = \vec{\mathcal{N}} \cdot \vec{\mathcal{A}}^n \qquad B^n = \vec{\mathcal{N}} \cdot \vec{\mathcal{B}}^n$$

Simple approach

The FE matrix structure is as follows:

$$\begin{pmatrix} \mathbb{K}_A & 0 \\ 0 & \mathbb{K}_B \end{pmatrix} \cdot \begin{pmatrix} \vec{\mathcal{A}} \\ \vec{\mathcal{B}} \end{pmatrix} = \begin{pmatrix} \vec{f}_A \\ \vec{f}_B \end{pmatrix}$$

Different approach

The rhs of the first equation contains the term $\gamma A^2 B$ so that we have to compute

$$\gamma \int \vec{\mathcal{N}}^T A^2 B dV$$

In the previous approach we take A and B as obtained from the previous nonlinear iteration. However, we could do things a little bit differently by only taking A from the previous iteration (so A_{k-1}) while keeping B_k.

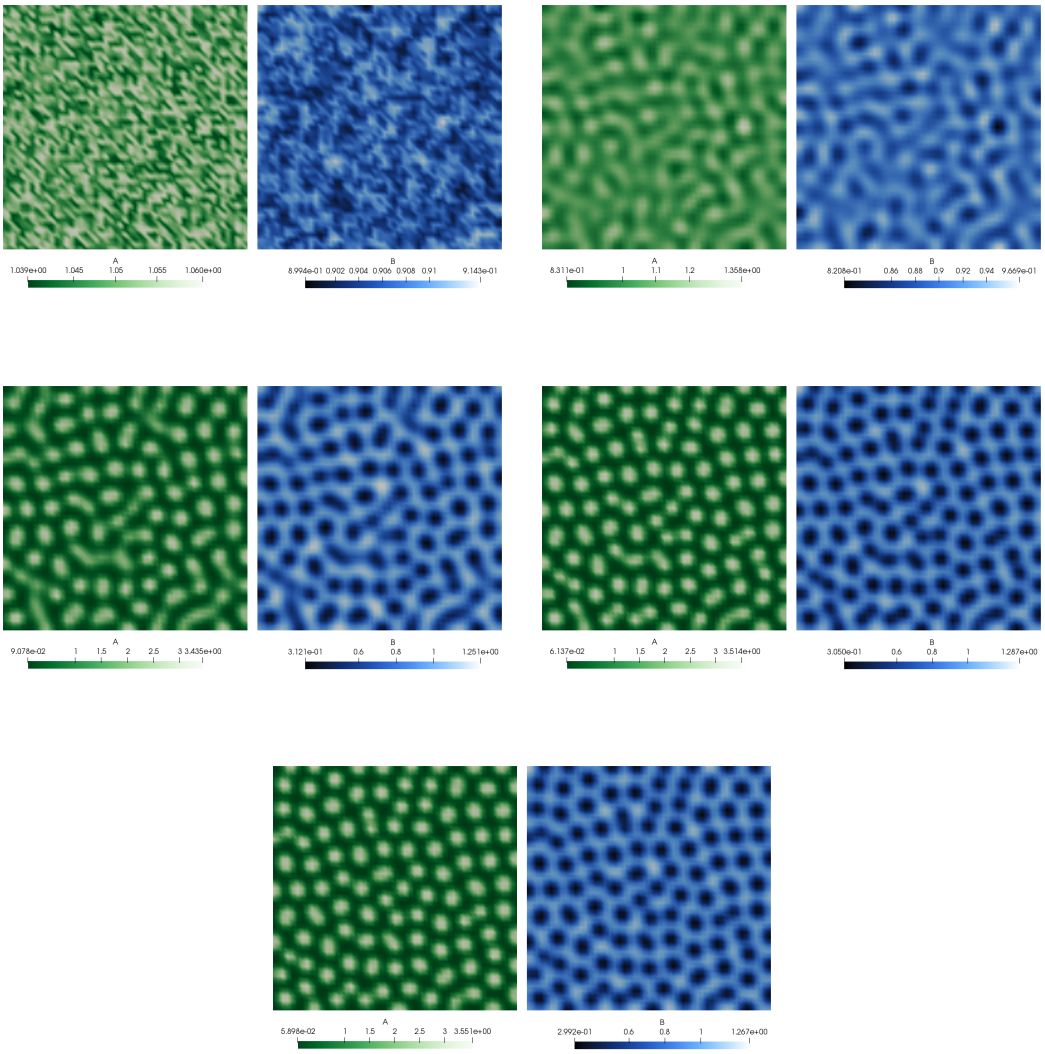
$$\mathbb{M}_A \cdot \vec{\mathcal{B}}_k^n = \int A_{k-1}^2 \vec{\mathcal{N}}^T \vec{\mathcal{N}} dV \cdot \vec{B}_k^n$$

The FE matrix structure is then as follows:

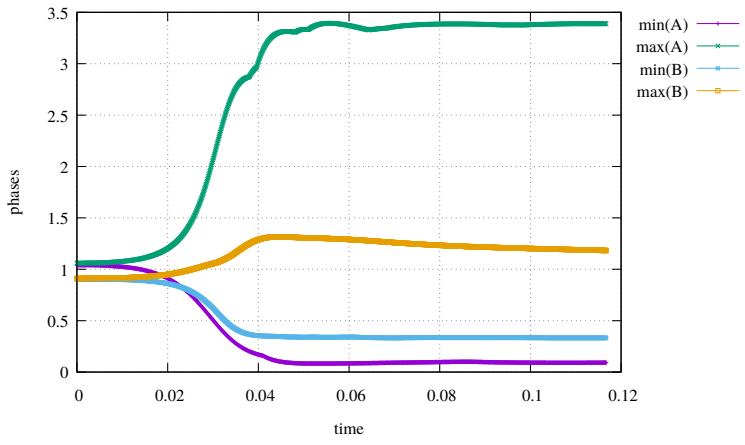
$$\begin{pmatrix} \mathbb{K}_A & \mathbb{M}_A \\ 0 & \mathbb{K}_B \end{pmatrix} \cdot \begin{pmatrix} \vec{\mathcal{A}}_k \\ \vec{\mathcal{B}}_k \end{pmatrix} = \begin{pmatrix} \vec{f}_A \\ \vec{f}_B \end{pmatrix}$$

Results

This is a rough take. I need nonlinear iterations to take care of the nonlinearities introduced by the rhs I need to explaore whether part of the rhs could not make its way in the matrix. Quid of the nl convergence ? This is 1st order implicit in time. Crank-Nicolson ? Check analytical solution talk about implementation of bc



Phases A and B at time steps 0,200,400,600,800.



Stone 131

creating triangular meshes with python



python_codes/fieldstone_131/text.tex

P Codes at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_131`

This stone was developed in collaboration with M. Blasweiler and J. Wolbers.

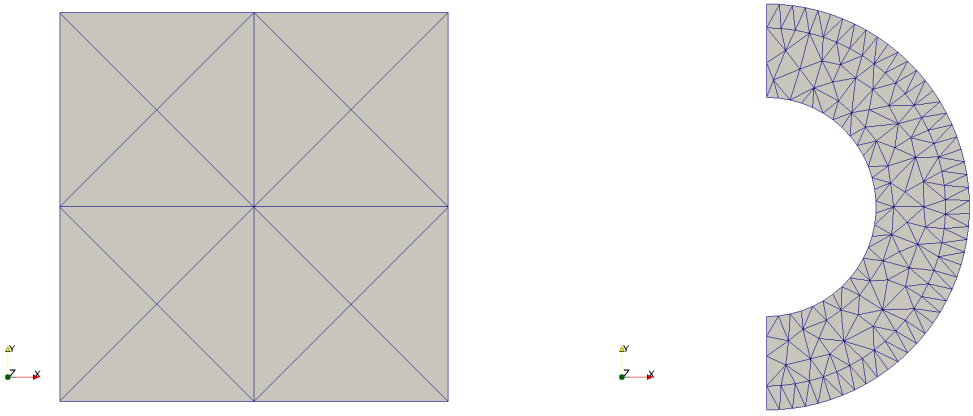
In previous stones we have used the `triangle` code `https://www.cs.cmu.edu/~quake/triangle.html`. Although very fast and very stable, this code is written in C and the interface with python is therefore not optimal. Typically one would generate the nodes on the convex hull and inner material boundaries, write these into a file, have `triangle` read these in, return the generated mesh into a file, and read the latter in the FEM python code.

However, there is a python module which is capable of generating Delaunay triangulations like `triangle` and we here explore its use. In order to install it:

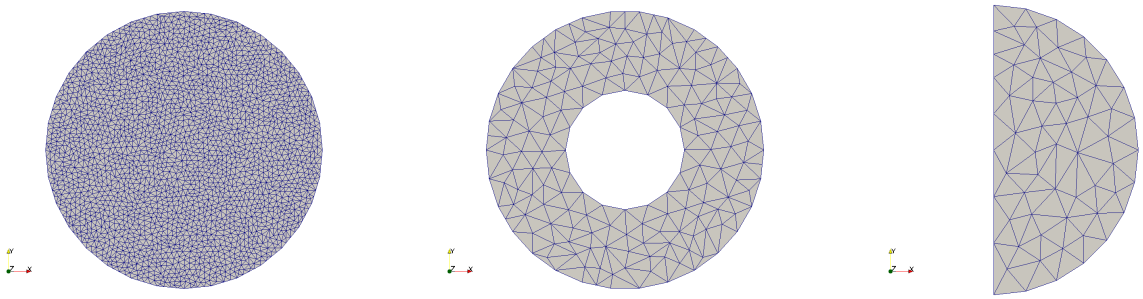
```
pip3 install triangle
```

`https://rufat.be/triangle/delaunay.html`

In the stone.py you will find five examples illustrating the various uses of this library. The code also provides an export to vtu file.



Example 1 (left), example 2 (right)



Example 3 (left), example 4 (middle), example 5 (right)

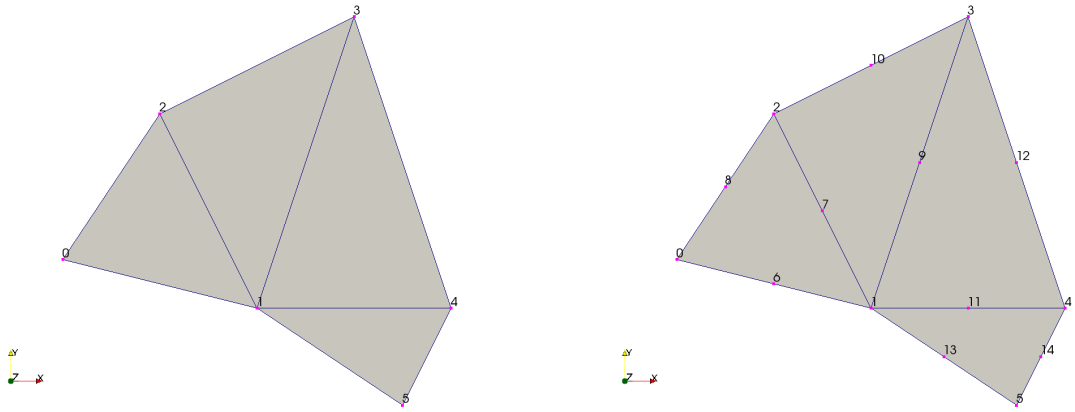
From linear triangles to quadratic ones

In the case quadratic triangular elements are to be used a conversion $P_1 \to P_2$ is provided.

Two methods are implemented. The first one is naive and therefore not efficient at all: It essentially creates all mid-edges points for all triangles and then looks for those that are situated at the same coordinates and remove doubles. The second one generates the mid-edge nodes progressively and checks that they do not exist yet (by means of a connectivity matrix which registers edges - stored as vertices pairs).

The file stone2.py contains multiple functions:

- export_elements_to_vtuP1
- export_elements_to_vtuP2
- mesh_P1_to_P2_naive
- mesh_P1_to_P2



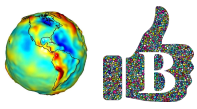
P_1 mesh (left), P_2 mesh (right)

In the case of a full disc, we vary the number of triangles and find that the second algorithm strongly outperforms the first one:

	'qa0.003'	'qa0.002'	'qa0.001'	'qa0.0005'
nel	1638	2408	4813	9677
P1→P2 naive	27.4s	53.8s	216.6s	958.5s
P1→P2 smart	0.071s	0.099s	0.195s	0.397s

Stone 132

gravity field of a mass circle

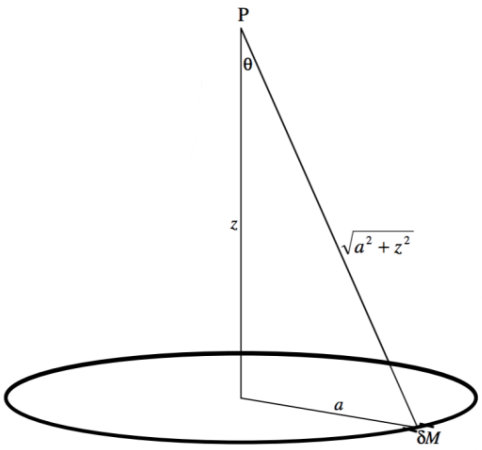


python_codes/fieldstone_132/text.tex

gravity

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_132

Let us consider a circle of mass M in the (x,y)-plane of radius R centered at the origin. Let us consider a point P of coordinates (x_P, y_P, z_P) at which we compute the gravity field generated by the circle:



Redo FIGURE

We are considering a small element of the ring of mass δm (with $\oint dm = M$). The contribution of this element to the field is

$$\delta\vec{g} = \frac{\mathcal{G}\delta m}{d^2}\vec{e}_{MP}$$

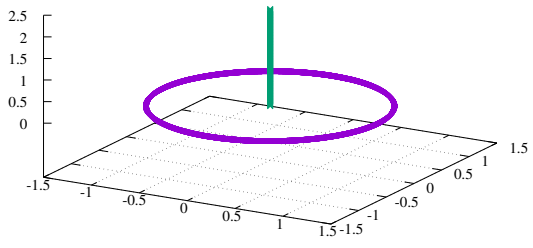
In what follow we arbitrarily set the linear density to $\rho = 10^6$ kg m^{-1} and the radius of the circle to $R = 1.123$

Case 1: P is above the center of the circle

In this case:

$$d = \sqrt{R^2 + z_P^2}$$

and because of symmetry we know that the resulting gravity components g_x and g_y will be zero.



For the mass δm the contribution on the z-axis is given by

$$\delta g_z = \frac{\mathcal{G}\delta m}{R^2 + z_P^2} \cos\theta$$

We see that

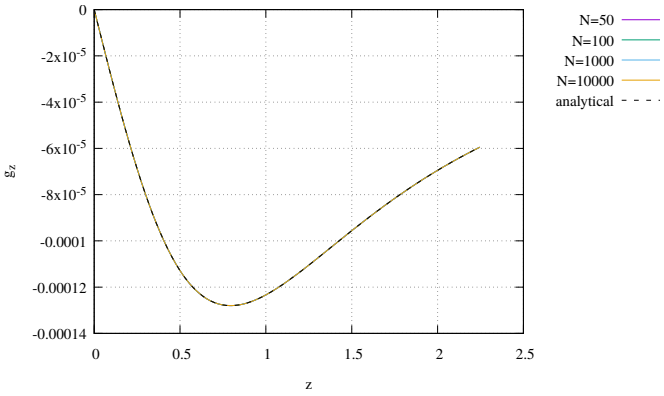
$$\cos\theta = \frac{z_P}{\sqrt{R^2 + z_P^2}}$$

so that

$$\delta g_z = \frac{\mathcal{G}\delta m}{R^2 + z_P^2} \frac{z_P}{\sqrt{R^2 + z_P^2}}$$

By integrating this expression for the entire circle we find[1]:

$$g_z = \frac{\mathcal{G}M z_P}{(R^2 + z_P^2)^{3/2}}$$

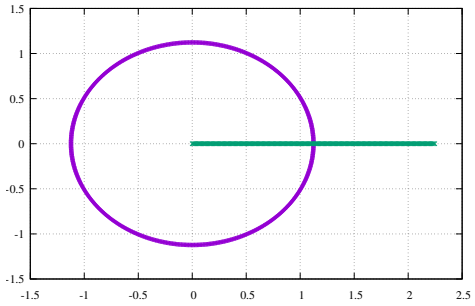


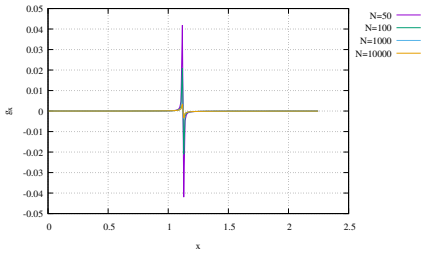
Vertical component of the gravity field as a function of z, for various circle resolution, plotted against the analytical solution.

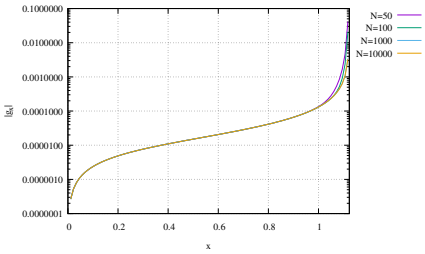
[1] See also video `https://www.youtube.com/watch?v=q5EWLNdv_pE`

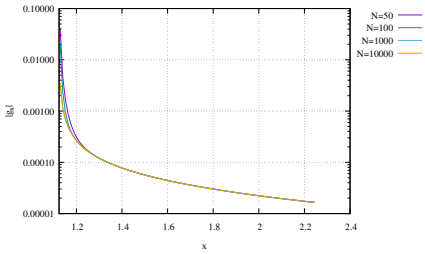
Case 2: P is in the plane circle

Unfortunately I was not able to find or derive an analytical solution for this case. Gravity is measured on a line starting at the center of the circle, following the x-axis and up to $x = 2R$:









Stone 133

The annulus benchmark ($Q_2 \times Q_1$) with Doube Jacobian



python_codes/fieldstone_133/text.tex

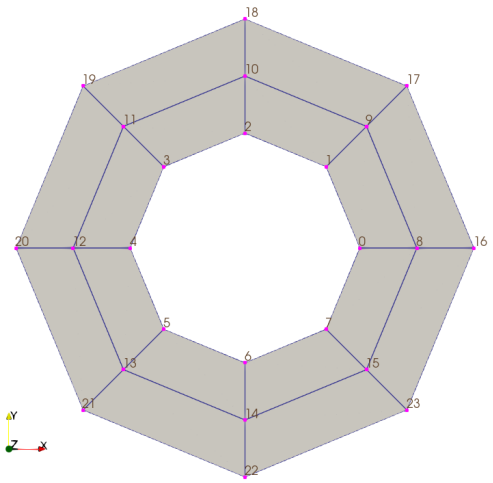
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_133`

The benchmark is described fully in Section 19.18. The following results have been obtained with $k = 4$. Taylor-Hood $Q_2 \times Q_1$ elements are used with an isoparametric mapping. This STONE is based on STONE 21 (please refer to this one first before reading any further).

The idea here is to test the Double Jacobian algorithm presented in Morgan, Taramón, and Hasenclever [237] (2020) as thoroughly described and worked out in Section 11.8.

In the code this can be toggled on/off by means of the DJ boolean. Also, the internal numbering of nodes has been completely reworked with regards to the one used in STONE 21 to match best with the theory notations.

The node layout is as follows (in this case nelr=1, nelt=4, so four Q_2 elements:



with

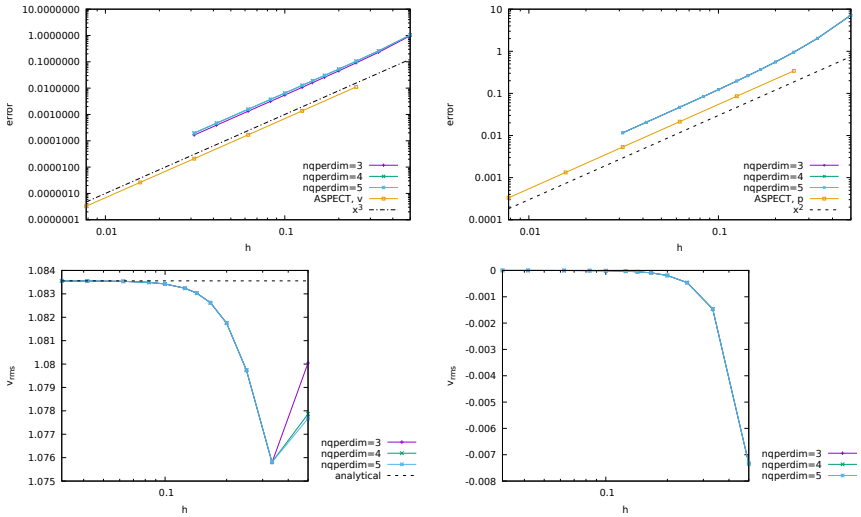
```
nelr= 1
nelt= 4
nel= 4
NfemV= 48
NfemP= 8
0 | [ 0 16 18  2  8 17 10  1  9]
1 | [ 2 18 20  4 10 19 12  3 11]
2 | [ 4 20 22  6 12 21 14  5 13]
3 | [ 6 22 16  0 14 23  8  7 15]
```

As mentioned in the paper, one of the drawbacks of the double Jacobian is the fact that the terms to be integrated are no more polynomials, so that it is not straightforward to determine the necessary number of quadrature points. I have introduced the nqperdim parameter which allows to control the number of quadrature points per dimension. We will be testing 3,4,5,6.

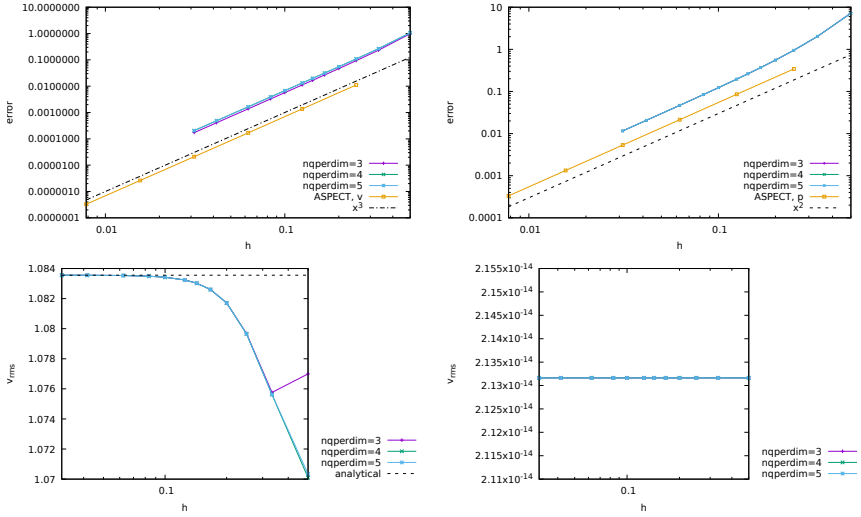
Results

benchmark 1

Using isoparametric mapping, we obtain the following results. As documented in STONE 21, we recover a 3rd order convergence for the velocity error and a 2nd order convergence for the pressure error.



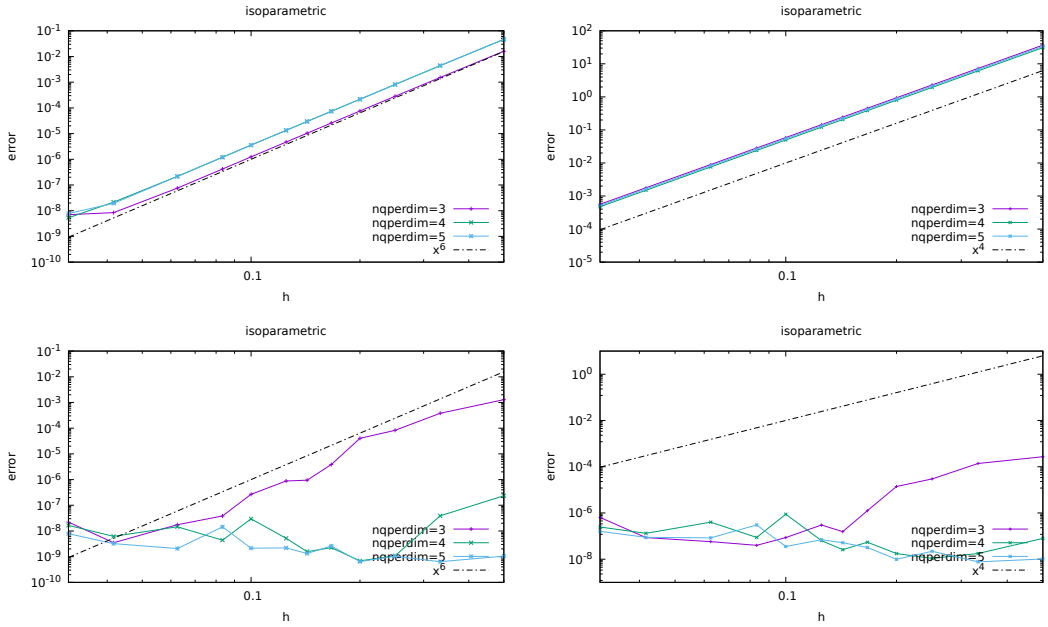
Now switching to DJ:



These results are somewhat disappointing.

benchmark 2

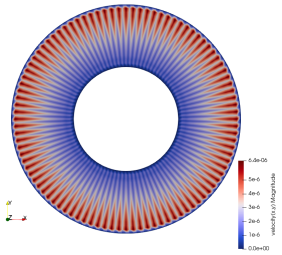
Let us now turn to the aquarium benchmark. Gravity is 1, density is 10^6 and no-slip boundary conditions are imposed. The solution is a zero velocity and a lithostatic pressure.



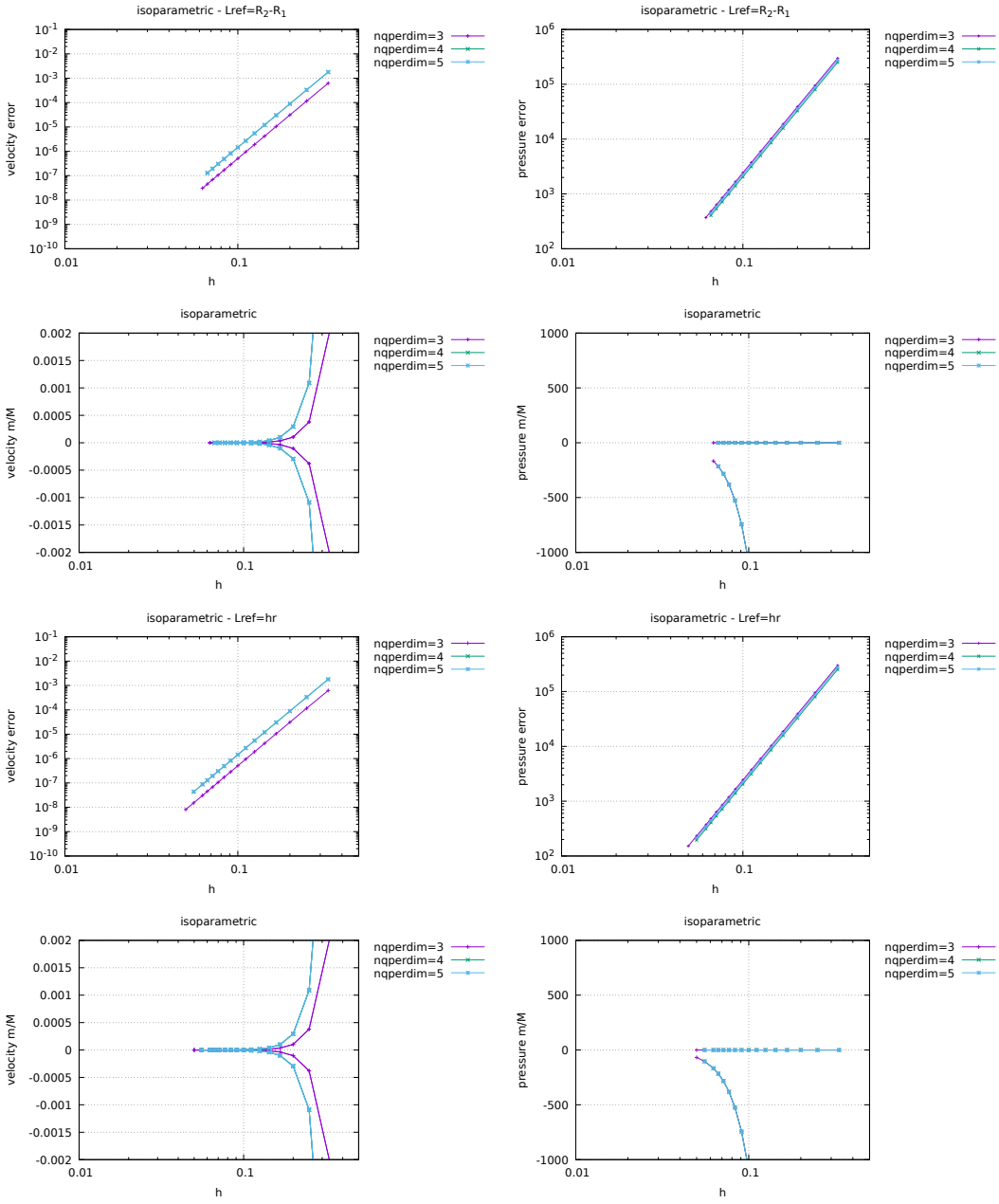
benchmark 3

Same idea as benchmark 2, but now it is Earth-like: $R_1 = 2890$ km, $R_2 = 6370$ km, $\rho = 4000\ \mathrm{kg\,m^{-3}}$, $\eta = 10^{21}$ Pa s, $g = 10\ \mathrm{m\,s^{-2}}$.

I screwed up! R_1 should have been 3480!

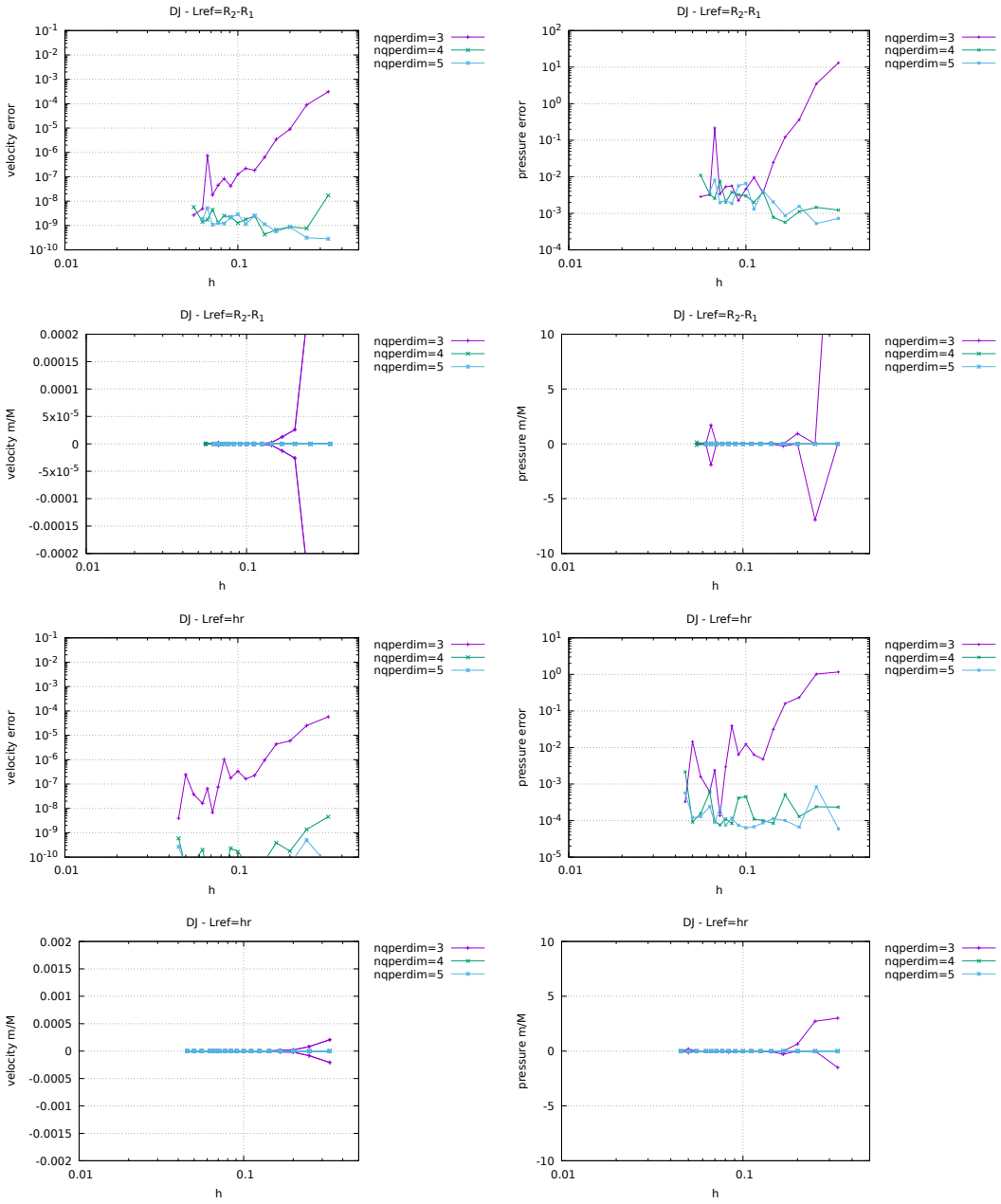


Velocity field for nelr=8 Q2Q1 elements, isoparametric mapping.



Velocities in cm/year.

Conclusions:



Velocities in cm/year.

Conclusions:

Stone 134

CMT Catalog Files



python_codes/fieldstone_134/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_134

This stone was developed in collaboration with Neil Ribe.

The seismic moment tensor is a complete description of the earthquake size and source geometry. The Centroid Moment Tensor (CMT) is a reliable method for calculating moment tensors, by which the Global Centroid Moment Tensor Project systematically studies global seismicity starting from the 80's.

Q: what is a seimic tensor? https://mxrap.com/moment-tensors-a-practical-guide/

All what follows is to be found at https://www.globalcmt.org/. Also check Ekström, Nettles, and Dziewoński [104] (2012) for an overview of the project.

The available data is stored in the "ndk" file format which used to store and distribute the Global Centroid-Moment-Tensor (CMT) catalog (formerly the Harvard CMT catalog). The format is ASCII and uses five 80-character lines per earthquake. Example[1]:

```
================================================================================
1234567890123456789012345678901234567890123456789012345678901234567890123456789

PDE  2005/01/01 01:20:05.4  13.78  -88.78 193.1 5.0 0.0 EL SALVADOR
C200501010120A   B:  4    4  40 S: 27   33  50 M:  0    0   0 CMT: 1 TRIHD:  0.6
CENTROID:     -0.3 0.9  13.76 0.06  -89.08 0.09 162.8 12.5 FREE S-20050322125201
23  0.838 0.201 -0.005 0.231 -0.833 0.270  1.050 0.121 -0.369 0.161  0.044 0.240
V10   1.581 56  12  -0.537 23 140  -1.044 24 241   1.312   9 29  142 133 72   66

PDE  2005/01/01 01:42:24.9   7.29   93.92  30.0 5.1 0.0 NICOBAR ISLANDS, INDIA R
C200501010142A   B: 17   27  40 S: 41   58  50 M:  0    0   0 CMT: 1 TRIHD:  0.7
CENTROID:     -1.1 0.8   7.24 0.04   93.96 0.04  12.0  0.0 BDY  S-20050322125628
23 -1.310 0.212  2.320 0.166 -1.010 0.241  0.013 0.535 -2.570 0.668  1.780 0.151
V10   3.376 16 149   0.611 43  44  -3.987 43 254   3.681 282 48  -23  28 73 -136

....

================================================================================
```

[1] I have added the blanklines. In the file there are none.

- First line: Hypocenter line
  [1-4] Hypocenter reference catalog (e.g., PDE for USGS location, ISC for ISC catalog, SWE for surface-wave location, Ekström [103] (2006))
  [6-15] Date of reference event
  [17-26] Time of reference event
  [28-33] Latitude
  [35-41] Longitude
  [43-47] Depth
  [49-55] Reported magnitudes, usually mb and MS
  [57-80] Geographical location (24 characters)

- Second line: CMT info (1)
  [1-16] CMT event name. This string is a unique CMT-event identifier. Older events have 8-character names, current ones have 14-character names. See note (1) below for the naming conventions used.
  [18-61] Data used in the CMT inversion. Three data types may be used: Long-period body waves (B), Intermediate-period surface waves (S), and long-period mantle waves (M). For each data type, three values are given: the number of stations used, the number of components used, and the shortest period used.
  [63-68] Type of source inverted for: "CMT: 0" - general moment tensor; "CMT: 1" - moment tensor with constraint of zero trace (standard); "CMT: 2" - double-couple source.
  [70-80] Type and duration of moment-rate function assumed in the inversion. "TRIHD" indicates a triangular moment-rate function, "BOXHD" indicates a boxcar moment-rate function. The value given is half the duration of the moment-rate function. This value is assumed in the inversion, following a standard scaling relationship (see note (2) below), and is not derived from the analysis.

- Third line: CMT info (2)
  [1-58] Centroid parameters determined in the inversion. Centroid time (C_time), given with respect to the reference time, centroid latitude (C_lat), centroid longitude (C_lon), and centroid depth (C_depth). The value of each variable is followed by its estimated standard error. See note (3) below for cases in which the hypocentral coordinates are held fixed.
  [60-63] Type of depth. "FREE" indicates that the depth was a result of the inversion; "FIX " that the depth was fixed and not inverted for; "BDY " that the depth was fixed based on modeling of broad-band P waveforms.
  [65-80] Timestamp. This 16-character string identifies the type of analysis that led to the given CMT results and, for recent events, the date and time of the analysis. This is useful to distinguish Quick CMTs ("Q-"), calculated within hours of an event, from Standard CMTs ("S-"), which are calculated later. The format for this string should not be considered fixed.

- Fourth line: CMT info (3)
  [1-2] The exponent for all following moment values. For example, if the exponent is given as 24, the moment values that follow, expressed in dyne-cm, should be multiplied by 10**24.
  [3-80] The six moment-tensor elements: M_{rr}, $M_{tt} = M_{\theta\theta}$, $M_{pp} = M_{\phi\phi}$, $M_{rt} = M_{r\theta}$, $M_{rp} = M_{r\phi}$, $M_{tp} = M_{\theta\phi}$, where r is up, $t = \theta$ is south, and $p = \phi$ is east. See Aki and Richards book [1] for conversions to other coordinate systems. The value of each moment-tensor element is followed by its estimated standard error. See note (4) below for cases in which some elements are constrained in the inversion.

- Fifth line: CMT info (4)
  [1-3] Version code. This three-character string is used to track the version of the program that generates the "ndk" file.
  [4-48] Moment tensor expressed in its principal-axis system: eigenvalue, plunge, and azimuth of the three eigenvectors. The eigenvalue should be multiplied by 10**(exponent) as given on line four.
  [50-56] Scalar moment, to be multiplied by 10**(exponent) as given on line four.
  [58-80] Strike, dip, and rake for first nodal plane of the best-double-couple mechanism, repeated for the second nodal plane. The angles are defined as in Aki and Richards book [1].

Notes (additional information):

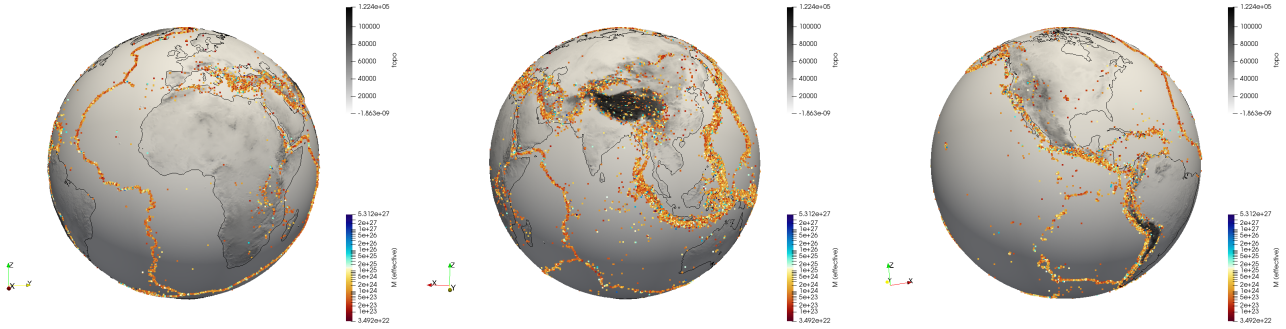
1. CMT event names follow two conventions. Older events use an 8-character name with the structure XMMDDYYZ, where MMDDYY represents the date of the event, Z is a letter (A-Z followed by a-z) distinguishing different events on the same day, and X is a letter (B,M,Z,C,...) used to identify the types of data used in the inversion. Newer events use 14-character event names with the structure XYYYYMMDDhhmmZ, in which the time is given to greater precision, and the initial letter is limited to four possibilities: B - body waves only, S - surface waves only, M - mantle waves only, C - a combination of data types.

2. The source duration is generally estimated using an empirically determined relationship such that the duration increases as the cube root of the scalar moment. Specifically, we currently use a relationship where the half duration for an event with moment 10^{24} is 1.05 seconds, and for an event with moment 10^{27} is 10.5 seconds.

3. For some small earthquakes for which the azimuthal distribution of stations with useful seismograms is poor, we constrain the epicenter of the event to the reference location. This is reflected in the catalog by standard errors of 0.0 for both the centroid latitude and the centroid longitude.

4. For some very shallow earthquakes, the CMT inversion does not well constrain the vertical-dip-slip components of the moment tensor ($M_{rt} = M_{r\theta}$ and $M_{rp} = M_{r\phi}$), and we constrain these components to zero in the inversion. The standard errors for $M_{rt} = M_{r\theta}$ and $M_{rp} = M_{r\phi}$ are set to zero in this case.

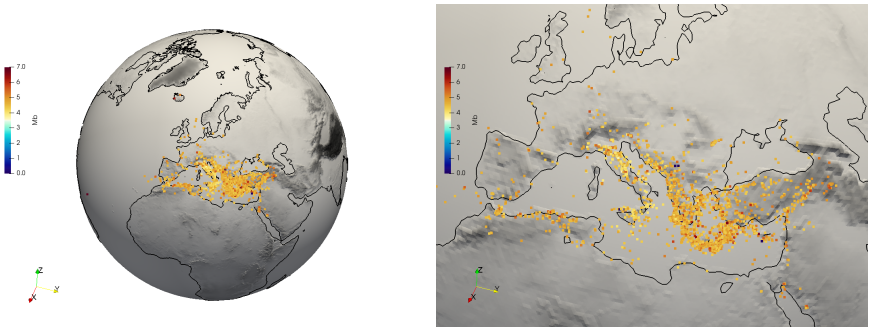
Problems I have noticed with the data file: line 69823: 1.060.0 becomes 1060.0 ? line 73622: 6.010.0 becomes 6010.0 ?



note that the background topogrpahy shell is obtained with STONE 69.

European-Mediterranean Regional Centroid-Moment Tensors

There is another catalogue at `http://rcmt2.bo.ingv.it/`.



Stone 135

SLAB2 data visualisation



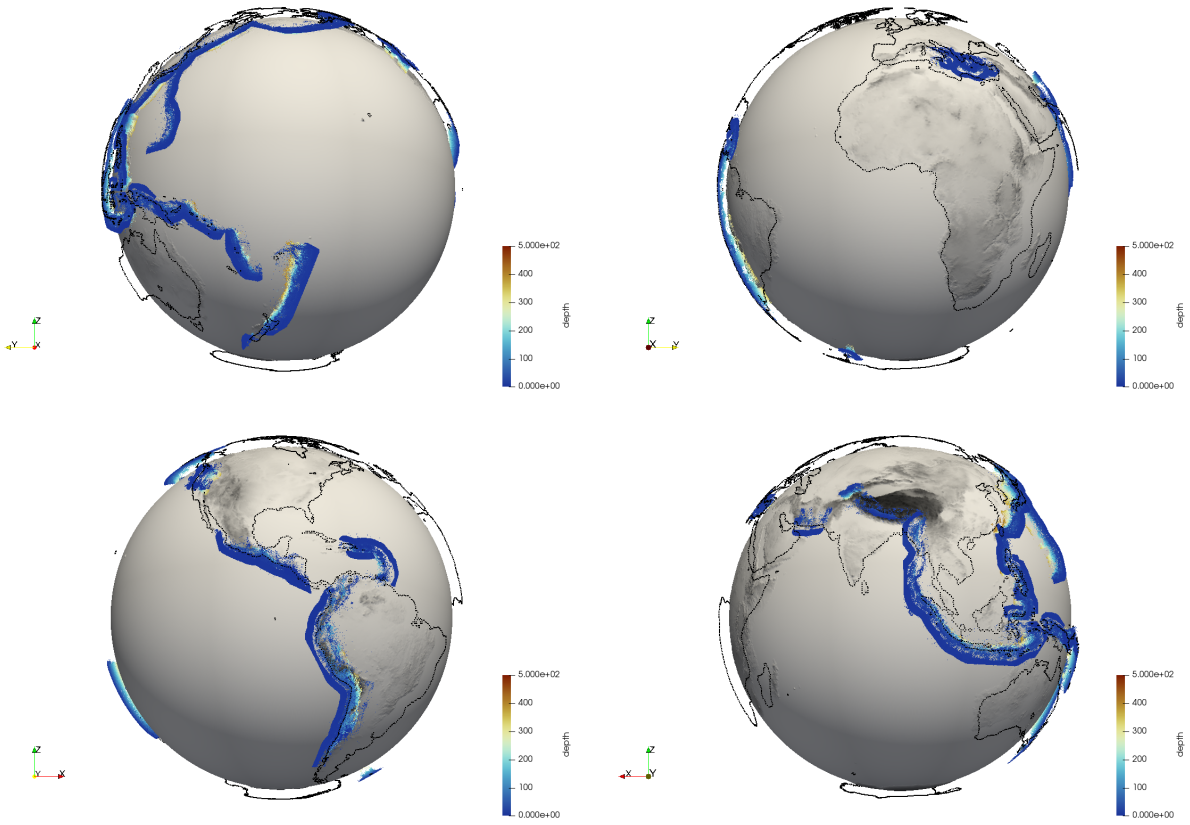
python_codes/fieldstone_135/text.tex

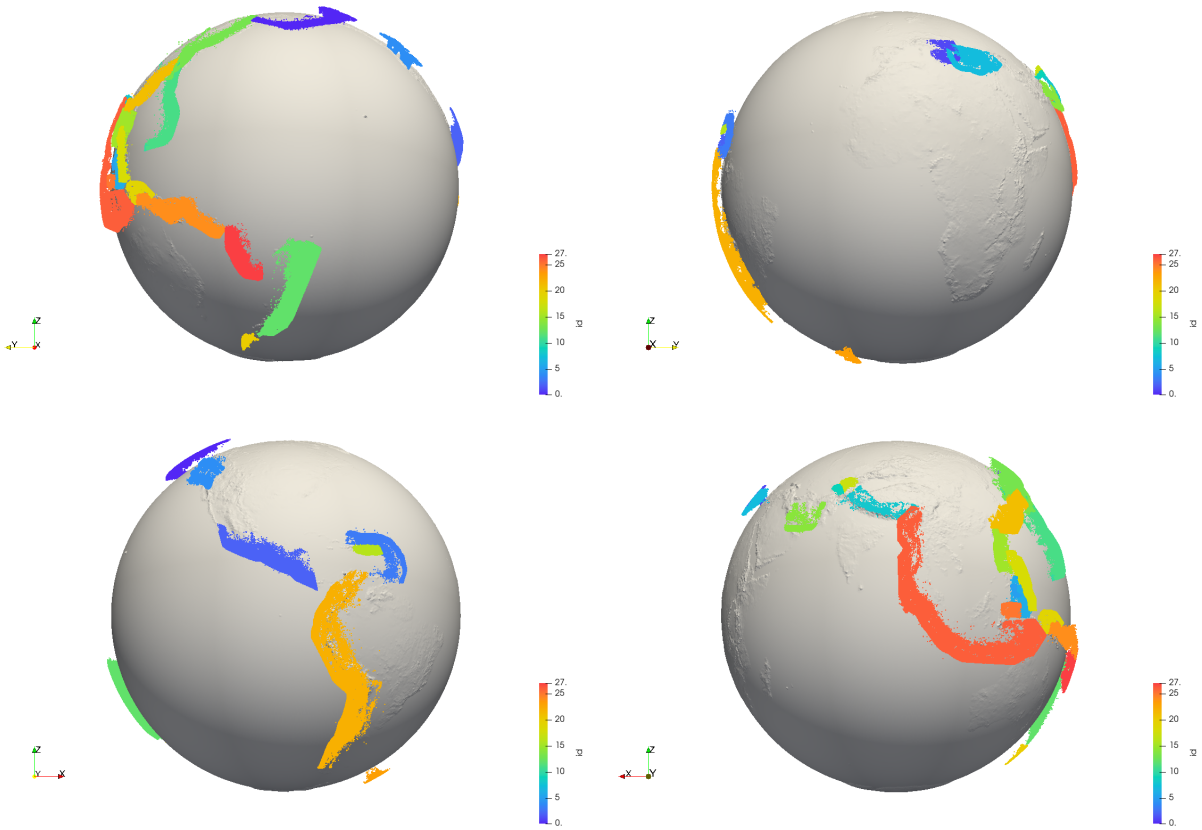
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_135`

This stone was developed in collaboration with Neil Ribe.

The data is obtained from `https://github.com/usgs/slab2`. More precisely, the csv files used here are in `slab2code/Input/09_21`. The data is several hundreds of Mb and it is not mine so it is not uploaded in the fieldstone repository. Go get it on the slab2 github and place it in the slab2 folder.

The 29 files are read in, the depth, latitude and longitude are extracted and exported to vtu.





shell+topo obtained from STONE 69, scaled 95%.

Stone 136

RUM slabs visualisation



python_codes/fieldstone_136/text.tex

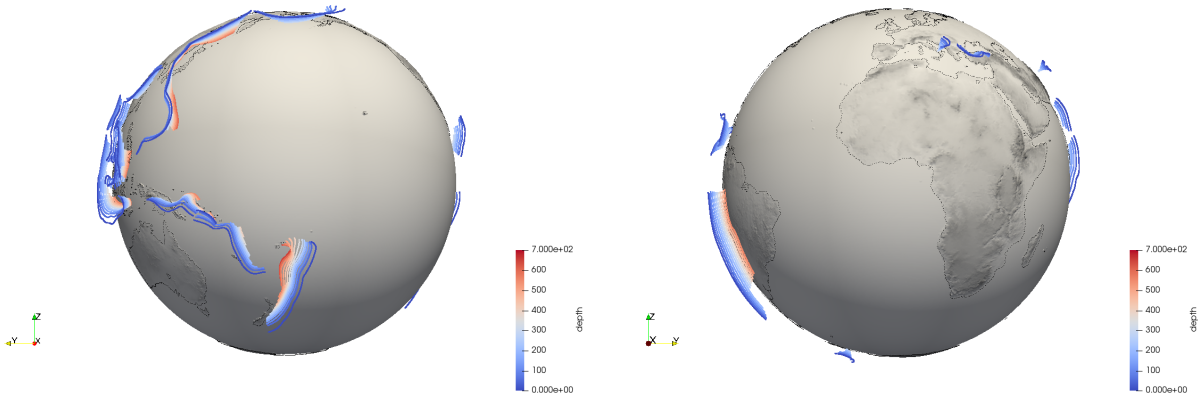
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_136`

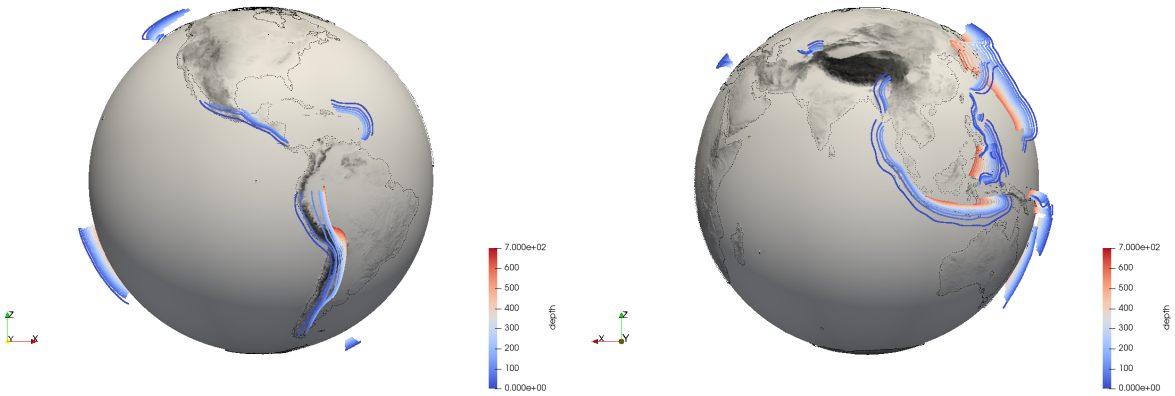
Gudmundsson and Sambridge [144] built a regionalized upper mantle (RUM) reference model. The form of the RUM model is a set of velocity profiles as functions of depth through the upper mantle for each of the different tectonic provinces of Earth. Together the profiles constitute a three-dimensional model which incorporates considerable structural detail. The RUM model includes subducting slabs as sharp fast features in the upper mantle. The RUM model is designed to represent as much of upper mantle heterogeneity as seen by body wave travel times as possible with a simple model. It can be useful as a reference model for individual tectonic regions. A companion paper [273] about the methodology is also available.

The RUM project has (an outdated) webpage: `http://rses.anu.edu.au/seismology/projects/RUM/`. Unfortunately the download links are all broken. After contacting Malcolm Sambridge, he was kind enough to send me all the RUM files.

From these files I use the slab contours in `slab/Contours/`. The files *.slb contain slab contours of individual coherent slab bodies. The first line contains a name which is the name of the file as they are named here. Subsequently come individual contours. First the number of samples on the contour. Then longitude (0-360), latitude, and depth. The contours are drawn such that they follow approximately the top of the seismogenic region.

The code reads all files and generates a vtu file for each as well as a single vtu file containing all slabs.




Z
Y
X
7.000e+02
600
500
400
300
200
100
0.000e+00
depth
Z
X
Y
7.000e+02
600
500
400
300
200
100
0.000e+00
depth

Stone 137

Configurational entropy



python_codes/fieldstone_137/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_137`

This stone was developed in collaboration with Erik van der Wiel and is used in the appendix of the submitted paper `https://doi.org/10.31223/X56971`.

What follows is taken from Camesasca, Kaufman, and Manas-Zloczower [62] (2006), entitled "Quantifying fluid mixing with the Shannon entropy".

The system is divided into 'bins' and for each bin $j = 1, ...M$ we evaluate the number $n_{j,c}$ of particles of species $c = 1, ...C$. The joint probability that a particle of species c is in bin j can be calculated by dividing $n_{j,c}$ by the overall system population.

$$p_{j,c} = \frac{\frac{n_{j,c}}{N_c}}{\sum_{i=1}^{M}\sum_{c=1}^{C}\frac{n_{i,c}}{N_c}} \tag{137.1}$$

with

$$N_c = \frac{n_c}{M}$$

where n_c is the total number of particles of c in the domain.

In the code, we define

$$\texttt{denom1} = \sum_{i=1}^{M}\sum_{c=1}^{C}\frac{n_{i,c}}{N_c}$$

Using the joint probabilities of the equation above we then calculate the entropy:

$$S = -\sum_{j=1}^{M}\sum_{c=1}^{C} p_{j,c} \ln p_{j,c} \tag{137.2}$$

Note that if one starts with $n_1 = n_2 = ...n_C$ then all values of P_i are equal and therefore can be removed from Eq. (137.1) and the denominator is then simply the total number of particles in the domain.

Eq. (137.2) can be expressed as the sum of two other entropies: the conditional entropy $S_{location}(species)$ and the entropy of spatial distribution $S(location)$, i.e.:

$$S = S_{location}(species) + S(location)$$

We have

$$p_{j,c} = p_{c|j} p_j \qquad \text{with} \qquad p_{c|j} = \frac{\frac{n_{j,c}}{N_c}}{\sum_{c=1}^{C} \frac{n_{j,c}}{N_c}} \qquad \text{and} \qquad p_j = \frac{\sum_{c=1}^{C} \frac{n_{j,c}}{N_c}}{\sum_{i=1}^{M} \sum_{c=1}^{C} \frac{n_{i,c}}{N_c}}$$

In the code we define

$$\texttt{denom2}[j] = \sum_{c=1}^{C} \frac{n_{j,c}}{N_c}$$

Substituting Eq. (??) into Eq. (??):

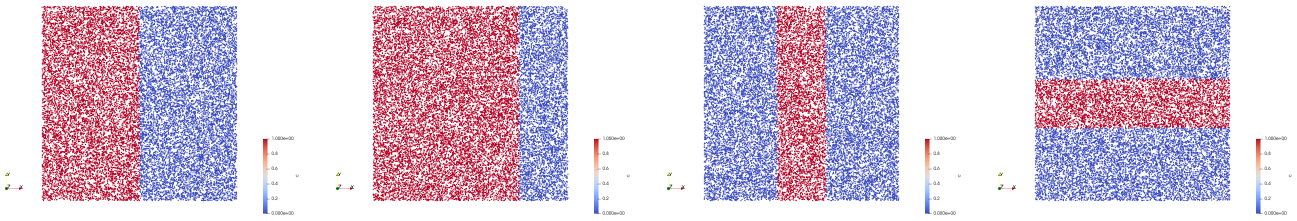
$$\begin{aligned} S &= -\sum_{j=1}^{M} \sum_{c=1}^{C} p_{j,c} \ln p_{j,c} \\ &= -\sum_{j=1}^{M} \sum_{c=1}^{C} \left[(p_{c|j} p_j) \ln (p_{c|j} p_j) \right] \\ &= -\sum_{j=1}^{M} \sum_{c=1}^{C} \left[(p_{c|j} p_j) \ln (p_j) \right] - \sum_{j=1}^{M} \sum_{c=1}^{C} \left[(p_{c|j} p_j) \ln (p_{c|j}) \right] \end{aligned} \tag{137.3}$$

Since $\sum_{c=1}^{C} p_{c|j} = 1$, then:

$$\begin{aligned} S &= -\sum_{j=1}^{M} \left[p_j \sum_{c=1}^{C} p_{c|j} \ln p_{c|j} \right] - \sum_{j=1}^{M} p_j \ln p_j \\ &= \underbrace{S_{location}(species)}_{S_3} + \underbrace{S(location)}_{S_2} \end{aligned} \tag{137.4}$$

As specified in the paper we also use a 4th order Runge-Kutta (in space only here) with a fixed timestep δt.

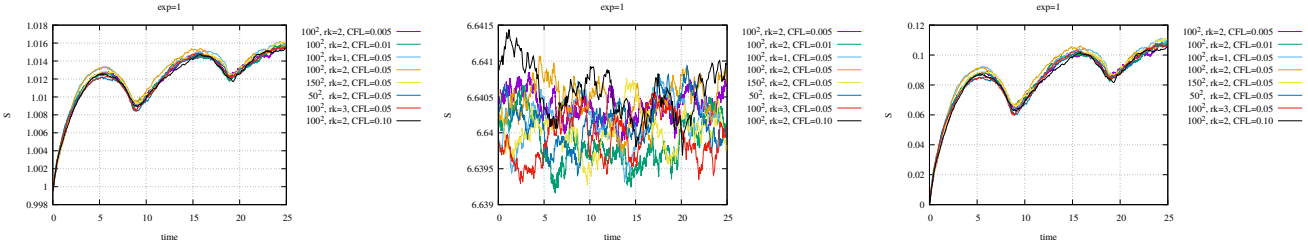
This STONE borrows from STONE 76 (Q2P-1 element), STONE 13 for PIC, STONE 67 for Runge-Kutta algorithms.

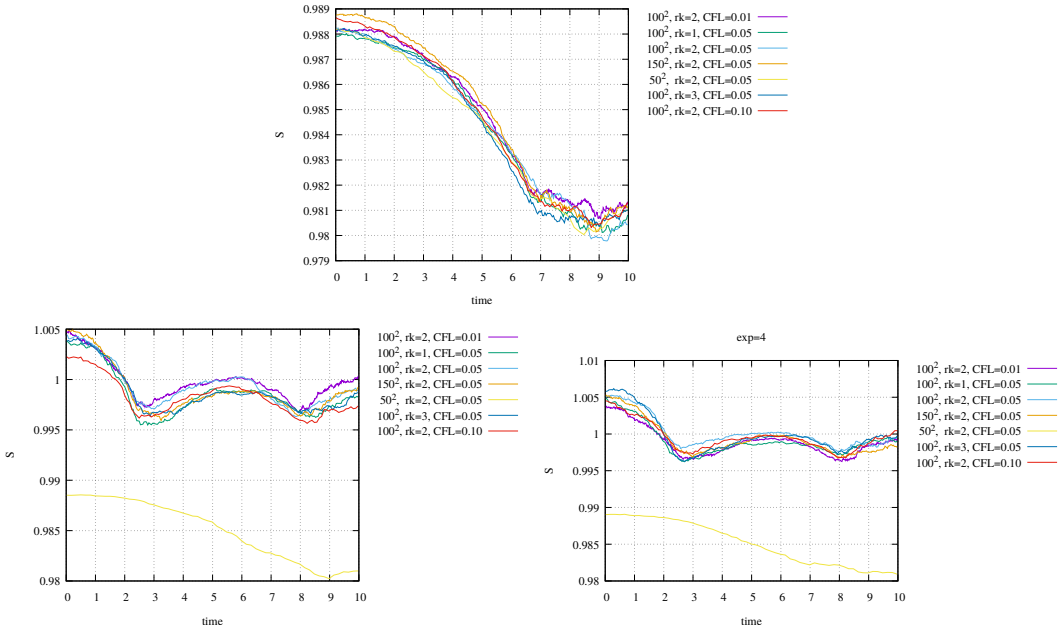


From left to right: exp=1,2,3,4.

Testing: Using the Donea & Huerta manufactured solution

This benchmark is taken from Donea & Huerta (2003) [94] and is described fully in section ??. We here explore the influence of the Runge-Kutta order on the entropy values, as well as the value of the CFL number and the mesh resolution for a given number of 10x10 bins.





Left: exp1, right: exp2. S normalised by $\ln M$.

Q: why is S equal to log(M) ?

Reproducing results of Camesasca, Kaufman, and Manas-Zloczower [62]?

The domain is a unit square. Boundary conditions are no slip on the sides, $\vec{\mathbf{v}} = (1, 0)$ at the top and $\vec{\mathbf{v}} = (-1, 0)$ at the bottom. The flow is assumed to be isoviscous ($\eta = 1$), incompressible and isothermal. Gravity is set to zero. Note that the original paper [62] solves the Navier-Stokes equations but with $Re = 1$ so as to avoid turbulence. Also the authors state that "the velocity is not a function of time" which I interprete as them solving the steady state N-S equations.

We then expect 3 stagnation points at $(0.5, 0.5)$ and $(0.5, 0.5 \pm 0.161)$.

Relevant Literature: Wichmann, Delandmeter, Dijkstra, and Sebille [352] (2019) use Camesasca, Kaufman, and Manas-Zloczower [62] (2006) in the context of "Mixing of passive tracers at the ocean surface and its implications for plastic transport modelling"

Stone 138

Magnetic Terrain Effect



python_codes/fieldstone_138/text.tex

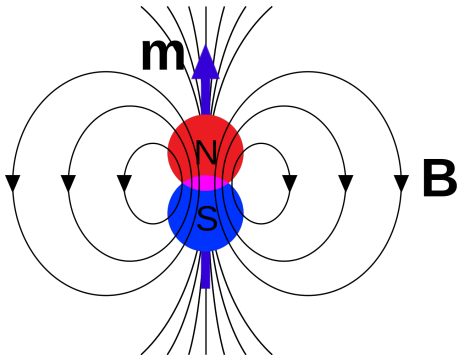
magnetostatics

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_138

This stone was developed in collaboration with Agnes Hendrickx and Lennart de Groot.

Rationale:

Maxwell equations for magnetostatics



Outside of the source region, this potential is the vector potential $\vec{A}$ of a magnetic dipole is ($\vec{r}$ is the vector from the position of the dipole to the position where the field is being measured):

$$\vec{A}(\vec{r}) = \frac{\mu_0}{4\pi}\frac{\vec{m}\times\vec{r}}{r^3}$$

and the magnetic flux density field $\vec{B}$ (in Tesla) is a vector quantity given by

$$\vec{B}(\vec{r}) = \vec{\nabla}\times\vec{A} = \frac{\mu_0}{4\pi}\left[\frac{3\vec{r}(\vec{m}\cdot\vec{r})}{r^5} - \frac{\vec{m}}{r^3}\right]$$

Note that a slightly different equation is presented on Wikipedia, which adds a term to the above one:

$$\vec{B}(\vec{r}) = \vec{\nabla} \times \vec{A} = \frac{\mu_0}{4\pi}\left[\frac{3\vec{r}(\vec{m}\cdot\vec{r})}{r^5} - \frac{\vec{m}}{r^3}\right] + \frac{2\mu_0}{3}\vec{m}\,\delta(\boldsymbol{r})$$

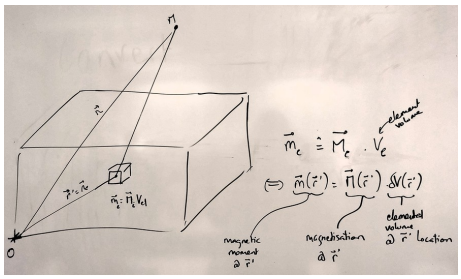
From Wiki[1]: In classical electromagnetism, magnetization or magnetic polarization is the vector field that expresses the density of permanent or induced magnetic dipole moments in a magnetic material. The origin of the magnetic moments responsible for magnetization can be either microscopic electric currents resulting from the motion of electrons in atoms, or the spin of the electrons or the nuclei. Net magnetization results from the response of a material to an external magnetic field, together with any unbalanced magnetic dipole moments that may be inherent in the material itself; for example, in ferromagnets. Magnetization is not always uniform within a body, but rather varies between different points. Physicists and engineers usually define magnetization as the quantity of magnetic moment per unit volume. It is represented by a pseudovector $\vec{M}$.

The magnetization field ($\vec{M}$-field) can be defined according to the following equation:

$$\vec{M} = \frac{\delta\vec{m}}{\delta V}$$

Where $\delta\vec{m}$ is the elementary magnetic moment and δV is the volume element; in other words, the $\vec{M}$-field is the distribution of magnetic moments in the region or manifold concerned. This is better illustrated through the following relation:

$$\vec{m} = \int\int\int \vec{M}\,dV$$



The 3D domain Ω is tesselated with many small volumes (cells). At any given position $\vec{r}$, the resulting fields $\vec{A}$ and $\vec{B}$ are the sum of the contributions of all cells, assuming each one can be seen as a dipole $\delta\vec{m}$. We then have:

$$\vec{A}(\vec{r}) = \sum_{i=1,cells} \frac{\mu_0}{4\pi}\frac{\delta\vec{m}_i \times (\vec{r}-\vec{r}_i)}{|\vec{r}-\vec{r}_i|^3} \tag{138.1}$$

$$= \sum_{i=1,cells} \frac{\mu_0}{4\pi}\frac{\vec{M}_i \times (\vec{r}-\vec{r}_i)}{|\vec{r}-\vec{r}_i|^3}\delta V_i \tag{138.2}$$

And then, when the number of cells becomes infinitely large, we obtain:

$$\vec{A}(\vec{r}) = \int_\Omega \frac{\mu_0}{4\pi}\frac{\vec{M}(\vec{r'}) \times (\vec{r}-\vec{r'})}{|\vec{r}-\vec{r'}|^3}d\vec{r'} \tag{138.3}$$

Likewise, the magnetic field is given by

$$\vec{B}(\vec{r}) = \sum_{i=1,cells} \frac{\mu_0}{4\pi}\left[\frac{3(\vec{r}-\vec{r}_i)(\delta\vec{m}_i\cdot(\vec{r}-\vec{r}_i))}{|\vec{r}-\vec{r}_i|^5} - \frac{\delta\vec{m}_i}{|\vec{r}-\vec{r}_i|^3}\right] \tag{138.4}$$

$$= \sum_{i=1,cells} \frac{\mu_0}{4\pi}\left[\frac{3(\vec{r}-\vec{r}_i)(\vec{M}_i\cdot(\vec{r}-\vec{r}_i))}{|\vec{r}-\vec{r}_i|^5} - \frac{\vec{M}_i}{|\vec{r}-\vec{r}_i|^3}\right]\delta V_i \tag{138.5}$$

$$\cdots$$

$$= \int_\Omega \frac{\mu_0}{4\pi}\left[\frac{3(\vec{r}-\vec{r'})(\vec{M}(\vec{r'})\cdot(\vec{r}-\vec{r'}))}{|\vec{r}-\vec{r'}|^5} - \frac{\vec{M}(\vec{r'})}{|\vec{r}-\vec{r'}|^3}\right]d\vec{r'} \tag{138.6}$$

[1] `https://en.wikipedia.org/wiki/Magnetization`

FROM grape function comment latex This function receives as argument the coordinates `x,y,z` of a point and returns the magnetic vector potential $\boldsymbol{A}$ and the magnetic field $\boldsymbol{B}$ defined by the equations

$$\boldsymbol{B}(\boldsymbol{r}) = \frac{\mu_0}{4\pi}\int\int\int \frac{1}{|\boldsymbol{r}-\boldsymbol{r}'|^3}\left[\frac{3}{|\boldsymbol{r}-\boldsymbol{r}'|^2}[\boldsymbol{M}(\boldsymbol{r}')\cdot(\boldsymbol{r}-\boldsymbol{r}')](\boldsymbol{r}-\boldsymbol{r}')-\boldsymbol{M}(\boldsymbol{r}')\right]d^3\boldsymbol{r}' \tag{138.7}$$

$$\boldsymbol{A}(\boldsymbol{r}) = \frac{\mu_0}{4\pi}\int\int\int \frac{\boldsymbol{M}(\boldsymbol{r}')\times(\boldsymbol{r}-\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|^3}d^3\boldsymbol{r}' \tag{138.8}$$

In the SI system, the units of A are $V.s.m^{-1}$ and are the same as that of momentum per unit charge.

We can also compute the magnetic scalar potential ψ as follows:

$$\Psi(\vec{r}) = \frac{1}{4\pi}\int\int\int \frac{\vec{M}(\vec{r}')\cdot(\vec{r}-\vec{r}')}{|\vec{r}-\vec{r}'|^3}d^3\vec{r}'$$

and the magnetic field vector $\vec{H}$:

$$\vec{H} = -\vec{\nabla}\psi = -\int\int\int \left[M_x(\vec{r}')\underbrace{\vec{\nabla}\frac{(x-x')}{|\vec{r}-\vec{r}'|^3}}_{\vec{\Theta}_x} + M_y(\vec{r}')\underbrace{\vec{\nabla}\frac{(y-y')}{|\vec{r}-\vec{r}'|^3}}_{\vec{\Theta}_y} + M_z(\vec{r}')\underbrace{\vec{\nabla}\frac{(z-z')}{|\vec{r}-\vec{r}'|^3}}_{\vec{\Theta}_z}\right]d^3\vec{r}'$$

$$\vec{\Theta}_x = \vec{\nabla}\frac{(x-x')}{[(x-x')^2+(y-y')^2+(z-z')^2]^{3/2}} \tag{138.9}$$

$$= \frac{1}{|\vec{r}-\vec{r}'|^6}\begin{pmatrix} 1\cdot|\vec{r}-\vec{r}'|^3-(x-x')\cdot 3(x-x')|\vec{r}-\vec{r}'| \\ 0\cdot|\vec{r}-\vec{r}'|^3-(x-x')\cdot 3(y-y')|\vec{r}-\vec{r}'| \\ 0\cdot|\vec{r}-\vec{r}'|^3-(x-x')\cdot 3(z-z')|\vec{r}-\vec{r}'| \end{pmatrix} \tag{138.10}$$

$$= \frac{1}{|\vec{r}-\vec{r}'|^5}\begin{pmatrix} |\vec{r}-\vec{r}'|^2-3(x-x')(x-x') \\ -3(x-x')(y-y') \\ -3(x-x')(z-z') \end{pmatrix} \tag{138.11}$$

$$\vec{\Theta}_y = \vec{\nabla}\frac{(y-y')}{[(x-x')^2+(y-y')^2+(z-z')^2]^{3/2}} \tag{138.12}$$

$$= \frac{1}{|\vec{r}-\vec{r}'|^5}\begin{pmatrix} -3(y-y')(x-x') \\ |\vec{r}-\vec{r}'|^2-3(y-y')(y-y') \\ -3(y-y')(z-z') \end{pmatrix} \tag{138.13}$$

$$\vec{\Theta}_z = \vec{\nabla}\frac{(z-z')}{[(x-x')^2+(y-y')^2+(z-z')^2]^{3/2}} \tag{138.14}$$

$$= \frac{1}{|\vec{r}-\vec{r}'|^5}\begin{pmatrix} -3(z-z')(x-x') \\ -3(z-z')(y-y') \\ |\vec{r}-\vec{r}'|^2-3(z-z')(z-z') \end{pmatrix} \tag{138.15}$$

$$\tag{138.16}$$

Computational methods

book fortran translation to python jit

stone_test: bunch of small tests to make sure all works as expected

stone.py: computes B on plane or line above 3D domain with mesh

two surface integral functions, one for cuboids, one for deformed hexahedra accomodating surface

jit much faster!

In magnetostatics.py:

```
def cross(a,b):
def qcoords_1D(nqpts):
def qweights_1D(nqpts):
def NNN(r,s,t):
def dNNNdr(r,s,t):
def dNNNds(r,s,t):
def dNNNdt(r,s,t):
def compute_B_quadrature(xmeas,ymeas,zmeas,x,y,z,icon,Mx,My,Mz,nqdim):
def plane(x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3):
def line(x0,y0,z0,x1,y1,z1,x2,y2,z2):
def rot(ax,ay,az,bx,by,bz,nx,ny,nz,px,py,pz):
def facmag(Mx,My,Mz,x0,y0,z0,x,y,z,n):
def compute_B_surface_integral_cuboid(xmeas,ymeas,zmeas,x,y,z,icon,Mx,My,Mz):
def compute_B_surface_integral_wtopo(xmeas,ymeas,zmeas,x,y,z,icon,Mx,My,Mz):
```

Surf vs volume integration

DEM processing & visualisation, and paths

provenance

resolutions 1,2,5,10 ?

processing / coordinate system

height offset DEM path

Benchmarking

small tests - volume vs surface integration

Benchmark 1 - magnetic field above a spherical anomaly/dipole

The domain is $2 \times 2 \times 2$ m. No background magnetisation. Single spherical inclusion of radius 1 m in the middle of the domain with magnetisation $\boldsymbol{M} = (0, 1, 0)$. Note that the computational domain is such that its top surface is at $z = 0$.

If one now computes the magnetic field on a vertical line centered on the sphere, one should recover the single dipole value at very large distances ($z >> L_z$).

We start from the vector potential of a single dipole (Eq. 5.83 of [143])

$$\boldsymbol{A}_{dip}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \frac{\boldsymbol{m} \times \boldsymbol{r}}{r^3}.$$

Very far away from the sphere, one can see the sphere as a single dipole of volume $V = 4\pi R^3/\pi$ and r is the distance between the measurement point and the center of the sphere, i.e. $r = z - z_c$:

$$\boldsymbol{A}_{dip}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \frac{\boldsymbol{M}V \times \boldsymbol{r}}{(z - z_c)^3}$$

Also, $\boldsymbol{M} = (0, 1, 0)$ and $\boldsymbol{r} = (x - x_c, y - y_c, z - z_c) = (0, 0, z - z_c)$ so that $\boldsymbol{M} \times \boldsymbol{r} = M(z - z_c)\boldsymbol{e}_x$. Finally:

$$A_x(z) = \frac{\mu_0}{4\pi} \frac{MV}{(z - z_c)^2} \qquad A_y = 0 \qquad A_z = 0$$

Turning to the magnetic field of a single dipole:

$$\boldsymbol{B}_{dip}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \frac{1}{r^3} \left[\frac{3}{r^2} (\boldsymbol{m} \cdot \boldsymbol{r}) \boldsymbol{r} - \boldsymbol{m} \right]$$

$$\boldsymbol{B}_{dip}(\boldsymbol{r}) = \frac{\mu_0 V}{4\pi} \frac{1}{r^3} \left[\frac{3}{r^2} (\boldsymbol{M} \cdot \boldsymbol{r}) \boldsymbol{r} - \boldsymbol{M} \right]$$

If we measure on a line perpendicular to $\vec{M}$, then $\boldsymbol{M} \cdot \boldsymbol{r} = 0$ so

$$\boldsymbol{B}_{dip}(\boldsymbol{r}) = -\frac{\mu_0 V}{4\pi} \frac{M}{r^3} \boldsymbol{e}_y$$

If we measure on a line along $\vec{M}$, and take $\vec{M} \propto \vec{e}_z$, then

$$\frac{3}{r^2} (\boldsymbol{M} \cdot \boldsymbol{r}) \boldsymbol{r} - \boldsymbol{M} = \frac{3}{r_z^2} \left[\begin{pmatrix} 0 \\ 0 \\ r_z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ M_z \end{pmatrix} \right] = 3 \begin{pmatrix} 0 \\ 0 \\ M_z \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ M_z \end{pmatrix} = 2 \begin{pmatrix} 0 \\ 0 \\ M_z \end{pmatrix}$$

M is actually (0,0,1) is code! adapt eqations above

Benchmark 2 - testing internal cancellation of surface integrals

random perturbation of internal nodes

add description and why it matters

2 tests:

- 2a: random perturbation of cubic elements
- 2b: random perturbation of elements with high aspect ratio

show(?) for both no change plane measurements

Benchmark 3 - magnetic field outside a magnetised sphere

The domain is $20 \times 20 \times 20$. $\vec{M} = (0, 0, 7.5)$ is prescribed inside a sphere of radius $R = 10$ centered in the domain. An analytical solution exists for this problem.

Because we discretise a sphere we need a rather high mesh resolution, i.e. 40**3, 80**3, 120**3, or even more (?). In order to insure a high volume integral accuracy we set nqperdim=6.

Instead of arbitrarily choosing a few points around the sphere we use a fibonacci spiral of 101 points. And in order to show that we can accurately compute $\vec{B}$ close to objects, we then set the spiral radius at $1.01 * R$ radius, i.e. 10cm above sphere.

Rather interestingly the volume integration seems to be the least accurate.

The paper

To Do, open questions, future work?

- minus sign: resulting B from vol and surface intergation are opposite signs?
- nqdim ?
- atan2? f77 python
- add dtype to arrrays
- compute field above vertical face!
- explain difference btw m and M (dipole moment vs magnetisation)


Relevant Literature:

Richard J Blakely. *Potential theory in gravity and magnetic applications*. Cambridge university press, 1996

Stone 139

Evolution of a 1D River Profile



python_codes/fieldstone_139/text.tex

surface_processes

Code at https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_139

This stone was developed in collaboration with Rayane Meghezi.

This STONE originates in Simpson [298] book, chapter 9. It investigates evolution of a one-dimensional (1D) river profile cutting into an uplifting block of crust.

$$\frac{\partial h}{\partial t}+\vec{\mathrm{v}}\cdot\vec{\nabla}h=-\vec{\nabla}\cdot(q_s\vec{n})+w$$

where h is the elevation of the topography, $\vec{\mathrm{v}}=(u,v)$ is the rate of horizontal tectonic motion in the x- and y-directions, w is the vertical rate of uplift or subsidence, q_s is the volumetric sediment flux per unit width (with units $\mathrm{m}^2\,\mathrm{s}^{-1}$), and $\vec{n}$ is a unit vector directed down the slope of the surface (i.e., $\vec{n}=-\vec{\nabla}h/S$ where S is the local surface slope defined as $|\vec{\nabla}h|$).

Simpson further explains that "The sediment flux depends on numerous factors, most notably the local slope and the discharge of water flowing over the surface. In situations where sediment transport is controlled by the supply of material rather than detachment processes (e.g., as would be the case in an alluvial river), one can write the sediment flux as

$$q_s=c_0S+b(q-q_c)^nS^m$$

where c_0 , b, n, and m are positive constants, S is the local slope, q is the local water discharge, and q_c is a critical discharge below which there is no sediment transport. This relation includes contributions from two terms, a *hillslope* flux that depends linearly on the local slope (term 1) and a *fluvial* flux that depends nonlinearly on both the water discharge and the local slope (term 2).

The discharge of water flowing over the surface accumulates, for example, due to run-off from rainfall. Here, rather than computing the discharge from the equations governing the conservation of mass and momentum of the surface water Simpson and Castelltort [299], we use a simple empirically based power law relationship between the water discharge and the upstream drainage area A, that is

$$q=kA^p$$

where k and p are positive constants. A similar relation can be introduced for the critical discharge

$$q_c = kA_c^p$$

where A_c is the critical drainage area below which there is no fluvial sediment transport. Substituting equations 9.2, 9.3, and 9.4 into 9.1 leads to

$$\frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y} = \vec{\nabla} \cdot \left[(c_0 + cA_e^r S^{m-1})\vec{\nabla} h\right] + w$$

where $c = bk^n$, $r = pn$ and A_e is the effective drainage area defined as $A_e = A - A_c$ for $A > A_c$ and $A_e = 0$ for $A \leq A_c$. ”

There are two python scripts doing about the same thing. The first one is a direct translation of the original matlab code to python, as carried out by MSc student R. Meghezi, and the second one is a complete rewrite of the original code in a style compliant with all the other stones.

1D linear elements are used.

See Yang, Chemia, Artemieva, and Thybo [355] (2018) for implementation based on Kooi and Beaumont [198] (1994)

To Do, open questions, future work?

- do smthg

Relevant Literature:

xxxxYY

Stone 140

Evolution of a Fluvially Dissected Landscape



python_codes/fieldstone_140/text.tex

2D
box
P1
surface_processes

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_140`

This stone was developed in collaboration with Rayane Meghezi.

This STONE idea originates in chapter 9.2 of Simpson [298] (2017).
We solve the full 2D surface evolution model

$$\frac{\partial h}{\partial t}+u\frac{\partial h}{\partial x}+v\frac{\partial h}{\partial y}=\vec{\nabla}\cdot\left[(c_0+cA_e^rS^{m-1})\vec{\nabla}h\right]+w \tag{140.1}$$

without lateral tectonic advection ($u=v=0$), that is,

$$\frac{\partial h}{\partial t}=\vec{\nabla}\cdot\left[(c_0+cA_e^rS^{m-1})\vec{\nabla}h\right]+w \tag{140.2}$$

where h is the surface elevation, c_0 is the "hillslope" diffusivity, c is a fluvial transport coefficient, A_e is the effective upstream drainage area (defined as $A_e=A-A_c$ where A is the upstream drainage area and A_c is a critical drainage area below which there is no fluvial sediment transport), S is the slope, r and m are positive exponents, and w is the rate of rock uplift (assumed to be uniform and constant).

As in the book, this equation will be solved on a square 50×50 km domain that has an initial random elevation of between 0 and 10 m, while the elevation of all four boundaries is fixed to zero. Additional setups will be added later. Here, we use linear three-node triangular elements.

The finite element formulation of the equation above is

$$\mathbb{M}\cdot\vec{\dot{h}}+\mathbb{K}\cdot\vec{h}=\vec{F}$$

with

$$\begin{aligned}\mathbb{M} &= \int_\Omega \vec{\mathcal{N}}^T \vec{\mathcal{N}} dV \\ \mathbb{K} &= \int_\Omega \boldsymbol{B}^T \underbrace{(c_0 + cA_e^r S^{m-1})}_{\kappa} \boldsymbol{B}\, dV \\ \vec{F} &= \int_\Omega \vec{\mathcal{N}}^T w\, dV \end{aligned} \tag{140.3}$$

where $\vec{h}$ is the vector of nodal elevations for an element, $\vec{\mathcal{N}}$ are the shape functions, $\boldsymbol{B}$ are the shape function derivatives expressed in terms of physical coordinates.

Using a first-order time derivative yields

$$\dot{\vec{h}} \simeq \frac{\vec{h}^n - \vec{h}^{n-1}}{\delta t}$$

so that

$$(\mathbb{M} + \mathbb{K}\delta t) \cdot \vec{h}^n = \vec{F}\delta t + \mathbb{M} \cdot \vec{h}^{n-1}$$

Except for the coefficient inside the $\mathbb{K}$ matrix, this is a rather standard diffusion equation. Looking closer, the upstream drainage area A appearing in the element matrix $\mathbb{K}$ (recall that $A_e = A - A_c$) is computed according to the following scheme:

1. Calculate the surface area of each finite element. In the program listed in the following text, the result is saved in the array `area`.
2. Find and save the three elements adjacent to each element, saved in the array `gnei`. Elements on the boundary that have only two neighbors are assigned their own element index to the missing third neighbor. If the mesh doesn't change through time, this step and the previous one need to be performed only once before the time loop.

change name!

3. At each step in time, calculate the average elevation of each element, based on the three corner nodes. The result is saved in the array `zc`.
4. Sort the average element elevation for the entire mesh from highest to lowest. The indices of the sorted elevations are saved in the array `sorted_indices`.
5. Using the sorted indices, sort the neighboring elements, the result of which is saved in `sorted_gnei`. At this stage, all elements in the mesh have been ordered, along with their three adjacent neighbors, from highest to lowest.
6. For each element, find and save the local index of the lowest of the three adjacent elements. The result (i.e., 1, 2, or 3) is saved in the array `min_index`.
7. Set the upstream drainage area `A` initially to the element area.
8. In a loop over all ordered elements from highest to lowest, "pass" the accumulated drainage area from each (donor) element to its lowest adjacent neighbor (receiver). The result after the loop has been completed is the accumulated surface area that "drains" to each element in the landscape, `A`. Note that this procedure implicitly assumes a spatially uniform rainfall, which can easily be accounted for if desired.

Furthermore, the slope S is computed as follows:

$$S = \sqrt{\left(\frac{\partial h}{\partial x}\right)^2 + \left(\frac{\partial h}{\partial y}\right)^2}$$

This means that the diffusion coefficient κ depends on (the gradient of) the solution, which renders the PDE nonlinear. As such we must then put in place a dedicated algorithm, so-called Picard iterations, to iterate things out and obtain the converged solution at each time step. We must then solve

$$\Big(\mathbb{M} + \mathbb{K}(\vec{h}^n)\delta t\Big) \cdot \vec{h}^n = \vec{F} + \mathbb{M} \cdot \vec{h}^{n-1}$$

and we will therefore introduce a nonlinear loop inside each time step so that

$$\Big(\mathbb{M} + \mathbb{K}(\vec{h}^n_k)\delta t\Big) \cdot \vec{h}^n_{k+1} = \vec{F} + \mathbb{M} \cdot \vec{h}^{n-1}$$

We will loop over k until $|\vec{h}^n_{k+1} - \vec{h}_k| < tol$.

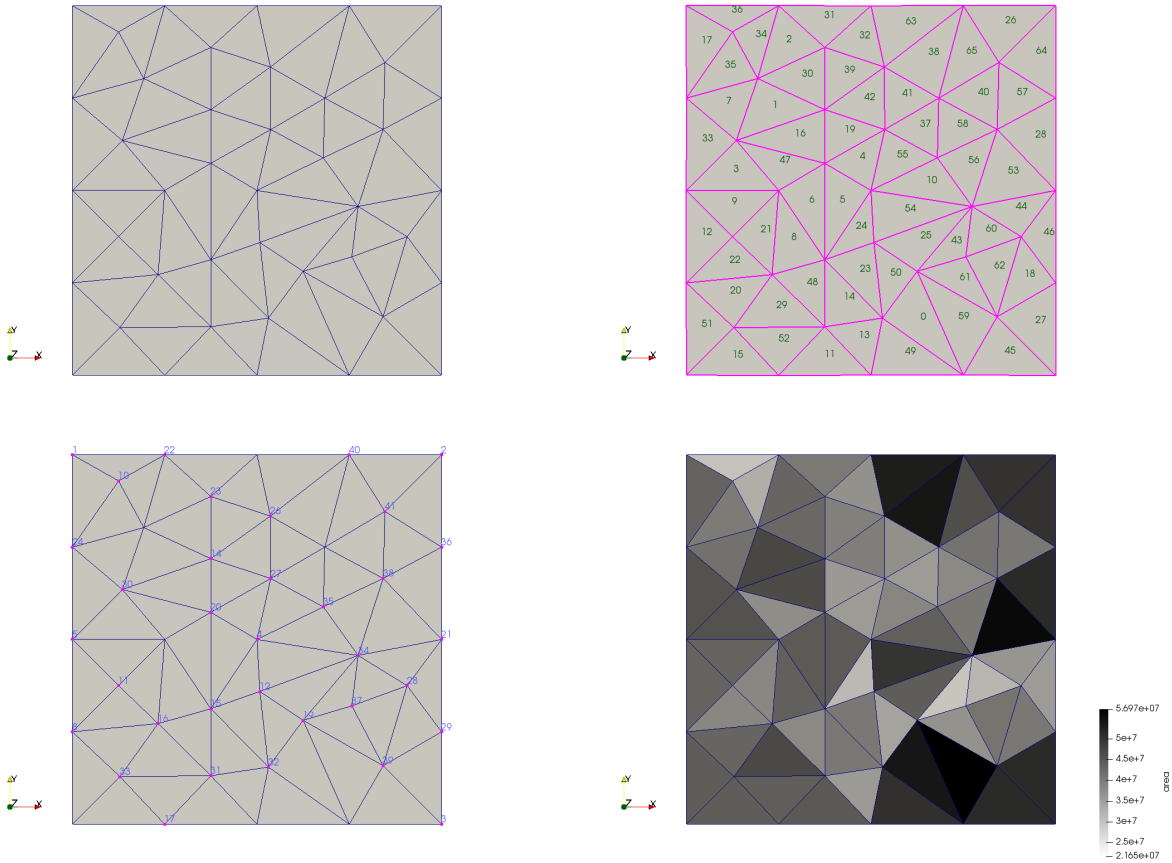
Note that when $m = 1$ and $r = 0$ then $\kappa = c_0$, the PDE is linear and is a simple 2D diffusion equation which is trivial to benchmark, see Section **??**.

Implementation & testing

In order to test/debug the steps highlighted previously we first generate a mesh that only counts a few dozen of elements by setting the maximum size of a triangle to a large value:

```
T1 = tr.triangulate(O1, 'pqa60000000')
```

We then obtain the following mesh counting 65 elements and 42 nodes:

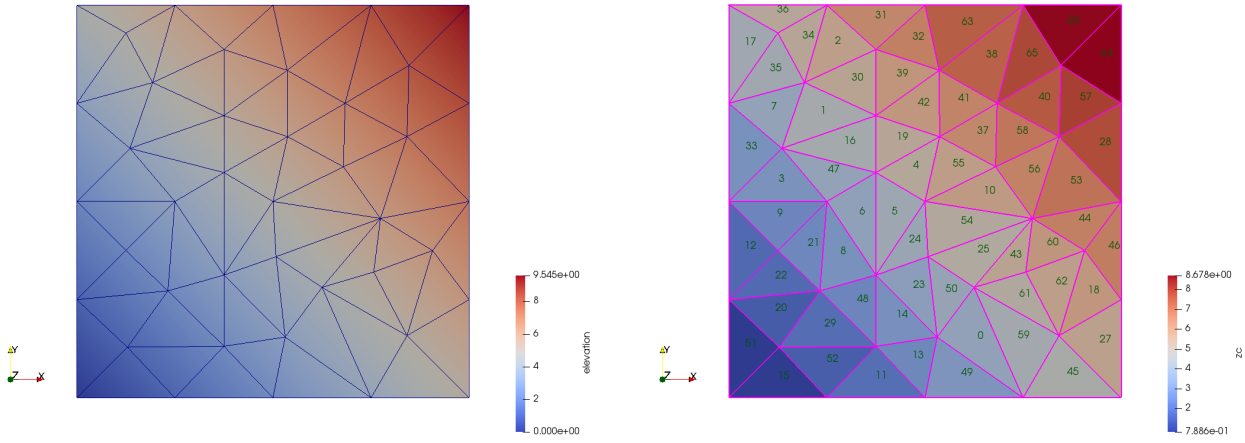


Note that the areas have been obtained by means of Heron's formula[1].

We then proceed by assigning the nodes the following elevation:

```
for i in range(0,N):
    z[i]=x[i]/10000+y[i]/11000
```

In this case we wish to establish a clear slope for the entire domain so as to obtain a model that is easy to debug, i.e. with a clear highest element at one corner and a lowest element in the opposite corner.



[1] https://en.wikipedia.org/wiki/Heron's_formula

Left: nodal elevation, Right: zc elevation array.

Step 2 is somewhat cumbersome but not difficult. In essence, we loop over all element with loop index `iel`. For each element iel we must find the element that shares a face `iface` with it. We therefore again loop over all elements again using the loop index `jel` and all their faces via `jface`. If the nodes making iface and jface are identical then we have found that jel is a neighbour of iel through iface and we then set:

```
gnei[iface,iel]=jel
```

If no neighbour was found this means that the face resides on the boundary of the domain and we then assign

```
gnei[iface,iel]=iel
```

In the end the `gnei` array of size 2×`nel` contains the following values:

```
50 30 34  9 19  4  8  1 21  3 55 52 12 14 23 15 19 35 46  4 51  8 20 14 ...
49  7 30 47  5  6  5 35 48 12 54 11 22 11 48 52  1 36 62 42 29  9 21 50...
59 16 31 33 55 24 47 33  6 21 56 13  9 49 13 51 47 17 27 16 22 22 12 24...
```

Looking at the figures above we see that element 0 indeed has the neighbours 50,49,59; element 1 has neighbours 30,7,16; etc ...

Step 4 consists of creating an array that contains the list of elements ordered from the highest elevation to the lowest. This is carried out in 2 simple lines of code:

```
sorted_indices=zc.argsort()
sorted_indices=np.flip(sorted_indices[:])
```

We can then ask to print the list of all `zc` values as ordered with `sorted_indices`:

```
print(zc[sorted_indices])
```

This yields the following list of elevations:

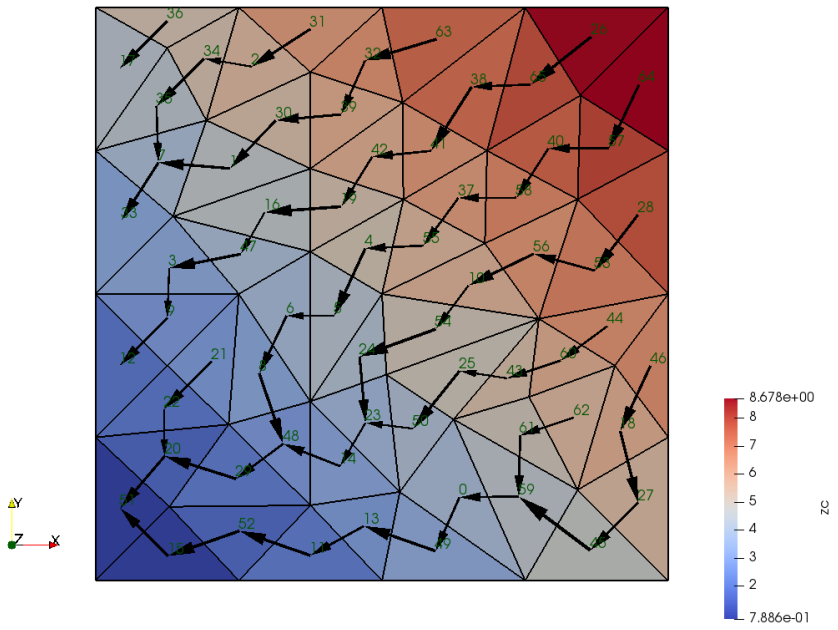
```
[8.67835429 8.64047551 7.90754376 7.73434388 7.63828038 7.38020001
 7.27131339 7.1985151  6.81720194 6.71087861 6.54905205 6.48554937
 6.47533275 6.41714392 6.33647163 6.24947391 6.202979   5.84086306
 5.80808307 5.77480023 5.76749316 5.59704537 5.51820154 5.46881474
 5.43960657 5.35480475 5.22180354 5.19653103 5.1113061  5.08682345
 5.06335745 4.98820004 4.9637227  4.85726269 4.84061664 4.5593502
 4.44070144 4.39626683 4.36048657 4.2913689  4.26790291 4.18357684
 4.08025568 3.97791121 3.86304993 3.85649411 3.83704947 3.76691017
 3.60859303 3.20257978 3.11988696 3.08200818 3.07224026 3.04159493
 2.77644341 2.75027056 2.72186147 2.70833333 2.07386364 2.03319022
 1.95481602 1.91287879 1.6503439  1.58811446 0.82648026 0.78860148]
```

Having verified that the ordering does work as intended, we can turn to step 5, i.e. `sorted_gnei`. DO we really need it ? as it turns out, there may be a pb with the original code `receiver = sorted_gnei(min_index(iel),i` so that this line was replaced with a line that only uses gnei.

for step 6, we must find behind which face 0, 1 or 2 the lowest neighbour is situated. The information is stored in `min_index`. The following algorithm is not particularly efficient nor elegant but it does just that:

```
for iel in range(0,nel):
    zc_0=zc[gnei[0,iel]]
    zc_1=zc[gnei[1,iel]]
    zc_2=zc[gnei[2,iel]]
    if zc_0<zc_1 and zc_0<zc_2:
       min_index[iel]=0
    if zc_1<zc_0 and zc_1<zc_2:
       min_index[iel]=1
    if zc_2<zc_0 and zc_2<zc_1:
       min_index[iel]=2
```

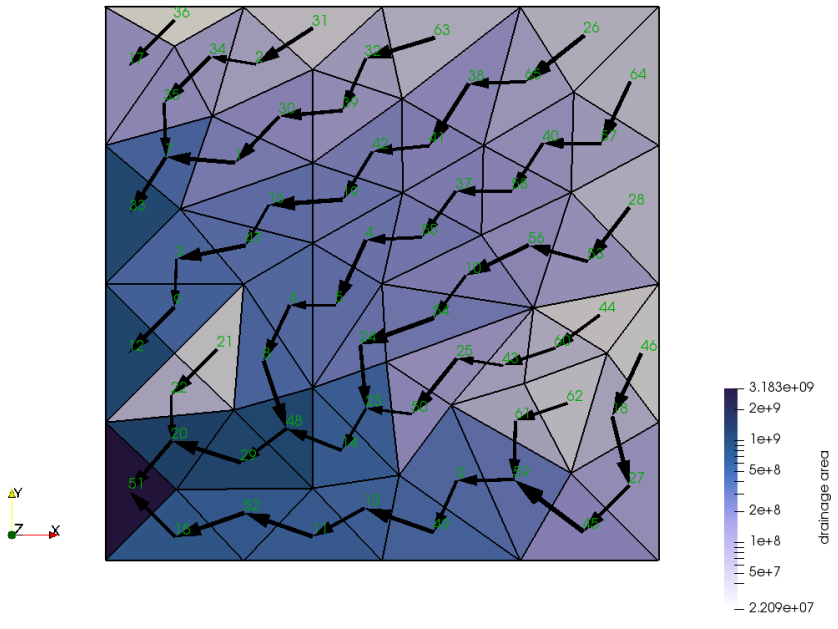
This information is exported to a vtu file so that we can visualise this bucket passing algorithm in action:



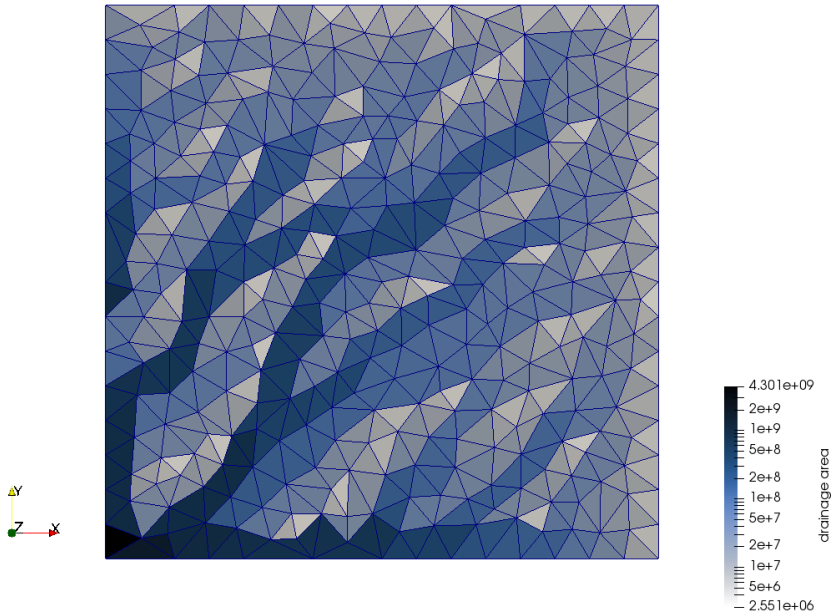
Finally steps 7 and 8 consist in identifying donors and receivers. We loop on elements from the highest to the lowest with loop index iel. The donor is the element in question (`sorted_indices[iel]`) and it gives to its lowest neighbour identified by `gnei[min_index[donor],donor]`. The result after the loop has been completed is the accumulated surface area that "drains" to each element in the landscape.

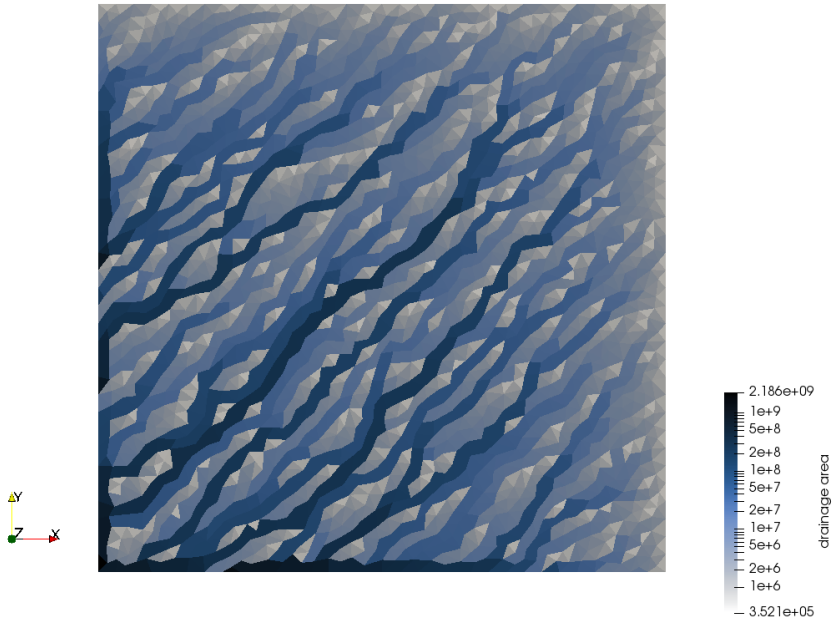
```
A[:]=area[:]
for iel in range(0,nel):
    donor = sorted_indices[iel]
    receiver=gnei[min_index[donor],donor]
    A[receiver] += A[donor]
```

In the end the field A looks like:



When increasing resolution one gets:





Computing the slope

In order to compute $\partial h/\partial x$ and $\partial h/\partial y$, one could resort to using basis functions derivatives in the middle of each element, but we instead use a different approach. Since P_1 basis functions are used, it means that inside an element the elevation h is represented by

$$h(x, y) = a + bx + cy$$

where the a, b, c coefficients are such that

$$h_0 = h(x_0, y_0) = a + bx_0 + cy_0 \tag{140.4}$$
$$h_1 = h(x_1, y_1) = a + bx_1 + cy_1 \tag{140.5}$$
$$h_2 = h(x_2, y_2) = a + bx_2 + cy_2 \tag{140.6}$$

where $0, 1, 2$ are the local numbers of the vertices. The equation above can be rewritten

$$\begin{pmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} h_0 \\ h_1 \\ h_2 \end{pmatrix}$$

After solving this system in each element, we have

$$\frac{\partial h}{\partial x} = b \qquad \frac{\partial h}{\partial y} = c \qquad S = \sqrt{b^2 + c^2}$$

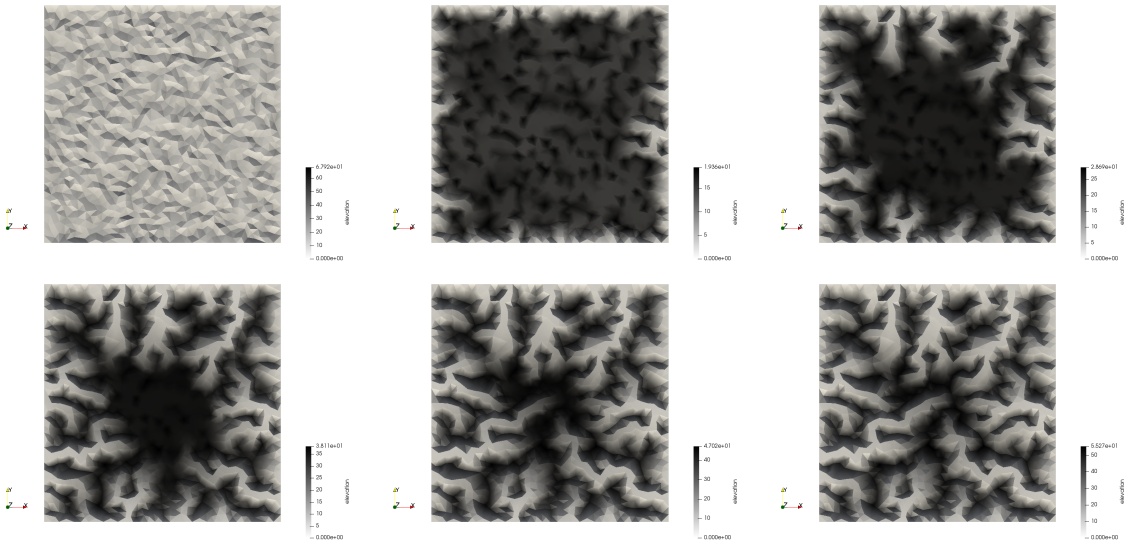
The calculations can be be benchmarked by assigning a simple linear planar topography and verifying that all elements are assigned the same correct slope.

Why lumping ?

As explained in the book, "mass lumping (or diagonalization) has been applied to the element mass matrix MM, which is performed to suppress numerical oscillations that can arise if the diffusivity in becomes too small. While there are no strict rules on how to perform mass lumping, here it is done by summing each column of MM separately, putting the result on the diagonal, and zeroing out all off-diagonal terms" as follows:

```
if lumping:
   MM[0,0]+=MM[0,1]+MM[0,2]  ; MM[0,1]=0  ; MM[0,2]=0
   MM[1,1]+=MM[1,0]+MM[1,2]  ; MM[1,0]=0  ; MM[1,2]=0
   MM[2,2]+=MM[2,0]+MM[2,1]  ; MM[2,0]=0  ; MM[2,1]=0
```

Experiment from Simpson's book (`experiment=0`)



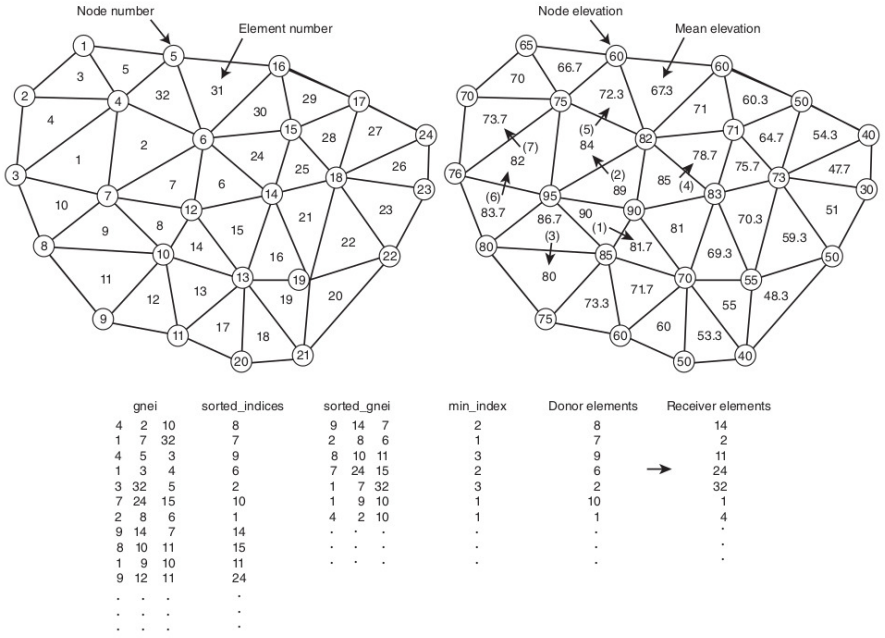


Figure 9.3 Illustration of drainage area calculation algorithm for a small triangular mesh. The mesh on the left shows global node (encircled) and element numbers. The mesh on the right shows node elevations (encircled) and element elevations computed as the average of the three triangle vertices. The arrows show the order (indicated with a bracketed number) and direction in which the cumulative drainage area is "passed" down the network (from donor to receiver) for part of the mesh. The Matlab variable `gnei` stores the three element neighbors for each element. The vector `sorted_indices` contains a list of elements, sorted on the basis of elevation, from highest to lowest. The array `sorted_gnei` contains the element neighbors, sorted according to the element list in `sorted_indices`. The vector `min_index` stores the local index (1, 2, or 3) of the lowest of the three neighboring elements for each element. The cumulative drainage area is redistributed progressively down the network from each (donor) element to its lowest neighbor (receiver), from the highest to the lowest element of the mesh.

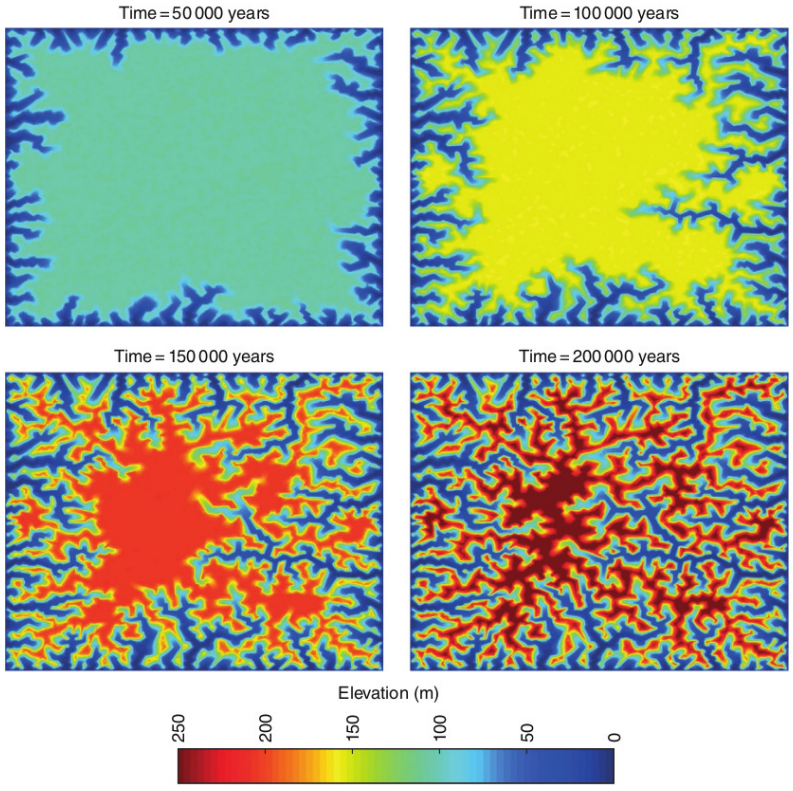
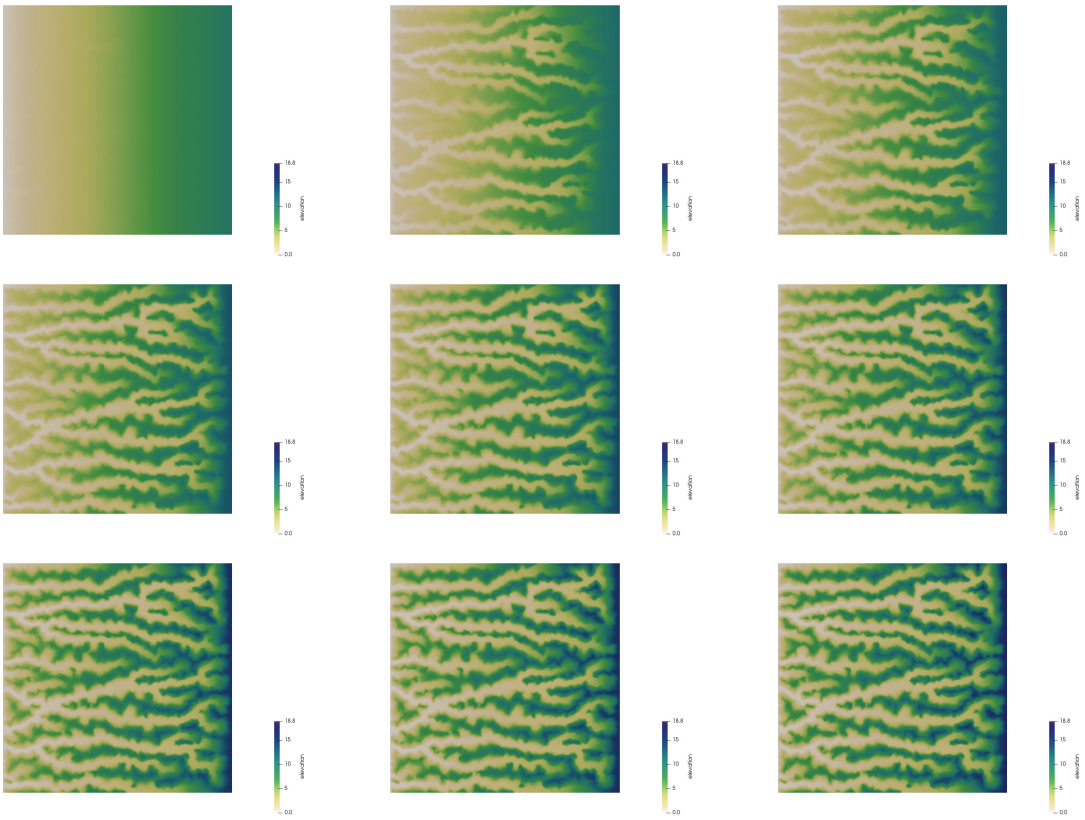
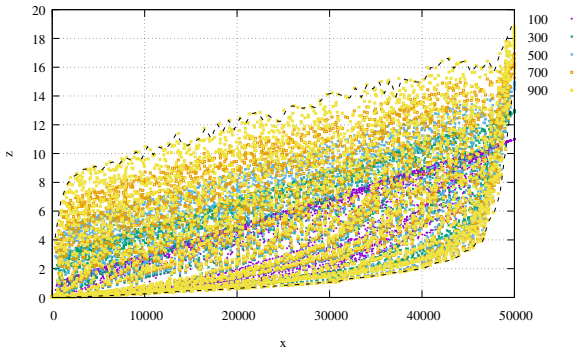


Figure 9.4 Topography after four different times based on the numerical solution of Equation 9.13 with a constant rock uplift rate and zero elevation prescribed on all boundaries. Results were computed with the script listed in Section 9.2 using a total of 19 968 linear triangular finite elements. Parameter values used were $w = 1$ mm year^{-1}, $c_0 = 3.171 \times 10^{-9}$ m^2 s^{-1}, $r = 2$, $m = 1$, $c = 3.171 \times 10^{-19}$ s^{-1}, and $\Delta t = 100$ years.

Taken from Simpson's book.

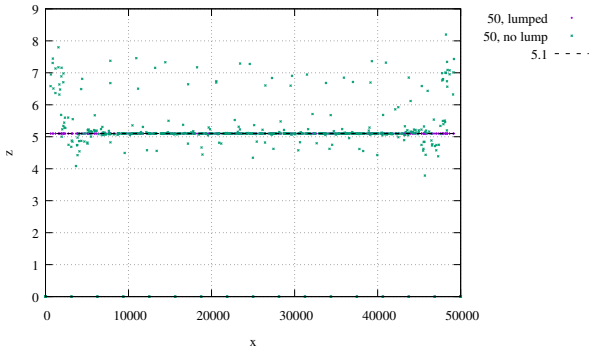
Single slope (`experiment=1`)



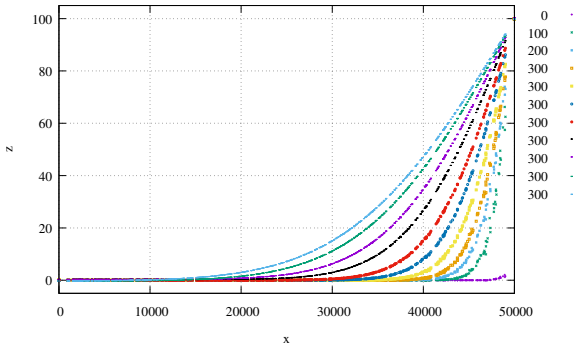


Benchmarking of uplift (`experiment=2`)

In this experiment we set the diffusion coefficient to zero and pin the boundary nodes to $h = 0$. The uplift rate is 1 mm yr^{-1}. As such we expect the elevation of the inside nodes to rise linearly with time. However, as explained before, we find that lumping the mass matrix is needed:



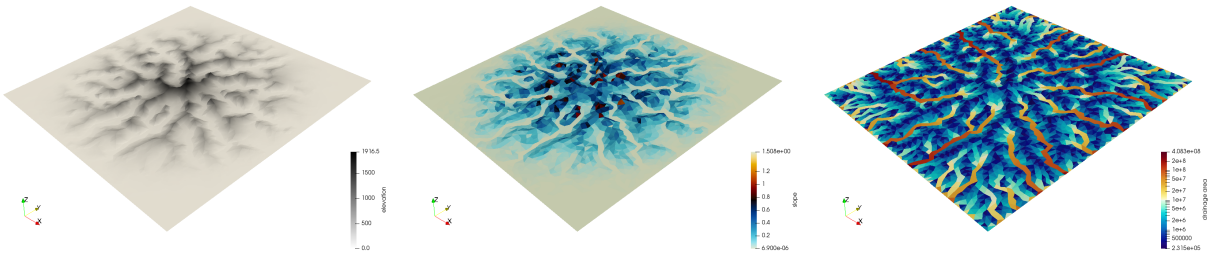
Benchmarking of linear diffusion (`experiment=3`)



Volcano-like uplift (`experiment=4`)

The domain is square of size $L_x = L_y = L = 50$ km.

$$w(x,y) = w_0 \left(1 - \frac{r}{L/2}\right) \qquad \text{for} \quad r = \sqrt{(x - L/2)^2 + (y - L/2)^2} < (L/2)$$

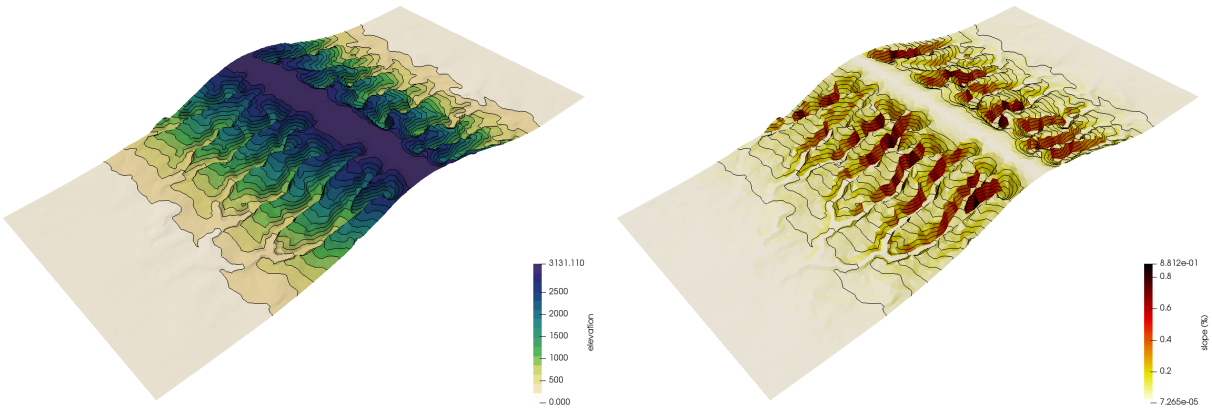


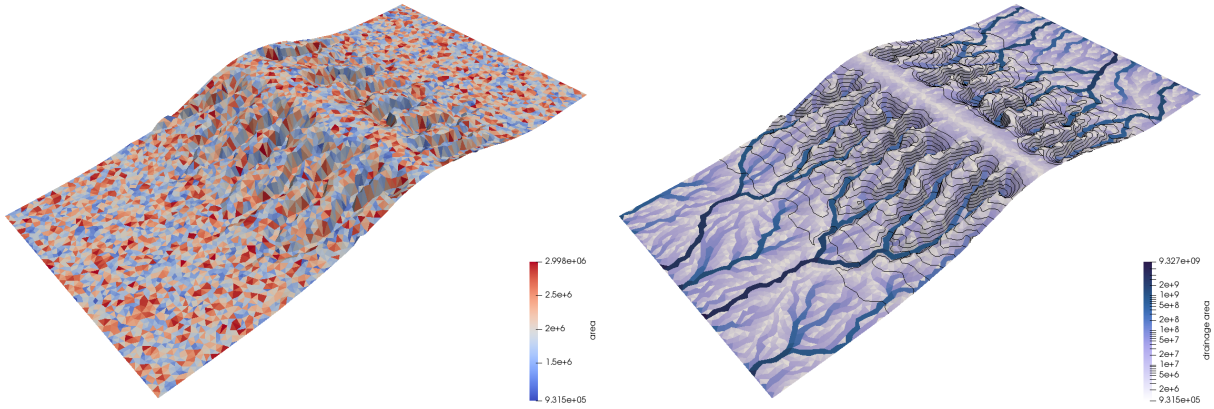
Orogen-like uplift (`experiment=5`)

The domain is a rectangle of size $L_x \times L_y = 200 \times 100$ km.

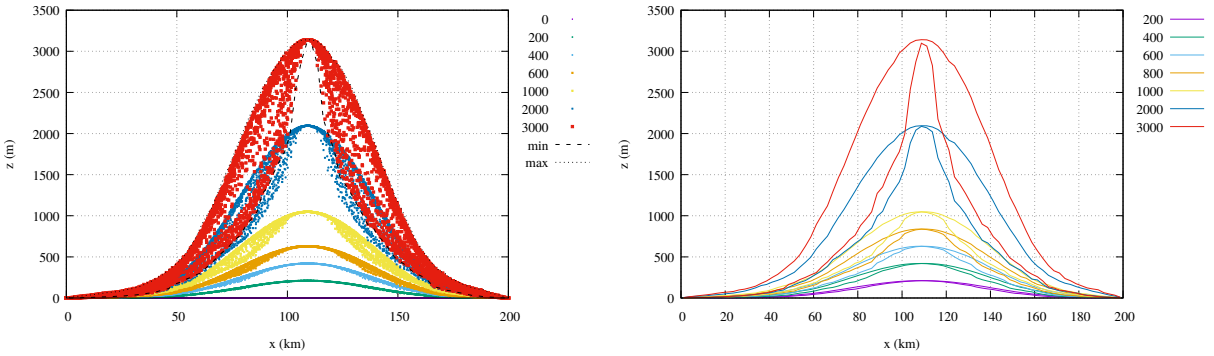
$$w(x,y) = w_0 \frac{x}{L_x} \left[\left(1 - \cos\frac{2\pi x}{L_x}\right)/2\right]^2$$

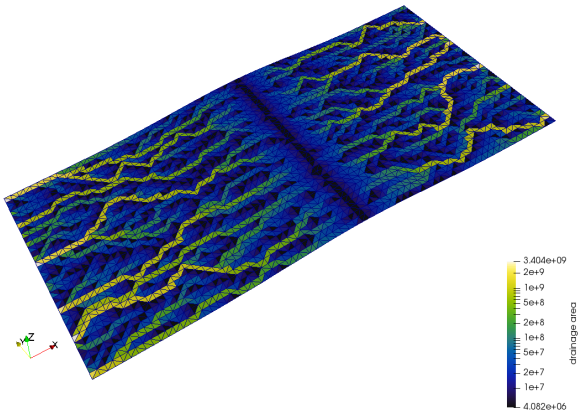
with $w_0 = 2$ cm yr^{-1}.





3000 time steps = 300kyr.

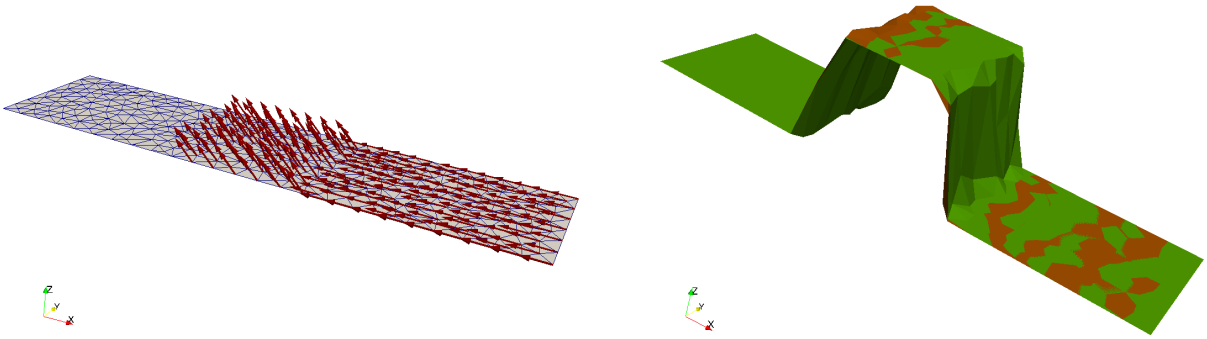




example of regular grid of quadrilaterals split in two triangles

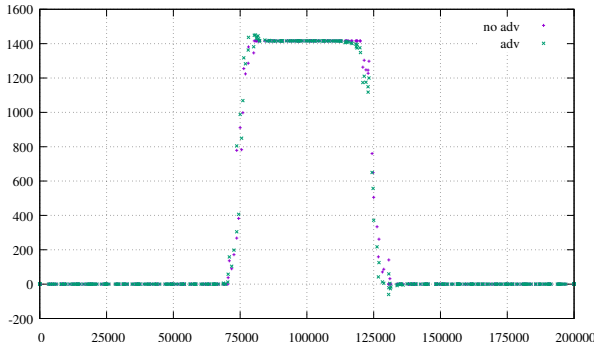
Crustal wedge with advection (`experiment=6`)

In this case Lx=200km, Ly=50km. Tectonic advection is included.



a) b)

Vertical scale x30. b) after 1000 timesteps = 100,000yr.



We find the hallmarks of numerical advection: under- and overrshoots.

To Do, open questions, future work?

- decide/homogenise colorscales for drainage area, slope, elevation...
- improve speed for gnei and for matrix build
- test/document Simpson's quadrature? 3 pts see table 7.2???
- CFL condition ?!
- benchmarks
- compute uplift
- compute catchment efficiently
- precipitation fct space coords?
- cylindrical uplift w, look at average slope
- avalanche rule, see page 244 of pelletier
- establish river path ?
- add nonlinear adv see pelletier ?
- what physics is in Cascade? Braun and Sambridge [35]
- inner lengthscale? one valley increase res?
- crank-nicolson ?

Relevant Literature:

Guy Simpson and Fritz Schlunegger. "Topographic evolution and morphology of surfaces evolving in response to coupled fluvial and hillslope sediment transport". In: *J. Geophys. Res.: Solid Earth* 108.B6 (2003). DOI: 10.1029/2002JB002162

Guy Simpson. "A dynamic model to investigate coupling between erosion, deposition, and three-dimensional (thin-plate) deformation". In: *J. Geophys. Res.: Earth Surface* 109.F2 (2004). DOI: 10.1029/2003JF000078

Guy Simpson. "Dynamic interactions between erosion, deposition, and three-dimensional deformation in compressional fold belt settings". In: *J. Geophys. Res.: Earth Surface* 109.F3 (2004). DOI: 10.1029/2003JF000111

Stone 141

The 'Beaumont' mantle temperature trick

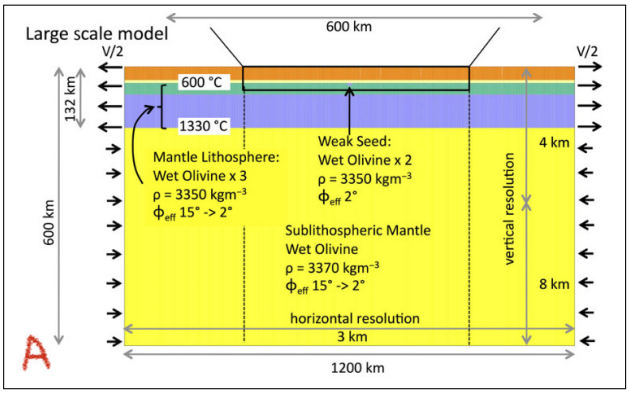


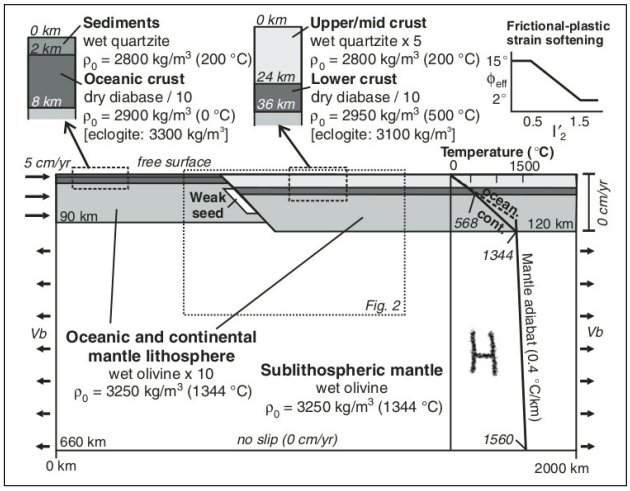
python_codes/fieldstone_141/text.tex

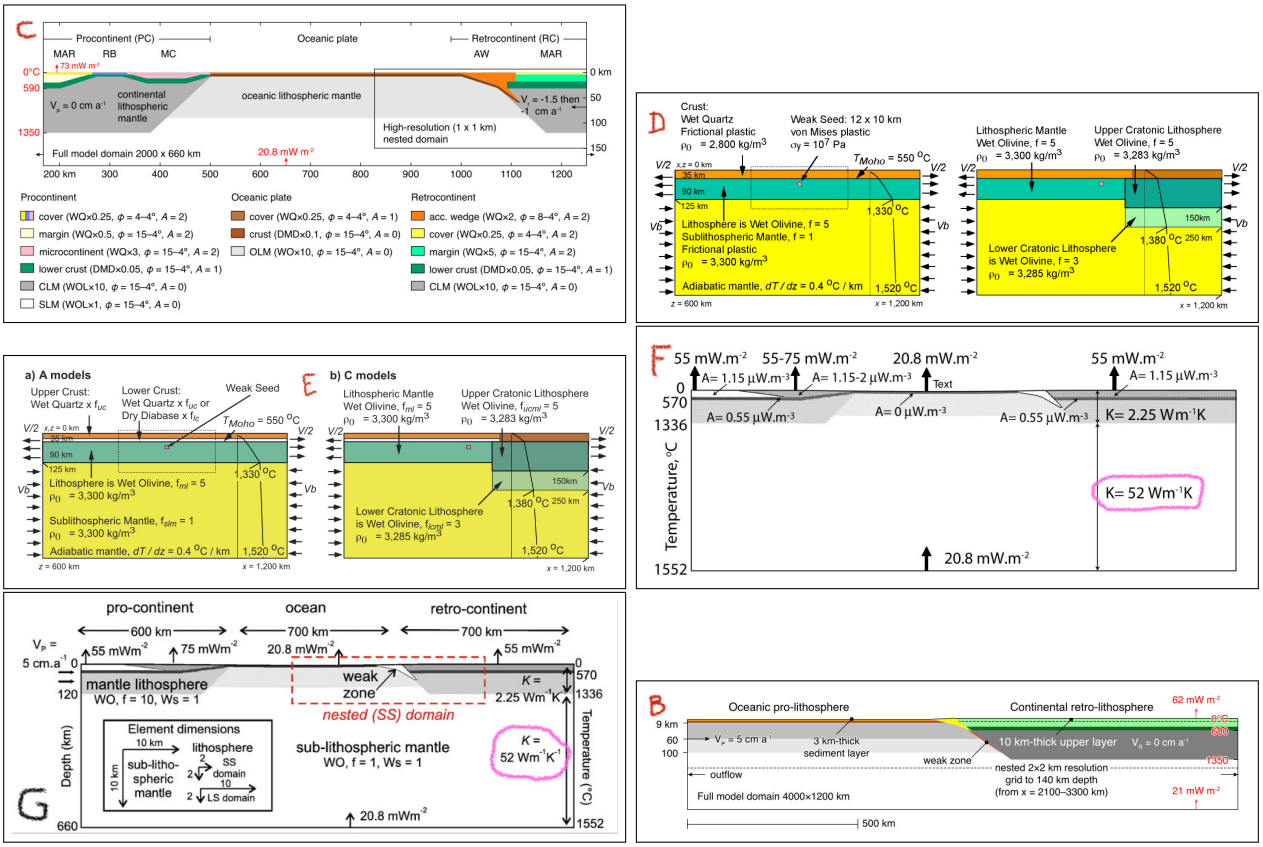
2D
box
Q1
analytical_benchmark
energy_equation
time_stepping

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_141`

The title of this STONE is voluntarily meant to be funny. Nobody calls what follows the 'Beaumont trick' but it has been prominent in many of his (and his collaborators') publications, all based on the SOPALE code. Let us look at the following figures from a few papers and focus on the initial temperature field.







A: Taken from Allen and Beaumont [3] (2015), B: Taken from Butler and Beaumont [60] (2017), C: Taken from Butler, Beaumont, and Jamieson [61] (2014). D: Taken from Huismans and Beaumont [162] (2011), E: Taken from Huismans and Beaumont [163] (2014), F: Taken from Warren, Beaumont, and Jamieson [346] (2008), G: Taken from Beaumont, Jamieson, Butler, and Warren [12] (2009), H: Taken from Currie, Beaumont, and Huismans [81] (2007).

What is common to all is (aside from margins, cratons, sediment layers, etc ...) the presence of a crust, a lithosphere and a mantle. The temperature at the top of the model is 0 °C, 550 °C at the base of the crust, $1330 - 1380$ °C at the base of the lithosphere (it depends on its thickness), and on some of the figures we see that the temperature is 1500 °C at the bottom of the domain (which is sometimes 600 or 660 km thick).

In between these depths the geotherm is linear (maybe parabolic in the crust where heat production is present, but not too relevant here).

At this stage it is worth noting that at the base of the model a heat flux value is prescribed (i.e. a Neumann boundary condition) of about 21 mW m^{-2}.

Another point worth noticing is the very large value of the heat conductivity $k = 52$ W m^{-1} K of the mantle in Warren, Beaumont, and Jamieson [346] (2008) (also in Warren, Beaumont, and Jamieson [347] and Warren, Beaumont, and Jamieson [348] - not shown here) and in Beaumont, Jamieson, Butler, and Warren [12] (2009) and other publications (see figure above). Indeed, we can look at the table of properties of all materials in the 2008 paper and confirm this is not a mistake in the figure:

Parameters	Units	Pro Interior Upper/Mid Crust	Pro Margin Upper/Mid Crust	Retro Upper/Mid Crust	Lower Crust	Oceanic Crust	Continental Mantle Lithosphere	Oceanic Mantle Lithosphere	Sublithospheric Mantle
Mechanical									
Thickness	km	24	0–24	24	12	8	84	82	to 660 km depth
Ref ρ	$kg.m^{-3}$	2800	2800	2800	2950	2900	3250	3250	3250
Ref ρ T	K	473	473	473	773	273	1609	1609	1609
ρ HP	$kg.m^{-3}$	2850	2850	2850	3100	3300	n/a	n/a	n/a
ρ UHP	$kg.m^{-3}$	2900	2900	2900	3100	3300	n/a	n/a	n/a
ϕ_{eff}	degrees	15-2	15-2	15-2	15-2	15-2	15-2	15-2	15-2
Cohesion	MPa	20	20	20	0	0	0	0	0
Flow law		wet quartzite	wet quartzite	wet quartzite	dry Maryland diabase	dry Maryland diabase	wet olivine	wet olivine	wet olivine
Flow law reference		*Gleason and Tullis* [1995]	*Gleason and Tullis* [1995]	*Gleason and Tullis* [1995]	*Mackwell et al.* [1998]	*Mackwell et al.* [1998]	*Karato and Wu* [1993]	*Karato and Wu* [1993]	*Karato and Wu* [1993]
f		5	1	5	0.1	0.1	10	10	1
W_s		10	10	10	10	10	1	1	1
ε		5–10	5–10	5–10	5–10	5–10	1	1	1
n		4	4	4	4.7	4.7	3	3	3
B*[a]	$Pa.s^{1/n}$	2.92×10^6	2.92×10^6	2.92×10^6	1.91×10^5	1.91×10^5	1.92×10^4	1.92×10^4	1.92×10^4
Q	$kJ.mol^{-1}$	223	223	223	485	485	430	430	430
V*	$m^3.mol^{-1}$	0	0	0	0	0	1.0×10^{-5}	1.0×10^{-5}	1.0×10^{-5}
Thermal									
Heat capacity	$m^2.K.s^{-2}$	750	750	750	750	750	1250	1250	1250
Thermal conductivity	$W.m^{-1}.K^{-1}$	2.25	2.25	2.25	2.25	2.25	2.25	2.25	52
Thermal diffusivity	$m^2.s^{-1}$	1.0×10^{-6}	1.0×10^{-6}	1.0×10^{-6}	1.0×10^{-6}	0.6×10^{-6}	0.6×10^{-6}	0.6×10^{-6}	1.4×10^{-5}
Thermal expansion	K^{-1}	3.0×10^{-5}	3.0×10^{-5}	3.0×10^{-5}	3.0×10^{-5}	3.0×10^{-5}	3.0×10^{-5}	3.0×10^{-5}	3.0×10^{-5}
Heat production	$\mu W.m^{-3}$	1.15	2	1.15	0.55	0	0	0	0

[a] B* (plane strain stress pre-exponential factor) = $2^{(1-n)/n}3^{-(n+1)/2n}A_{uni}^{-1/n}$, where A_{uni} is the uniaxial pre-exponential factor.

Taken from Warren, Beaumont, and Jamieson [346] (2008).

The purpose of this STONE is to answer these two questions: why did the authors set the conductivity of the mantle to such high values and why did they use heat flux boundary conditions?

Let us focus for a minute on Beaumont, Jamieson, Butler, and Warren [12] (figure 'G'): at 120 km depth the temperature is 1336 °C while it is set to 1552 °C at the base of the model. The temperature gradient is then

$$\frac{\Delta T}{\Delta z} = \frac{1552 - 1336}{660 - 120} = 0.4 \text{ °C km}^{-1}$$

The heat flux is then $|q_y| = k\Delta T/\Delta y = 52 * 0.4 = 20.8$ mW m^{-2}, which is exactly the value prescribed at the bottom. Turning now to Huismans and Beaumont [163] (figure 'E') or Huismans and Beaumont [162] (figure 'D'): The temperature difference is $\Delta T = 1520 - 1330 = 190$ °C over $\Delta y = 600 - 125 = 475$ km, so that once again the temperature gradient is 0.4 °C km^{-1}.

These values for the temperature gradient in the upper mantle do agree well with adiabatic temperature profiles in the (upper) mantle, see for example Katsura, Yoneda, Yamazaki, Yoshino, and Ito [177].

In what follows we'll focus for simplicity on the setup of Huismans and Beaumont [162] (2011): a 35km crust, a 90km lithosphere, a 600km deep domain, with no variation in the horizontal direction.

Remarks about the code

The first obvious thing to be mentioned is that this is inherently a 1D problem in the y-direction. However, for practical reasons I have reused an existing 2D code. As such, we need to specify a lateral extent L_x to the domain which can be only a few kilometers.

The code is a FEM code relying on bi-linear elements (Q_1). Both Dirichlet and Neumann boundary conditions are implemented. Before we use it to investigate the problem highlighted above, we need to make sure it works as intended, i.e. we need to carry out analytical benchmarks.

Each element is assigned a heat conductivity k, heat capacity C_p and density ρ. The timestep value is computed by means of a CFL condition for diffusive processes, i.e. $dt = 0.75 * \min(h_x, h_y)^2 / \max(\kappa)$ where h_x, h_y is the size of an element, and κ is the heat diffusivity given by $\kappa = k/\rho C_p$. The computed timestep is capped to 10^5 yr.

The heat flux vector is computed in each corner of each element using the elemental heat conductivity and then averaged out on the nodes.

All results are exported to ascii and vtu files. Note that the latter have been stretched by a factor 20 in the horizontal direction so as to facilitate visualisation in paraview.

Note that if temperature is prescribed at the bottom of the domain then the heat flux is unconstrained. On the other hand, if the heat flux is prescribed on the boundary then it is the temperature that is unconstrained.

benchmark #1 (`test=1`)

The domain is 5×100 km. Temperature boundary conditions are imposed at the top and bottom with $T_{top} = 0$ °C and $T_{bottom} = 100$ °C. The initial temperature $T(x, y, t = 0)$ is set to T_{top}.

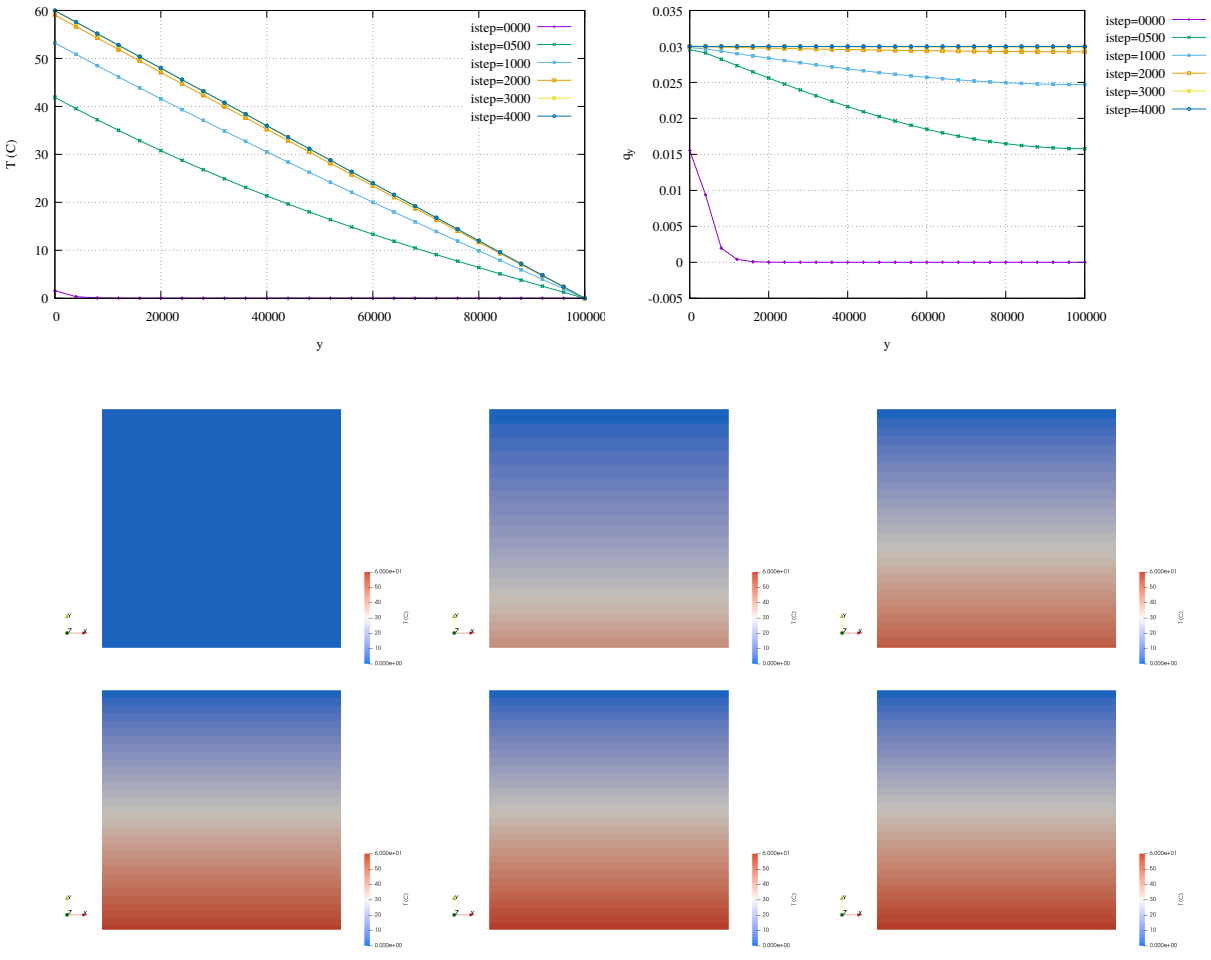
A time-dependent analytical solution exists for this problem and is given ...

to do !

benchmark #2 (`test=2`)

The domain is 5×100 km. Temperature boundary conditions are imposed at the top with $T_{top} = 0$°C and a heat flux is prescribed at the bottom $q_{bottom} = 0.03$ W m^{-2}. The initial temperature $T(x, y, t = 0)$ is set to T_{top}.

The expected steady state solution is such that the temperature is $T(x, y = L_y, t) = T_{top}$ and $q_y(x, y, t) = q_{bottom}$. Since then $|q_y| = k\Delta T/L_y$, then we expect a temperature at the bottom of 60 °C.



Time evolution of the temperature field.

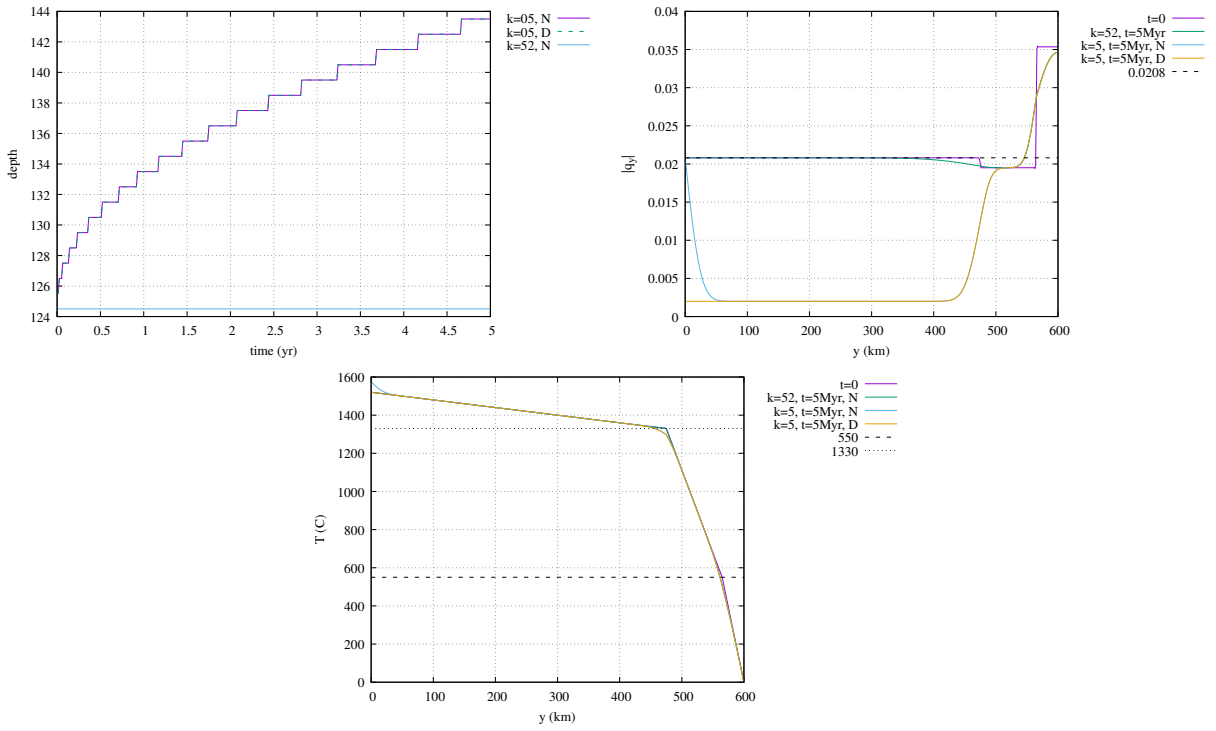
Real application (`test=0`)

We want to track the depth of the 1330 °C isotherm. We then run three models:

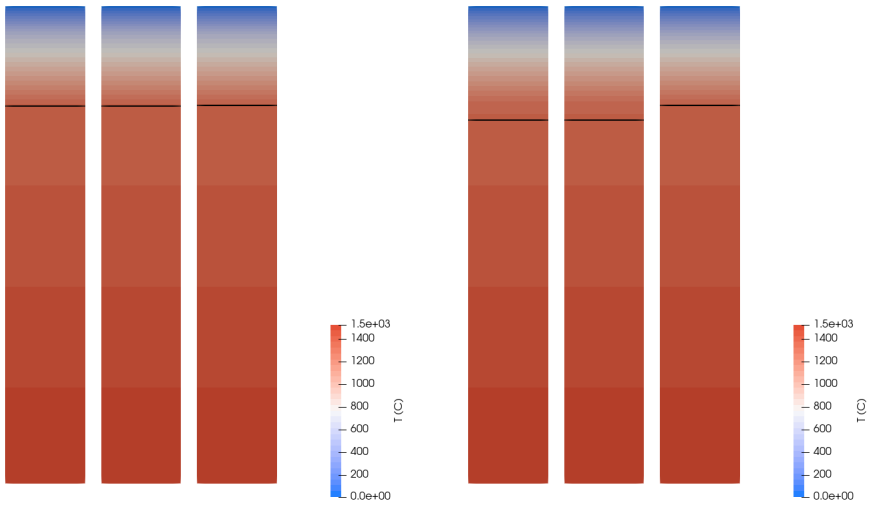
- $k = 5$ in the mantle, Neumann b.c. at the bottom with $q_{bottom} = 0.0208$
- $k = 5$ in the mantle, Dirichlet b.c. at the bottom with $T_{bottom} = 1520$

- $k = 52$ in the mantle, Neumann b.c. at the bottom with $q_{bottom} = 0.0208$

Note that in the case $k = 52$ the time step dt is essentially 10x smaller than in the $k = 5$ case so that the code needs to run longer.



We see that the depth of the 1330 isotherm increases with time if $k = 5$ values are used. Using a higher k value prevents that.



Left: temperatures at t=0; Right: temperatures at t=5Myr. from left to right: k05D, k05N, k52N.

Stone 142

Calculating stress and strain rate in 2D viscous inclusion-matrix systems



python_codes/fieldstone_142/text.tex

2D
box
Q2Q1
P2pPm1
sparse_matrix
numerical_benchmark
stokes_equation
use_mesher
nonlinear
paraview_output

 Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_142`

This STONE is based on Halter, Macherel, and Schmalholz [147] (2022) published in the Journal of Structural Geology. As for every STONE aiming at reproducing results off a publication I here include de abstract of the article:

Computer-based numerical solutions of geomechanical problems are important to understand the processes forming rock structures as well as to quantify the associated pressure, stresses and strain rates. However, the development of such computer programs and the underlying mathematical methods are commonly not taught in a standard structural geology curriculum. Here, we present a simple computer program to calculate the stress, pressure, velocity and strain rate fields for two-dimensional (2D) viscous inclusion-matrix systems under pure shear and simple shear. The main aim of our contribution is to explain this computer program in a simple and transparent way, so that it can be used for introductory courses on geomechanical numerical modelling in structural geology. We present the governing equations of 2D viscous deformation and program the equations in the same order and style, so that the equations are still recognizable in the computer program. The computer program can treat stiff and weak inclusions of any shape and considers both linear and power-law viscous flow laws. We present numerical calculations for various inclusion-matrix scenarios. The program is written with the software MATLAB, is provided as supplementary material, and can also be run with the freely available software GNU Octave.

In the paper the effective viscosity for a power-law viscous fluid, termed here η_{PL}, can be written as

$$\eta_{PL} = \eta_L \left(\frac{\tau_e}{\tau_R} \right)^{1-n}$$

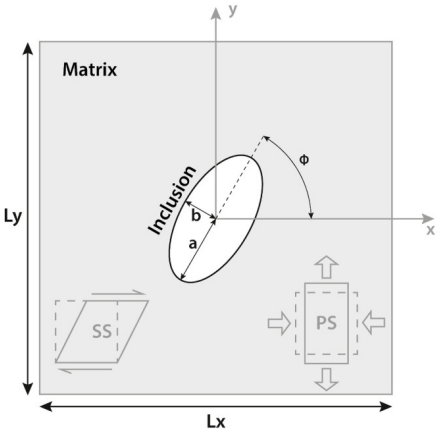
where η_L is the linear viscosity, τ_R is a constant reference stress, n is the stress exponent (which for rocks is $\geq$1), and τ_e is the effective deviatoric stress.

Often, a combination of linear viscous and power-law viscous flow is assumed for ductile rocks. The viscosity for such combined flow law can be given by the pseudo-harmonic mean of η_L and η_{PL}:

$$\eta_C = \left(\frac{1}{\eta_L} + \frac{1}{\eta_{PL}} \right)^{-1}$$

The effective viscosity, η in the momentum conservation equation can, hence, either be equal to η_L, η_{PL} or η_C depending on whether a linear viscous, power-law viscous or combined flow law, respectively, is used.

The paper relies on the Finite Difference method to solve the Stokes equations but we will here rely on the Finite Element method based on an unstructured mesh.



Taken from [147]. Model configuration and boundary conditions. An elliptical inclusion of different viscosity than its surrounding matrix is located in the center of a square box of size Lx by Ly. The ellipse has a semi-major axis a and semi-minor axis b and is rotated by angle ϕ from the horizontal axis. Both simple (SS) and pure (PS) shear boundary conditions can be applied.

Boundary conditions are either a far-field pure shear velocity field at the model boundaries or a simple shear velocity field. Hence, both the horizontal and vertical components of the velocity are defined as boundary conditions at the model boundaries.

The model domain consists of a matrix in which stiffer or weaker inclusions are embedded. As initial guess for the unknown velocities, which is required for the first iteration step, the authors assume that the velocity field in the model domain corresponds to either a homogeneous pure shear or homogeneous simple shear velocity field, depending which far-field velocity boundary conditions are applied.

Comments about the article

A few remarks:

- The authors also state that "The pressure values at the model boundaries remain zero." uh??!
- The viscosity field is smoothed but this is never mentioned in the paper!

```
for smo=1:2; Ii  = [2:nx-1]; Ij  = [2:ny-1]; % Smoothing of the initial viscosity field
    ETA(Ii,:)    = ETA(Ii,:) + 0.4*(ETA(Ii+1,:)-2*ETA(Ii,:)+ETA(Ii-1,:));
    ETA(:,Ij)    = ETA(:,Ij) + 0.4*(ETA(:,Ij+1)-2*ETA(:,Ij)+ETA(:,Ij-1));
end
```

  This looks like a discretised diffusion operator...

- no real benchmarking, no error convergence on SolVi, instead: eyeball norm

- no solver in code ... why use Richardson rather than backslah operator of matlab? This finally explained in Discussion but this should have been addressed earlier in the paper since it is rather uncommon.
- no comment on viscous relaxation in fig 7 ?

  ```
  ETA_PL = exp(log(ETA_PL)*0.5+log(ETA_PL_it)*0.5);
  ```

- in the Octave code:

  ```
  subplot(224),pcolor(X,Y,DXX/(2*eta_B*D_B)),shading interp,axis equal,axis tight,colorbar,col
           title(['D) Horizontal strain rate [ ], D_{xx}/D_B']), xlabel('Width [ ]')
  ```

  We see that there is a mismatch between what is plotted and the legend (`DXX/(2*eta_B*D_B)` vs. `D_{xx}/D_B`)
- As we will see in the ellipse inclusion case, the code fails to converge for modest viscosity contrasts.
- authors state: Considering more irregular geometries is important, because "the geometrical form of the competent inclusion clearly has an influence on the pattern of stress orientations in the surrounding medium". Yet not a single stress or principal stress components calculations in the wole paper?
- 'spacial' ? 'quadratic model domain' ?
- fig 5: why change the radius inside the figure? why viscosity contrast at 100, not 1000 ?
- The provided code does not allow to reproduce experiments directly. Only ellipse.
- fig 9: resolution? applied velocity ? "Except for the circular inclusion, all ellipses possess the same aspect ratio." Which is equal to ?
- nothing about convergence pattern ? 1000's of iterations ? round off errors.
- about fig 10: 'The value of Dxx inside the garnet inclusion is zero, indicating that the inclusion is rigid and not deforming.' No, Dxx cannot be and is not zero.
- in the corrigendum only ellipse figures are corrected, not all the other ones so that the figs in the paper are useless to us when it comes to nonlinear rheologies as they cannot be used for comparison/benchmarking.
- often in figure captions: 'The arrows display the velocity field, they are not to scale and only for direction.' However looking at the figures arrows have various lengths which reflect the velocity field.
- in some plots, Dxx is described as 'horizontal strain rate'. Not a common name imo

Important remark: this stone was written in 2023. I sent my comments to the authors and in 2024 they published a second corrigendum acknowledging my comments and fixing the issue (`https://doi.org/10.1016/j.jsg.2024.105071`). What follows has not been adapted since.

Comments about the stone

My code solves the incompressible Stokes equations with the Finite Element Method on an unstructured grid that is built so that it fits the shape of the inclusion(s). $P_2 \times P_1$ or Crouzeix-Raviart elements can be used. The Delaunay triangulation is explained in STONE 131, and some of the tools are also used in STONE 96 and 143. In what follows I focus only on the Newtonian cases.

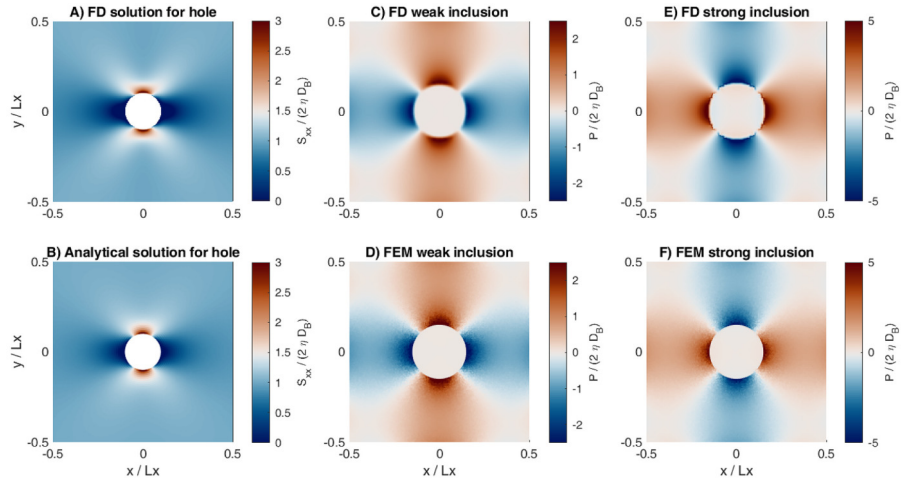
I also make use of the ASPECT code[1] and the prm file is provided in results/case1/aspect. This experiment is also a cookbook[2] in ASPECT .

[1] `https://github.com/geodynamics/aspect`

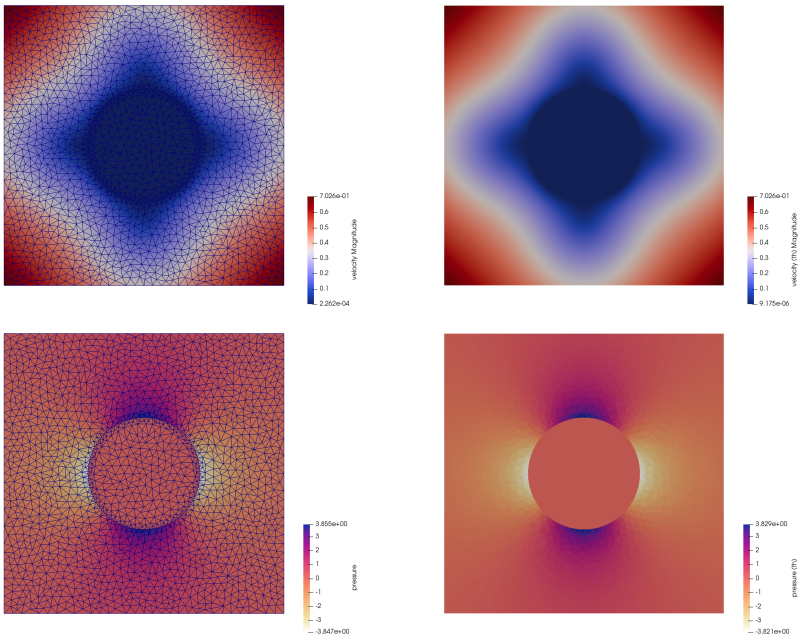
[2] `https://aspect-documentation.readthedocs.io/en/latest/user/cookbooks/cookbooks/inclusions/doc/inclusions.html`

Case 0: Circular inclusion and comparison with analytical solution

The boundary conditions are pure shear extension. This is the well known SolVi benchmark that has been already encountered in STONE 7,120.



Taken from [147]. Panels A) and B): Comparison of the normalized horizontal total stress field generated with the presented iterative finite difference (FD) program against an analytical solution for a circular hole within a matrix subjected to horizontally extensional pure shear. The hole in the FD program is approximated by a viscosity contrast of 1000. The hole radius is 0.1 times the model width. Panels C) to F): Comparison of the normalized pressure field generated with the presented iterative FD program against a tested finite element method (FEM) program (Schmalholz and Schmid, 2012) for both a weak and a strong circular inclusion under horizontally compressive pure shear. The regions of high pressure (red color) and the low pressure (blue color) are inverted from the weak inclusion to the strong inclusion setup. The viscosity of the inclusion is 100 times lower, respectively higher, than the surrounding matrix viscosity. The circle radius is 0.15 times the model width.

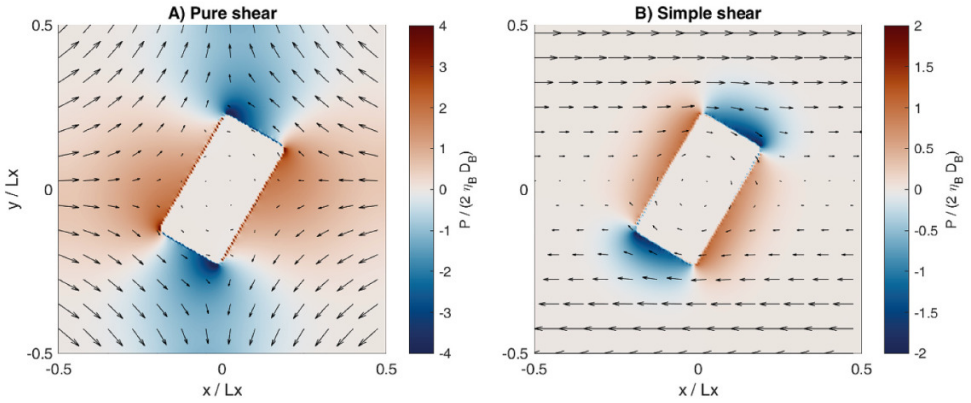


Left: solution using Crouzeix-Raviart elements; Right: analytical solution.

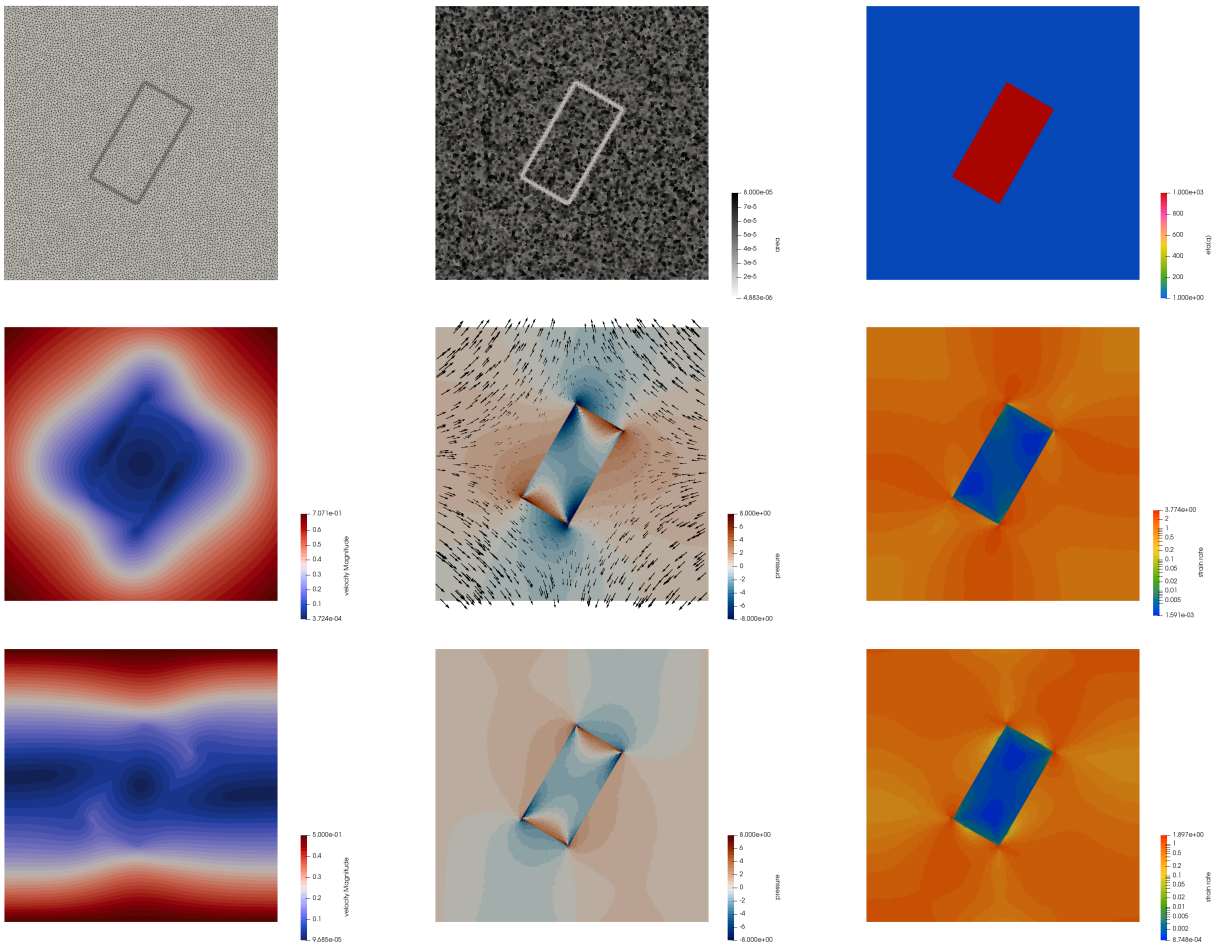
This proves that the code is correct. One should do a proper error analysis as a function of resolution but this has been done before in many other stones (see STONE 7 for example).

Case 1: Strong rectangular inclusion under pure shear and simple shear

The authors state: "For illustration purposes, we calculated the pressure fields for a strong and inclined rectangular inclusion for pure shear and simple shear. For simple shear, the pressure variations are more localized around the inclusion compared to the pressure variations for pure shear. Also, for simple shear the velocity field indicates a rotational flow of the inclusion with a clock-wise sense, in agreement with the applied simple shear."

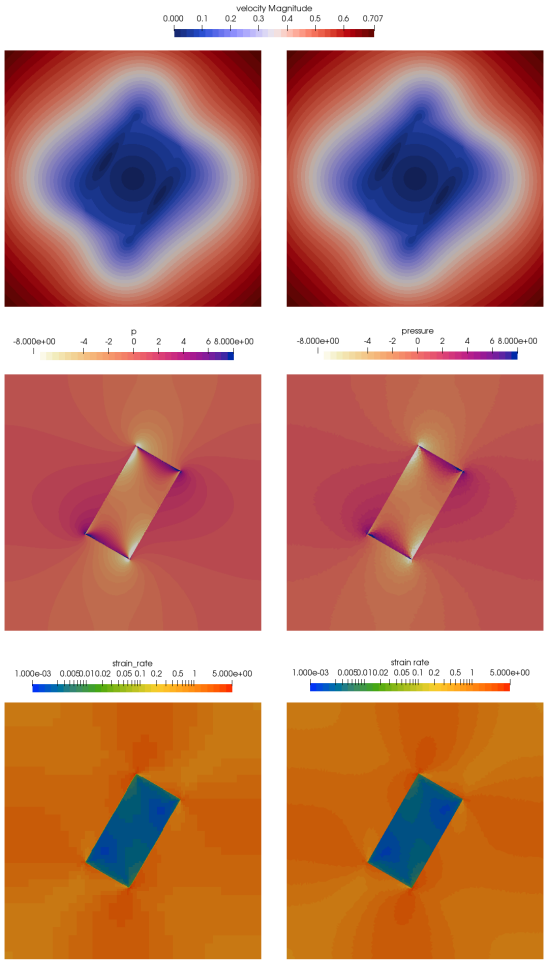


Taken from [147]. Comparison between horizontally compressive pure shear and horizontal simple shear boundary conditions for a strong (viscosity contrast of 1000) rectangular inclusion. Regions of pressure accumulation are depicted in red, pressure shadow zones in blue. The arrows display the velocity field, they are not to scale and only for direction. The long side of the rectangle is twice the size of its short side and corresponds to 0.4 times the model width. The long side of the rectangle is rotated by an angle of 60 from the horizontal axis.



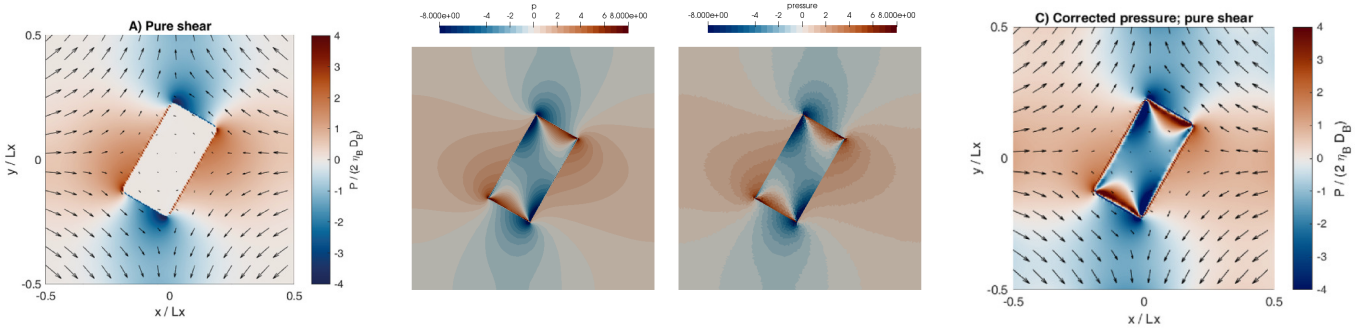
In order to favor comparison with the figure of the paper I for once use a different pressure colorscale ('vik') than normal. Second line is Pure Shear (PS), third line is simple shear (SS).

To validate the results of this stone further, I have run this model with the ASPECT code, and my results agree nicely with those of ASPECT :



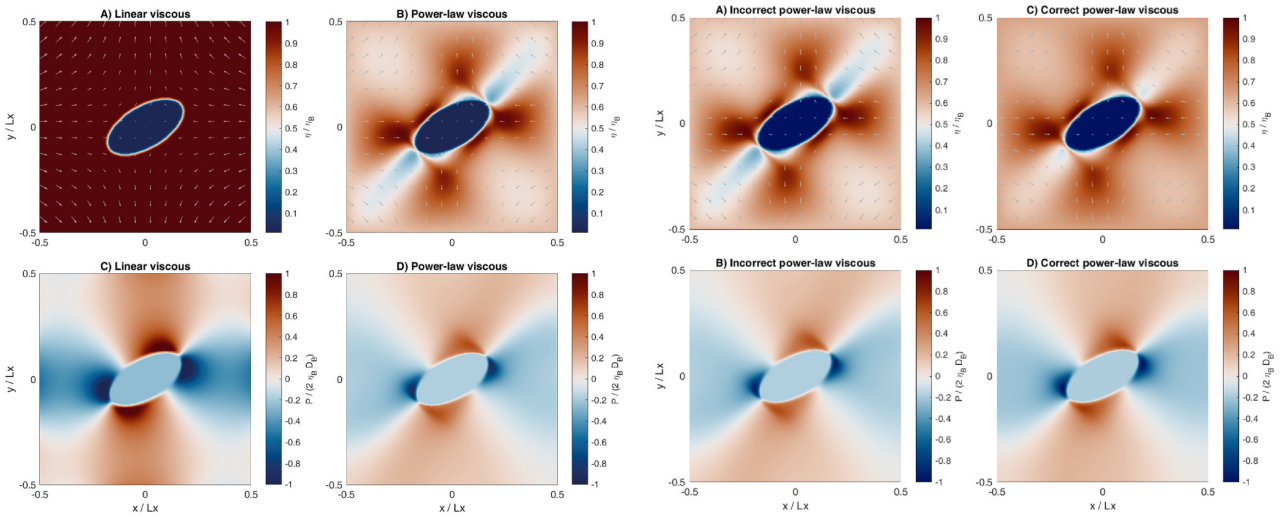
Left: ASPECT results; Right: fieldstone results.

Plotting the ASPECT and fieldstone results with a similar color scale yields the unescapable conclusion that the pressure field in the figure of the paper is wrong inside the inclusion:



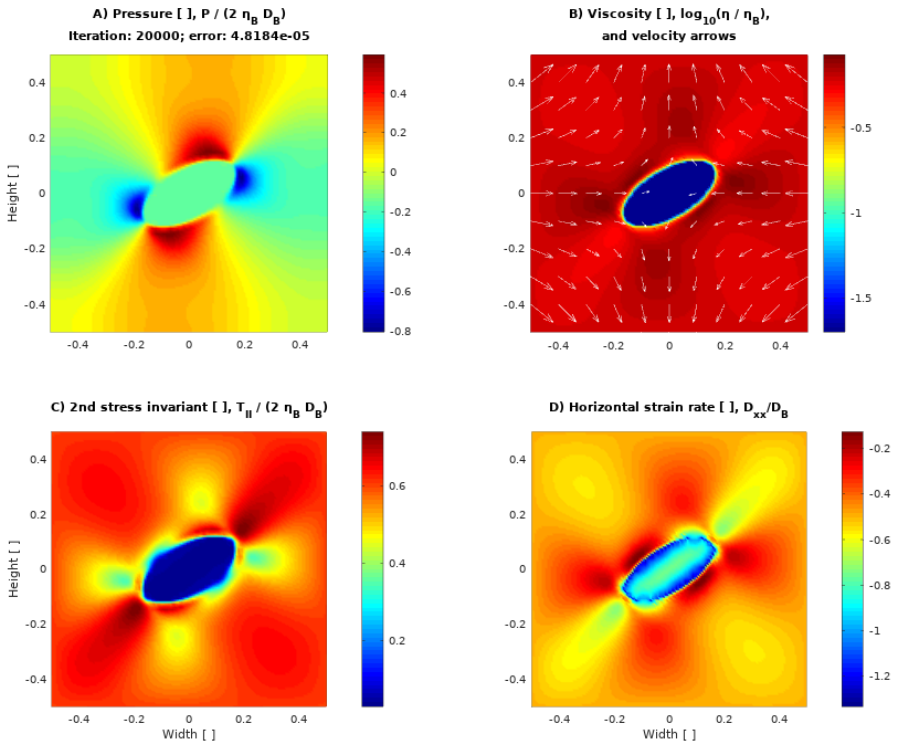
In the paper $\eta_B = 1$ and $D_B = 1$ so that the pressure range in their figure is in fact [-8:8]. Left: fig 6 of the paper; Middle: ASPECT and fieldstone; Right: figure of the paper after correction in 2024.

Case 2: Power-law viscous matrix and weak elliptical inclusion



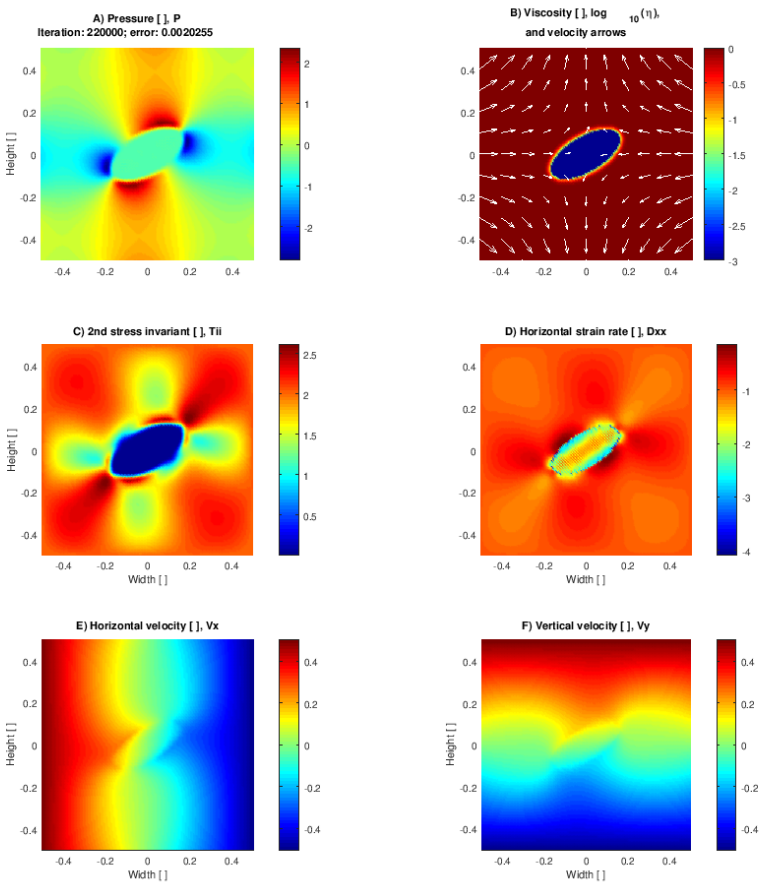
Taken from [147]. Viscosity and pressure field of a weak (viscosity ratio of 1000) elliptical inclusion under pure shear. A model with a purely linear viscosity (left panels) is compared against a model with an effective viscosity including power-law viscosity (right panels). The power-law viscosity is calculated using a stress exponent of 5 and a constant reference stress of 1.5 (see equation (12)). The arrows display the velocity field, they are not to scale and only for direction. The ellipse's semi-major axis is twice as long as its semi-minor axis and corresponds to 0.2 times the model width. The semi-major axis is rotated by 30 degrees from the horizontal axis.

The paper comes with two corrected Matlab and Octave codes that are available online under `https://github.com/halterw/A_simple_computer_program`. I have run the Octave version as is (I do not have Matlab). Rather annoyingly, it runs, shows results every 2000 iterations, ultimately converges but does not produce a plot. Also there is no visualisation of velocity?



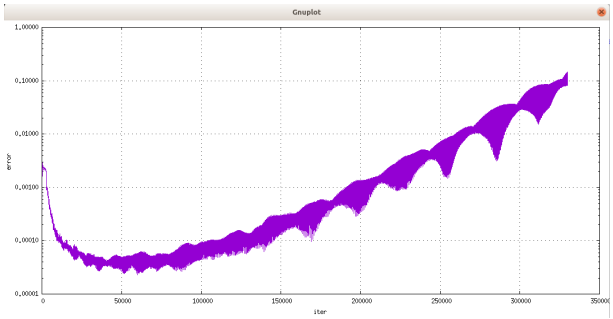
Screen capture of the Octave code output when run as is off github.

I have therefore modified the code to add velocity visualisation and export to png. I have also removed the normalisation of results. If I now set `n_exp=1` with a 1000 viscosity ratio the model now becomes Newtonian, but the code fails to converge. After more than 200,000 iterations the results are as follows:

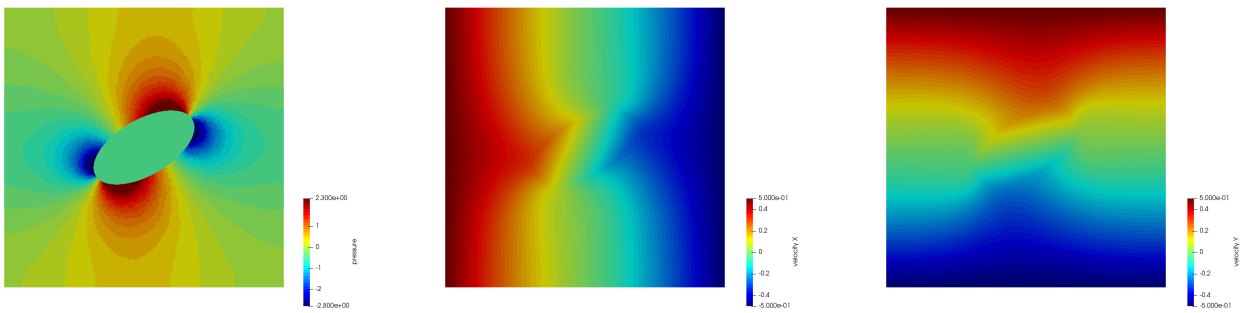


Screen capture of the Octave code output for a viscosity ratio of 1000. Modified output. Dxx looks somewhat suspicious...

As it turns out, when one lets the code run for a *long* time, the error looks like:

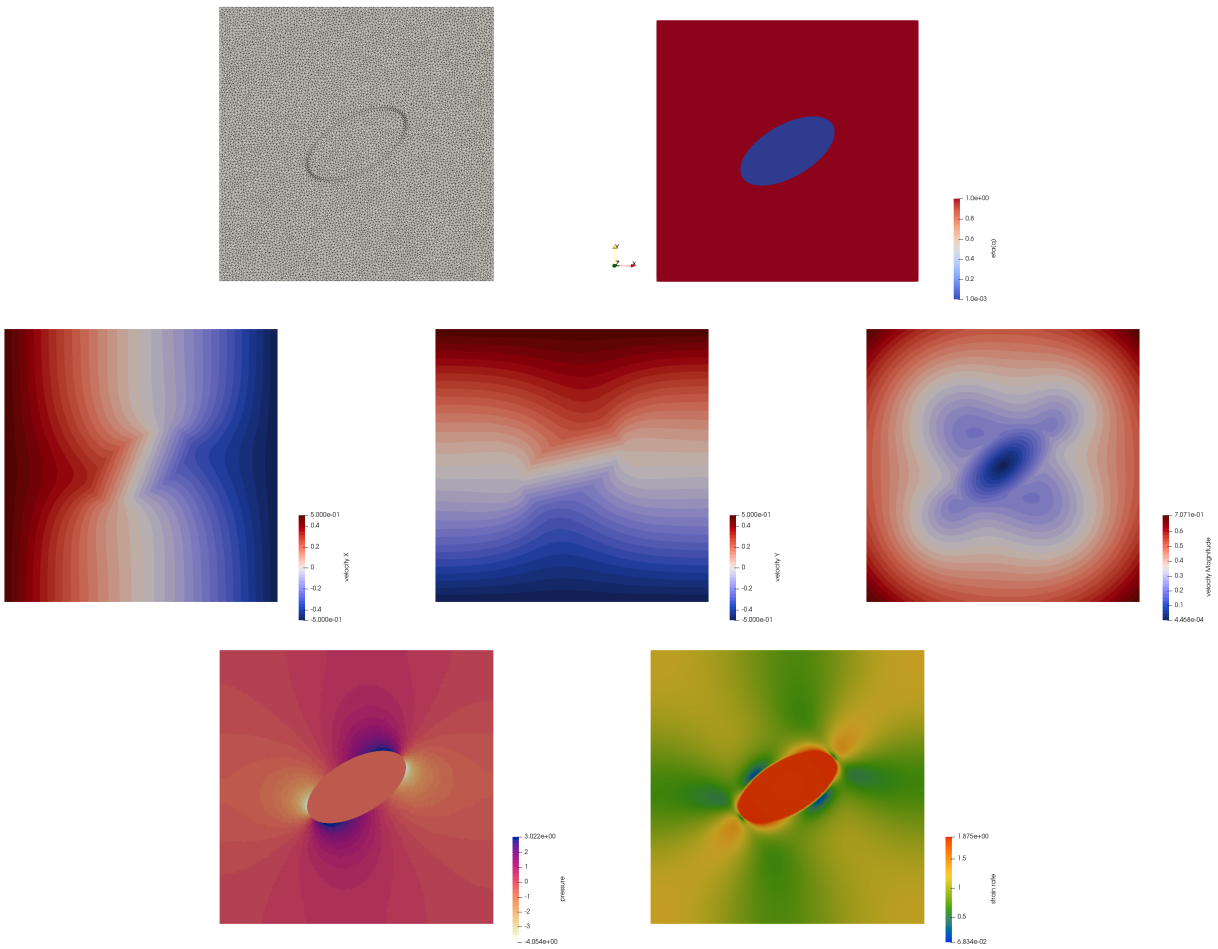


In summary, the solver is not converging and the results above are suspicious.

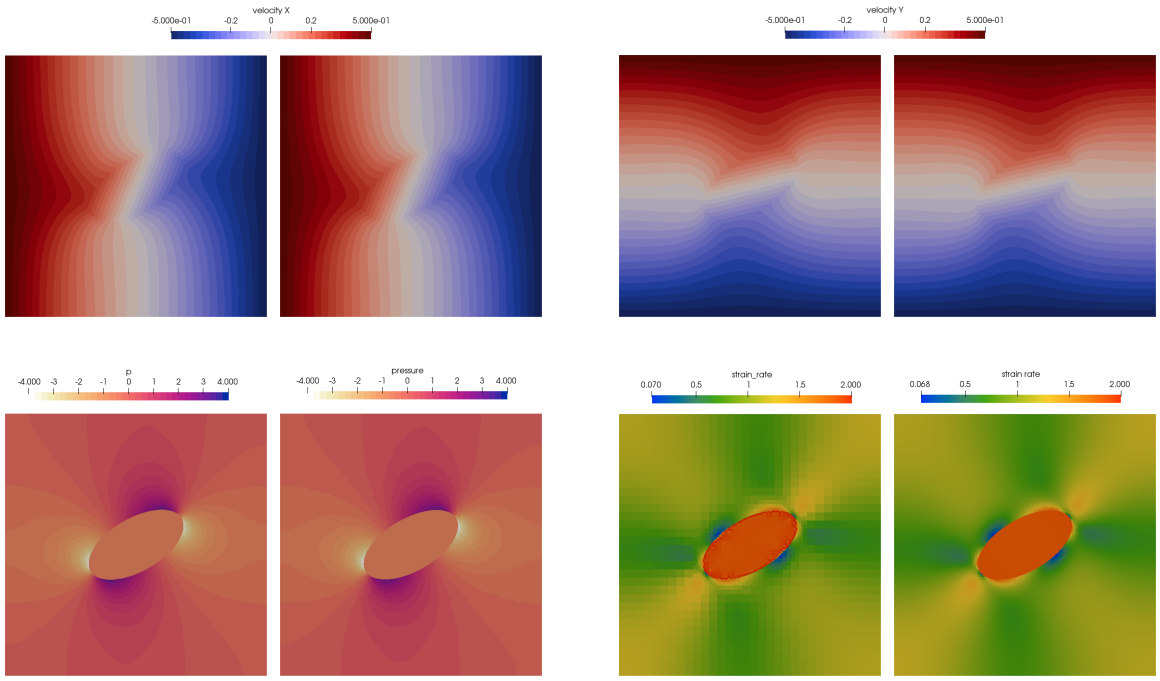


I get a reasonable match for pressure and velocity - provided I use the same colorscale and range.

Here is the full set of fields with appropriate color scales obtained with this STONE :

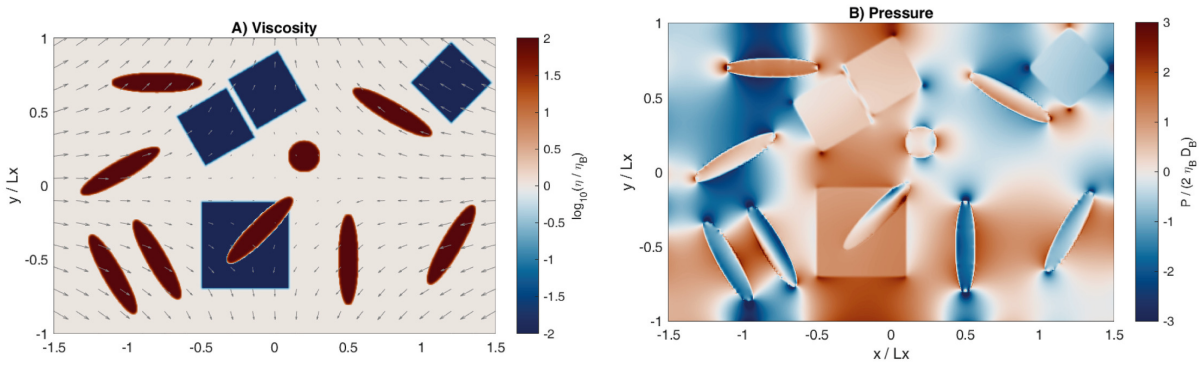


To validate the results of this stone further, I have run this model with the ASPECT code:



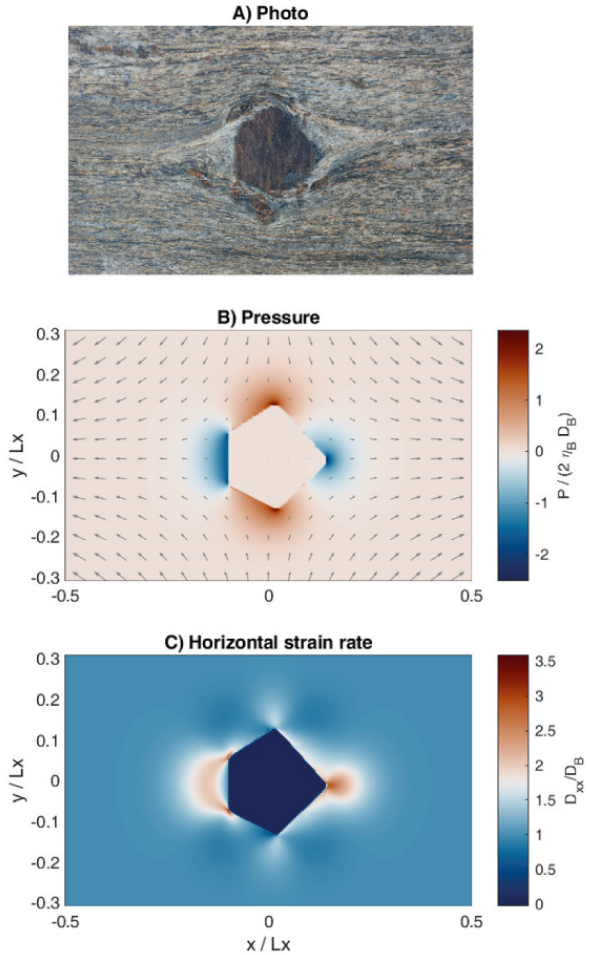
Left: ASPECT results; Right: fieldstone results.

Exp4: Multiple inclusions of different shape



Viscosity and pressure for a combination of different geometrical weak and strong inclusions subjected to horizontally compressive pure shear. The arrows in panel A) display the velocity field, they are not to scale and only for direction. The ellipses are 100 times more viscous, the squares are 100 times less viscous than the surrounding matrix. Except for the circular inclusion, all ellipses possess the same aspect ratio.

Exp5: Inclusion with a more natural shape



Numerical modelling of the pressure and strain rate field around a garnet inclusion using a photo from an actual inclusion (Fig. 1 C). In the model the strong inclusion has a viscosity contrast of 1000 and is subjected to horizontally extensional pure shear. The garnet was redrawn as a polygon using the Matlab ginput function. The arrows in panel B) display the velocity field, they are not to scale and only for direction.

Stone 143

Passive margins getting squeezed in the mantle convection vice



2D
box
Q2Q1
sparse_matrix
numerical_benchmark
stokes_equation
paraview_output

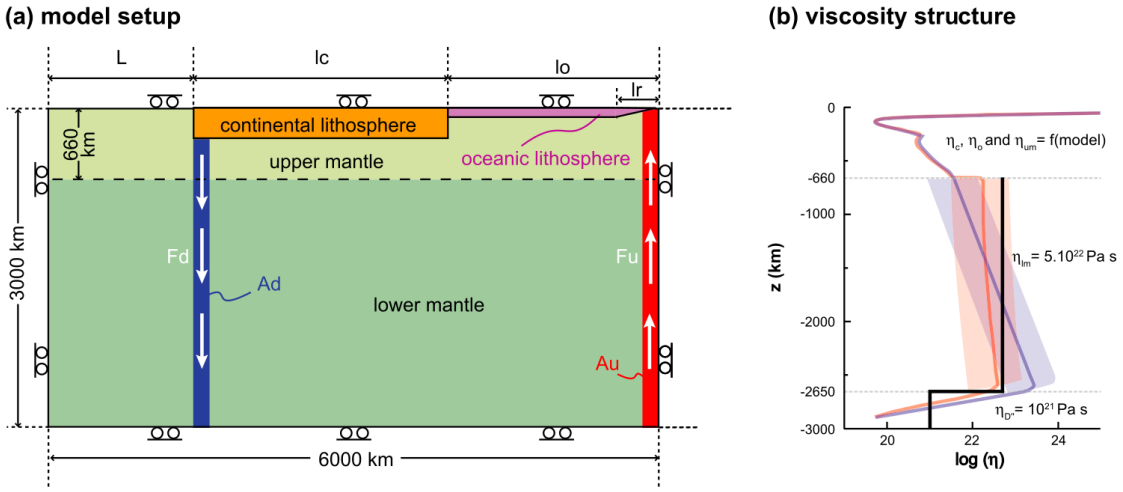
P Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_143`

This STONE is based on Yamato, Husson, Becker, and Pedoja [354] (2013) published in Tectonics. As for every STONE aiming at reproducing results off a publication I here include de abstract of the article:

Passive margins often exhibit uplift, exhumation, and tectonic inversion. We speculate that the compression in the lithosphere gradually increased during the Cenozoic, as seen in the number of mountain belts found at active margins during that period. Less clear is how that compression increase affects passive margins. In order to address this issue, we design a 2-D viscous numerical model wherein a lithospheric plate rests above a weaker mantle. It is driven by a mantle conveyor belt, alternatively excited by a lateral downwelling on one side, an upwelling on the other side, or both simultaneously. The lateral edges of the plate are either free or fixed, representing the cases of free convergence, and collision (or slab anchoring), respectively. This distinction changes the upper mechanical boundary condition for mantle circulation and thus, the stress field. Between these two regimes, the flow pattern transiently evolves from a free-slip convection mode toward a no-slip boundary condition above the upper mantle. In the second case, the lithosphere is highly stressed horizontally and deforms. For a constant total driving force, compression increases drastically at passive margins if upwellings are active. Conversely, if downwellings alone are activated, compression occurs at short distances from the trench and extension prevails elsewhere. These results are supported by Earth-like models that reveal the same pattern, where active upwellings are required to excite passive margins compression. Our results substantiate the idea that compression at passive margins is in response to the underlying mantle flow that is increasingly resisted by the Cenozoic collisions.

In the paper the authors state that their simple setup "also ensures the reproducibility of our results that can be easily tested by any code- solving Stokes equations with free-slip boundary conditions." This is *exactly* what we are going to do!

The authors state: "The Cartesian box is 6000 km wide and 3000 km deep. The grid resolution used is of 601×301 nodes which corresponds to a spatial resolution of 10 km. [...] All the mechanical boundary conditions are free slip and all the viscosities in our model are linear viscous." This makes indeed for a simple model.



Taken from [354]. Initial configuration of the model. (a) Model setup. L: length of mantle space between the left-hand side of the box and the continental lithosphere; l_c : length of continental lithosphere; l_o: length of oceanic lithosphere; $l_r = 500$ km: length corresponding to the thinning of the lithosphere due to the ridge; F_d: downwelling force; F_u: upwelling force. These buoyancy forces are implemented by setting variable densities in the red and blue columns. The area of the columns is A_d and A_u for the downwelling and the upwelling force, respectively. (b) Viscosity structure. η_c, η_o, and η_{um} correspond to the viscosity of the continental lithosphere, oceanic lithosphere, and upper mantle, respectively. The viscosity of the lower mantle corresponds to a simplified profile (black line). The viscosity models from [76] (2012) are also provided (Family-A model in red and Family-B model in blue) as references.

Please read the original paper for a justification of the chosen values for most of the parameters.

"The oceanic lithosphere is not attached to the right-hand side of the box, for one upper mantle cell separates it from the model box edge. This ensures that the lithosphere is free to move laterally and is not affected by the free-slip boundary condition imposed at the right side [...]."

"We [...] fixed typical thicknesses to 200 km and 100 km [...] for the continental and the oceanic lithosphere, respectively. The width of the continental lithosphere is similarly set in all our models to 2000 km."

The density is set to $\rho_{ref} = 3250$ kg m^{-3} everywhere in the model box. Density differences are only assigned to the upwelling and the downwelling zones that drive, by buoyancy, the convection cell. The density value in the upwelling (ρ_u) and in the downwelling (ρ_d) areas depends on the buoyancy force applied and are expressed as follows:

$$\rho_u = \rho_{ref} - \frac{F_u}{\rho_{ref} g A_u} \tag{143.1}$$

$$\rho_d = \rho_{ref} + \frac{F_d}{\rho_{ref} g A_d} \tag{143.2}$$

where F_u and F_d correspond to the absolute values of the upwelling and downwelling forces, respectively, and g is the gravitational acceleration set to 9.81 m s^{-2}. Note that the values for A_u and A_d is never specified! Looking closely at Fig 6, we find that the blue and red zones are as wide as the thickness of the oceanic lithosphere, i.e. 100km.

These formula provide us with a learning opportunity: If one fills the values for F_u, ρ_{ref}, g and A_u or A_d in the formula we find *very* small density variations, i.e. $\rho_u = 3249.99895449$ and $\rho_d = 3250.00104551$. These values are suspicious, and given the rather realistic viscosity values and dimensions of the setup this is not likely to yield plausible velocities or strain rates.

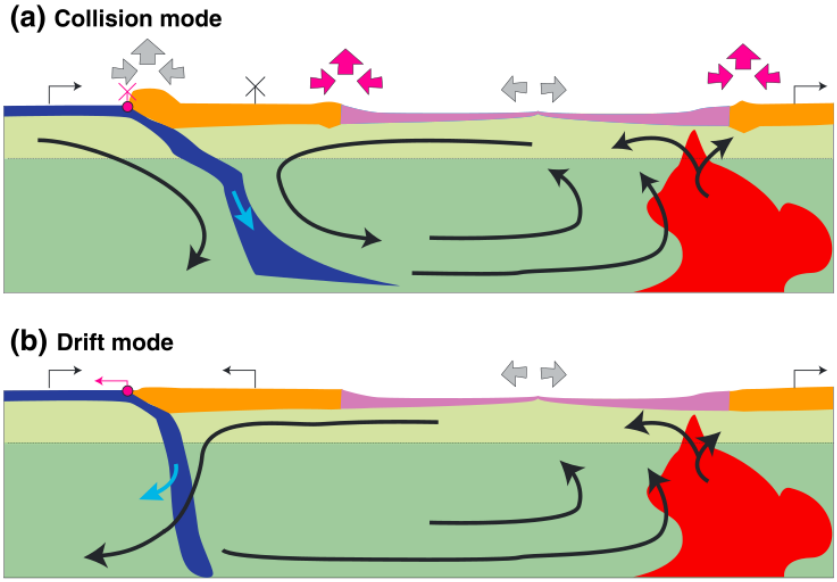
Looking closely, the nominator and denominator have the same dimensions, so that the ratio is dimensionless and can therefore not be added or subtracted to a density! Upon my contacting the authors they confirmed my suspicion: the density term in the denominator is wrong. The correct formula then read:

$$\rho_u = \rho_{ref} - \frac{F_u}{gA_u} \simeq 3246.60211 \text{ kg m}^{-3} \tag{143.3}$$

$$\rho_d = \rho_{ref} + \frac{F_d}{gA_d} \simeq 3253.64059 \text{ kg m}^{-3} \tag{143.4}$$

The viscosity in the lower mantle corresponds to a very simplified profile: From 660 km to 2650 km depth, the viscosity is set to $\eta_{lm} = 5 \cdot 10^{22}$ Pa s. Below, the viscosity is set to 10^{21} Pa s (D" layer).

Above the lower mantle, the viscosities are not well resolved and therefore need to be tested. The authors "considered the viscosity of the continental lithosphere, of the oceanic lithosphere, and of the upper mantle as linear and constant."



Taken from [354]. Sketch of the mantle convective cell flow in (a) collision mode and (b) drift mode. In the collision mode, the anchored slab does not retreat and prevents the trench-ward motion of the upper plate and is thus blocked. In drift mode, the subducting slab is retreating and accommodates the oceanic litho- sphere convergence that can freely move. Thick black and blue arrows show mantle flow and slab motion, respectively. Thin black and magenta arrows correspond to the lithosphere and trench displacements, respectively. These thin arrows are replaced by crosses when the motion is zero with respect to the upwelling. Large grey and magenta arrows correspond to compressional and extensional areas at active and passive plate margins, respectively.

About the code

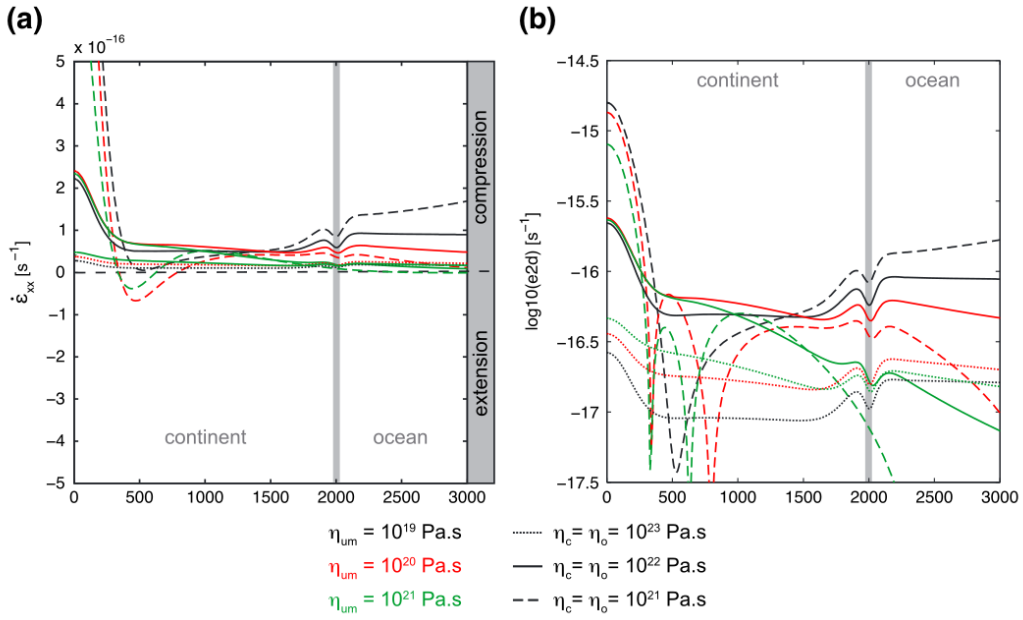
The code relies on the Finite Element method to solve the mass and momentum conservation equations for an isothermal incompressible fluid. Quadrilateral elements are used and the stable $Q_2 \times Q_1$ pair is used. Note that in the light of the first round of results I have also implemented the $Q_2 \times P_{-1}$ pair (unless mentioned the first element is used). Since free slip boundary conditions are used on all sides there is a pressure nullspace which is removed by requiring that $\int_\Omega p \, dV = 0$. Note that this code is rather inefficient and that although resolutions above 300×150 are attainable, the required cpu time makes them undesirable[1]. For simplicity elements are isoviscous so that the question of averaging does not come into play. Before building the FE matrix, the density and viscosity are evaluated at the center of each element.

[1] On my laptop a 400x200 mesh yields a matrix that is built in approx. 1000s! and memory requirements seem to be above 10Gb. A mesh of 600x300 takes 1000s to build the matrix, 650s to solve and about 30Gb of RAM.

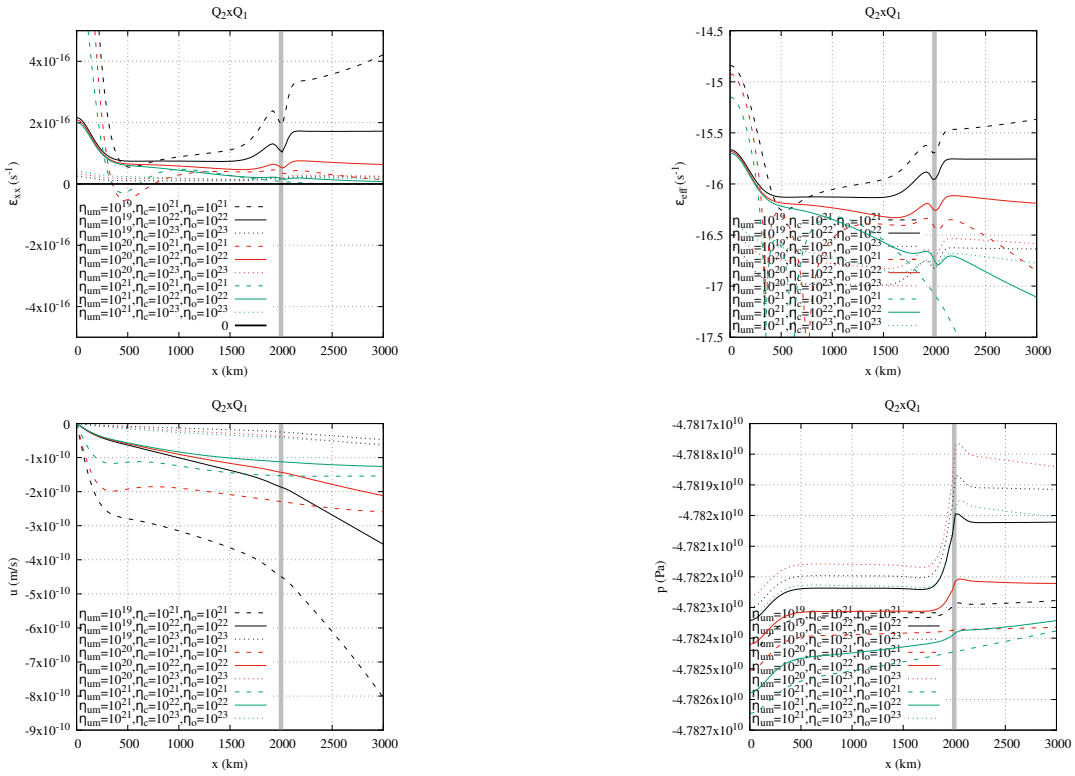
Collision mode

This corresponds to paragraph 19 and after in the paper.

In the collision mode, $L = 0$ and the lithosphere cannot move freely above the convection cell. The net driving force involved in this example amounts to $2 \cdot 10^{13}$ N m^{-1}. This total force is evenly distributed ($F_u = F_d = 10^{13}$ N m^{-1}).



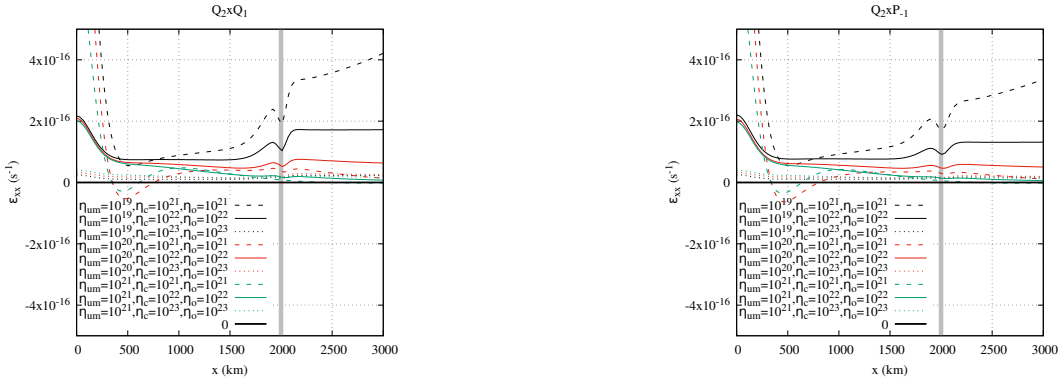
(a) Shortening rate $\dot{\varepsilon}_{xx}$ (horizontal component of the strain rate tensor) at the surface across the 3000 km long continental and oceanic lithospheres ($L = 0$ km, $l_c = 2000$ km, $F_d = F_u = 10^{13}$ N m^{-1}. Positive values of $\dot{\varepsilon}_{xx}$ correspond to compression[2]. (b) Logarithm of the second invariant of the strain rate (e2d) along the same profile.



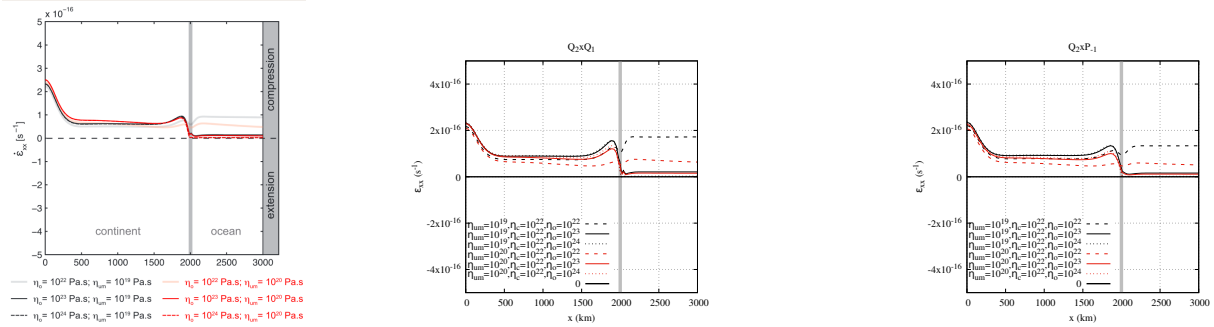
Data obtained with `script_fig4` with resolution 200 × 100. Replication is possible and aside from 19/21/21 all curves agree.

[2]Their convention is then opposite of mine so I plot the negative of my measurements to match theirs.

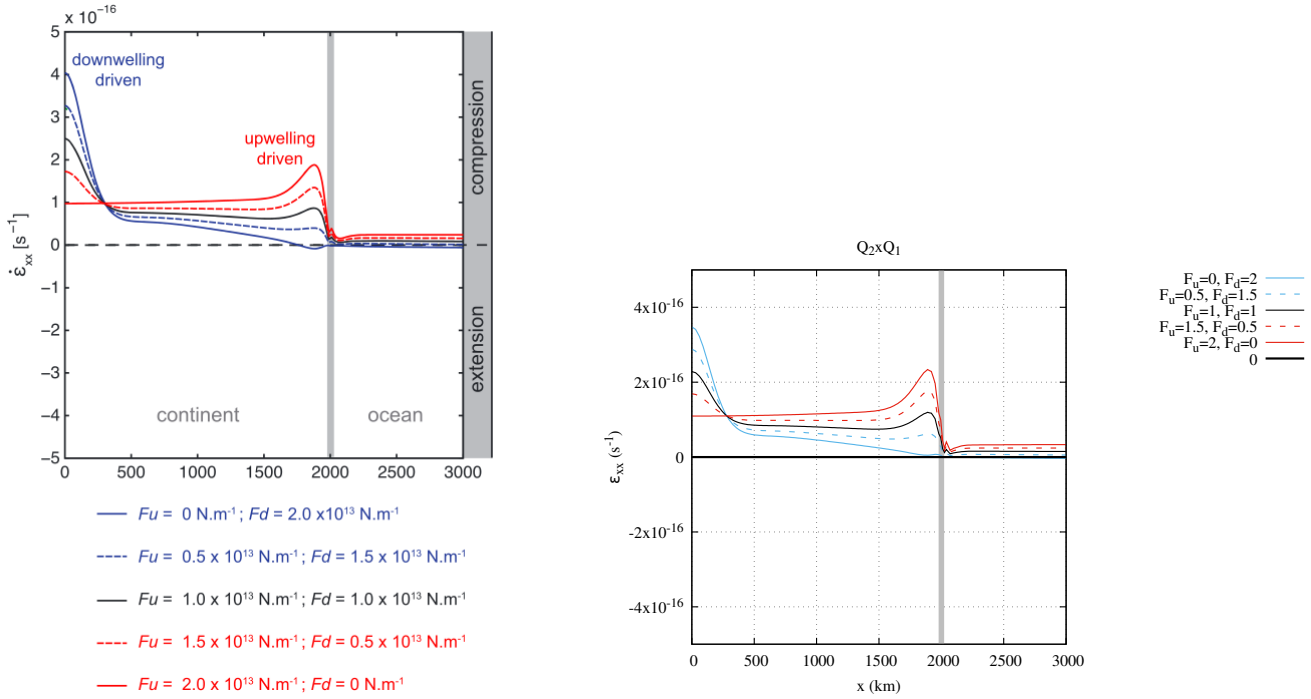
If we now run the same models using the discontinuous pressure element, we observe substantial differences in the results:



Left: $Q_2 \times Q_1$; Right: $Q_2 \times P_{-1}$. Both at resolution 200 × 100.



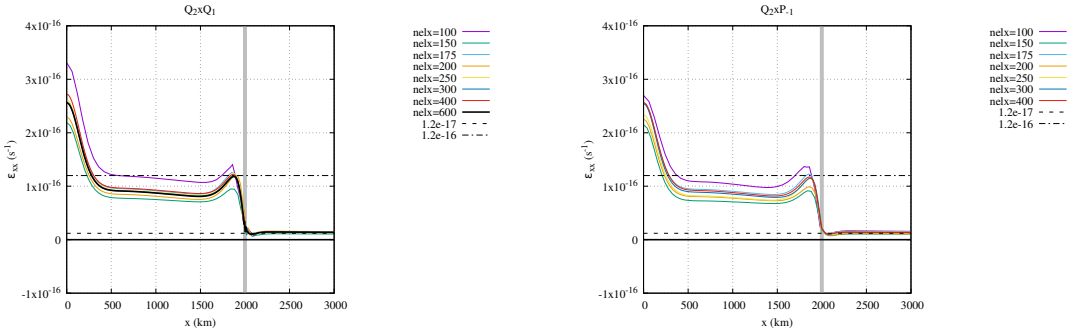
Horizontal component of the strain rate $\dot{\varepsilon}_{xx}$ across the 3000 km long continental and oceanic lithospheres ($L = 0$ km and $l_c = 2000$ km), for a variety of oceanic η_o and upper mantle $\eta_u m$ viscosities. The viscosity of the continental lithosphere η_c is set to 10^{22} Pa s. Left: fig 5 from the paper. Middle: obtained with this `script_fig5`, $Q_2 \times Q_1$; Right: same with $Q_2 \times P_{-1}$.



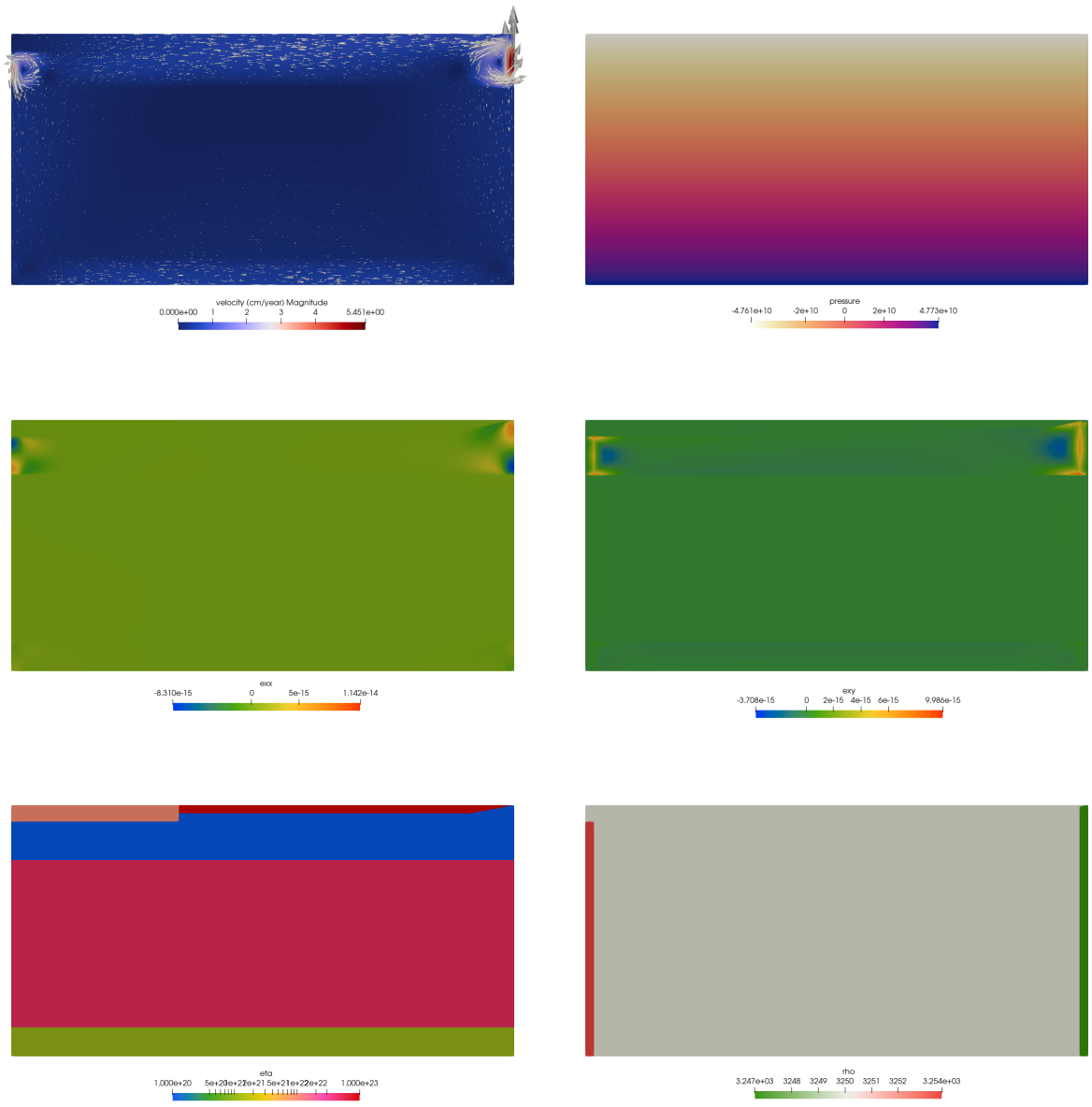
Strain rate at the surface across the 3000 km long continental and oceanic lithospheres ($L = 0$ km and $l_c = 2000$ km). Reference model, $\eta_{um} = 10^{20}$ Pa s, $\eta_c = 10^{22}$ Pa s, $\eta_o = 10^{23}$ Pa s. Left: Fig 6 from [354], Right: obtained with this `script_fig6`.

Testing influence of resolution

Results are not fundamentally different between the two types of elements.



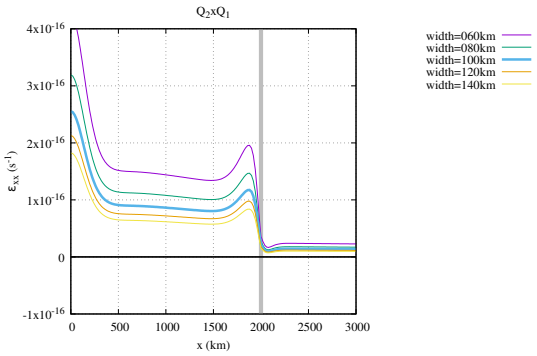
Reference model, $\eta_{um} = 10^{20}$ Pa s, $\eta_c = 10^{22}$ Pa s, $\eta_o = 10^{23}$ Pa s. Obtained with this `script_resolutions`. Left: $Q_2 \times Q_1$; Right: $Q_2 \times P_{-1}$.



Reference model, nelx=300. Resolution is then 20 km while theirs is 10 km.

Testing the influence of width of areas

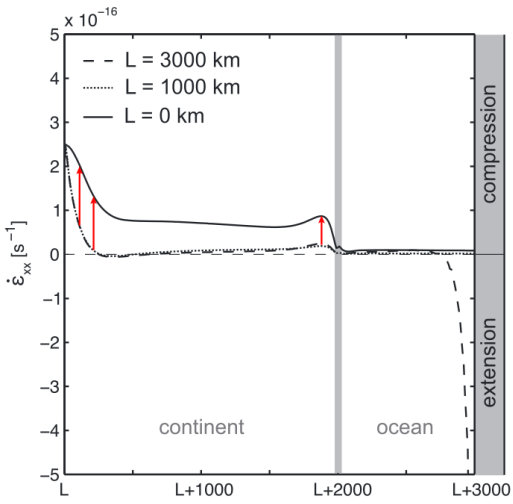
In the paper the exact width of the high/low density zones is not specified. Based on the figures we have assumed that there were 100 km wide, so it is time to explore this parameter.

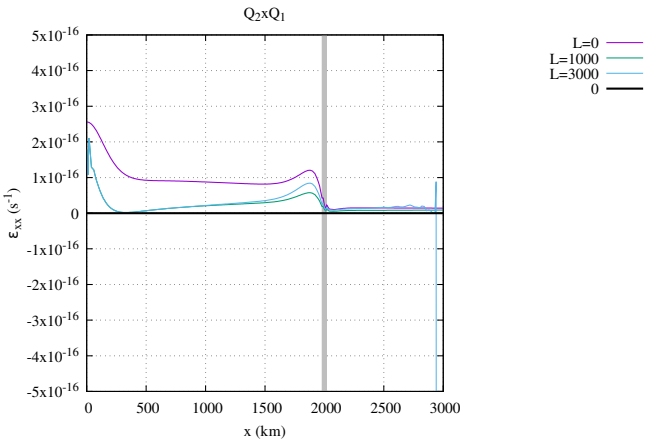


Obtained with `script_widths` with $Q_2 \times P_{-1}$ elements. Resolution 300x150, h=20km.

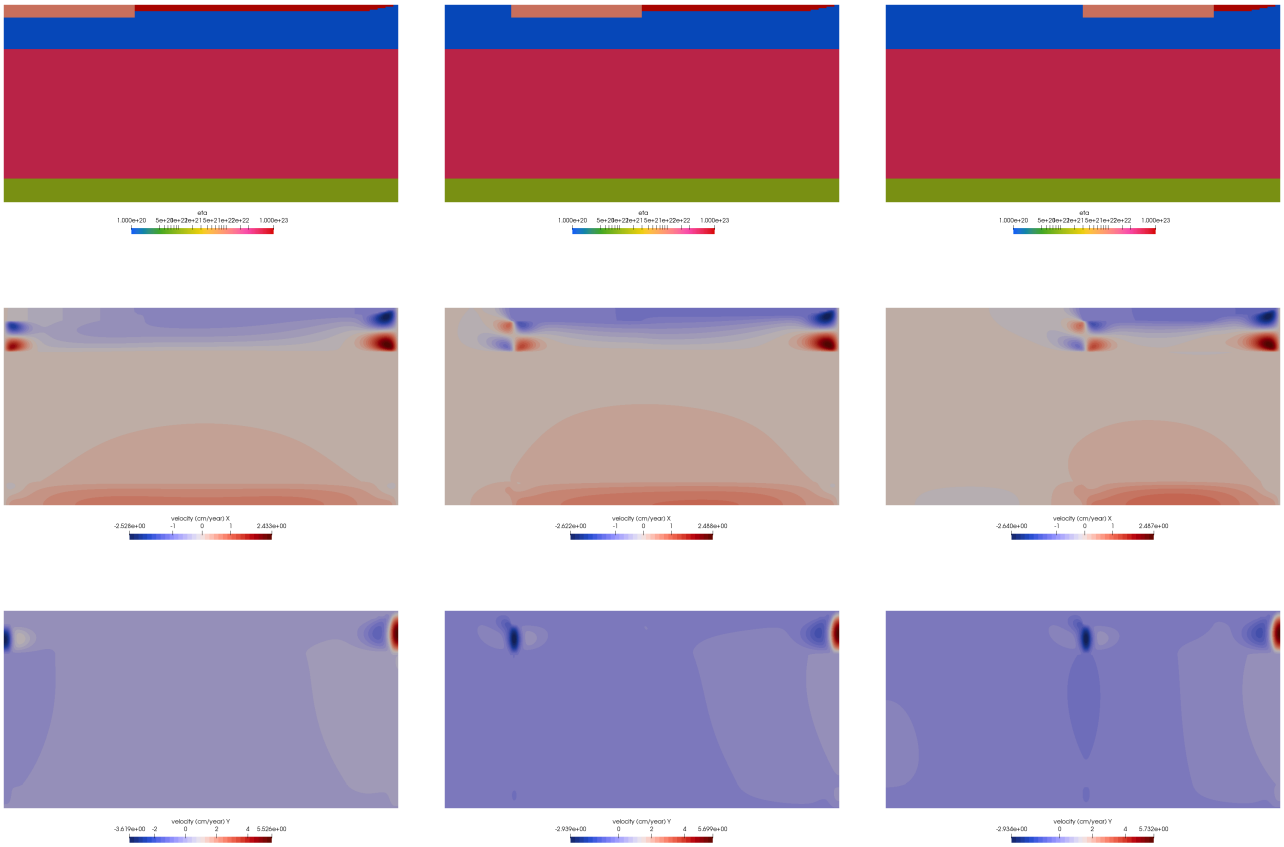
Drift versus collision mode

In this case we explore $L = 0, 1000, 3000$ km, see paper for geodynamical implications. Also I have not bothered to carry out the measurements of their fig 8.





Horizontal component of the strain rate $\dot{\varepsilon}_{xx}$ at the surface across the 3000 km long continental and oceanic lithospheres. Red arrows show the jump that occurs at the transition from drift mode to collision mode. Left: fig 9 of the paper. Right: obtained with this stone, nelx=300.



From left to right: L=0km, L=10000km, L=3000km.

ToDo:
- redo figure 8 of paper
- make code faster?
- stretch mesh ?

Stone 144

Dave's magma

Great seeing you in Vienna, and sorry for the delay in getting back to you. I attach a diagram that shows the overall geometry of the magma intrusion heat transfer problem we're interested in.

In terms of the values not shown on the model, here are a few numbers that might be useful:

Channel depth: 10.5 km Channel thickness: 100 m Channel length: 110 km (perhaps up to 130 km, if possible) Intrusion temperature: 1250C Surface temperature: 0C (would be helpful if this could be a variable) Basal temperature: 1480C Model thickness: 25 km

Does that cover everything you would need to know? I was planning to use an analytical solution for the channel velocity for different power-law exponents. Would this be something you need up front, or would I be able to code the velocity model? The main question is how fast the magma needs to flow to not cool during transport to not cool over the transportation of 110 km. So, we would want to vary the velocity magnitudes and pattern (i.e., power law viscous flow).

Thanks for offering to assist with this, and please let me know if you need anything else.

25000/100=nely=250elemnts
25000/500=nely=50elemnts

Stone 145

Converting vtu files to png or ascii



python_codes/fieldstone_145/text.tex

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_145`

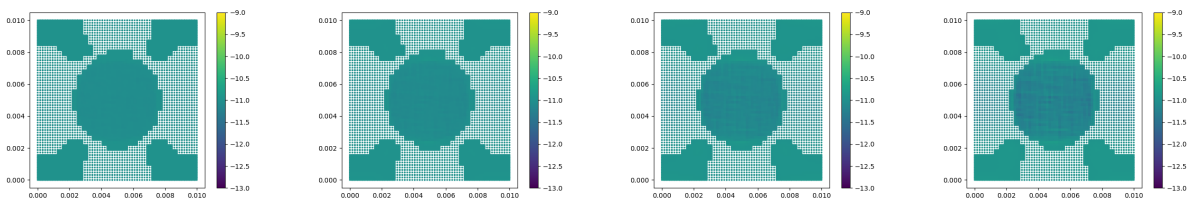
This stone was developed in collaboration with Frederic Gueydan.

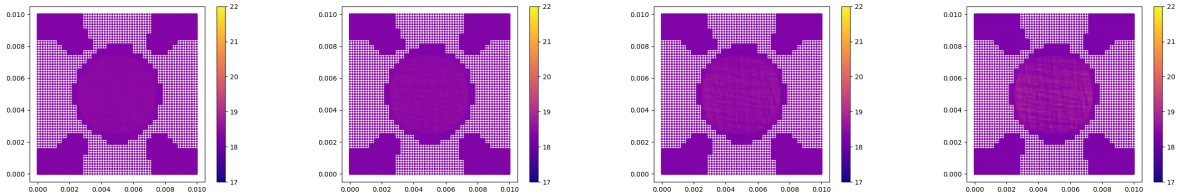
This STONE is a simple example of how one can easily read in (p)vtu files, extract the raw data for the fields and then for example post-process them, or plot them in a different way. I cannot predict your exact requirements when it comes to colorscale, aspect ratio, legends, types of data, etc ... so you will need to figure a few things out on your own.

In this example there are 4 `.pvtu` files coming from an article in prep. These are read in, and converted to png in a crude way. This is what the code displays when running

```
found  4  pvtu files
processing  solution-00000.pvtu which contains  22905  data points
-----> strain rate m/M: 8.305214e-12 1.1551771e-11
-----> viscosity m/M: 1.8379192e+18 3.4957697e+18
processing  solution-00002.pvtu which contains  22905  data points
-----> strain rate m/M: 5.3416676e-12 1.4275901e-11
-----> viscosity m/M: 1.4940624e+18 6.7333757e+18
processing  solution-00001.pvtu which contains  22905  data points
-----> strain rate m/M: 7.17326e-12 1.27030435e-11
-----> viscosity m/M: 1.6681961e+18 4.2696392e+18
processing  solution-00003.pvtu which contains  22905  data points
-----> strain rate m/M: 3.2133198e-12 1.5906169e-11
-----> viscosity m/M: 1.3733053e+18 2.1652515e+19
```

and here are the files produced





as well as 4 ascii files.

Stone 146

Jan's project



python_codes/fieldstone_146/text.tex

Code at `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_146`

This stone was developed in collaboration with J. Veenhof.

geometry

Temperature field

Grain size

Rheology

Stone 147

Solving the Stokes system with LINALG and more ($Q_2 \times Q_1$)



python_codes/fieldstone_147/text.tex

```
2D
box
Q2Q1
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
vrms
paraview_output
pressure_normalisation
schur_complement
```

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_147

The idea behind this STONE is the comparison of various approaches to solve the Stokes system. We here consider simple setups in a 2D rectangular domain. The code relies on the $Q_2 \times Q_1$element pair and is based on STONE 18.

In almost all previous stones I have used spsolve to solve the linear system but looking online we find that there are many more methods available[1]. Of all available I highlight the following ones:

- Direct methods for linear equation systems
  - spsolve: It looks like it is the SuperLU solver[2].
  - spsolve+UMFPACK: (solver=11)
- Iterative methods for linear equation systems:
  - bicg(A, b[, x0, tol, maxiter, M, callback, atol]) (solver=9)
  - bicgstab(A, b[, x0, tol, maxiter, M, ...]) (solver=10)

[1] https://docs.scipy.org/doc/scipy/reference/sparse.linalg.html
[2] https://caam37830.github.io/book/02_linear_algebra/sparse_linalg.html

- cg(A, b[, x0, tol, maxiter, M, callback, atol])
- cgs(A, b[, x0, tol, maxiter, M, callback, atol]) (solver=12)
- gmres(A, b[, x0, tol, restart, maxiter, M, ...]) (solver=2)
- lgmres(A, b[, x0, tol, maxiter, M, ...]) (solver=3)
- minres(A, b[, x0, shift, tol, maxiter, M, ...]) (solver=5)
- qmr(A, b[, x0, tol, maxiter, M1, M2, ...]) (solver=6)
- gcrotmk(A, b[, x0, tol, maxiter, M, ...]) (solver=8)
- tfqmr(A, b[, x0, tol, maxiter, M, callback, ...]) (solver=7)

If we build the system as a single (sparse) two-dimensional array we can right away rule out cg which requires a SPD matrix[3]. We will then try all the other ones.

Additionally we can build the $\mathbb{K}$ and $\mathbb{G}$ blocks and use conjugate gradient method on the Schur complement equation ('SC-CG'), as explained in Section 10.13.3. This solver is implemented in `schur_complement_cg_solver` It requires an inner solver called once per iteration on the $\mathbb{K}$ block. We then have the choice between using a direct solver (solver=4) or using a conjugate gradient approach since $\mathbb{K}$ is SPD (solve=13). By using CG we pretty much ensure that memory requirements remain very low. Another variant consists in acknowledging that inside the iterations the block $\mathbb{K}$ does not change, so that we can compute its LU decomposition and use this decomposition every time that we need to solve a linear system with $\mathbb{K}$. This is solver=14. This option means that the LU factors are stored in the memory longer than in the standard case but with equal memory use.

All these methods should also be used with a preconditioner, which we will investigate later on.

Iterative solver require a tolerance value and it is set to 1e-7 unless mentioned otherwise.

[3] `https://en.wikipedia.org/wiki/Conjugate_gradient_method`

1. the first solver is the usual one I use in all the stones

```
sol=sps.linalg.spsolve(sparse_matrix,rhs,use_umfpack=False)
```

2. the second solver is the gmres from linalg

```
sol,info = scipy.sparse.linalg.gmres(sparse_matrix, rhs, restart=2000,tol=tolerance,M=Mprec)
```

3. the third one is the lgmres from linalg

```
sol = scipy.sparse.linalg.lgmres(sparse_matrix, rhs,atol=1e-16,tol=tolerance)[0]
```

4. the fourth one is my Schur complement CG solver using direct solver for inner solve

```
solV,p,niter=schur_complement_cg_solver(K_mat,G_mat,M_mat,f_rhs,h_rhs,\
                                        NfemV,NfemP,niter_max,tolerance,use_precond,'direct')
```

5. the fifth one is minres from linalg

```
sol = scipy.sparse.linalg.minres(sparse_matrix, rhs, tol=1e-10)[0]
```

6. the sixth one is the qmr from linalg

```
sol = scipy.sparse.linalg.qmr(sparse_matrix, rhs, tol=tolerance)[0]
```

7. the seventh one is the tfqmr from linalg

```
sol = scipy.sparse.linalg.tfqmr(sparse_matrix, rhs, tol=1e-10)[0]
```

8. the eighth one is the gcrotmk from linalg

```
sol = scipy.sparse.linalg.gcrotmk(sparse_matrix, rhs,atol=1e-16, tol=1e-10)[0]
```

9. the ninth one is the bicg from linalg

```
sol = scipy.sparse.linalg.bicg(sparse_matrix, rhs, tol=tolerance)[0]
```

10. the tenth one is bicgstab from linalg

```
sol = scipy.sparse.linalg.bicgstab(sparse_matrix, rhs, tol=tolerance)[0]
```

11. the eleventh one is spsolve+umfpack

```
sol=sps.linalg.spsolve(sparse_matrix,rhs,use_umfpack=True)
```

12. the twelfth one is cgs from linalg

```
sol = scipy.sparse.linalg.cgs(sparse_matrix, rhs, tol=tolerance)[0]
```

13. the thirteenth one is my Schur complement CG solver using cg for inner solve

```
solV,p,niter=schur_complement_cg_solver(K_mat,G_mat,M_mat,f_rhs,h_rhs,\
                                        NfemV,NfemP,niter_max,tolerance,use_precond,'cg')
```

14. the fourteenth one is my Schur complement CG solver using splu for inner solve

```
solV,p,niter=schur_complement_cg_solver(K_mat,G_mat,M_mat,f_rhs,h_rhs,\
                                        NfemV,NfemP,niter_max,tolerance,use_precond,'splu')
```

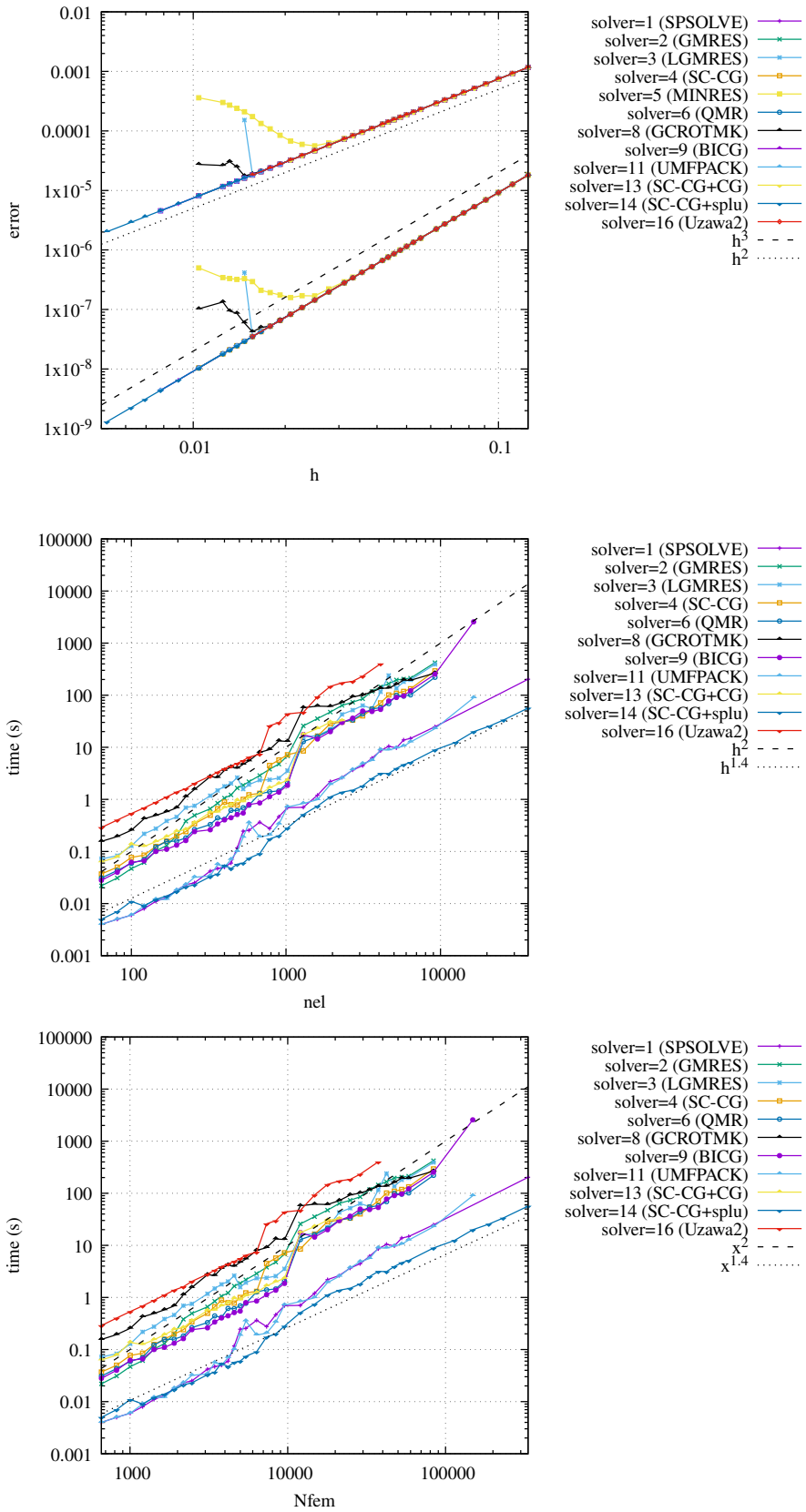
15. uzawa 1: Section 5.1 [33]. Parameterised by α(=2000 default)

```
alpha=2000
solV,p,niter=uzawa1_solver(K_mat,G_mat,M_mat,f_rhs,h_rhs,\
                           NfemV,NfemP,niter_max,tolerance,use_precond,'direct',alpha)
```

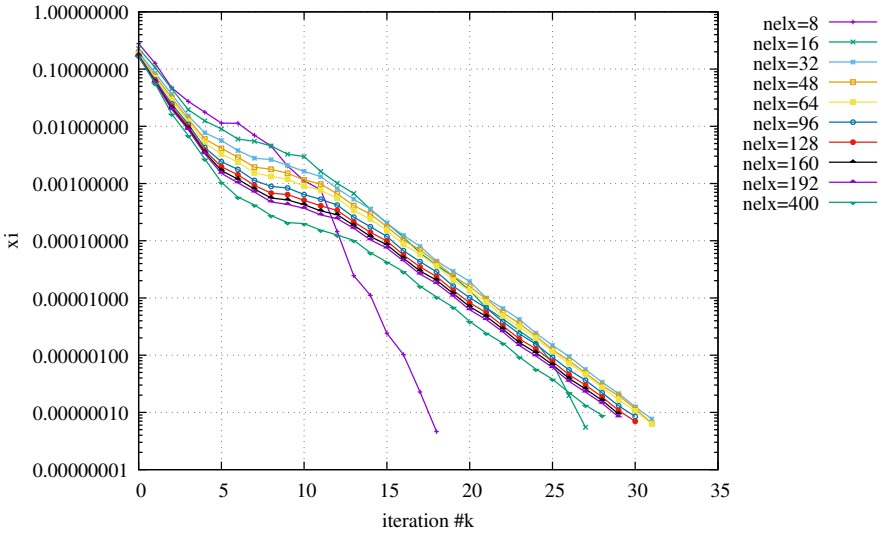
16. uzawa 2: Section 5.2 [33].

```
solV,p,niter=uzawa2_solver(K_mat,G_mat,M_mat,f_rhs,h_rhs,\
                           NfemV,NfemP,niter_max,tolerance,use_precond,'direct')
```

Donea & Huerta mms Without preconditioner

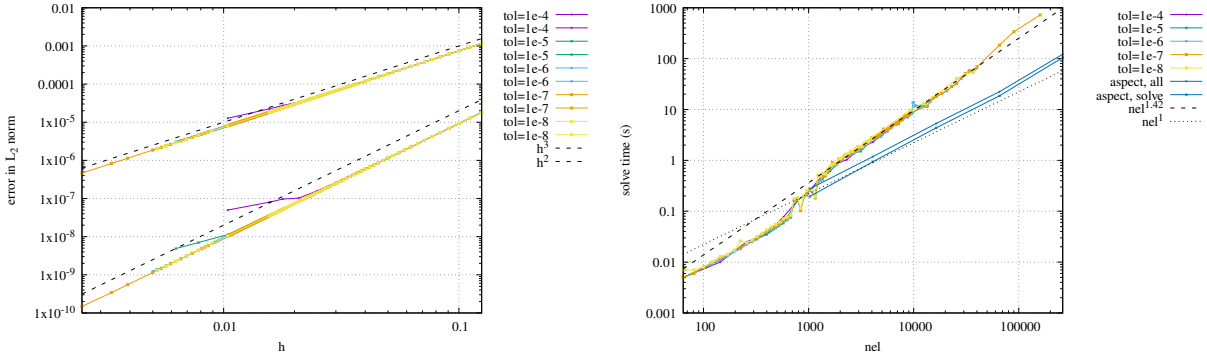


Time to solve linear system. Only results for usable solvers are shown. Left: time as a function of the number of elements; Right: time as a function of the total number of dofs.



Convergence of the SC-CG solver as function of resolution.

On the following figure I plot the velocity and pressure errors for SC-CG+ilu solver for different values of the relative tolerance. We find that the solve time is not much influenced by the tolerance, but unsurprisingly the accuracy of the solution is:

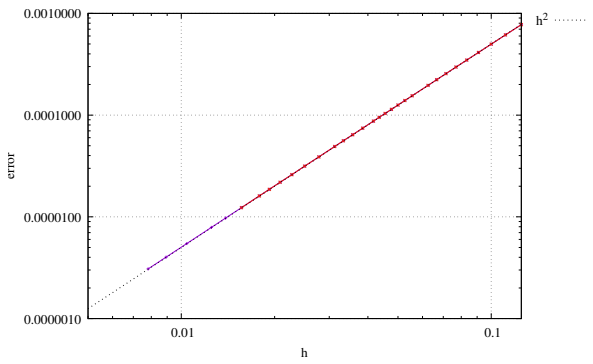


Left: Influence of the relative tolerance on the accuracy of the solution obtained with the SC-CG+ilu (solver=14). Right: solve time in seconds.

It appears that a relative tolerance of 10^{-6} is enough, but I set it to 10^{-7} default to be on the safe side (when looking closley at high resolution it looks like the blue curve of 10^{-6} seems to separate from the orange one from 10^{-7}).

I have also run Aspect on the same problem (with the same type of elements) and on only 1 core (while the spsolve inner solve tends to use all available cores of my laptop). Since Aspect reports the assembly time, the preconditioner time and the solve time separately I add them together. We find that the solver outperforms mine (obviously!) and more importantly that is scales linearly with the number of elements. The highest resolutions so far are 300×300 elements and the memory requirements were about half of my laptop RAM, so about 16Gb, and 400×400 which requires about 20+ Gb.

We can also look at the L_2 error on the velocity divergence:



About the Uzawa solvers

U1 is uzawa of Braess, section 5.1: simple to implement but a parameter is needed. U2 is uzawa of Braess, section 5.2 U3 is conjugate gradient on schur complement, section 5.3.

The three algorithms U1, U2, U3 are reproduced from [33] hereunder from left to right for convenience:

5.1 The Uzawa Algorithm. Let $\lambda_0 \in \mathbb{R}^m$. Find u_k and λ_k so that

$$\left.\begin{aligned} Au_k &= f - B^t\lambda_{k-1}, \\ \lambda_k &= \lambda_{k-1} + \alpha(Bu_k - g), \end{aligned}\right\} \quad k = 1, 2, \ldots \tag{5.4}$$

Here we assume that the step size parameter α is sufficiently small. □

5.2 Uzawa Algorithm *(the variant equivalent to the gradient method).*
Let $\lambda_0 \in \mathbb{R}^m$ and $Au_1 = f - B^t\lambda_0$.
For $k = 1, 2, \ldots$, compute

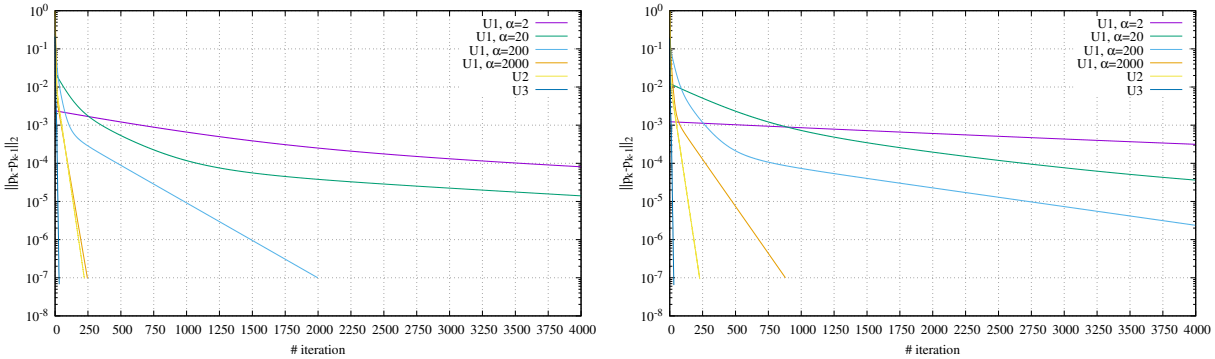
$$\begin{aligned} q_k &= g - Bu_k, \\ p_k &= B^t q_k, \\ h_k &= A^{-1}p_k, \\ \lambda_k &= \lambda_{k-1} - \alpha_k q_k, \qquad \alpha_k = \frac{q_k^t q_k}{p_k^t h_k}, \\ u_{k+1} &= u_k + \alpha_k h_k. \end{aligned}$$

5.3 Uzawa Algorithm with Conjugate Directions.
Let $\lambda_0 \in \mathbb{R}^m$ and $Au_1 = f - B^t\lambda_0$. Set $d_1 = -q_1 = Bu_1 - g$.
For $k = 1, 2, \ldots$, find

$$\begin{aligned} p_k &= B^t d_k, \\ h_k &= A^{-1}p_k, \\ \lambda_k &= \lambda_{k-1} + \alpha_k d_k, \qquad \alpha_k = \frac{q_k^t q_k}{p_k^t h_k}, \\ u_{k+1} &= u_k - \alpha_k h_k, \\ q_{k+1} &= g - Bu_{k+1}, \\ d_{k+1} &= -q_{k+1} + \beta_k d_k, \qquad \beta_k = \frac{q_{k+1}^t q_{k+1}}{q_k^t q_k}. \end{aligned}$$

Taken from Braess [33].

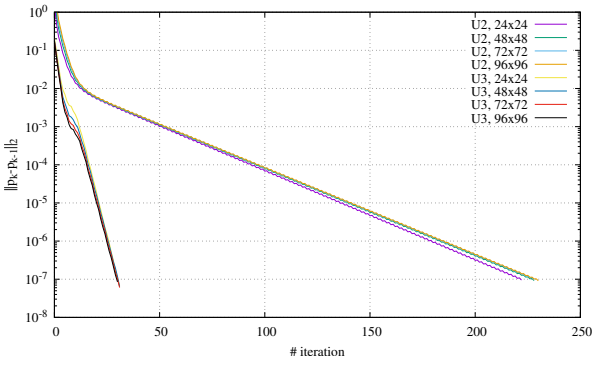
The following figure shows the convergence of this solver for different values of this parameter. 20000 makes the solver explode.



Convergence on 24x24 mesh (left) and 48x48 mesh (right) for Donea & Huerta manufactured solution.

We see that the α parameter should be a function of resolution, and that the uzawa2 converges in the same number of iterations because it automatically computes the step.

Actually when focusing now on U2 and U3 only and running the experiment at increasing resolutions I find (*as expected for stable element pairs*) that the number of iterations is independent of the resolution:



This shows that
1) U2 is a much better/safe algorithm than U1 while being equally simple to implement, thereby proving that U1 should be discarded altogether;
2) U3 is the obvious starting point if one wishes to offer an improvement on Uzawa-type solvers as it is by far the best of the three and is marginally more complex to implement than U1 or U2.

Conclusions

- direct method spsolve about 10x faster than all others (except solver 14!)
- UMFPACK option in spsolve does not yield substantial differences
- my own Schur complement CG solver rivals other LINALG solvers (solver 4, 13) !
- MINRES, even with very low tolerance 1e-10, ultimately fails
- LGMRES seems to work fine, very similar to GMRES but does not always converge.
- TFQMR, BICGSTAB, CGS are out: they fail to converge and/or return very wrong results
- GCROTMK usable but slowest, also fails at high resolution
- number of iterations of SC-CG is independent of resolution, as expected from a stable element pair (for isoviscous flow)
- solve time for iterative method rather comparable, no clear best
- SC-CG with direct or cg inner solve are comparable in terms of solve time. There is still something wrong about my having to set tol of cg extremely low, and even then it ends up crashing at nel=52 ...
- my own SC-CG+ilu is faster than spsolve!

In the end, I keep SPSOLVE, SC-CG, GMRES, QMR, BICG.

Donea & Huerta mms With preconditioner

Following this site[4] we build an ILU preconditioner as follows:

```
ILUfact = sla.spilu(sparse_matrix)
M = sla.LinearOperator(
    shape = sparse_matrix.shape,
    matvec = lambda b: ILUfact.solve(b))
```

However, I find that the preconditioner does not work with gmres, minres, cg, ...
This means that I need to build my own preconditioner. For the SC-CG
We have designed X preconditioners:

- `precond_type=0` It is the unit matrix (so it does nothing).
- `precond_type=2`
$$\boldsymbol{M} = \mathbb{G}^T(diag[\mathbb{K}])^{-1}\mathbb{G}$$
- `precond_type=3`
$$\boldsymbol{M} = diag\left[\mathbb{G}^T(diag[\mathbb{K}])^{-1}\mathbb{G}\right]$$
- `precond_type=4` Same as 2, but instead of using the diagonal of $\mathbb{G}^T(diag[\mathbb{K}])^{-1}\mathbb{G}$ we lump the matrix instead.

unfinished

To Do, open questions, future work?

- do smthg

Relevant Literature:
xxxxYY

[4] https://caam37830.github.io/book/02_linear_algebra/sparse_linalg.html

Stone 148

Erik's project



python_codes/fieldstone_148/text.tex

2D
box
Q2Q1
Q2Pm1
stokes_equation

P Code at https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_148

This stone was developed in collaboration with Erik van der Wiel.

Setup

The domain is a 2D Cartesian box of size $L_x \times L_y$ which is discretised by means of $nelx \times nely$ elements. Both the $Q_2 \times Q_1$ and $Q_2 \times P_{-1}$ element pairs are implemented, but given the discontinuous nature of the viscosity field (see hereafter) the latter is to be preferred.

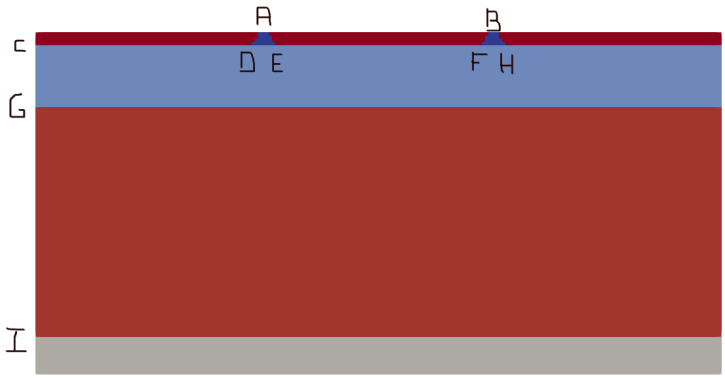
Boundary conditions are free slip at the top and bottom while in/out-flow kinematical boundary conditions are prescribed on the sides (see Section **??**). The in- to out-flow transition profile starts at a 150 km depth and ends at a depth of 350 km. The prescribed velocities on the left and right lithosphere are v_L and v_R respectively. Also an option ('symm bc') has been created so that the same outflow velocity is prescribed on the left and the right even though $|v_L| \neq |v_R|$. Volume conservation is of course still ensured but it is in fact a case of absolute plate motion. After solving the Stokes system, strain rate and deviatoric stress components are computed and projected onto nodes.

There are 6 materials in the domain (viscosity and density values are the default ones):

- Oceanic lithosphere ('o'): $\eta_o = 10^{23}$ Pa s, $\rho_o = 3300$ kg m^{-3}
- Upper mantle ('um'): $\eta_{um} = 10^{20}$ Pa s, $\rho_{um} = 3500$ kg m^{-3}
- Lower mantle ('lm'): $\eta_{lm} = 5 \cdot 10^{22}$ Pa s, 3900 kg m^{-3}
- a D" layer (dpp): $\eta_{dpp} = 10^{21}$ Pa s, $\rho_{dpp} = 4000$ kg m^{-3}
- two oceanic ridges coined A and B ('A', 'B'): $\eta_A = \eta_B = 10^{19}$, $\rho_A = \rho_B = 3300$ kg m^{-3}

For simplicity both viscosity and density are elemental fields, possibly discontinuous from one element to another.

Given the nature of the boundary conditions on all four sides the pressure can be computed up to a constant (nullspace) so that we require that $\frac{1}{L_x}\int_{y=L_y} p dx = 0$. LINK to Jourdon and May [174] (2022) - may be do it, as it would be much more consistent?



Experiments

The code has been written in such a way that it can be scripted and the list of conducted experiments is presented in the following table. For reasons that need to be justified, experiment 4 has been deemed the reference one.

experiment	v_L	v_R	η_{um}	rA	rB	o	v_{mantle}	γ	
1	-4	4	20	19	19	23	0	1	
2	-2	4	20	19	19	23	0	1	
3	-4	8	20	19	19	23	0	1	
4	-6	12	20	19	19	23	0	1	REF
5	-8	16	20	19	19	23	0	1	
6	-4	10	20	19	19	23	0	1	
7	-8	8	20	19	19	23	0	1	
8	-2	8	20	19	19	23	0	1	
9	-4	15	20	19	19	23	0	1	
10	-4	20	20	19	19	23	0	1	
11	-6	12	20	**18**	**18**	23	0	1	
12	-6	12	20	**20**	**20**	23	0	1	
13	-6	12	**19**	19	19	23	0	1	
14	-6	12	**21**	19	19	23	0	1	
15	-6	12	20	19	19	**22**	0	1	
16	-6	12	20	19	19	**24**	0	1	
17	-6	12	20	19	19	**25**	0	1	
18	-6	12	20	19	19	23	0	1	symm bc
19	-6	12	20	19	**18**	23	0	1	
20	-6	12	20	**18**	19	23	0	1	
21	-6	12	20	**20**	19	23	0	1	
22	-6	12	20	19	**20**	23	0	1	
23	-6	12	20	19	19	23	**1**	1	
24	-6	12	20	19	19	23	**3**	1	
25	-6	12	20	19	19	23	**6**	1	
26	-6	12	20	19	19	23	0	**1.5**	
27	-6	12	20	19	19	23	0	**2**	
27	-6	12	20	19	19	23	0	**4**	

Additionally, mantle wind can be addedd (v_{mantle}).

There are two simple ridges in the system, coined A and B. Each can have its own properties. They are isosceles triangles. Ridge A is parameterised by points A,D,E while ridge B is parameterised by points B,F,H.

The parameter γ controls of wide the rifts are at 100km depth.



from left to right: $\gamma = 1, 2, 4$.

Measurements

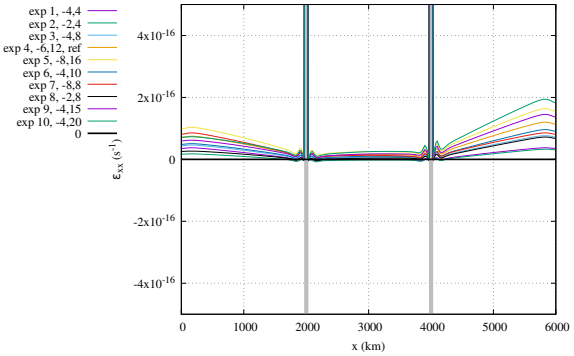
We focus on two profiles: $y = L_y$ and $x = L_x/2$ for which we export the velocity, the strain rate components, the pressure and the stress components.

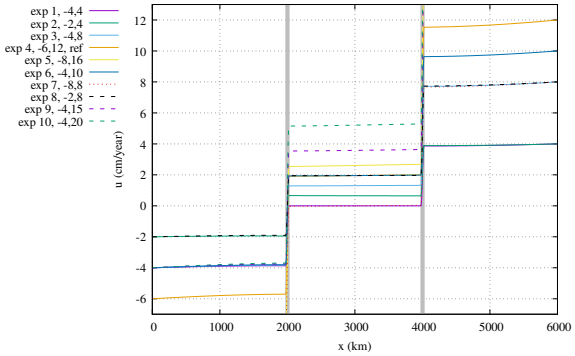
Results are for now obtained with 300x150 but should be re-run with 600x300 resolutions. Each model takes about 5min to run at the current resolution. The assembly process could be made faster following the explanations provided in STONE 150.

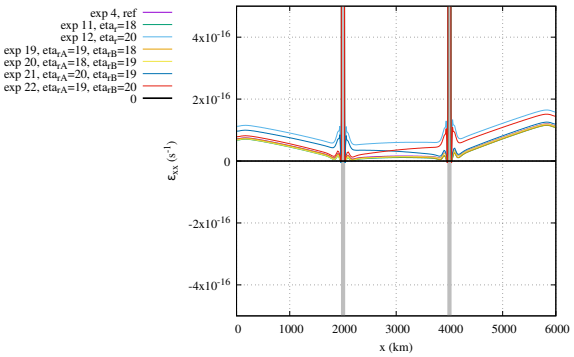
Todo/totry

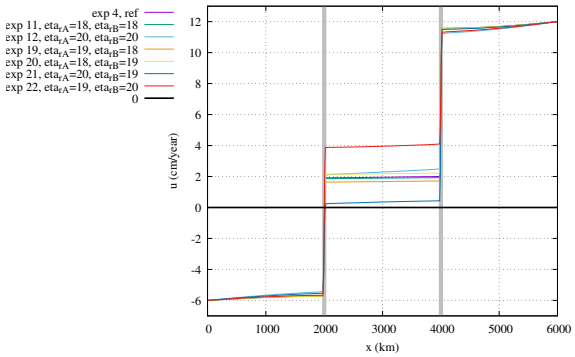
- vary middle plate length
- open bc + traction

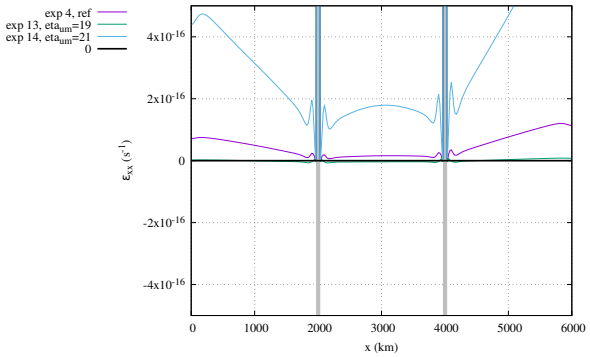
Horizontal profile at $y = Ly$

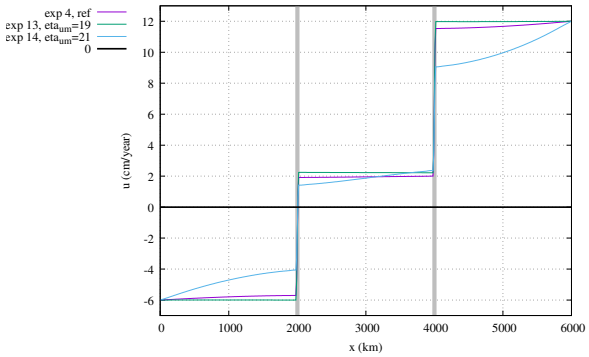


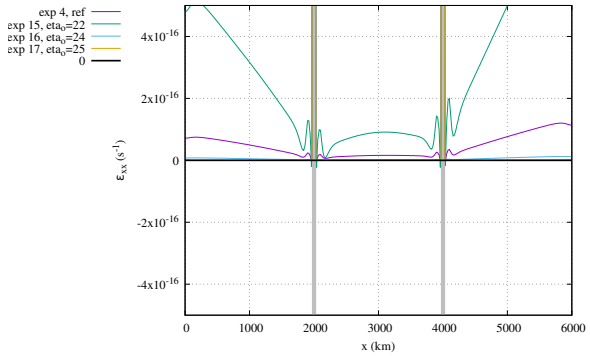


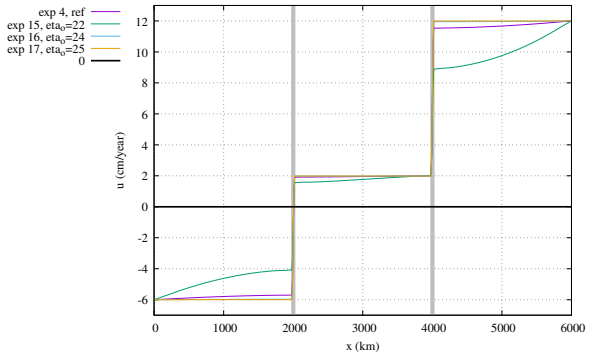


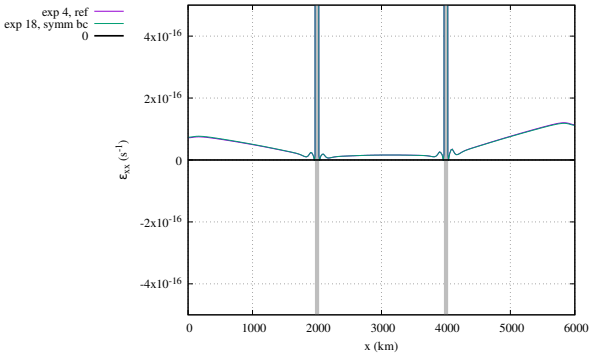


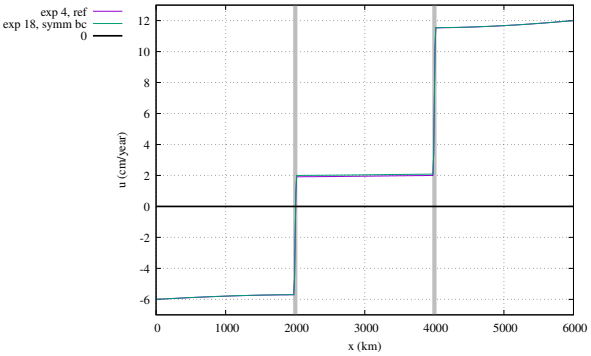


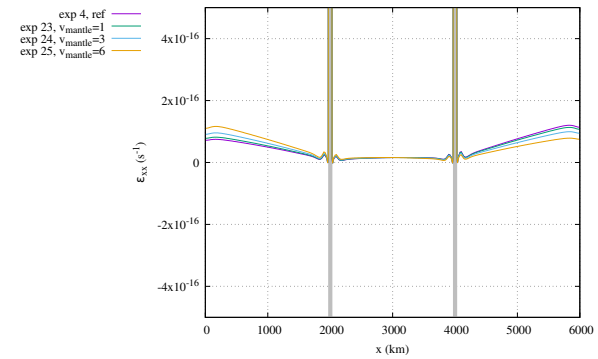


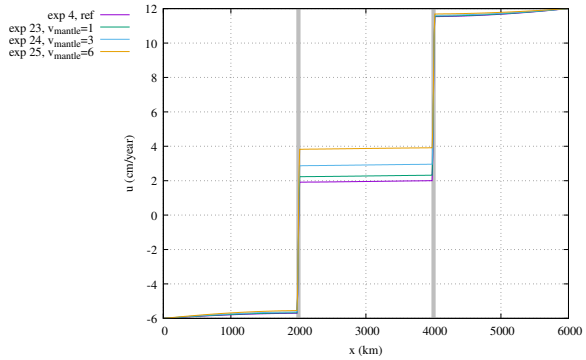


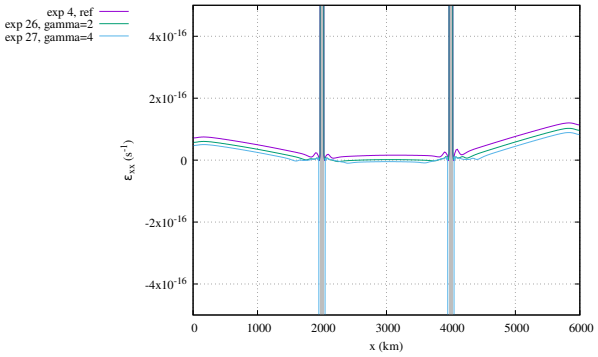


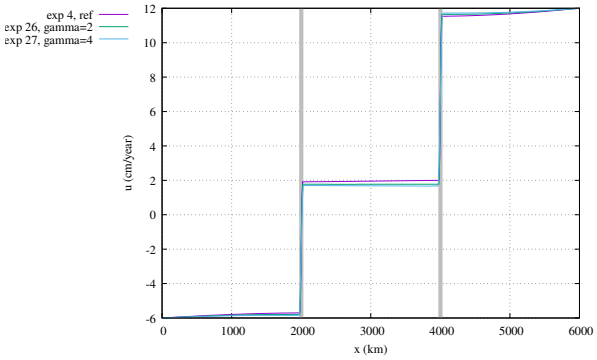




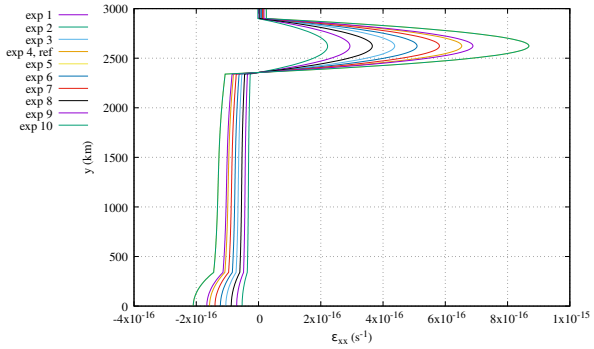


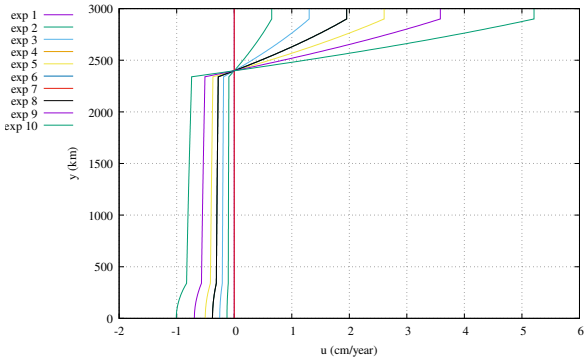


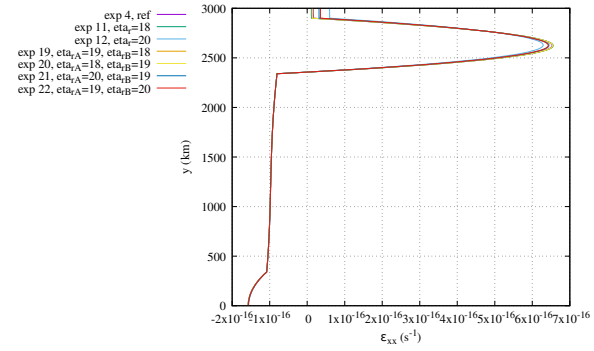


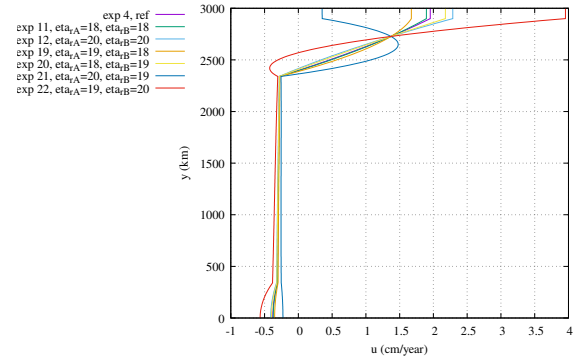


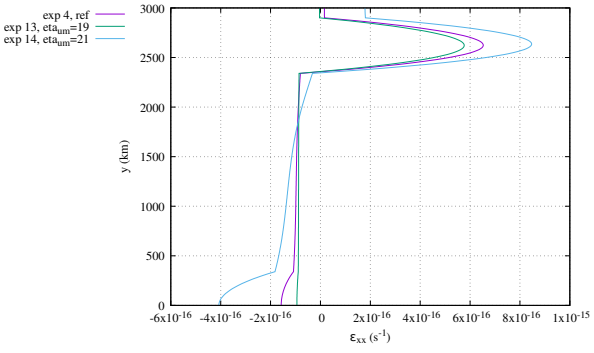
Vertical profile at $x = L_x/2$

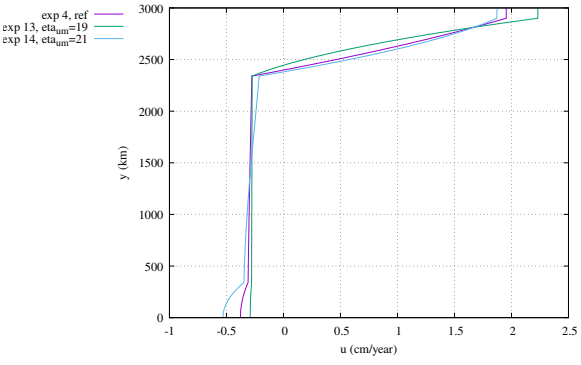


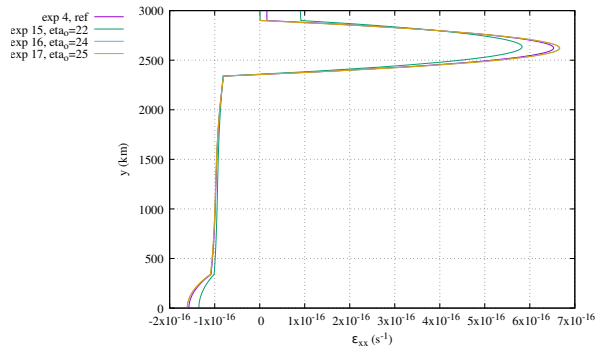


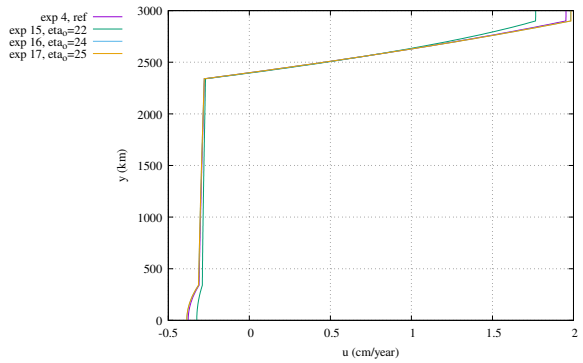


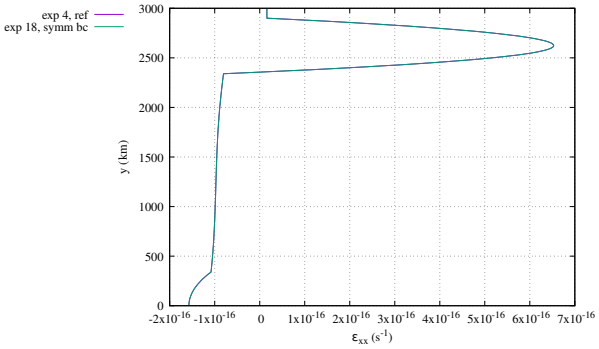


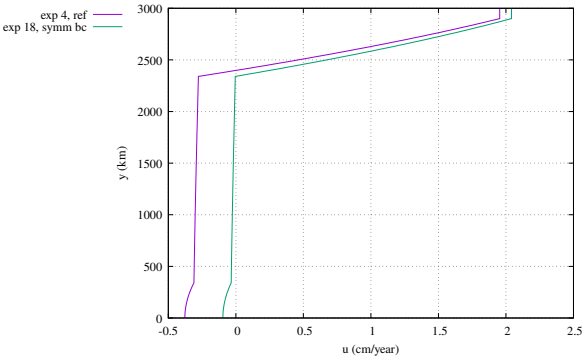


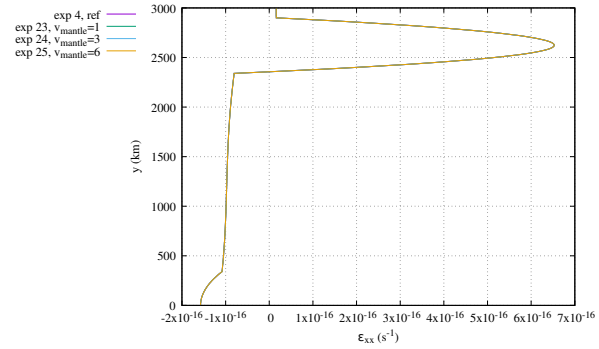


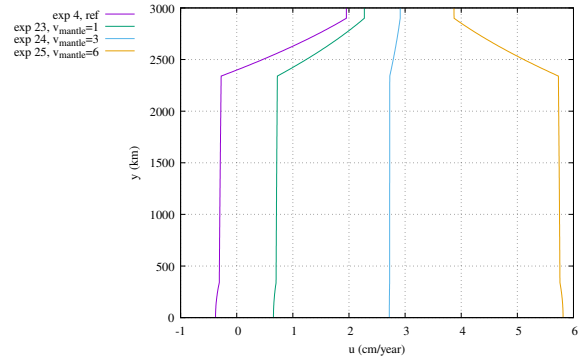

exp 4, ref
exp 15, eta0=22
exp 16, eta0=24
exp 17, eta0=25
y (km)
εxx (s-1)

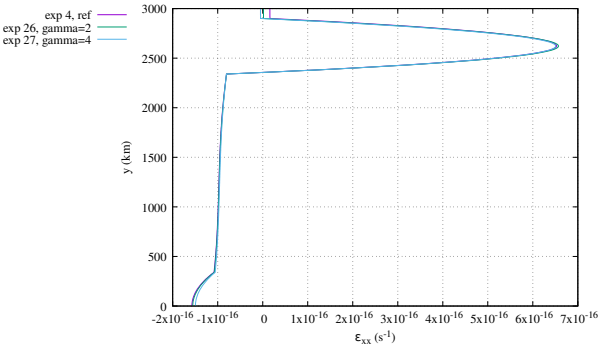

exp 4, ref
exp 15, eta0=22
exp 16, eta0=24
exp 17, eta0=25
y (km)
u (cm/year)

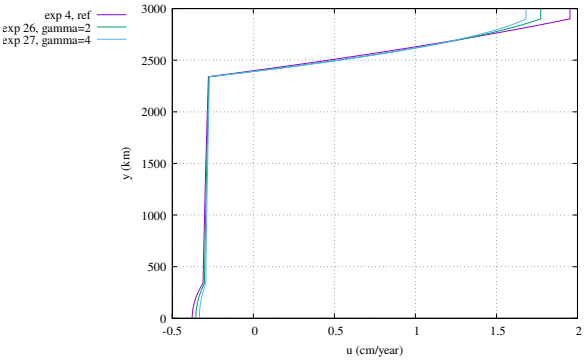

exp 4, ref
exp 18, symm bc
y (km)
εxx (s-1)


exp 4, ref
exp 18, symm bc
y (km)
u (cm/year)

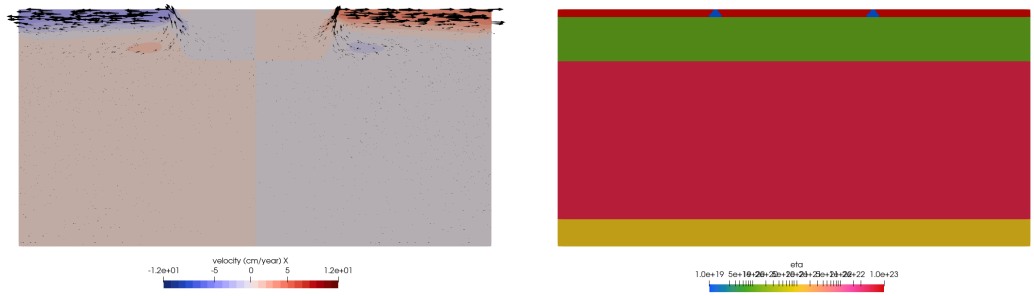

exp 4, ref
exp 23, vmantle=1
exp 24, vmantle=3
exp 25, vmantle=6
y (km)
εxx (s-1)


exp 4, ref
exp 23, vmantle=1
exp 24, vmantle=3
exp 25, vmantle=6
y (km)
u (cm/year)

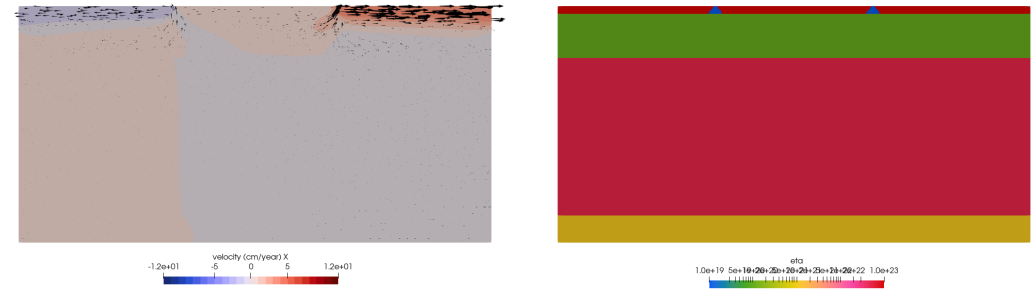

exp 4, ref
exp 26, gamma=2
exp 27, gamma=4
y (km)
εxx (s-1)


exp 4, ref
exp 26, gamma=2
exp 27, gamma=4
y (km)
u (cm/year)

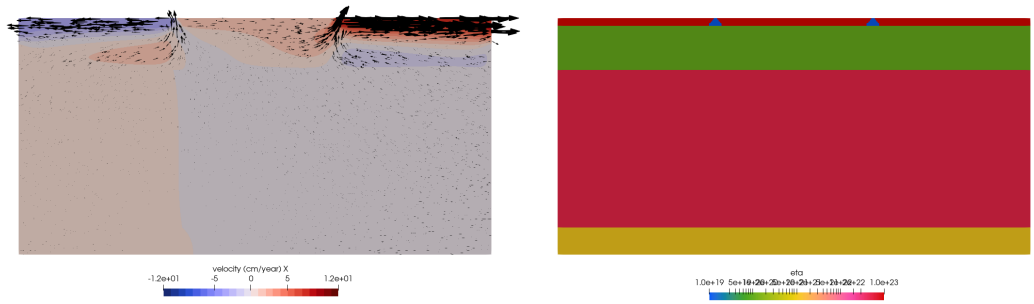
- experiment 1:



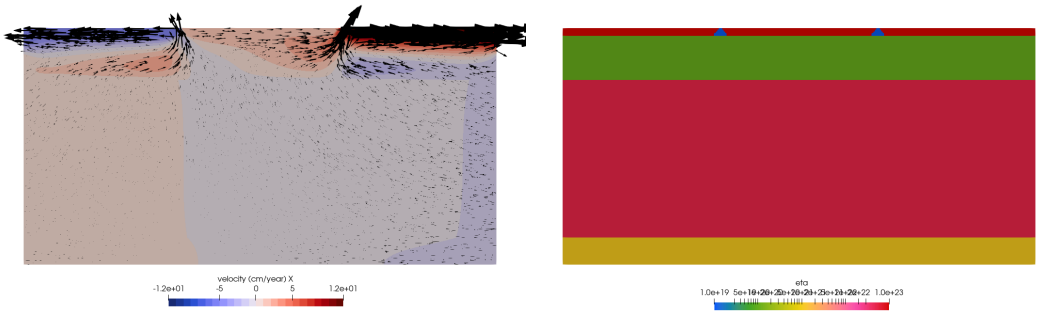
- experiment 2:



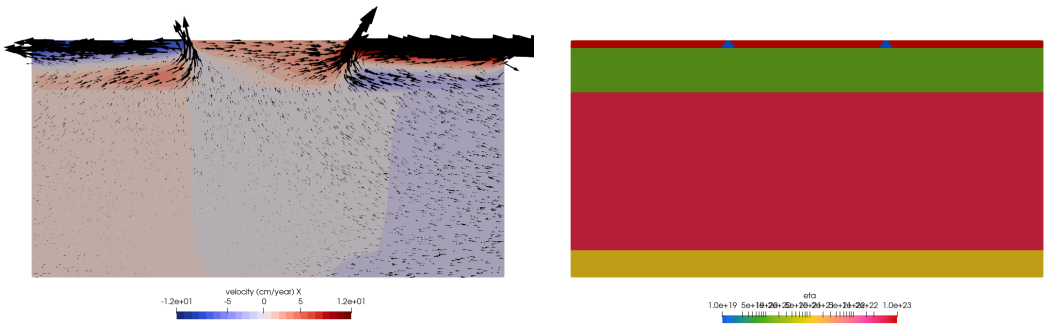
- experiment 3:



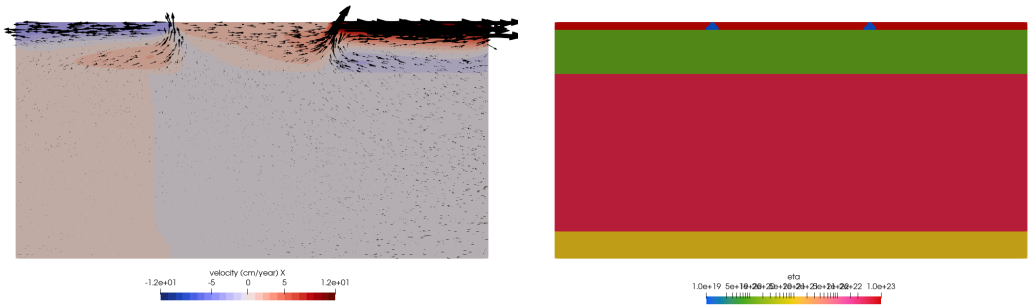
- experiment 4:



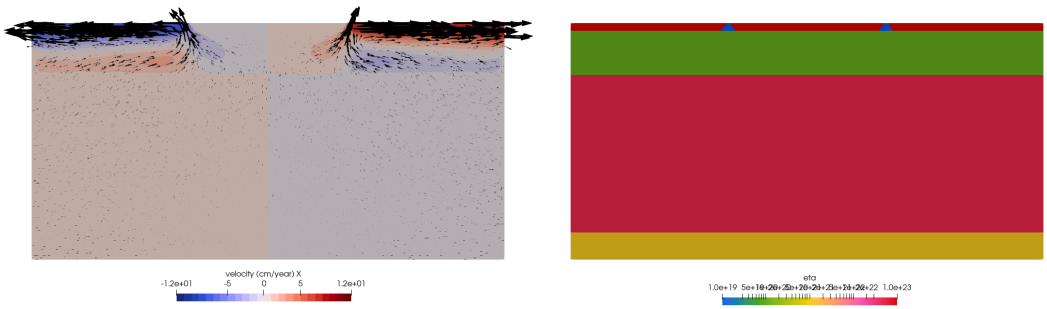
- experiment 5:



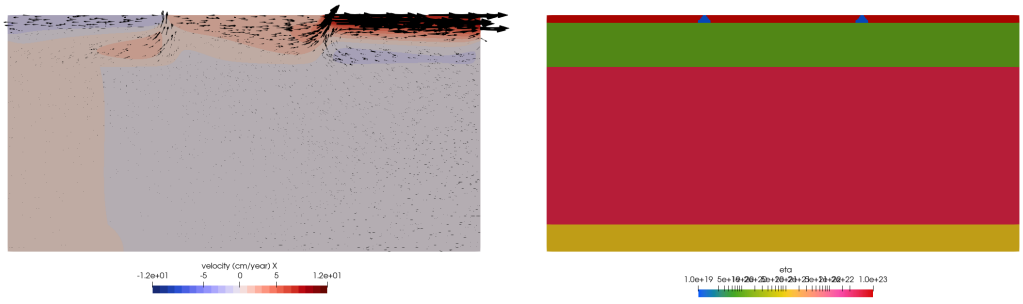
- experiment 6:



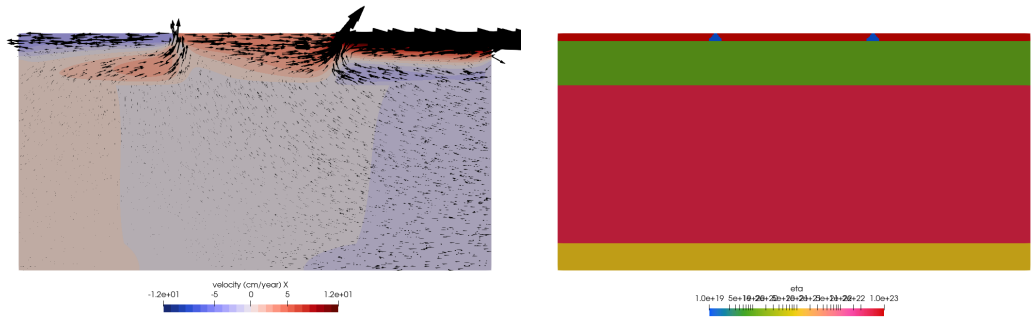
- experiment 7:



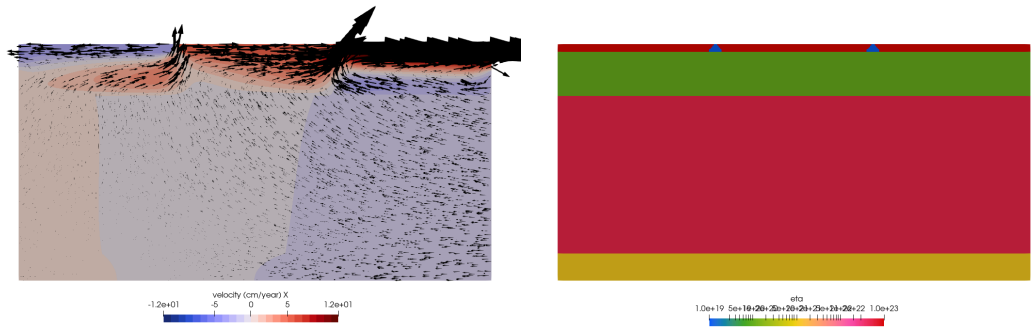
- experiment 8:



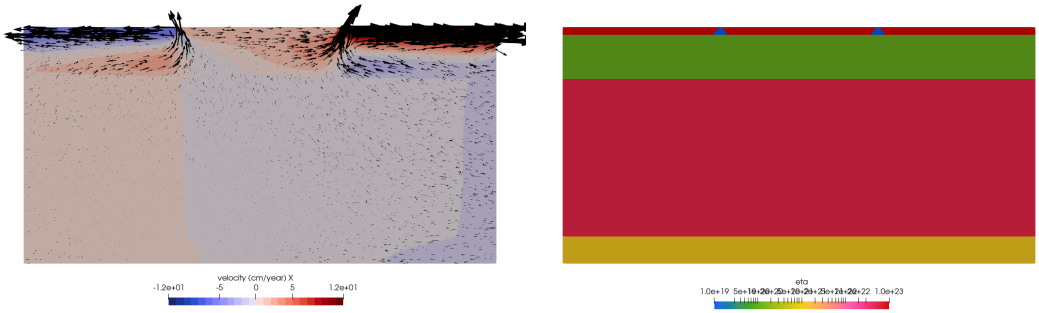
- experiment 9:



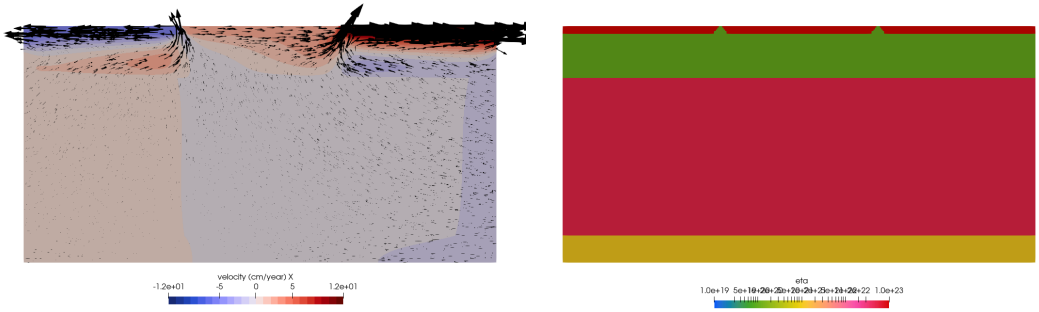
- experiment 10:



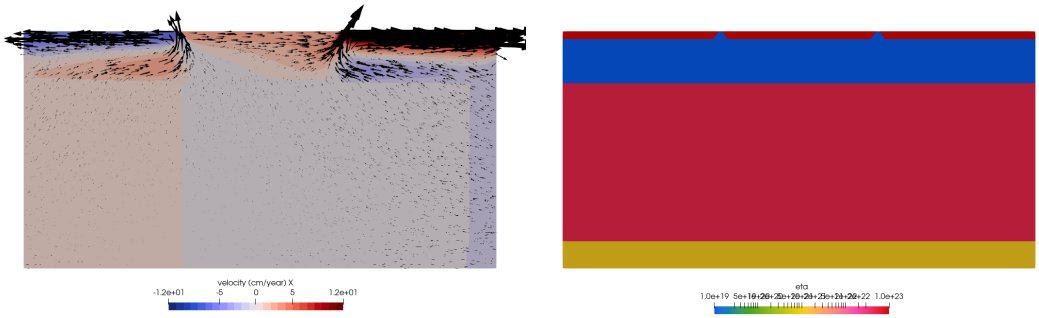
- experiment 11:



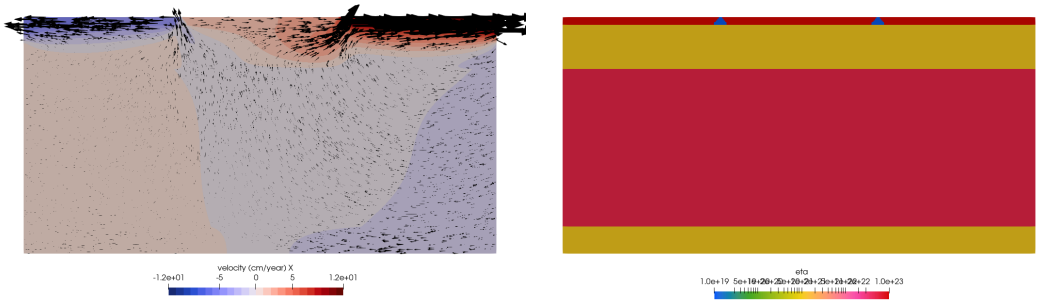
- experiment 12:



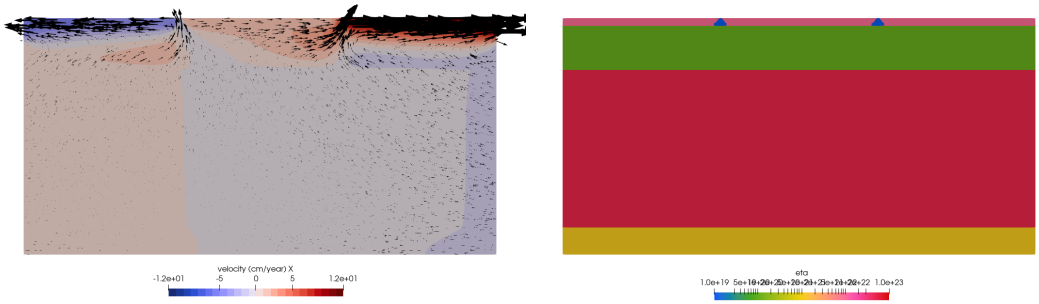
- experiment 13:



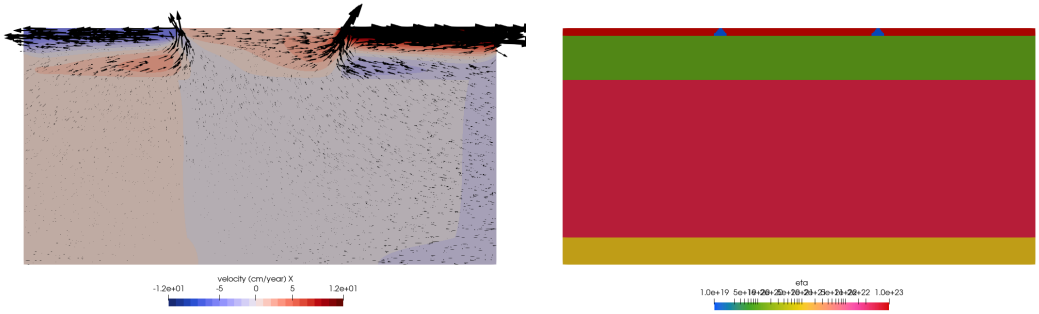
- experiment 14:



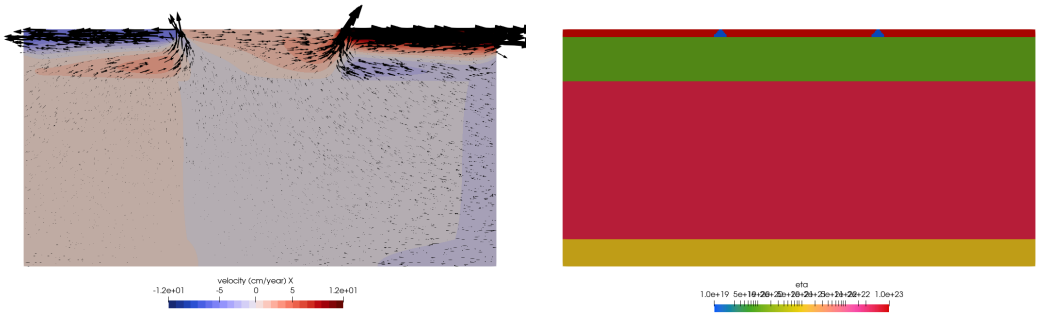
- experiment 15:



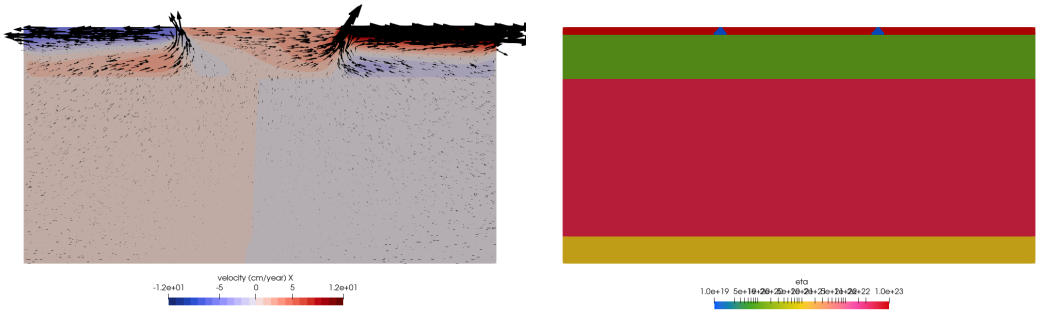
- experiment 16:



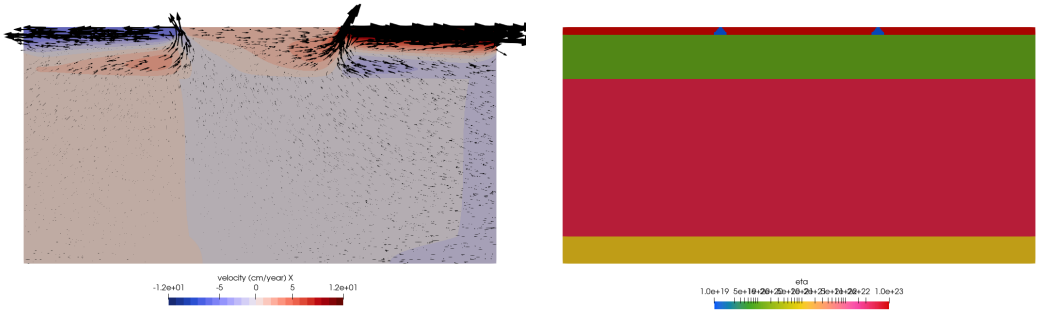
- experiment 17:



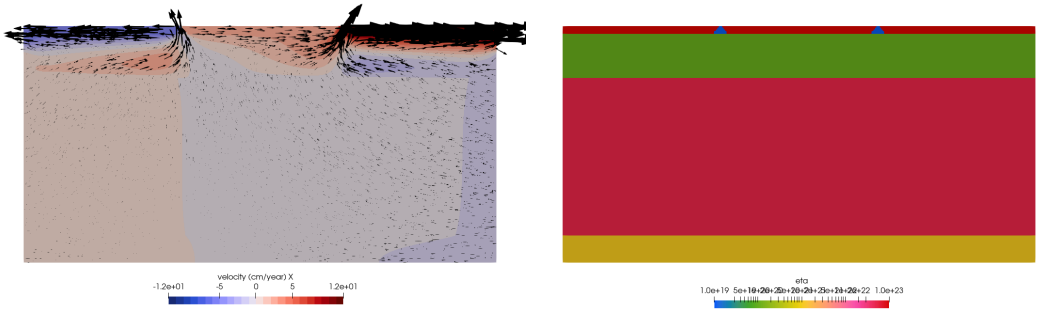
- experiment 18:



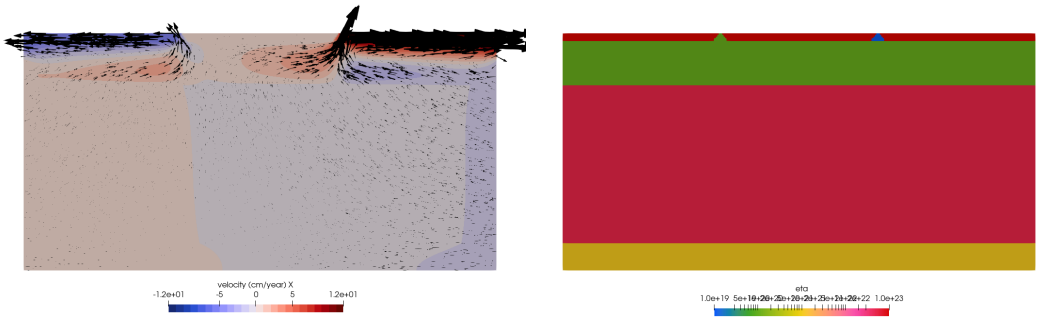
- experiment 19:



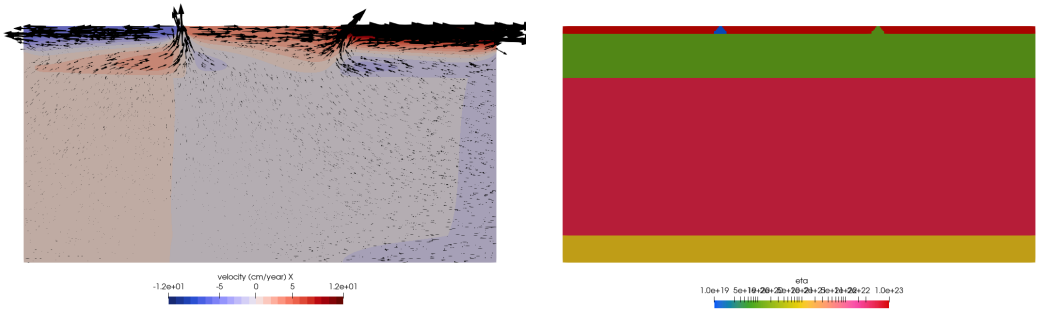
- experiment 20:



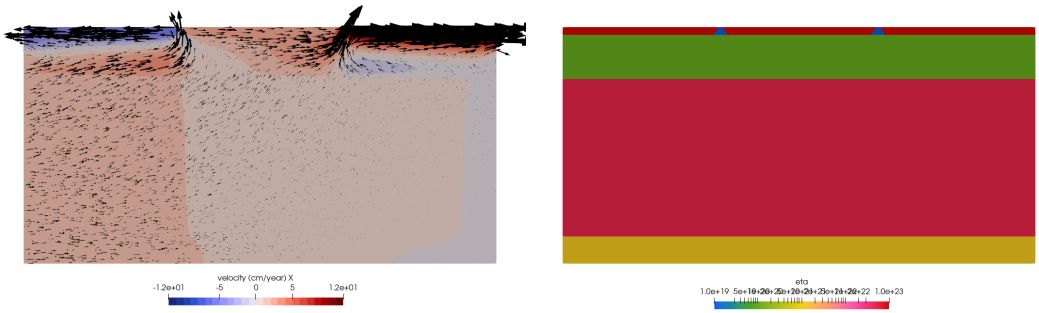
- experiment 21:



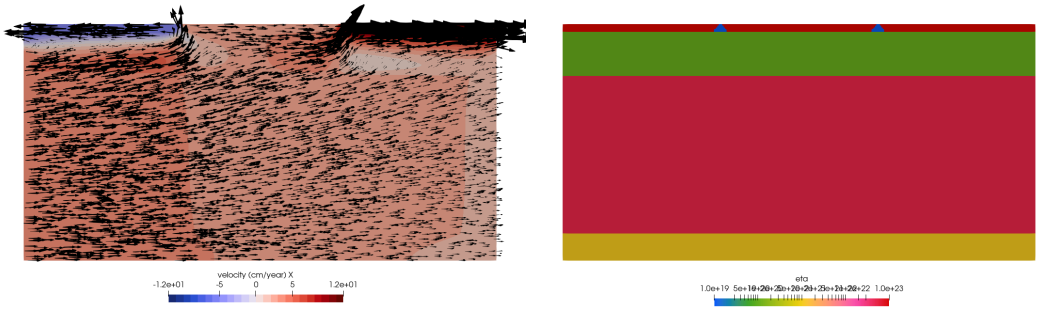
- experiment 22:



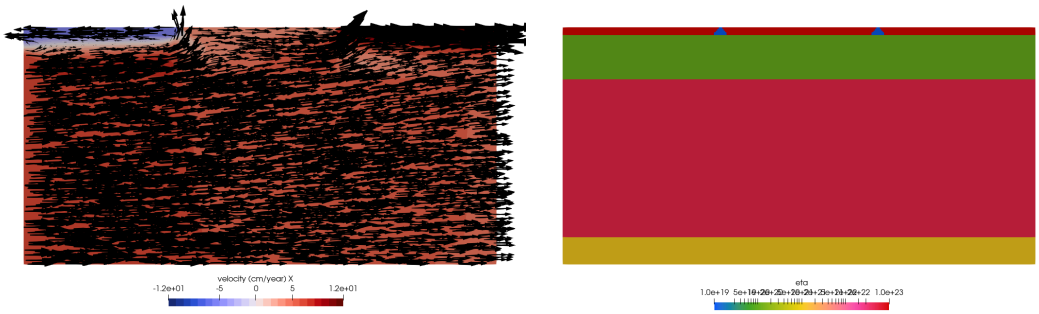
- experiment 23:



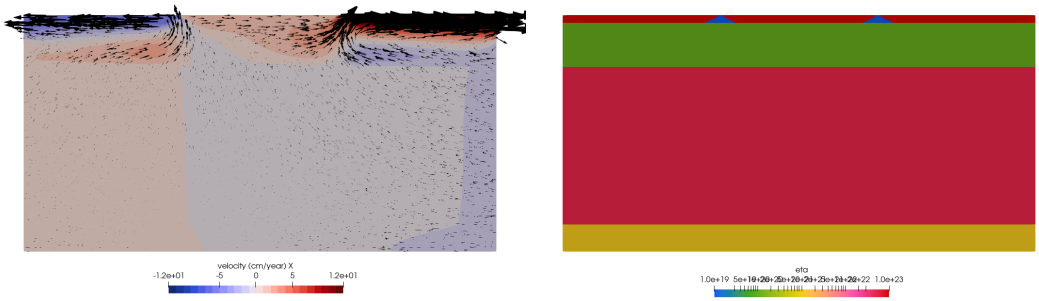
- experiment 24:



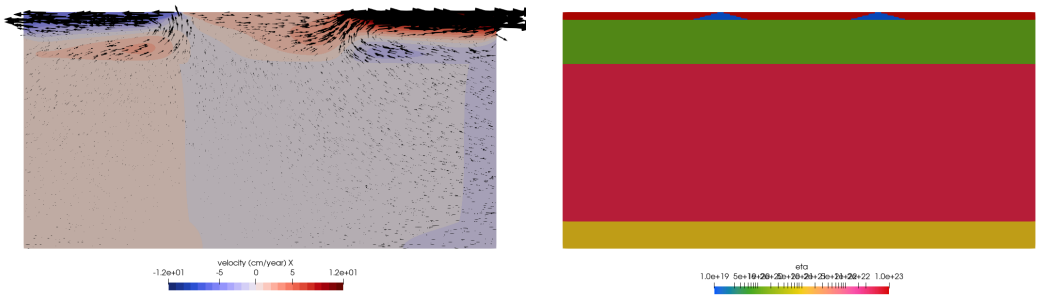
- experiment 25:



- experiment 26:



- experiment 27:



To Do, open questions, future work?

- better lithostatic pressure ?
- better open bc implementation

Relevant Literature:
xxxxYY

Stone 149

van Keken et al 2008 benchmark with bespoke quad mesh



python_codes/fieldstone_149/text.tex

2D
box
Q2Q1
numerical_benchmark
thermo_mechanical
time_stepping
use_mesher

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_149`

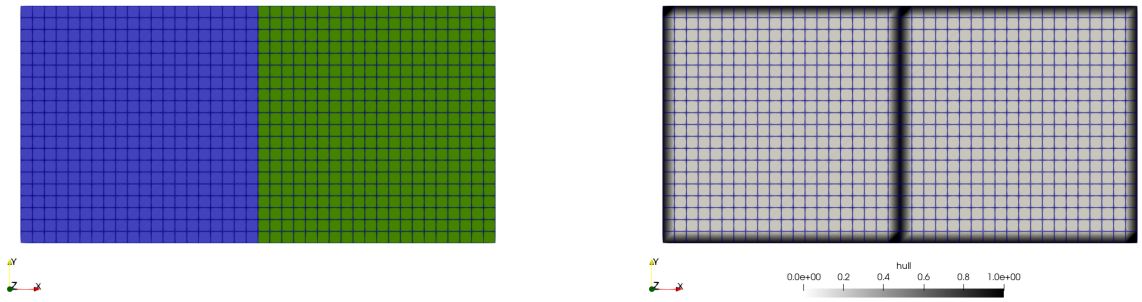
This stone was developed in with input from Daniel Douglas.

In the tools.py there are two functions:

- `merge_two_blocks`: it takes as arguments two meshes and returns a single mesh in which the overlapping nodes of both have been merged and the global numbering and connectivity array computed.
- `export_to_vtu`: it takes a mesh and exports it to vtu format.

Joining two simple and identical meshes

We start with two square meshes of resolution 20×20. This is implemented in stone_demo.py. Note that the nodes of a block that are on the boundary ('the hull') are flagged: indeed, only these are likely to be overlapping with a neighbouring block and merged.



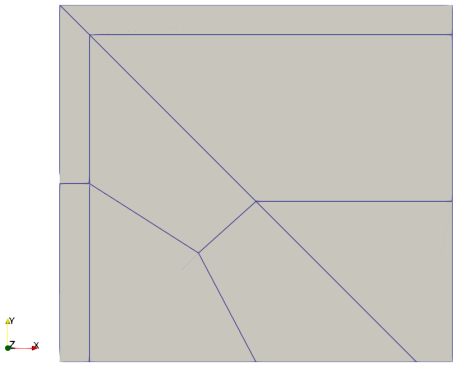
a) b)

a) two individual meshes. b) final assembled mesh showing the hull flag. Note that the current algorithm does not unflag nodes that are no more on the boundary, it simply merges the two hull fields.

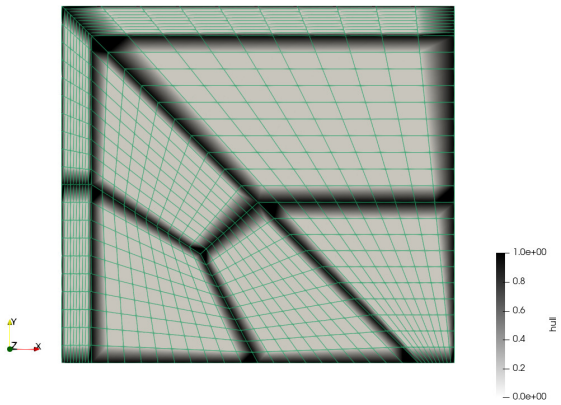
Joining multiple meshes

This proto-mesh contains 8 elements and 14 points.

```
12---------------13
| \               |
|  10------------11 icon[0:m,0]=[0,1,6,5]
|   | \           | icon[0:m,1]=[1,2,7,6]
|   |  \          | icon[0:m,2]=[2,3,8,7]
|   |   \         | icon[0:m,3]=[3,4,9,8]
|   |    \        | icon[0:m,4]=[5,6,10,12]
5---6     8-------9 icon[0:m,5]=[6,7,8,10]
|   | \   / \     | icon[0:m,6]=[8,9,11,10]
|   |   7-   \    | icon[0:m,7]=[10,11,13,12]
|   |     \   \   |
0---1-----2----3--4
```

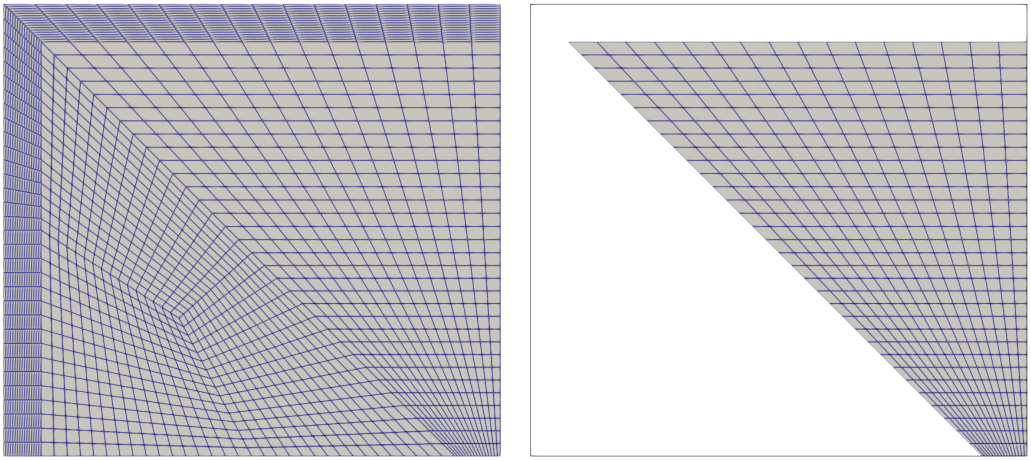


We then proceed by creating 8 blocks of resolution 10×10. We then map them and finally assemble them one by one to obtain:



About the code

This is a rather peculiar approach that we take here. We wish to solve the temperature equation on the full domain $L_x \times L_y$ while we only wish to solve the Stokes equations on the domain composed of blocks 3 and 6.



For simplicity (and because I have not yet implemented a version of the merging algorithm for second order elements), I will use the $Q_1^+ \times Q_1$ pair for Stokes and Q_1 elements for the temperature.

Also, the steady state energy equation is solved. Since the (linear) Stokes equations do not depend on temperature, we then only need to solve the Stokes equations first and the energy equation second.

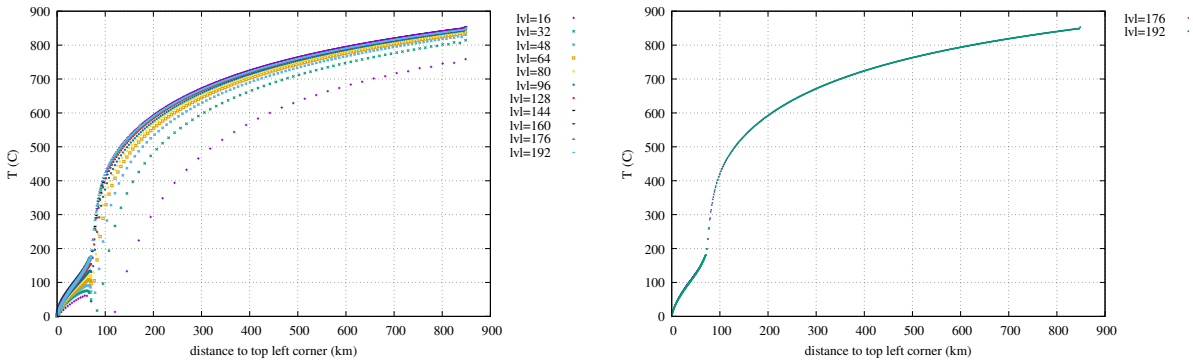
After the Stokes equations are solved we must transfer the velocity solution onto the temperature mesh. I have then created the `mapping` array that links each Stokes mesh point with a temperature mesh point. Note that this algorithm is *very* slow and could certainly be improved (currently about 500s for level=128!).

Once the temperature equation is solved I also compute the nodal heat flux vector $\vec{q} = -k\vec{\nabla}T$.

Case 1a

The Stokes equations are not solved. Instead the velocity field is prescribed analytically everywhere in the domain, using the corner flow solution in the mantle wedge.

The code uses analytical velocity from corner flow theory (see STONE 68) so the Stokes equations are (for now) not solved.



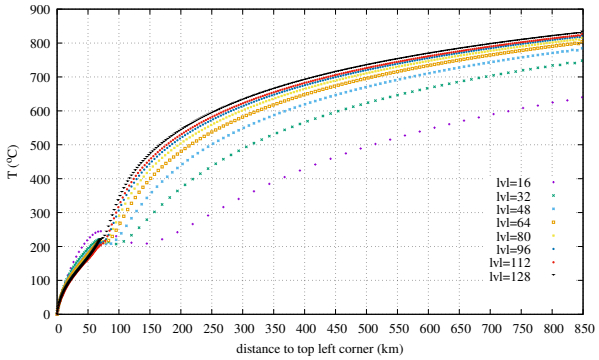
We see that results seem to become resolution independent when level> 160.

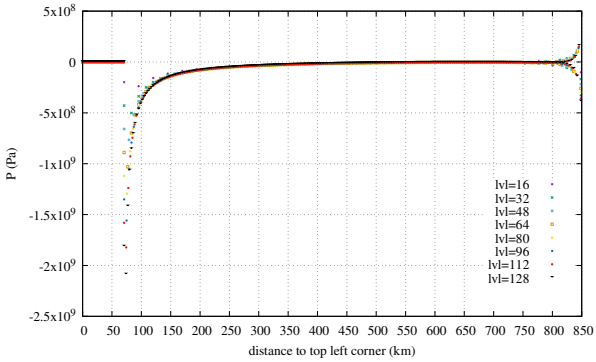
Case 1b - dynamical flow in isoviscous wedge I

This case is the same as 1a, except that the solution for the wedge flow is determined by solving the Stokes equations while the Batchelor solution is imposed on the inflow and outflow boundaries. This case tests the ability of the numerical method to accurately reproduce the corner flow solution.

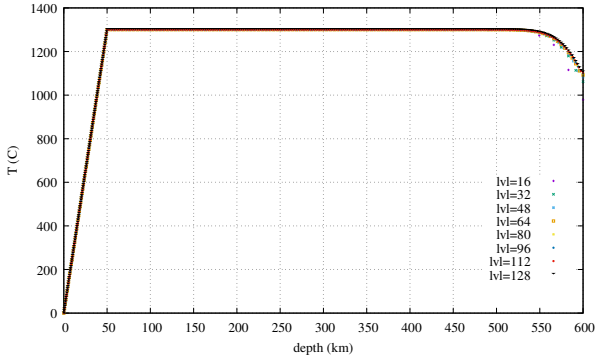
Case 1c - dynamical flow in isoviscous wedge II

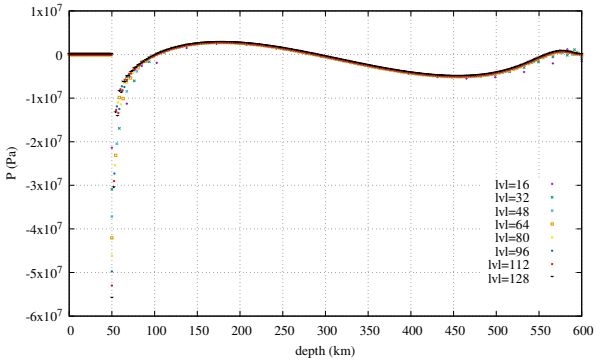
Same as case 1b, but with stress-free boundary conditions on the mantle wedge.

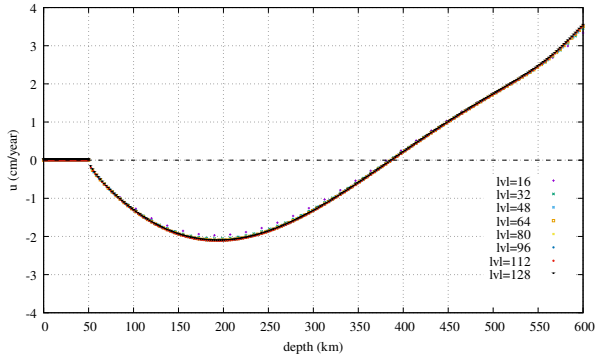


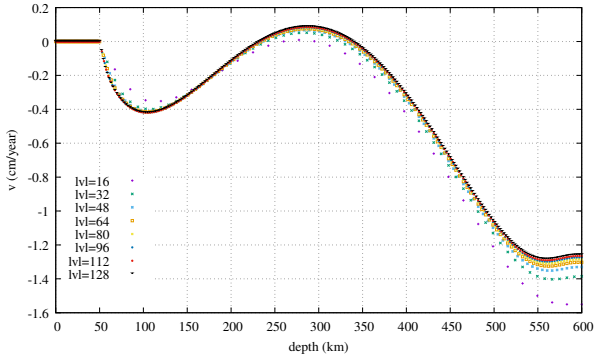


Measurements on the diagonal.









Measurements on the right side of the domain.

There seems to be a pressure instability/mode towards the bottom?! Is this because of the element?

To Do, open questions, future work?

- use f131 P1 to P2 tool to make Q1 to Q2 ?
- redo figures with appropriate colors
- redo (some) benchmark measurements?
- improve mapping algo?
- is it worth moving to Q2Q1 ? what about the weird pressure error of Q1+xQ1 ?

 Relevant Literature:

P.E. van Keken et al. “A community benchmark for subduction zone modelling”. In: *Phys. Earth. Planet. Inter.* 171 (2008), pp. 187–197. DOI: 10.1016/j.pepi.2008.04.015

Iris van Zelst, Cedric Thieulot, and Timothy J. Craig. “The effect of temperature-dependent material properties on simple thermal models of subduction zones”. In: *Solid Earth* 14 (2023), pp. 683–707. DOI: 10.5194/se-14-683-2023

Stone 150

Optimisation of FE matrix building process



python_codes/fieldstone_250/text.tex

```
2D
box
Q2Q1
sparse_matrix
analytical_benchmark
error_convergence
stokes_equation
vrms
paraview_output
```

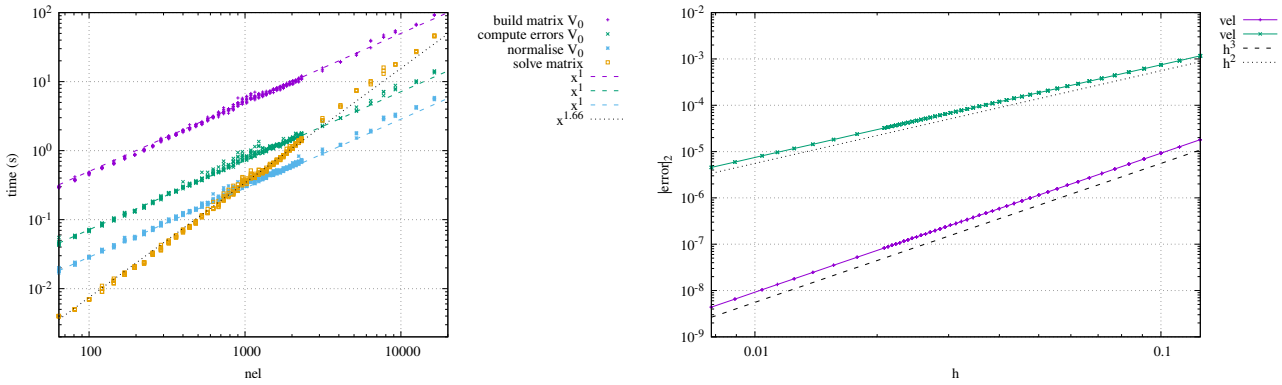
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_150`

This STONE is an attempt at looking into the optimisation of how (FE) stones have been coded up until now. In particular I want to try to build the (Stokes) FE matrix much faster. If successful, this would enable either higher resolution runs, or longer time series, or a quicker way to steady state, etc ...

We then start from the stone_V0.py script which solves the Donea & Huerta benchmark (see **??**) in a unit square by means of $Q_2 \times Q_1$elements. It is very similar to STONE 48 for example.

Standard approach

Running the model at resolution 128×128 elements, we find that building the matrix takes approximately 11.7s and solving the system approximatively 1.4s. Although the solve time grows like $nel^{1.66}$, even running a 256×256 elements model would still yields a build time that is substantially larger than the solve time. Also, the part of the code which computes the L_2 errors can take up to a few seconds (in general 10x less time than building the matrix). This is reason enough to try to make building the matrix *much* faster. Although there is no hard rule, it is commonly accepted that the solve is the most expensive part of a Stokes FE code.



Left: Timings of 4 parts of the code as a function of the number of elements. Right: L_2-error convergence plot as a function of the element size h.

Dealing with the Jacobian matrix code bits

To start with I here assume that we are dealing with rectangular elements, all of the same size (h_x, h_y). This allows to replace this typical piece of code

```
for iel in range(0,nel):
    for iq in range(0,nqperdim):
        for jq in range(0,nqperdim):
            ...
            jcb=np.zeros((ndim,ndim),dtype=np.float64)
            for k in range(0,mV):
                jcb[0,0] += dNNNVdr[k]*xV[iconV[k,iel]]
                jcb[0,1] += dNNNVdr[k]*yV[iconV[k,iel]]
                jcb[1,0] += dNNNVds[k]*xV[iconV[k,iel]]
                jcb[1,1] += dNNNVds[k]*yV[iconV[k,iel]]
            jcob = np.linalg.det(jcb)
            jcbi = np.linalg.inv(jcb)
```

by something much simpler. Indeed, in this case the Jacobian matrix is

$$\boldsymbol{J} = \begin{pmatrix} h_x/2 & 0 \\ 0 & h_y/2 \end{pmatrix}$$

so that the determinant is $|\boldsymbol{J}| = h_x h_y/4$ and its inverse:

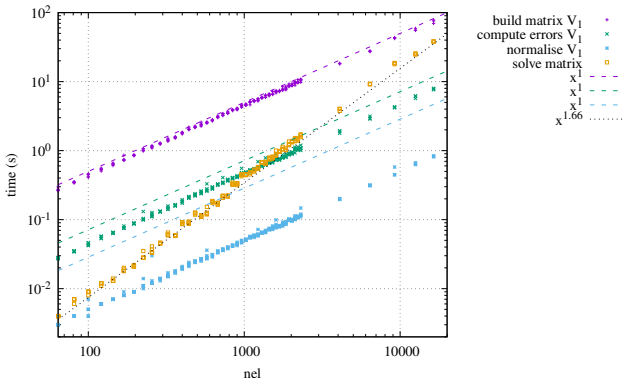
$$\boldsymbol{J}^{-1} = \begin{pmatrix} 2/h_x & 0 \\ 0 & 2/h_y \end{pmatrix}$$

We can then replace the code excerpt above (which is usually called at every quadrature of every element) by:

```
jcbi=np.zeros((ndim,ndim),dtype=np.float64)
jcbi[0,0]=2/hx
jcbi[1,1]=2/hy
jcob=hx*hy/4

for iel in range(0,nel):
    for iq in range(0,nqperdim):
        for jq in range(0,nqperdim):
            ...
```

Note that the explicit calculation of the Jacobian determinant and inverse is also present in the pressure normalisation and error calculations.

This is very simple and is carried out in stone_V1.py.



Timings of 4 parts of the code as a function of the number of elements. The dotted lines correspond to the timings of version V0.

	V0	V1
build matrix (128×128)	77.3-82.4 s	69.7-78 s
normalise pressure(128×128)	s	0.81-0.84 s
compute errors (128×128)	12.2-12.9 s	7.6-8 s

We see that this change has not really lead to a substantial improvement in computational time, especially for the build time (only 10% improvement)...

Dealing with the basis functions

In about every FE code so far, we encounter this bit:

```
NNNV     = np.zeros(mV, dtype=np.float64)
NNNP     = np.zeros(mP, dtype=np.float64)
dNNNVdr = np.zeros(mV, dtype=np.float64)
dNNNVds = np.zeros(mV, dtype=np.float64)

for iel in range(0,nel):
    for iq in range(0,nqperdim):
        for jq in range(0,nqperdim):
            rq=qcoords[iq]
            sq=qcoords[jq]
            weightq=qweights[iq]*qweights[jq]

            NNNV[0:mV]=NNV(rq,sq)
            dNNNVdr[0:mV]=dNNVdr(rq,sq)
            dNNNVds[0:mV]=dNNVds(rq,sq)
            NNNP[0:mP]=NNP(rq,sq)
```

If we look at it more closely, we see that the basis functions are evaluated for every element at the (here) 9 reduced coordinates of the quadrature points. We should then instead precompute the values of the basis functions at the reduced coordinates of the `nqel=nqperdim**ndim` quadrature points as follows:

```
NNNV     = np.zeros((nqel,mV),dtype=np.float64)
NNNP     = np.zeros((nqel,mP),dtype=np.float64)
dNNNVdr = np.zeros((nqel,mV),dtype=np.float64)
dNNNVds = np.zeros((nqel,mV),dtype=np.float64)
rq = np.zeros(nqel,dtype=np.float64)
sq = np.zeros(nqel,dtype=np.float64)
weightq = np.zeros(nqel,dtype=np.float64)

counterq=0
for iq in range(0,nqperdim):
    for jq in range(0,nqperdim):
        rq[counterq]=qcoords[iq]
        sq[counterq]=qcoords[jq]
        weightq[counterq]=qweights[iq]*qweights[jq]
        NNNV[counterq,0:mV]=NNV(rq,sq)
        dNNNVdr[counterq,0:mV]=dNNVdr(rq,sq)
        dNNNVds[counterq,0:mV]=dNNVds(rq,sq)
        NNNP[counterq,0:mP]=NNP(rq,sq)
        counterq+=1
```

so that we can later reuse these where needed (note that we have transformed the double for loop on quadrature points into a single one):

```
for iel in range(0,nel):
    for iq in range(0,nqel):
        ...
        rq[iq],sq[iq],weightq[iq]
        NNNV[iq,0:mV]
        dNNNVdr[iq,0:mV]
        dNNNVds[iq,0:mV]
        NNNP[iq,0:mP]
```

Looking further down we then see that the inverse of the Jacobian matrix is used to compute the derivatives of the basis functions in the real coordinate space (x, y):

```
for k in range(0,mV):
    dNNNVdx[k]=jcbi[0,0]*dNNNVdr[k]+jcbi[0,1]*dNNNVds[k]
    dNNNVdy[k]=jcbi[1,0]*dNNNVdr[k]+jcbi[1,1]*dNNNVds[k]
```

From what follows above we know that `jcbi[0,1]` and `jcbi[1,0]` are both zero, so that in fact we have

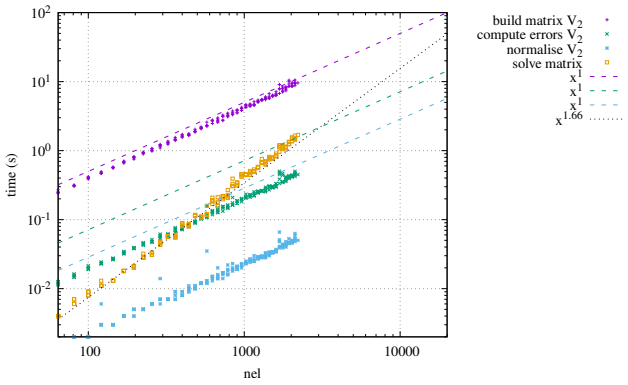
```
for k in range(0,mV):
    dNNNVdx[k]=jcbi[0,0]*dNNNVdr[k]
    dNNNVdy[k]=jcbi[1,1]*dNNNVds[k]
```

which means that these too can be precomputed at the quadrature points before entering the loop over elements:

```
NNNV     = np.zeros((nqel,mV),dtype=np.float64)
NNNP     = np.zeros((nqel,mP),dtype=np.float64)
dNNNVdr = np.zeros((nqel,mV),dtype=np.float64)
dNNNVds = np.zeros((nqel,mV),dtype=np.float64)
dNNNVdx = np.zeros((nqel,mV),dtype=np.float64)
dNNNVdy = np.zeros((nqel,mV),dtype=np.float64)
rq = np.zeros(nqel,dtype=np.float64)
sq = np.zeros(nqel,dtype=np.float64)
weightq = np.zeros(nqel,dtype=np.float64)

counterq=0
for iq in range(0,nqperdim):
    for jq in range(0,nqperdim):
        rq[counterq]=qcoords[iq]
        sq[counterq]=qcoords[jq]
        weightq[counterq]=qweights[iq]*qweights[jq]
        NNNV[counterq,0:mV]=NNV(rq,sq)
        dNNNVdr[counterq,0:mV]=dNNVdr(rq,sq)
        dNNNVds[counterq,0:mV]=dNNVds(rq,sq)
        NNNP[counterq,0:mP]=NNP(rq,sq)
        dNNNVdx[counterq,0:mV]=jcbi[0,0]*dNNNVdr[counterq,0:mV]
        dNNNVdy[counterq,0:mV]=jcbi[1,1]*dNNNVds[counterq,0:mV]
        counterq+=1

for iel in range(0,nel):
    for iq in range(0,nqel):
        ...
```

All of this now happens only once, before entering the element loop! This means that inside the element loop and the quadrature point loop, only the calculation of $\mathbb{K}_e$, the boundary conditions and the assembly must take place. In the very specific case that the viscosity is constant (at least within each element) one could also precompute $\mathbb{K}_e/\eta_e$ beforehand for example. Another interesting option would be to compute $\mathbb{K}_e$ for a linear viscosity in the element given by $\eta_e(x, y) = a + bx + cy$ (think symbolic calculation ?).

This is carried out in stone_V2.py.



Timings of 4 parts of the code as a function of the number of elements. The dotted lines correspond to the timings of version V0.

Here two, we see that we have improved the timings of the normalisation and error calculations but not of the matrix building. One possible benefit from this is that the code is getting more compact. Note that of course we can verify that the measured errors themselves have not changed.

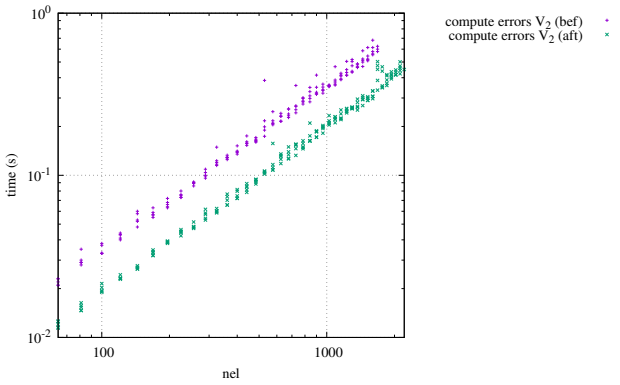
Side note: in the calculation of errors (and elsewhere), we had so far these explicit loops:

```
for iel in range (0,nel):
    for iq in range(0,nqel):
        xq=0.
        yq=0.
        uq=0.
        vq=0.
        qq=0.
        for k in range(0,mV):
            xq+=
            yq+=NNNV[iq,k]*yV[iconV[k,iel]]
            uq+=NNNV[iq,k]*u[iconV[k,iel]]
            vq+=NNNV[iq,k]*v[iconV[k,iel]]
            qq+=NNNV[iq,k]*q[iconV[k,iel]]
        errv+=((uq-velocity_x(xq,yq))**2+(vq-velocity_y(xq,yq))**2)*weightq[iq]*jcob
        errq+=(qq-pressure(xq,yq))**2*weightq[iq]*jcob
```

This is easily replaced by a much more compact code:

```
for iel in range (0,nel):
    for iq in range(0,nqel):
        xq=np.dot(NNNV[iq,:],xV[iconV[:,iel]])
        yq=np.dot(NNNV[iq,:],yV[iconV[:,iel]])
        uq=np.dot(NNNV[iq,:],u[iconV[:,iel]])
        vq=np.dot(NNNV[iq,:],v[iconV[:,iel]])
        qq=np.dot(NNNV[iq,:],q[iconV[:,iel]])
        errv+=((uq-velocity_x(xq,yq))**2+(vq-velocity_y(xq,yq))**2)*weightq[iq]*jcob
        errq+=(qq-pressure(xq,yq))**2*weightq[iq]*jcob
```

Such approach has been taken where ever possible in the stone. This makes bits of code even more compact (we have lost 30+ lines so far). As it turns out, this is also much faster (about 30%), as shown here for the error calculations:



All in all, there is an improvement in the timings, but nothing really substantial with regards to the matrix building. Let's dig deeper.

Improving matrix building (1)

We see that in the V2 version we have

```
for i in range(0,mP):
    N_mat[0,i]=NNNP[iq,i]
    N_mat[1,i]=NNNP[iq,i]
    N_mat[2,i]=0.
```

which can be replaced by

```
N_mat[0,0:mP]=NNNP[iq,0:mP]
N_mat[1,0:mP]=NNNP[iq,0:mP]
```

but it barely registers in terms on compute time. As this stage we have the building matrix code that looks as follows:

```
for iel in range(0,nel):
    loop over qpts to build K_el,G_el,f_el
    impose boundary conditions
    assemble
```

The question is then: which part is the most costly? In order to have an idea, we can implement something like this:

```
for iel in range(0,nel):
    start1 = timing.time()
    loop over qpts to build K_el,G_el,f_el
    end1 = timing.time()
    --
    start2 = timing.time()
    impose boundary conditions
    end2 = timing.time()
    --
    start3 = timing.time()
    assemble
    end3 = timing.time()
    --
    t1+=end1-start1
    t2+=end2-start2
    t3+=end3-start3
```

This is carried out in stone_V3.py. This is probably not ideal since these calls probably slow the execution of the element loop somewhat, but it proves to be informative:

resolution	t1	t2	t3	t1+t2+t3	t3/t1
32×32	0.90 s	0.06 s	3.12 s	4.08 s	3.3
48×48	1.93 s	0.11 s	6.61 s	9.24 s	3.4
96×96	7.86 s	0.40 s	26.34 s	35.64 s	3.4
128×128	13.87 s	0.68 s	47.04 s	66.38 s	3.5

The assembly seems to be the part that takes the most time (about 3x the quadrature points loop) so this is now our focus.

The assembly for each element looks like:

```
for k1 in range(0,mV):
    for i1 in range(0,ndofV):
        ikk=ndofV*k1           +i1
        m1 =ndofV*iconV[k1,iel]+i1
        for k2 in range(0,mV):
            for i2 in range(0,ndofV):
                jkk=ndofV*k2           +i2
                m2 =ndofV*iconV[k2,iel]+i2
                A_sparse[m1,m2] += K_el[ikk,jkk]
```

Effectively the loops over `k1+i1` and `k2+i2` loop over all terms of the `K_el`. Since we know that explicit for loops are slow in python, we could try to rewrite this as follows:

```
for ikk in range(0,ndofV*mV):
    for jkk in range(0,ndofV*mV):
        A_sparse[X,Y] += K_el[ikk,jkk]
```

where the values X and Y have been precomputed for each element. In the case of $Q_2 \times Q_1$elements in 2D, we have mV=9 and ndofV=2 so that the size of the matrix `K_el` is 18×18. For each of these 18 dofs I need to know its corresponding global dof number. I can precompute this for each element as follows:

```
local_to_globalV=np.zeros((ndofV_el,nel),dtype=np.int32)
for iel in range(0,nel):
    for k1 in range(0,mV):
        for i1 in range(0,ndofV):
            ikk=ndofV*k1           +i1
            m1 =ndofV*iconV[k1,iel]+i1
            local_to_globalV[ikk,iel]=m1
```

and the assembly has now become more compact:

```
    for ikk in range(ndofV_el):
        m1=local_to_globalV[ikk,iel]
        for jkk in range(ndofV_el):
            A_sparse[m1,local_to_globalV[jkk,iel]] += K_el[ikk,jkk]
        for jkk in range(0,mP):
            m2 =iconP[jkk,iel]
            A_sparse[m1,NfemV+m2]+=G_el[ikk,jkk]
            A_sparse[NfemV+m2,m1]+=G_el[ikk,jkk]
        f_rhs[m1]+=f_el[ikk]
    for k2 in range(0,mP):
        m2=iconP[k2,iel]
        h_rhs[m2]+=h_el[k2]
```

This is carried out in stone_V4.py. Unfortunately, the improvement is just not very substantial again (a few percents at best).

resolution	t1	t2	t3	t1+t2+t3	t3/t1
32×32	0.88 s	0.05 s	2.11 s	3.04 s	2.40
48×48	2.04 s	0.11 s	4.96 s	7.12 s	2.42
96×96	7.90 s	0.40 s	18.80 s	27.11 s	2.38
128×128	14.32 s	0.72 s	35.93 s	50.97 s	2.51

Turning now to the calculation of the elemental matrices, I put a timer around each piece of code:

```
    for iq in range(0,nqel):

        start11 = timing.time()
        xq=np.dot(NNNV[iq,:],xV[iconV[:,iel]])
        yq=np.dot(NNNV[iq,:],yV[iconV[:,iel]])
        end11 = timing.time()

        start12 = timing.time()
        for i in range(0,mV):
            b_mat[0:3, 2*i:2*i+2] = [[dNNNVdx[iq,i],0.          ],
                                     [0.          ,dNNNVdy[iq,i]],
                                     [dNNNVdy[iq,i],dNNNVdx[iq,i]]]
        end12 = timing.time()

        start13 = timing.time()
        K_el+=b_mat.T.dot(c_mat.dot(b_mat))*eta*weightq[iq]*jcob
        end13 = timing.time()

        start14 = timing.time()
        for i in range(0,mV):
            f_el[ndofV*i  ]+=NNNV[iq,i]*jcob*weightq[iq]*bx(xq,yq)
            f_el[ndofV*i+1]+=NNNV[iq,i]*jcob*weightq[iq]*by(xq,yq)
        end14 = timing.time()

        start15 = timing.time()
        N_mat[0,0:mP]=NNNP[iq,0:mP]
        N_mat[1,0:mP]=NNNP[iq,0:mP]
        end15 = timing.time()

        start16 = timing.time()
        G_el-=b_mat.T.dot(N_mat)*weightq[iq]*jcob
        end16 = timing.time()

        t11+=end11-start11
```

```
        t12+=end12-start12
        t13+=end13-start13
        t14+=end14-start14
        t15+=end15-start15
        t16+=end16-start16

    # end for iq
```

I found that t14 was rather large, but it actually comes from the complicated polynomial calculations inside bx and by. The second most costly was t12. A lucky guess had me replace

```
for i in range(0,mV):
    b_mat[0:3, 2*i:2*i+2] = [[dNNNVdx[iq,i],0.              ],
                             [0.           ,dNNNVdy[iq,i]],
                             [dNNNVdy[iq,i],dNNNVdx[iq,i]]]
```

by

```
for i in range(0,mV):
    b_mat[0,2*i  ]=dNNNVdx[iq,i]
    b_mat[1,2*i+1]=dNNNVdy[iq,i]
    b_mat[2,2*i  ]=dNNNVdy[iq,i]
    b_mat[2,2*i+1]=dNNNVdx[iq,i]
```

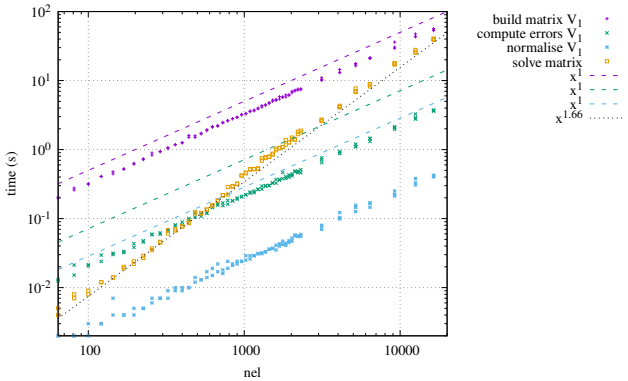
and I obtained an improvement for t1:

resolution	t1	t2	t3	t1+t2+t3	t3/t1
32×32	0.77	0.05	2.06	2.89	2.69
48×48	1.75	0.11	4.92	6.79	2.81
96×96	7.26	0.42	19.97	27.65	2.75
128×128	12.66	0.67	35.13	48.46	2.77

Again, nothing spectacular. Looking at this block again, we see that we have 4 times the occurence of the product `jcob*weightq[iq]`. We should therefore compute it once and store it:

```
JxW=jcob*weightq[iq]
```

All in all, this is where we are at:



The matrix build for 128x128 used to take about 80s for V0, now it takes about 55s. Let us be honest, I was hoping for at least a factor 2.

Dealing with the matrix storage

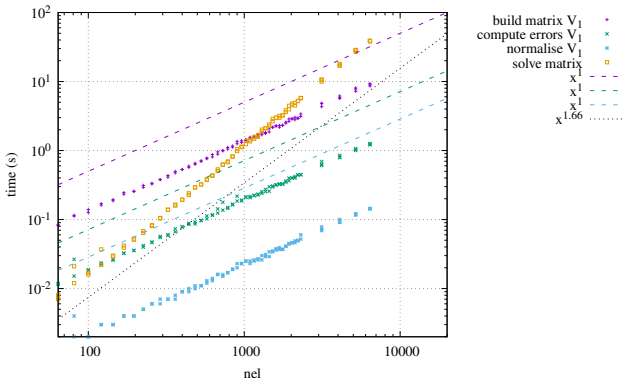
One thing we have not tried yet is to not use a linked list approach to store the FE matrix but instead use a full array approach, i.e.

```
A_sparse=np.zeros((Nfem,Nfem),dtype=np.float64)
...
sparse_matrix=sps.csr_matrix(A_sparse)
```

instead of

```
A_sparse=lil_matrix((Nfem,Nfem),dtype=np.float64)
...
sparse_matrix=A_sparse.tocsr()
```

This is carried out in stone_V5.py.



The build time is improved BUT there are two problems: a) converting the matrix to csr format before passing it to the solver takes more time and therefore shifts the solve time up; b) the simulation stops at resolution 96x96 as the python interpreter complains about the memory to store the matrix: "Unable to allocate 165. GiB for an array with shape (148739, 148739) and data type float64".

Not giving up on matrix assembly

Upon searching online, I came across this example:

```
from scipy import sparse
from numpy import array
I = array([0,0,1,3,1,0,0])
J = array([0,2,1,3,1,0,0])
V = array([1,1,1,1,1,1,1])
B = sparse.coo_matrix((V,(I,J)),shape=(4,4)).tocsr()
```

This is straightforward: keep a list of the integer coordinates of the nonzeros and their values in a large array and let the class `coo_matrix` assemble it all in a large matrix. This is also explained there[1].

In our case the size of these arrays will be large. They will contain *all* the values of all $\mathbb{K}_e$ and $\mathbb{G}_e$! I have written a short script compute_sizes.py which computes the required memory to store these arrays:

```
nel=256**2
ndofV=2
mV=9
mP=4
size=nel*( (mV*ndofV)**2 + 2*(mV*ndofV*mP) )
print('array size=',size)
print('--> arrays I,J are int32')
print('    ',size*32,'bits')
print('    ',size*32/8,'bytes')
print('    ',size*32/8/1024,'kbytes')
print('    ',size*32/8/1024/1024,'Mbytes')
print('    ',size*32/8/1024/1024/1024,'Gbytes')

print('--> array V is float64')
print('    ',size*64,'bits')
print('    ',size*64/8,'bytes')
print('    ',size*64/8/1024,'kbytes')
print('    ',size*64/8/1024/1024,'Mbytes')
print('    ',size*64/8/1024/1024/1024,'Gbytes')
```

We then get for a 256x256 mesh of $Q_2 \times Q_1$elements:

```
array size= 30670848
--> arrays I,J are int32
```

[1] https://stackoverflow.com/questions/10522296/how-to-assemble-large-sparse-matrices-effectively-in-python-scipy

```
     981467136 bits
     122683392.0 bytes
     119808.0 kbytes
     117.0 Mbytes
     0.1142578125 Gbytes each
--> array V is float64
     1962934272 bits
     245366784.0 bytes
     239616.0 kbytes
     234.0 Mbytes
     0.228515625 Gbytes
```

or, less than a Gb of memory to store it all!

This is carried out in stone_V6.py. and the assembly then looks like:

```
bignb=nel*((mV*ndofV)**2+2*(mV*ndofV*mP))

I=np.zeros(bignb,dtype=np.int32)
J=np.zeros(bignb,dtype=np.int32)
V=np.zeros(bignb,dtype=np.float64)

counter=0
for iel in range(0,nel):

    ... build K_el
    ... apply boundary conditions

    for ikk in range(ndofV_el):
        m1=local_to_globalV[ikk,iel]
        for jkk in range(ndofV_el):
            m2=local_to_globalV[jkk,iel]
            I[counter]=m1
            J[counter]=m2
            V[counter]=K_el[ikk,jkk]
            counter+=1
        for jkk in range(0,mP):
            m2 =iconP[jkk,iel]+NfemV
            I[counter]=m1
            J[counter]=m2
            V[counter]=G_el[ikk,jkk]
            counter+=1
            I[counter]=m2
            J[counter]=m1
            V[counter]=G_el[ikk,jkk]
            counter+=1
        f_rhs[m1]+=f_el[ikk]
    for k2 in range(0,mP):
        m2=iconP[k2,iel]
        h_rhs[m2]+=h_el[k2]
    end3 = timing.time()

sparse_matrix = sparse.coo_matrix((V,(I,J)),shape=(Nfem,Nfem)).tocsr()
```

Also, filling `I` and `J` can be done once and for all if the mesh topology does not change which is always the case in our codes. This therefore takes place before the loop over elements which would represent a gain in the case of time stepping or nonlinear iterations when the FE matrix needs to be rebuilt multiple times.

In the end, the assembly process is very compact:

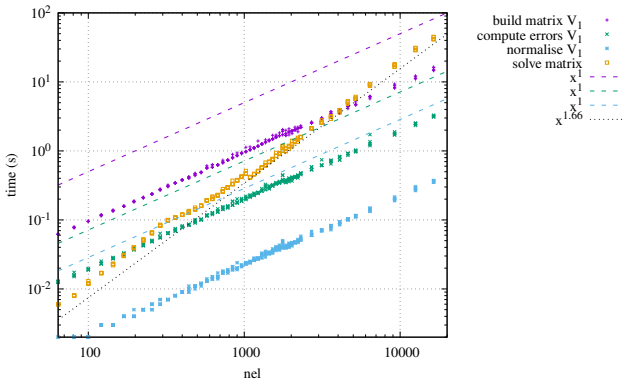
```
bignb=nel*((mV*ndofV)**2+2*(mV*ndofV*mP))

I=np.zeros(bignb,dtype=np.int32)
J=np.zeros(bignb,dtype=np.int32)
V=np.zeros(bignb,dtype=np.float64)

...fill I,J

counter=0
for iel in range(0,nel):

    ... build K_el
```

```
    ... apply boundary conditions

    for ikk in range(ndofV_el):
        m1=local_to_globalV[ikk,iel]
        for jkk in range(ndofV_el):
            V[counter]=K_el[ikk,jkk]
            counter+=1
        for jkk in range(0,mP):
            V[counter]=G_el[ikk,jkk]
            counter+=1
            V[counter]=G_el[ikk,jkk]
            counter+=1
        f_rhs[m1]+=f_el[ikk]
    for k2 in range(0,mP):
        m2=iconP[k2,iel]
        h_rhs[m2]+=h_el[k2]
    end3 = timing.time()

sparse_matrix = sparse.coo_matrix((V,(I,J)),shape=(Nfem,Nfem)).tocsr()
```

Note that the last line has been incorporated in the solve timing since it replaces the previous command which transformed the lil matrix into csr format.

resolution	t1	t2	t3	t1+t2+t3	t3/t1
32×32	0.70	0.05	0.14	0.90	0.21
48×48	1.62	0.11	0.33	2.06	0.21
96×96	6.33	0.38	1.28	8.00	0.20
128×128	12.24	0.71	2.55	15.50	0.21

Compared to V3, t3 has gone from 47s to less than 3s! Results are finally substantial! the matrix build has nearly decreased by a factor of 5 with respect to V0 while the solve time remains indentical! But now we need to look again at t1, i.e. the making of the element matrices $\mathbb{K}_e$ and $\mathbb{G}_e$. In the 128×128 case the call to bx,by costs about 7s from the 12s of t1, so that in the case of a buoyancy driven flow with for example $g_x = 0$ and $g_y = -g_0$ then t1 would be about 5s.



Back to computing elemental matrices / looking at b.c.

I have tried replacing in stone_V7.py

```
for i in range(0,mV):
    b_mat[0,2*i  ]=dNNNVdx[iq,i]
    b_mat[1,2*i+1]=dNNNVdy[iq,i]
    b_mat[2,2*i  ]=dNNNVdy[iq,i]
    b_mat[2,2*i+1]=dNNNVdx[iq,i]
```

by

```
for i in range(0,mV):
    dNdx=dNNNVdx[iq,i]
    dNdy=dNNNVdy[iq,i]
    b_mat[0,2*i  ]=dNdx
    b_mat[1,2*i+1]=dNdy
```

```
    b_mat[2,2*i  ]=dNdy
    b_mat[2,2*i+1]=dNdx
```

and found a minor improvement (few percents on t12).

I have then tried

```
b_mat[0,:]=[dNNNVdx[iq,0],0,dNNNVdx[iq,1],0,dNNNVdx[iq,2],0,\
            dNNNVdx[iq,3],0,dNNNVdx[iq,4],0,dNNNVdx[iq,5],0,\
            dNNNVdx[iq,6],0,dNNNVdx[iq,7],0,dNNNVdx[iq,8],0]
b_mat[1,:]=[0,dNNNVdy[iq,0],0,dNNNVdy[iq,1],0,dNNNVdy[iq,2],\
            0,dNNNVdy[iq,3],0,dNNNVdy[iq,4],0,dNNNVdy[iq,5],\
            0,dNNNVdy[iq,6],0,dNNNVdy[iq,7],0,dNNNVdy[iq,8]]
b_mat[2,:]=[dNNNVdy[iq,0],dNNNVdx[iq,0],\
            dNNNVdy[iq,1],dNNNVdx[iq,1],\
            dNNNVdy[iq,2],dNNNVdx[iq,2],\
            dNNNVdy[iq,3],dNNNVdx[iq,3],\
            dNNNVdy[iq,4],dNNNVdx[iq,4],\
            dNNNVdy[iq,5],dNNNVdx[iq,5],\
            dNNNVdy[iq,6],dNNNVdx[iq,6],\
            dNNNVdy[iq,7],dNNNVdx[iq,7],\
            dNNNVdy[iq,8],dNNNVdx[iq,8]]
```

which is of course ugly and not extendable to 3D or higher order elements. It did not yield any improvement at all so I abandoned this.

Likewise I replaced

```
for i in range(0,mV):
    f_el[ndofV*i  ]+=NNNV[iq,i]*JxW*bx(xq,yq)
    f_el[ndofV*i+1]+=NNNV[iq,i]*JxW*by(xq,yq)
```

by

```
for i in range(0,mV):
    Ni=NNNV[iq,i]*JxW
    f_el[ndofV*i  ]+=Ni*bx(xq,yq)
    f_el[ndofV*i+1]+=Ni*by(xq,yq)
```

and shave off a few percents off t14.

Finally, we can also make use of the `local_to_globalV` array in the boundary conditions, so that

```
for k1 in range(0,mV):
    for i1 in range(0,ndofV):
        ikk=ndofV*k1           +i1
        m1 =ndofV*iconV[k1,iel]+i1
        if bc_fix[m1]:
            ...
```

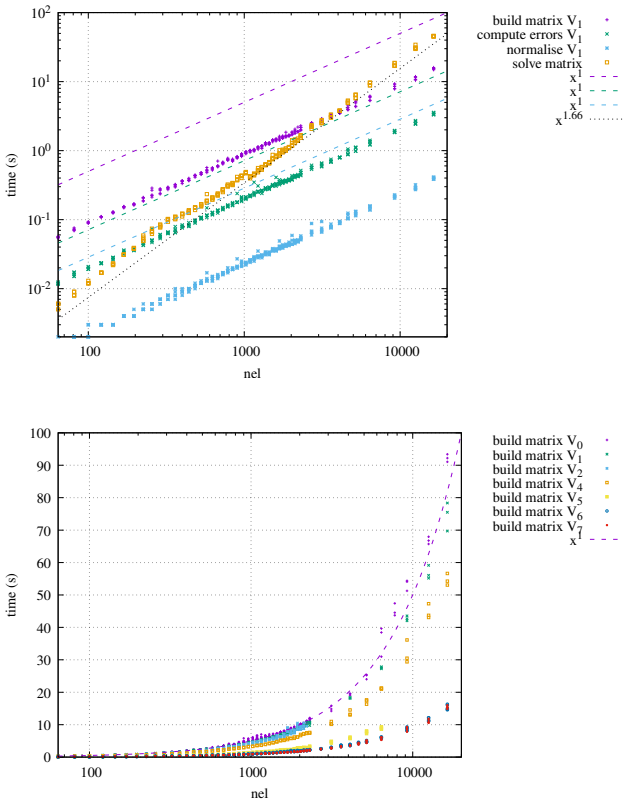
is replaced by

```
for ikk in range(0,ndofV_el):
    m1=local_to_globalV[ikk,iel]
    if bc_fix[m1]:
        ...
```

Once again, this is more compact and contains less nested loops.

resolution	t1	t2	t3	t1+t2+t3	t3/t1
32×32	0.71 s	0.019 s	0.14 s	0.87 s	0.20
48×48	1.54 s	0.034 s	0.32 s	1.90 s	0.21
96×96	6.23 s	0.094 s	1.38 s	7.71 s	0.22
128×128	10.82 s	0.140 s	2.29 s	13.25 s	0.21

We see that the boundary conditions time has been divided by 5!



To Do, open questions, future work?

- do smthg

Note that there is quite some literature about how to carry out the assembly in a very efficient manner.

- François Cuvelier, Caroline Japhet, and Gilles Scarella. "An efficient way to assemble finite element matrices in vector languages". In: *BIT Numerical Mathematics* 56 (2016), pp. 833–864. DOI: 10.1007/s10543-015-0587-4
- Yannis Voet. "On the fast assemblage of finite element matrices with application to nonlinear heat transfer problems". In: *Applied Mathematics and Computation* 436 (2023), p. 127516. DOI: 10.1016/j.amc.2022.127516

Stone 151

Free slip on curved geometries ($Q_2 \times Q_1$)



python_codes/fieldstone_151/text.tex

2D
annulus
Q2Q1
sparse_matrix
analytical_benchmark
error_convergence

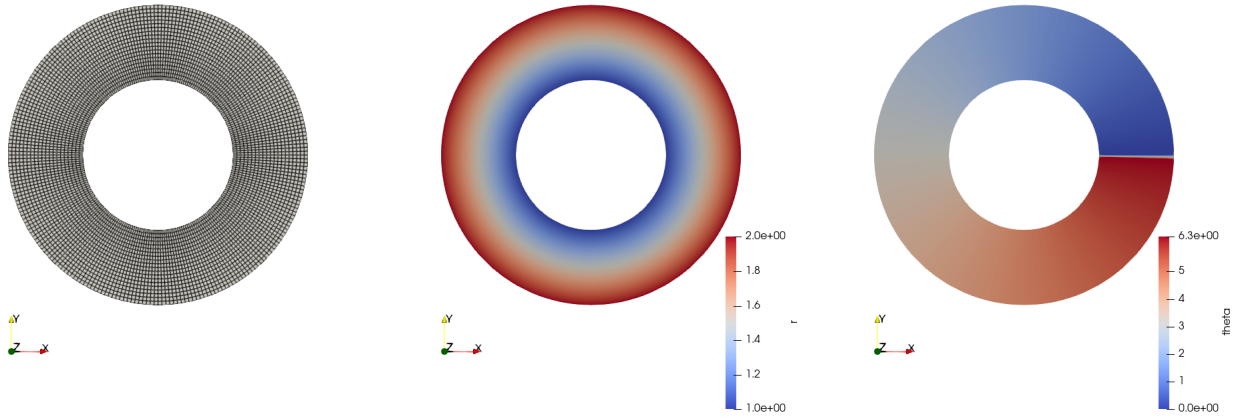
P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_151

This STONE builds on STONE 33 in the sense that in STONE 33 free slip boundary conditions were implemented and tested on an annulus discretised by means of $Q_1 \times P_0$ elements. We here implement and test the implementation of such boundary conditions on an annulus discretised by means of $Q_2 \times Q_1$ elements.

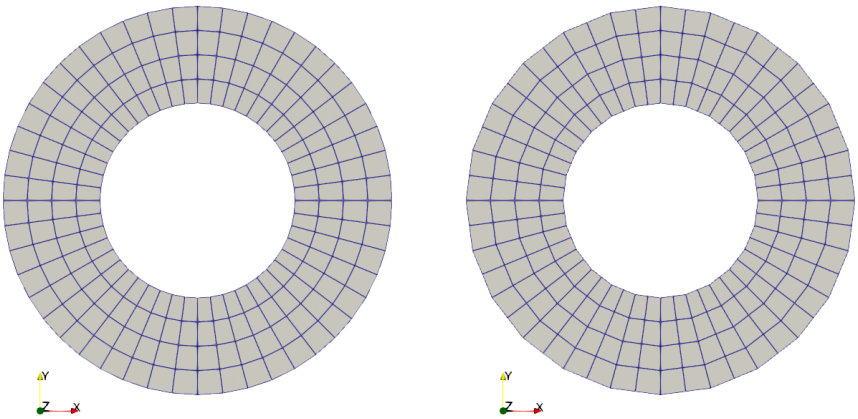
One could envisage three ways to implemenent these boundary conditions:

- the rotating method (see Section 10.14.4),
- constraints applied to elemental matrix,
- constraints via Lagrange multipliers

The code follows a standard approach when it comes to meshing, FEM, isoparametric mapping, quadrature, solving, etc ...and the fluid is isothermal and incompressible. Note that the code computes the strain rate tensor components by means of 3 different methods (it is based on STONE 21) but this is not relevant here. Also the mesh exported to the vtu file is based on a subdivision of the elements into four bilinear quadrilaterals. In other words it means that 4 elements in paraview form a single 'real' element in the code. By default all node are placed on concentric circles. However when the trapezes integer variable is set to 1 then all mid-edges nodes are then placed exactly in the middle of the corresponding corner nodes (obviously the middle node is then also moved to be at the center of the trapeze/element). Free slip boundary conditions are only implemented at the surface ($r = R_2$).



Mesh, r and θ coordinates.



Left: trapezes=0; right: trapezes=1.

The goal of this STONE is twofold:

- test the implementation of the free slip boundary conditions algorithms,
- explore the differences between straight edge elelements vs curved ones.

Method 1: rotating elemental matrices

This is explained in Section 10.14.4.

Method 2: constraints applied to elemental matrix

After asking W. Bangerth about the deal.II strategy for such boundary conditions his email answer was

```
------------------------------------------------------------
It's this function:

https://dealii.org/developer/doxygen/deal.II/group__constraints.html#gae3b53d69bd7afe08f0d0334978

In short, we compute constraints on individual nodes that correspond to the
  u.n = 0
constraint. In 2d, you get something like
  ux*nx + uy*ny = 0
which you convert into a constraint of the kind
  ux = - uy*ny/nx
or
  uy = - ux*nx/ny
depending on whether nx>ny or nx<ny, for numerical stability.
------------------------------------------------------------
```

What follows is mine only.

$$\begin{pmatrix} \mathbb{K}_e & \mathbb{G}_e \\ \mathbb{G}_e^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{V} \\ \vec{P} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ \vec{h} \end{pmatrix}$$

Let us assume that node 2 is on the boundary and we therefore wish to impose $\vec{v}_2 \cdot \vec{n} = 0$, or $u_2 n_x + v_2 n_y = 0$.

The equation above writes (for simplicity we take $m_v = 4$ and $m_p = 3$ which does not correspond to a real FE pair) with K_e of size 8×8 and G_e of size 8×3 in 2D:

$$\begin{array}{llllllllllll}
K_{11}u_1 & +K_{12}v_1 & +K_{13}u_2 & +K_{14}v_2 & +K_{15}u_3 & +K_{16}v_3 & +K_{17}u_4 & +K_{18}v_4 & +G_{11}p_1 & +G_{12}p_2 & +G_{13}p_3 & = f_1 \\
K_{21}u_1 & +K_{22}v_1 & +K_{23}u_2 & +K_{24}v_2 & +K_{25}u_3 & +K_{26}v_3 & +K_{27}u_4 & +K_{28}v_4 & +G_{21}p_1 & +G_{22}p_2 & +G_{23}p_3 & = f_2 \\
K_{31}u_1 & +K_{32}v_1 & +K_{33}u_2 & +K_{34}v_2 & +K_{35}u_3 & +K_{36}v_3 & +K_{37}u_4 & +K_{38}v_4 & +G_{31}p_1 & +G_{32}p_2 & +G_{33}p_3 & = f_3 \\
K_{41}u_1 & +K_{42}v_1 & +K_{43}u_2 & +K_{44}v_2 & +K_{45}u_3 & +K_{46}v_3 & +K_{47}u_4 & +K_{48}v_4 & +G_{41}p_1 & +G_{42}p_2 & +G_{43}p_3 & = f_4 \\
K_{51}u_1 & +K_{52}v_1 & +K_{53}u_2 & +K_{54}v_2 & +K_{55}u_3 & +K_{56}v_3 & +K_{57}u_4 & +K_{58}v_4 & +G_{51}p_1 & +G_{52}p_2 & +G_{53}p_3 & = f_5 \\
K_{61}u_1 & +K_{62}v_1 & +K_{63}u_2 & +K_{64}v_2 & +K_{65}u_3 & +K_{66}v_3 & +K_{67}u_4 & +K_{68}v_4 & +G_{61}p_1 & +G_{62}p_2 & +G_{63}p_3 & = f_6 \\
K_{71}u_1 & +K_{72}v_1 & +K_{73}u_2 & +K_{74}v_2 & +K_{75}u_3 & +K_{76}v_3 & +K_{77}u_4 & +K_{78}v_4 & +G_{71}p_1 & +G_{72}p_2 & +G_{73}p_3 & = f_7 \\
K_{81}u_1 & +K_{82}v_1 & +K_{83}u_2 & +K_{84}v_2 & +K_{85}u_3 & +K_{86}v_3 & +K_{87}u_4 & +K_{88}v_4 & +G_{81}p_1 & +G_{82}p_2 & +G_{83}p_3 & = f_8 \\
G_{11}u_1 & +G_{21}v_1 & +G_{31}u_2 & +G_{41}v_2 & +G_{51}u_3 & +G_{61}v_3 & +G_{71}u_4 & +G_{81}v_4 & +0 & +0 & +0 & = h_1 \\
G_{12}u_1 & +G_{22}v_1 & +G_{32}u_2 & +G_{42}v_2 & +G_{52}u_3 & +G_{62}v_3 & +G_{72}u_4 & +G_{82}v_4 & +0 & +0 & +0 & = h_2 \\
G_{13}u_1 & +G_{23}v_1 & +G_{33}u_2 & +G_{43}v_2 & +G_{53}u_3 & +G_{63}v_3 & +G_{73}u_4 & +G_{83}v_4 & +0 & +0 & +0 & = h_3
\end{array}$$

We must now find a way to include the contraint above in these equations. Typically we usually have $v_2 = v_{bc}$ and the system above is transformed as follows

$$\begin{array}{llllllllllll}
K_{11}u_1 & +K_{12}v_1 & +K_{13}u_2 & +0 & +K_{15}u_3 & +K_{16}v_3 & +K_{17}u_4 & +K_{18}v_4 & +G_{11}p_1 & +G_{12}p_2 & +G_{13}p_3 & = f_1 - K_{14}v_{bc} \\
K_{21}u_1 & +K_{22}v_1 & +K_{23}u_2 & +0 & +K_{25}u_3 & +K_{26}v_3 & +K_{27}u_4 & +K_{28}v_4 & +G_{21}p_1 & +G_{22}p_2 & +G_{23}p_3 & = f_2 - K_{24}v_{bc} \\
K_{31}u_1 & +K_{32}v_1 & +K_{33}u_2 & +0 & +K_{35}u_3 & +K_{36}v_3 & +K_{37}u_4 & +K_{38}v_4 & +G_{21}p_1 & +G_{32}p_2 & +G_{33}p_3 & = f_3 - K_{34}v_{bc} \\
0 & +0 & +0 & +K_{44}v_2 & +0 & +0 & +0 & +0 & +0 & +0 & +0 & = \\
K_{51}u_1 & +K_{52}v_1 & +K_{53}u_2 & +0 & +K_{55}u_3 & +K_{56}v_3 & +K_{57}u_4 & +K_{58}v_4 & +G_{21}p_1 & +G_{52}p_2 & +G_{53}p_3 & = f_5 - K_{54}v_{bc} \\
K_{61}u_1 & +K_{62}v_1 & +K_{63}u_2 & +0 & +K_{65}u_3 & +K_{66}v_3 & +K_{67}u_4 & +K_{68}v_4 & +G_{21}p_1 & +G_{62}p_2 & +G_{63}p_3 & = f_6 - K_{64}v_{bc} \\
K_{71}u_1 & +K_{72}v_1 & +K_{73}u_2 & +0 & +K_{75}u_3 & +K_{76}v_3 & +K_{77}u_4 & +K_{78}v_4 & +G_{21}p_1 & +G_{72}p_2 & +G_{73}p_3 & = f_7 - K_{74}v_{bc} \\
K_{81}u_1 & +K_{82}v_1 & +K_{83}u_2 & +0 & +K_{85}u_3 & +K_{86}v_3 & +K_{87}u_4 & +K_{88}v_4 & +G_{21}p_1 & +G_{82}p_2 & +G_{83}p_3 & = f_8 - K_{84}v_{bc} \\
G_{11}u_1 & +G_{21}v_1 & +G_{31}u_2 & +0 & +G_{51}u_3 & +G_{61}v_3 & +G_{71}u_4 & +G_{81}v_4 & +0 & +0 & +0 & = h_1 - G_{41}v_{bc} \\
G_{12}u_1 & +G_{22}v_1 & +G_{32}u_2 & +0 & +G_{52}u_3 & +G_{62}v_3 & +G_{72}u_4 & +G_{82}v_4 & +0 & +0 & +0 & = h_2 - G_{42}v_{bc} \\
G_{13}u_1 & +G_{23}v_1 & +G_{33}u_2 & +0 & +G_{53}u_3 & +G_{63}v_3 & +G_{73}u_4 & +G_{83}v_4 & +0 & +0 & +0 & = h_3 - G_{43}v_{bc}
\end{array}$$

This approach yields an elemental Stokes matrix that is still strictly symmetric. Since we only build G_e (and not G_e^T) in the code then zeroing a line in G_e means zeroing a line in G_e^T so the approach above reconciles this pb.

If $v_{bc} = 0$ then we obtain

$$\begin{array}{llllllllllll}
K_{11}u_1 & +K_{12}v_1 & +K_{13}u_2 & +0 & +K_{15}u_3 & +K_{16}v_3 & +K_{17}u_4 & +K_{18}v_4 & +G_{11}p_1 & +G_{12}p_2 & +G_{13}p_3 & = f_1 \\
K_{21}u_1 & +K_{22}v_1 & +K_{23}u_2 & +0 & +K_{25}u_3 & +K_{26}v_3 & +K_{27}u_4 & +K_{28}v_4 & +G_{21}p_1 & +G_{22}p_2 & +G_{23}p_3 & = f_2 \\
K_{31}u_1 & +K_{32}v_1 & +K_{33}u_2 & +0 & +K_{35}u_3 & +K_{36}v_3 & +K_{37}u_4 & +K_{38}v_4 & +G_{21}p_1 & +G_{32}p_2 & +G_{33}p_3 & = f_3 \\
0 & +0 & +0 & +K_{44}v_2 & +0 & +0 & +0 & +0 & +0 & +0 & +0 & = 0 \\
K_{51}u_1 & +K_{52}v_1 & +K_{53}u_2 & +0 & +K_{55}u_3 & +K_{56}v_3 & +K_{57}u_4 & +K_{58}v_4 & +G_{21}p_1 & +G_{52}p_2 & +G_{53}p_3 & = f_5 \\
K_{61}u_1 & +K_{62}v_1 & +K_{63}u_2 & +0 & +K_{65}u_3 & +K_{66}v_3 & +K_{67}u_4 & +K_{68}v_4 & +G_{21}p_1 & +G_{62}p_2 & +G_{63}p_3 & = f_6 \\
K_{71}u_1 & +K_{72}v_1 & +K_{73}u_2 & +0 & +K_{75}u_3 & +K_{76}v_3 & +K_{77}u_4 & +K_{78}v_4 & +G_{21}p_1 & +G_{72}p_2 & +G_{73}p_3 & = f_7 \\
K_{81}u_1 & +K_{82}v_1 & +K_{83}u_2 & +0 & +K_{85}u_3 & +K_{86}v_3 & +K_{87}u_4 & +K_{88}v_4 & +G_{21}p_1 & +G_{82}p_2 & +G_{83}p_3 & = f_8 \\
G_{11}u_1 & +G_{21}v_1 & +G_{31}u_2 & +0 & +G_{51}u_3 & +G_{61}v_3 & +G_{71}u_4 & +G_{81}v_4 & +0 & +0 & +0 & = h_1 \\
G_{12}u_1 & +G_{22}v_1 & +G_{32}u_2 & +0 & +G_{52}u_3 & +G_{62}v_3 & +G_{72}u_4 & +G_{82}v_4 & +0 & +0 & +0 & = h_2 \\
G_{13}u_1 & +G_{23}v_1 & +G_{33}u_2 & +0 & +G_{53}u_3 & +G_{63}v_3 & +G_{73}u_4 & +G_{83}v_4 & +0 & +0 & +0 & = h_3
\end{array}$$

Let us now turn to free slip and start with $v_2 = -u_2 n_x/n_y$. Then we could write:

$$\begin{array}{llllllllllll}
K_{11}u_1 & +K_{12}v_1 & +(K_{13} - \frac{n_x}{n_y}K_{14})u_2 & +0 & +K_{15}u_3 & +K_{16}v_3 & +K_{17}u_4 & +K_{18}v_4 & +G_{11}p_1 & +G_{12}p_2 & +G_{13}p_3 & = f_1 \\
K_{21}u_1 & +K_{22}v_1 & +(K_{23} - \frac{n_x}{n_y}K_{24})u_2 & +0 & +K_{25}u_3 & +K_{26}v_3 & +K_{27}u_4 & +K_{28}v_4 & +G_{21}p_1 & +G_{22}p_2 & +G_{23}p_3 & = f_2 \\
K_{31}u_1 & +K_{32}v_1 & +(K_{33} - \frac{n_x}{n_y}K_{34})u_2 & +0 & +K_{35}u_3 & +K_{36}v_3 & +K_{37}u_4 & +K_{38}v_4 & +G_{31}p_1 & +G_{32}p_2 & +G_{33}p_3 & = f_3 \\
0 & +0 & +K_{44}\frac{n_x}{n_y}u_2 & +K_{44}v_2 & +0 & +0 & +0 & +0 & +0 & +0 & +0 & = 0 \\
K_{51}u_1 & +K_{52}v_1 & +(K_{53} - \frac{n_x}{n_y}K_{54})u_2 & +0 & +K_{55}u_3 & +K_{56}v_3 & +K_{57}u_4 & +K_{58}v_4 & +G_{51}p_1 & +G_{52}p_2 & +G_{53}p_3 & = f_5 \\
K_{61}u_1 & +K_{62}v_1 & +(K_{63} - \frac{n_x}{n_y}K_{64})u_2 & +0 & +K_{65}u_3 & +K_{66}v_3 & +K_{67}u_4 & +K_{68}v_4 & +G_{61}p_1 & +G_{62}p_2 & +G_{63}p_3 & = f_6 \\
K_{71}u_1 & +K_{72}v_1 & +(K_{73} - \frac{n_x}{n_y}K_{74})u_2 & +0 & +K_{75}u_3 & +K_{76}v_3 & +K_{77}u_4 & +K_{78}v_4 & +G_{71}p_1 & +G_{72}p_2 & +G_{73}p_3 & = f_7 \\
K_{81}u_1 & +K_{82}v_1 & +(K_{83} - \frac{n_x}{n_y}K_{84})u_2 & +0 & +K_{85}u_3 & +K_{86}v_3 & +K_{87}u_4 & +K_{88}v_4 & +G_{81}p_1 & +G_{82}p_2 & +G_{83}p_3 & = f_8 \\
G_{11}u_1 & +G_{21}v_1 & +(G_{31} - \frac{n_x}{n_y}G_{41})u_2 & +0 & +G_{51}u_3 & +G_{61}v_3 & +G_{71}u_4 & +G_{81}v_4 & +0 & +0 & +0 & = h_1 \\
G_{12}u_1 & +G_{22}v_1 & +(G_{32} - \frac{n_x}{n_y}G_{42})u_2 & +0 & +G_{52}u_3 & +G_{62}v_3 & +G_{72}u_4 & +G_{82}v_4 & +0 & +0 & +0 & = h_2 \\
G_{13}u_1 & +G_{23}v_1 & +(G_{33} - \frac{n_x}{n_y}G_{43})u_2 & +0 & +G_{53}u_3 & +G_{63}v_3 & +G_{73}u_4 & +G_{83}v_4 & +0 & +0 & +0 & = h_3
\end{array}$$

which is not very useful since the symmetry has been destroyed. Note that when $n_x \to 0$ then we recover the system above.

May be the simplest approach is then to do

$$\begin{array}{llllllllllll}
K_{11}u_1 & +K_{12}v_1 & +K_{13}u_2 & +K_{14}v_2 & +K_{15}u_3 & +K_{16}v_3 & +K_{17}u_4 & +K_{18}v_4 & +G_{11}p_1 & +G_{12}p_2 & +G_{13}p_3 & = f_1 \\
K_{21}u_1 & +K_{22}v_1 & +K_{23}u_2 & +K_{24}v_2 & +K_{25}u_3 & +K_{26}v_3 & +K_{27}u_4 & +K_{28}v_4 & +G_{21}p_1 & +G_{22}p_2 & +G_{23}p_3 & = f_2 \\
K_{31}u_1 & +K_{32}v_1 & +K_{33}u_2 & +K_{34}v_2 & +K_{35}u_3 & +K_{36}v_3 & +K_{37}u_4 & +K_{38}v_4 & +G_{31}p_1 & +G_{32}p_2 & +G_{33}p_3 & = f_3 \\
0 & +0 & +K_{44}\frac{n_x}{n_y}u_2 & +K_{44}v_2 & +0 & +0 & +0 & +0 & +0 & +0 & +0 & = 0 \\
K_{51}u_1 & +K_{52}v_1 & +K_{53}u_2 & +K_{54}v_2 & +K_{55}u_3 & +K_{56}v_3 & +K_{57}u_4 & +K_{58}v_4 & +G_{51}p_1 & +G_{52}p_2 & +G_{53}p_3 & = f_5 \\
K_{61}u_1 & +K_{62}v_1 & +K_{63}u_2 & +K_{64}v_2 & +K_{65}u_3 & +K_{66}v_3 & +K_{67}u_4 & +K_{68}v_4 & +G_{61}p_1 & +G_{62}p_2 & +G_{63}p_3 & = f_6 \\
K_{71}u_1 & +K_{72}v_1 & +K_{73}u_2 & +K_{74}v_2 & +K_{75}u_3 & +K_{76}v_3 & +K_{77}u_4 & +K_{78}v_4 & +G_{71}p_1 & +G_{72}p_2 & +G_{73}p_3 & = f_7 \\
K_{81}u_1 & +K_{82}v_1 & +K_{83}u_2 & +K_{84}v_2 & +K_{85}u_3 & +K_{86}v_3 & +K_{87}u_4 & +K_{88}v_4 & +G_{81}p_1 & +G_{82}p_2 & +G_{83}p_3 & = f_8 \\
G_{11}u_1 & +G_{21}v_1 & +G_{31}u_2 & +G_{41}v_2 & +G_{51}u_3 & +G_{61}v_3 & +G_{71}u_4 & +G_{81}v_4 & +0 & +0 & +0 & = h_1 \\
G_{12}u_1 & +G_{22}v_1 & +G_{32}u_2 & +G_{42}v_2 & +G_{52}u_3 & +G_{62}v_3 & +G_{72}u_4 & +G_{82}v_4 & +0 & +0 & +0 & = h_2 \\
G_{13}u_1 & +G_{23}v_1 & +G_{33}u_2 & +G_{43}v_2 & +G_{53}u_3 & +G_{63}v_3 & +G_{73}u_4 & +G_{83}v_4 & +0 & +0 & +0 & = h_3
\end{array}$$

but it means that after imposing the b.c. I end up with a system like

$$\begin{pmatrix} K_e & G_1 \\ G_1^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{V} \\ \vec{P} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ \vec{h} \end{pmatrix}$$

After writing & sending my doubts to Wolfgang:

```
C: My instinct(memory?) tells me that having the discretised gradient and
divergence operators not being the transpose of one another (after applying
boundary conditions) is wrong. Is my hunch correct? If so, what is the common
procedure?
W: Yes, you need to keep things symmetric.
```

```
If you have constraints that couple multiple DoFs, the procedure is
substantially more complex. I've never really seen a good description of how
this is done (even though it's been done for decades), so I wrote it up a few
years ago. Take a look at section 5 here:
https://www.math.colostate.edu/~bangerth/publications/2006-hp.pdf
```

This publication is: Wolfgang Bangerth and Oliver Kayser-Herold. "Data structures and requirements for hp finite element software". In: *ACM Transactions on Mathematical Software (TOMS)* 36.1 (2009), pp. 1–31. DOI: 10.1145/1486525.1486529.

This therefore means that this approach is not viable and fs_method=2 should not be used.

method 3: constraints via Lagrange multipliers

Once assembled (without the free slip constraints) the Stokes system is

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} \\ \mathbb{G}^T & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{V} \\ \vec{P} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ \vec{h} \end{pmatrix}$$

Since we wish to impose free slip at all nodes on the outer surface[1] of the annulus by means of Lagrange mutlipliers the new system will look like

$$\begin{pmatrix} \mathbb{K} & \mathbb{G} & \mathbb{L} \\ \mathbb{G}^T & 0 & 0 \\ \mathbb{L}^T & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \vec{V} \\ \vec{P} \\ \vec{\Lambda} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ \vec{h} \\ \vec{0} \end{pmatrix}$$

The vector $\vec{\Lambda}$ is N_{lm} long where N_{lm} is the number of nodes on the outer surface. Since the constraints do not involve the pressure dofs there is only one block added to the global matrix, coined $\mathbb{L}$ of size $(ndofV * N_V) \times N_{lm}$.

The total size of the Stokes matrix is then $NfemV + NfemP + N_{lm}$. The non zero entries of $\mathbb{L}$ are obtained by looping over all surface nodes, computing the normal vector (n_x, n_y) there, and writing n_x and n_y in $\mathbb{L}$ at the right line (corresponding to the surface node identity) so that $n_x u + n_y v = 0$:

```
counter=NfemV+NfemP
for i in range(0,nnp):
    if surface[i]:
       A_sparse[counter,2*i  ]=nx[i]
       A_sparse[counter,2*i+1]=ny[i]
       A_sparse[2*i  ,counter]=nx[i]
       A_sparse[2*i+1,counter]=ny[i]
       counter+=1
```

This turns out to be a very simple method to implement. However, it has the disadvantage that the Stokes matrix is now larger and contains yet another zero block on the diagonal.

Q: what is the meaning of the vector Λ that we recover after solving the system?

[1] We restrict ourselves to the outer boundary

Computing the normal vector

In the case of an annulus, the normal to the surface is easily calculated. If a node i has an associated angle θ_i, then its normal vector is $\vec{n}_i = (\cos\theta_i, \sin\theta_i)$. This simply translates as follows in the code:

```
nx1=np.zeros(NV,dtype=np.float64)
ny1=np.zeros(NV,dtype=np.float64)
for i in range(0,NV):
    if rV[i]/R2>1-eps:
        nx1[i]=np.cos(theta[i])
        ny1[i]=np.sin(theta[i])
```

However, as explained in Engelman, Sani, and Gresho [106], the normal vector at a node i can be computed using the basis function derivatives:

$$\begin{aligned} n_{x,i} &= \frac{1}{n_i}\int_\Omega \frac{\partial \mathcal{N}_i}{\partial x} dV \\ n_{y,i} &= \frac{1}{n_i}\int_\Omega \frac{\partial \mathcal{N}_i}{\partial y} dV \end{aligned}$$

with

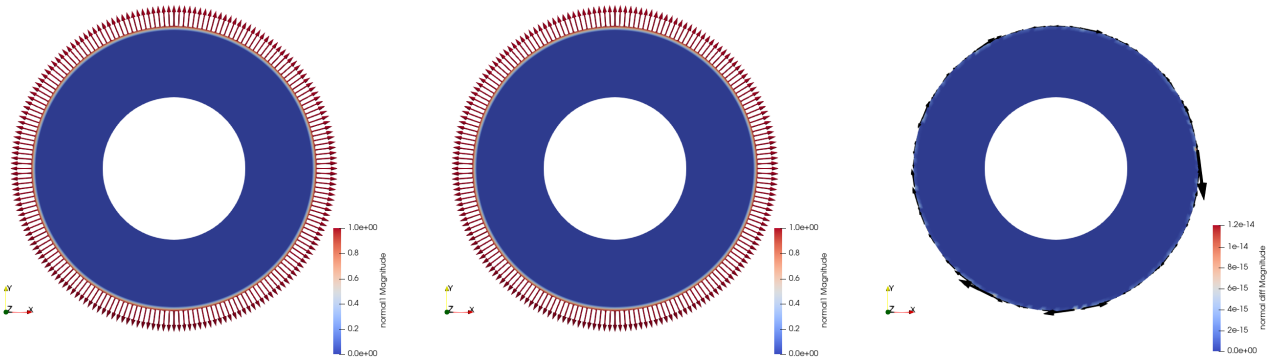
$$n_i = \left[\left(\int_\Omega \frac{\partial \mathcal{N}_i}{\partial x} dV \right)^2 + \left(\int_\Omega \frac{\partial \mathcal{N}_i}{\partial y} dV \right)^2 \right]^{1/2}$$

This second method yields the nodal fields

```
nx2=np.zeros(NV,dtype=np.float64)
ny2=np.zeros(NV,dtype=np.float64)
```

and the normal_type indicates which of the two normal vectors is to be used.

Actually, when computing $\vec{n}_1$ and $\vec{n}_2$ on the outer surface of the annulus we find not so surprisingly that the two vectors are nearly identical (the difference is basically machine precision):



Mesh 8 × 96 elements.

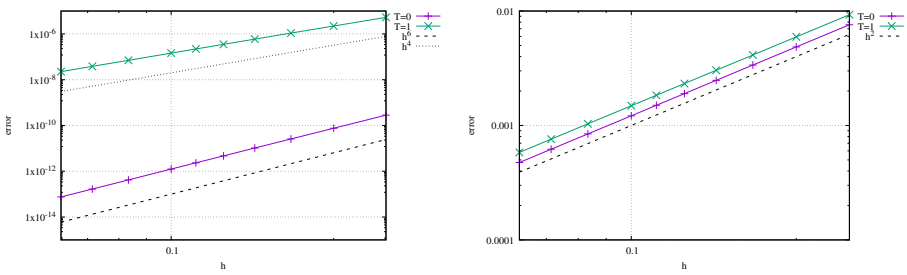
In what follows models are run with $\vec{n}_1$.

Various benchmarks/tests

In what follows we set $R_1 = 1$ and $R_2 = 2$. Unless otherwise specified we also set $\rho = 1$ and $\eta = 1$. The following tests are such that there is no rotational nullspace. Note that the argument n passed as argument to the code is the number of Q_1 elements in the radial direction.

test #1: the aquarium - no slip

Results are obtained by running `script1`. Boundary conditions are no slip on the surface and the cmb. The analytical solution is a zero velocity field and an hydrostatic pressure field. The following figures show the error convergence for both velocity and pressure.

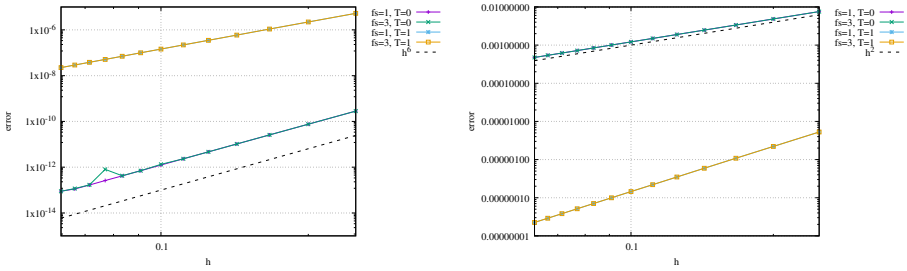


T=1 means trapezes=True.

We obtain a supra-convergence for velocity and quadratic convergence for pressure. This experiment serves as a reference for the next one.

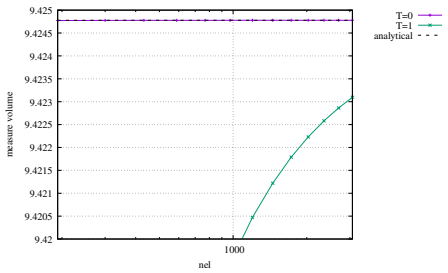
test #2: the aquarium - free slip

Results are obtained by running `script2`. This is the same experiment as test 1, but we now prescribe free slip on the outer surface by means of method 1 or 3. The analytical solution is identical and we recover near identical results as before, for both methods.



T=1 means trapezes=True.

Unsurprisingly we also find that not using trapezoidal elements yields a much higher accuracy for the volume of the domain:



total area of the domain measured by means of Gauss integration. T=1 means trapezes=True.

test #3: shearing - no slip

Results are obtained by running `script3`. Boundary conditions are no slip on the surface and a prescribed constant velocity $\mathrm{v}_{bc} = 1$ tangent to the inner boundary. The analytical solution is hydrostatic pressure and a rotational velocity going from 1 at the cmb to 0 on the outer surface

TODO

test #4: shearing - free slip

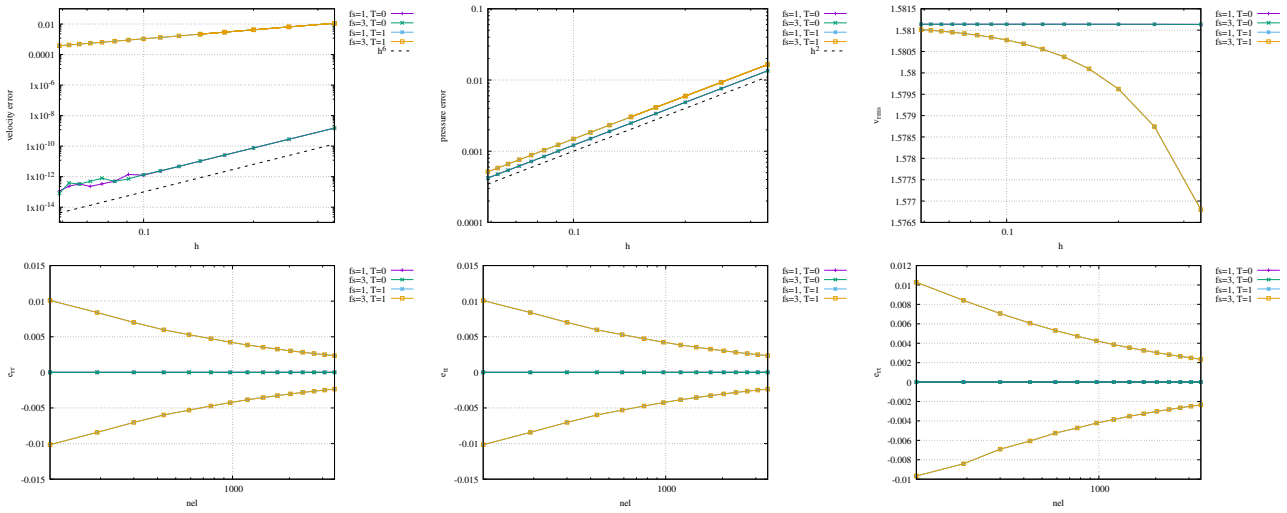
Results are obtained by running `script4`. Boundary conditions are free slip on the surface and a prescribed constant velocity $\mathrm{v}_{bc} = 1$ tangent to the inner boundary.

The analytical solution is described in Section 18.22. It is a hydrostatic pressure and a rotational velocity given by $\vec{\mathrm{v}} = \mathrm{v}_r \vec{e}_r + \mathrm{v}_\theta \vec{e}_\theta = r\ \vec{e}_\theta$. Turning to Section 4.1.2:

$$\dot{\varepsilon}_{rr} = \frac{\partial \mathrm{v}_r}{\partial r} = 0 \tag{151.1}$$

$$\dot{\varepsilon}_{\theta\theta} = \frac{\mathrm{v}_r}{r} + \frac{1}{r}\frac{\partial \mathrm{v}_\theta}{\partial \theta} = 0 \tag{151.2}$$

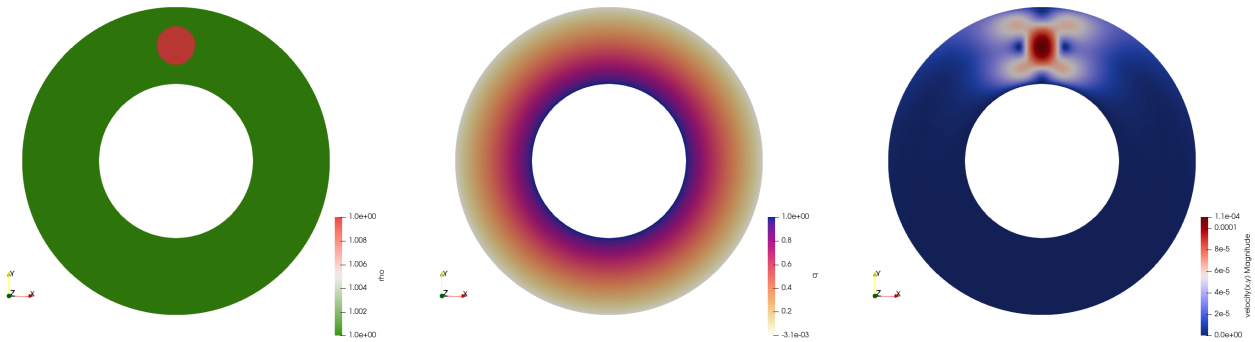
$$\dot{\varepsilon}_{\theta r} = \dot{\varepsilon}_{r\theta} = \frac{1}{2}\left(\frac{\partial \mathrm{v}_\theta}{\partial r} - \frac{\mathrm{v}_\theta}{r} + \frac{1}{r}\frac{\partial \mathrm{v}_r}{\partial \theta}\right) = \frac{1}{2}(1-1) = 0 \tag{151.3}$$

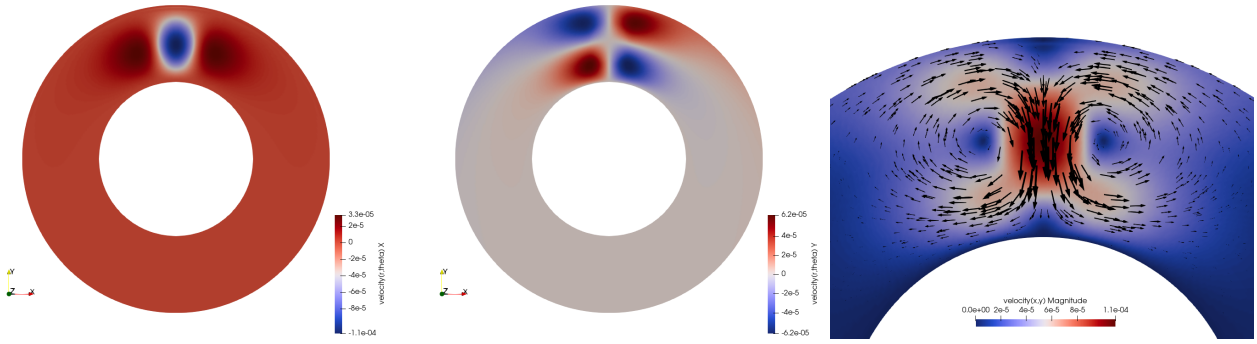


T stands for trapezes.

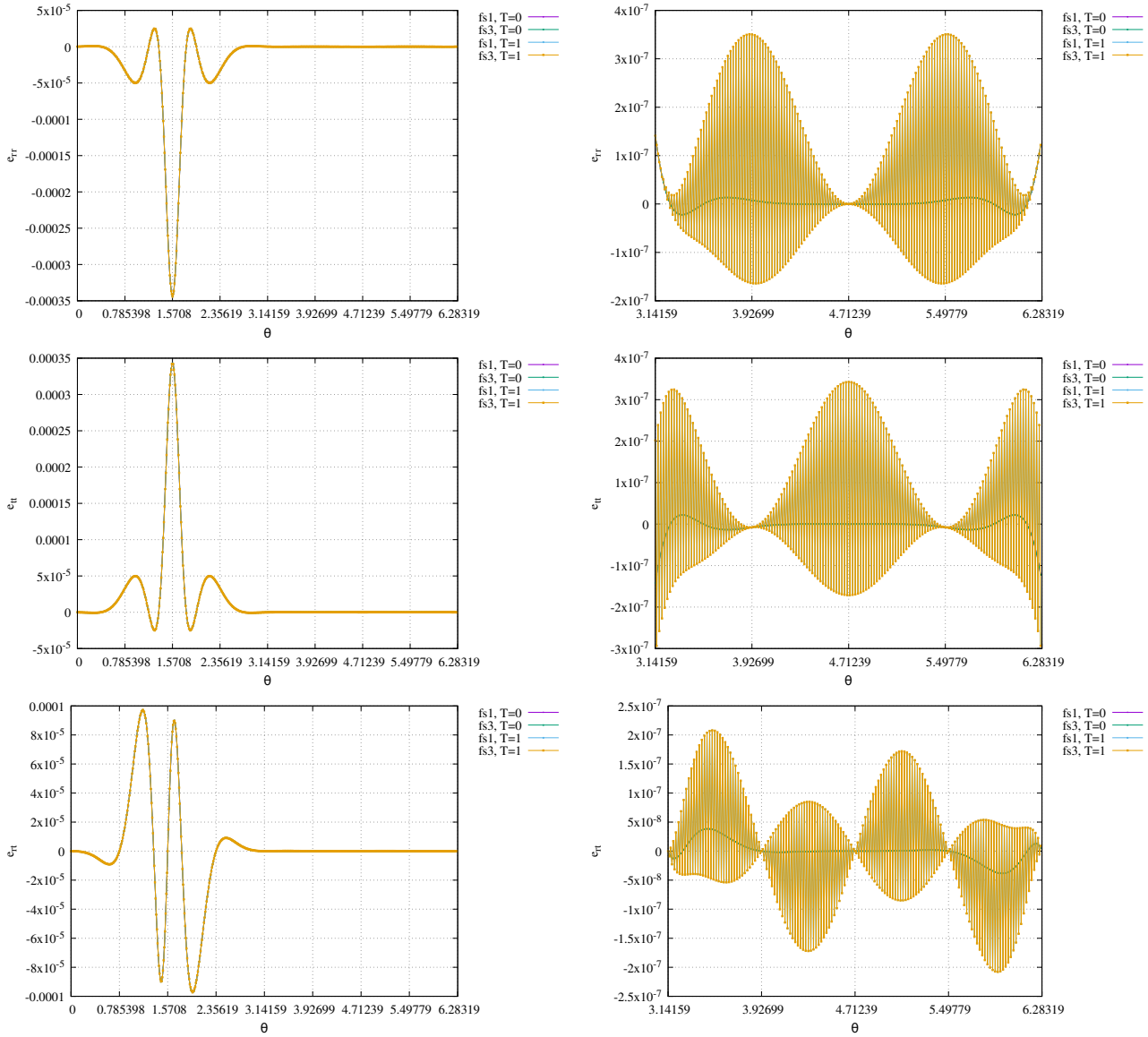
test #4: sinking blob - free slip

There is a disc of slightly higher density ($\delta\rho/\rho_0 = 0.01$) centered at $(0, 1.5)$ with a radius 0.25.

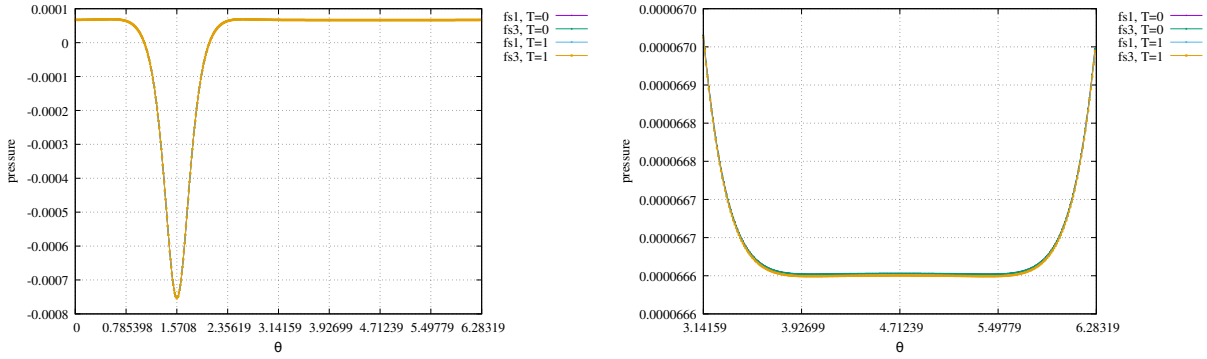




resolution 32x384.



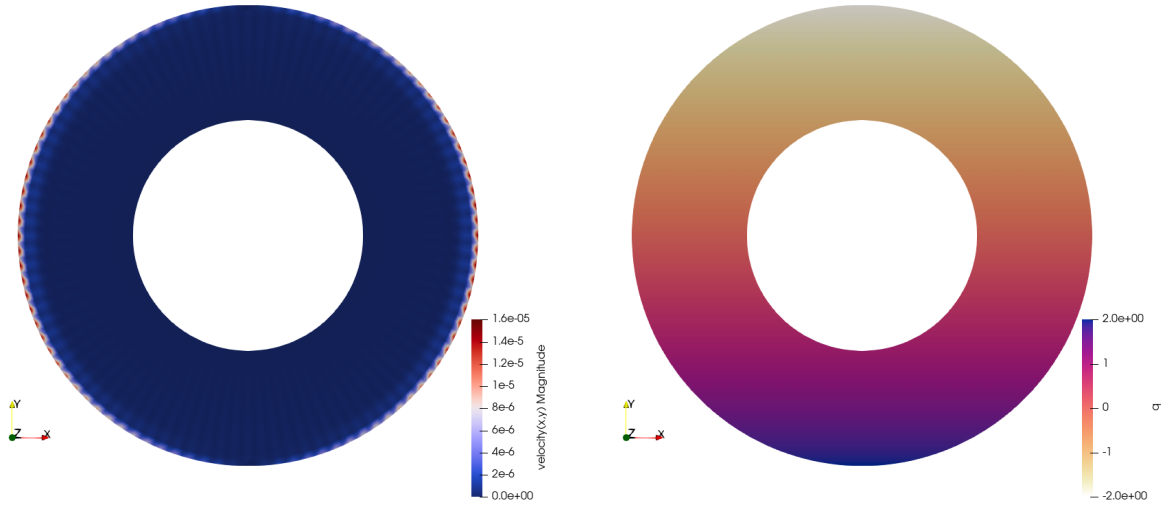
Strain rate components at the surface, resolution 24x288. The first column of plots is the strain rate components all around the annulus at the surface, i.e. $\theta \in [0, 2\pi]$. The second column is the same components but plotted for $\theta \in [\pi, 2\pi]$. Zoom in to better see the results. 'T' stands for the trapezes variable.



Pressure at the surface, resolution 24x288.

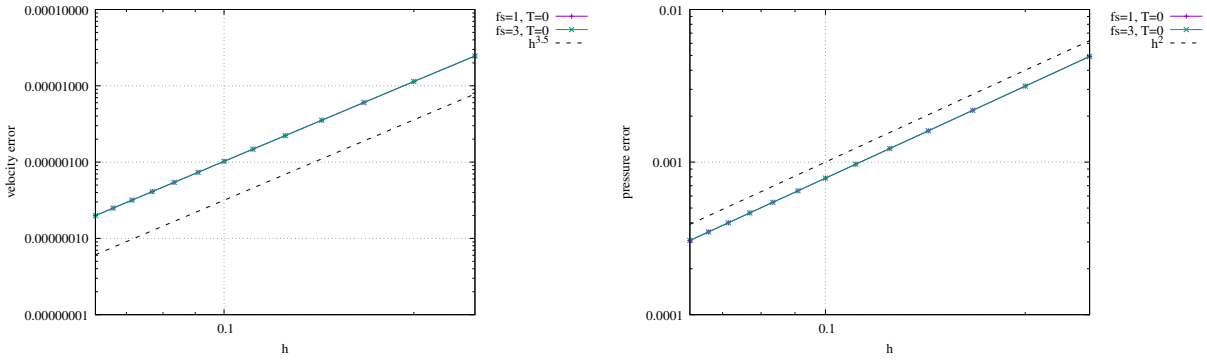
test #6: aquarium - free slip

This test is inspired by Behr [15]. Results obtained with `script6`. Density is constant, viscosity is constant, but now gravity is given by $\vec{g} = (0, -g_0)$. Free slip boundary conditions are used. We expect zero flow and yet ...



Velocity and pressure fields, resolution 8×96.

The analytical solution is $\vec{\mathbf{v}} = (0, 0)$ and $p(x, y) = -R_2 + i\rho_0 g_0 (R_2 - y)$ so we can compute the velocity and pressure errors as a function of the mesh size h:



Conclusions

The conclusion is unescapable: either way of prescribing free slip is fine, they yield virtuallly identical results. The trapeze option should not be used.

To Do, open questions, future work?

- analytical solution test 3 ?
- design more appropriate flow solution and test 3 methods to compute strain rate
- implement axisymmetric calculations
- check M.S. Engelman, R.L. Sani, and P.M. Gresho. "The implementation of normal and/or tangential boundary conditions in finite element codes for incompressible fluid flow". In: *Int. J. Num. Meth. Fluids* 2 (1982), pp. 225–238. DOI: 10.1002/fld.1650020302
- make sure that conversion between cartesian strain rate and cylindrical strain rate is correct
- from my experience with stone 152, it is not clear how the additional terms in the matrix wrt Lagrange multipliers should be scaled
- check Nathan Sime and Cian R Wilson. "Automatic weak imposition of free slip boundary conditions via Nitsche's method: application to nonlinear problems in geodynamics". In: *arXiv preprint arXiv:2001.10639* (2020)
- check J.M. Urquiza, A. Garon, and M.-I. Farinas. "Weak imposition of the slip boundary condition on curved boundaries for Stokes flow". In: *J. Comp. Phys.* 256 (2014), pp. 748–767. DOI: 10.1016/j.jcp.2013.08.045
- check I. Dione, C. Tibirna, and J. Urquiza. "Stokes equations with penalised slip boundary conditions". In: *International Journal of Computational Fluid Dynamics* 27.6-7 (2013), pp. 283–296. DOI: 10.1080/10618562.2013.821114
- check Ibrahima Dione and José M Urquiza. "Penalty: finite element approximation of Stokes equations with slip boundary conditions". In: *Numerische Mathematik* 129.3 (2015), pp. 587–610. DOI: 10.1007/s00211-014-0646-9
- check W. Layton. "Weak imposition of 'no-slip' conditions in finite element methods". In: *Computers and Mathematics with Applications* 38 (1999), pp. 129–142. DOI: 10.1016/S0898-1221(99)00220-5

Following Dione and Urquiza [91] we add the following term to the weak form of the momentum equation:

$$\int_\Gamma \vec{u}\cdot\vec{n}\;\vec{v}\cdot\vec{n}d\Gamma \qquad \forall\vec{v}$$

where $\epsilon > 0$ has a small value destined to converge to 0 and Γ is the portion of the boundary where slip boundary conditions are to be implemented. We have

$$\begin{aligned}
\vec{u}\cdot\vec{n} \quad \vec{v}\cdot\vec{n} &= (u_x n_x + u_y n_y)(v_x n_x + v_y n_y) && (151.4)\\
&= u_x v_x n_x^2 + u_y v_y n_y^2 + (u_x v_y + u_y v_x) n_x n_y && (151.5)\\
&= v_x(n_x^2 u_x + n_x n_y u_y) + v_y(n_x n_y u_x + n_y^2 u_y) && (151.6)\\
&= (v_x\ v_y)\cdot\begin{pmatrix} n_x^2 u_x + n_x n_y u_y \\ n_x n_y u_x + n_y^2 u_y \end{pmatrix} && (151.7)\\
&= (v_x\ v_y)\cdot\begin{pmatrix} n_x^2 & n_x n_y \\ n_x n_y & n_y^2 \end{pmatrix}\cdot\begin{pmatrix} u_x \\ u_y \end{pmatrix} && (151.8)\\
&= (v_x\ v_y)\cdot\begin{pmatrix} n_x^2 & n_x n_y \\ n_x n_y & n_y^2 \end{pmatrix}\cdot\begin{pmatrix} \sum_i \mathcal{N}_i u_{x,i} \\ \sum_i \mathcal{N}_i u_{y,i} \end{pmatrix} && (151.9)\\
& && (151.10)
\end{aligned}$$

Stone 152

Temperature in mantle wedge (van Keken et al 2008)



python_codes/fieldstone_152/text.tex

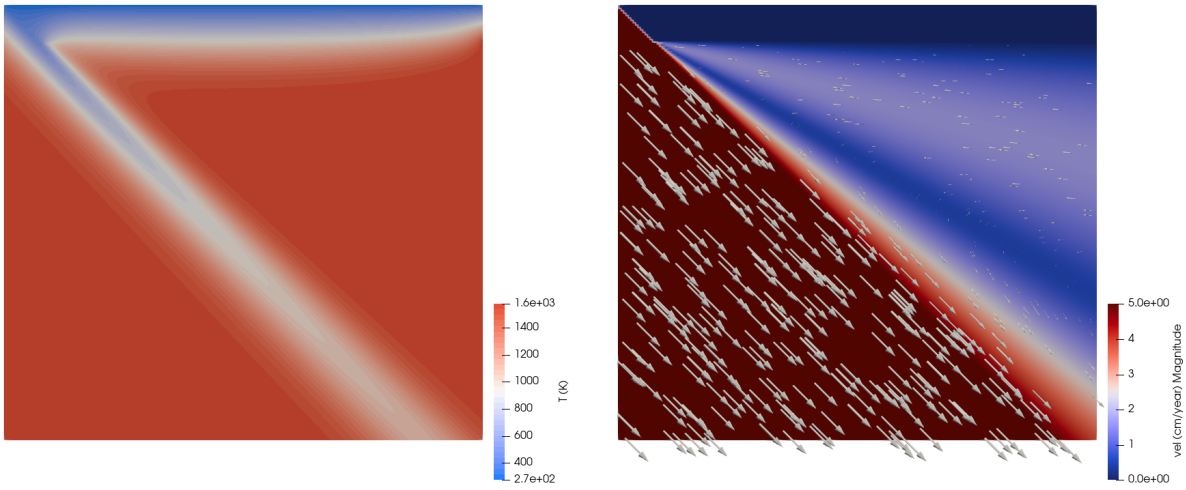
P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_152`

This stone was developed in collaboration with C. Leblanc and F. Gueydan.

The setup is based on van Keken et al. [339] (2008). This benchmark is carried out in STONE 68 and 149. The code was developed during the MSc thesis of C. Leblanc.

DESCRIBE

this is case 1a.



Stone 153

Vorticity-Streamfunction - isoviscous 2D case with free slip



python_codes/fieldstone_153/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_153

The theory and derivations of the vorticity-stream function approach is presented in detail in Chapter 20.15.

We start from the Poisson equation for the vorticity ω

$$\vec{\nabla}^2\omega = -\frac{1}{\eta_0}\left(-\frac{\partial(\rho g_x)}{\partial y}+\frac{\partial(\rho g_y)}{\partial x}\right) \tag{153.1}$$

In what follows we assume that the domain is a Cartesian box aligned with the x,y axis. The gravity is constant and vertical so that $g_x=0$ and then

$$\vec{\nabla}^2\omega = -\frac{g_y}{\eta_0}\frac{\partial\rho}{\partial x} \tag{153.2}$$

As shown in Section 6.8, the Laplacian of ω can be discretised with a second order central difference stencil:

$$\vec{\nabla}^2\omega|_{i,j} = \frac{\omega_{i+1,j}-2\omega_{i,j}+2\omega_{i-1,j}}{h_x^2}+\frac{\omega_{i,j+1}-2\omega_{i,j}+2\omega_{i,j-1}}{h_y^2}+\mathcal{O}(h_x^2,h_y^2)$$

while the density gradient is approximated by a second order central difference:

$$\frac{\partial\rho}{\partial x}|_{i,j} = \frac{\rho_{i+1,j}-\rho_{i-1,j}}{2h_x}+\mathcal{O}(h_x^2)$$

In the end,

$$\frac{\omega_{i+1,j}-2\omega_{i,j}+2\omega_{i-1,j}}{h_x^2}+\frac{\omega_{i,j+1}-2\omega_{i,j}+2\omega_{i,j-1}}{h_y^2} = -\frac{g_y}{\eta_0}\frac{\rho_{i+1,j}-\rho_{i-1,j}}{2h_x}$$

We then define the global indices

$$\begin{aligned} k &= j\cdot nnx+i \\ k_E &= j\cdot nnx+i+1 \\ k_W &= j\cdot nnx+i-1 \\ k_N &= (j+1)\cdot nnx+i \\ k_S &= (j-1)\cdot nnx+i \end{aligned}$$

so that

$$\frac{\omega_{k_E} - 2\omega_k + 2\omega_{k_W}}{h_x^2} + \frac{\omega_{k_N} - 2\omega_k + 2\omega_{k_S}}{h_y^2} = -\frac{g_y}{\eta_0}\frac{\rho_{k_E} - \rho_{k_W}}{2h_x}$$

or,

$$\frac{\omega_{k_E}}{h_x^2} + \frac{\omega_{k_W}}{h_x^2} + \frac{\omega_{k_N}}{h_y^2} + \frac{\omega_{k_S}}{h_y^2} - \left(\frac{2}{h_x^2} + \frac{2}{h_y^2}\right)\omega_k = -\frac{g_y}{\eta_0}\frac{\rho_{k_E} - \rho_{k_W}}{2h_x}$$

This stencil is used for all interior nodes. All nodes on the boundaries have $\omega = 0$.

Having obtained ω we can solve for Ψ by solving $\vec{\nabla}^2\Psi = -\omega$. Finally we can recover the velocity field with $u = \partial_y\Psi$ and $v = -\partial_x\Psi$. Once again using a second order centered approach for the internal nodes:

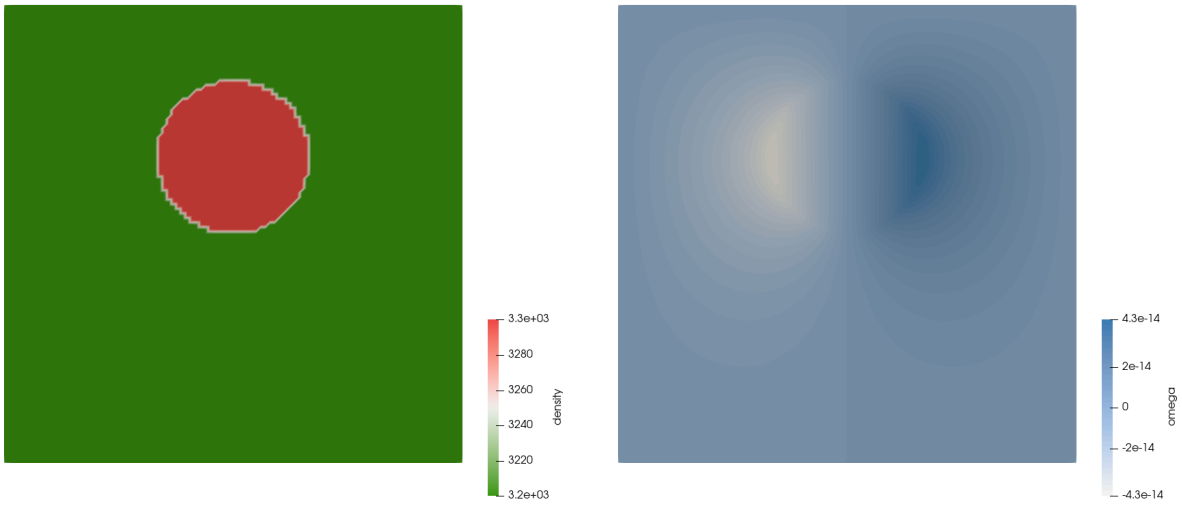
$$u_{i,j} = \frac{\Psi_{i,j+i} - \Psi_{i,j-i}}{2h_y}$$

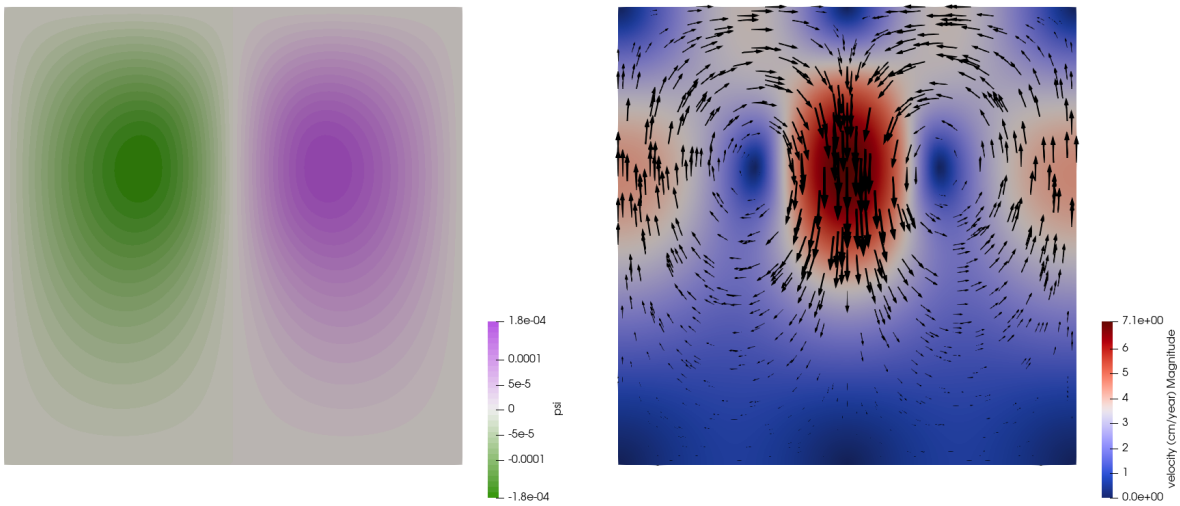
$$v_{i,j} = -\frac{\Psi_{i+1,j} - \Psi_{i-1,j}}{2h_x}$$

For the nodes on the boundary we see that these formulations are not suited: indeed, for example, if $j = 0$ (bottom boundary), then u is not zero (free slip b.c.) and should be computed, but in the equation above we find the term $\Psi_{i,j-1}$ which is undefined. In the end, for each node on the boundary we use a first order stencil that uses only internal nodes.

Experiment 1

The domain is a square of size 600 km. It contains two fluids, the 'mantle' and the 'slab'. The mantle has a density of 3200 $\mathrm{kg\,m^{-3}}$ and fills the domain except for a circle (the slab) of radius 100 km centered at $x = L_x/2$ and $y = \frac{2}{3}L_y$ of density 3300 $\mathrm{kg\,m^{-3}}$. $g_0 = 10\ \mathrm{m\,s^{-2}}$ and $\eta_0 = 10^{21}\ \mathrm{Pa\,s}$.





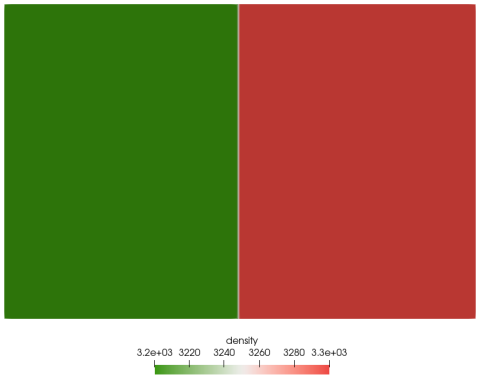
Obtained on 101x101 mesh.

Experiment 2

This is inspired by Exercise 5.2 of Gerya [129] which reads as follows:

Solve the momentum and continuity equations with finite differences using the stream function-vorticity formulation on a regular grid of 51×41 points.

Program a numerical model for buoyancy-driven flow in a vertical gravity field ($g_x = 0$, $g_y = 10\ \mathrm{m\,s^{-2}}$) for a density structure with two vertical layers ($3200\ \mathrm{kg\,m^{-3}}$ and $3300\ \mathrm{kg\,m^{-3}}$ for the left and right layers, respectively). The model size is 1000 km $\times$ 1500 km. Use a constant viscosity $\eta = 10^{21}$ Pa s for the entire model.



Compose a matrix of coefficients $\mathbb{K}$ and a right hand side vector $\vec{b}$ and obtain the solution vector $\vec{X}$ for the two Poisson equations (first for ω, then for Ψ).

Use $\omega = 0$ and $\Psi = 0$ as boundary conditions for all external nodes of the grid. Note that velocity components should only be computed for the internal nodes of the grid (i.e. for 49×39 points).

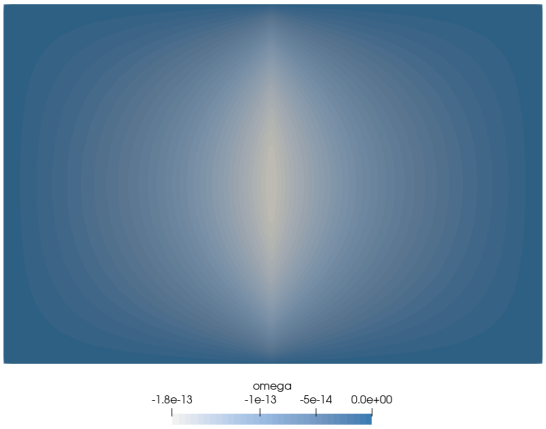
The MATLAB code of the book is reproduced hereafter:

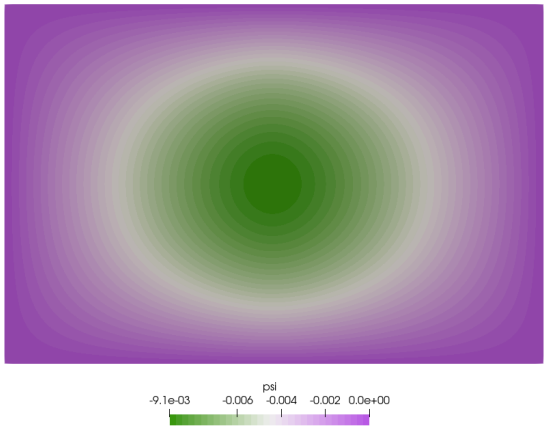
```
xsize=1000000; % Horizontal
ysize=1500000; % Vertical
xnum=41; % Horizontal
ynum=31; % Vertical
xstp=xsize/(xnum-1); % Horizontal
ystp=ysize/(ynum-1); % Vertical
eta=1e+21;
gy=10; % m/s^2

% Making vectors for nodal points positions
```

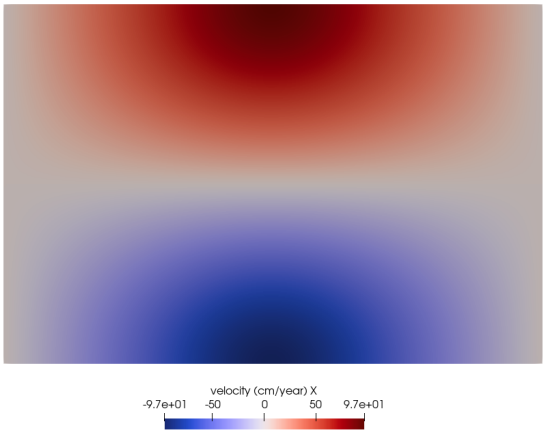
```
x=0:xstp:xsize; % Horizontal
y=0:ystp:ysize; % Vertical

% Creating array for density structure (two vertical layers)
rho=zeros(ynum,xnum);
for i=1:1:ynum
  for j=1:1:xnum
    % Horizontal position of the nodal point
    if(x(j)<xsize/2)
        rho(i,j)=3200;  % left layer
    else
        rho(i,j)=3300;  % right layer
    end
  end
end

% Matrix of coefficients initialization
L=sparse(xnum*ynum,xnum*ynum);
R=zeros(xnum*ynum,1);

% Solving Poisson equation for worticity
% d2OMEGA/dx2+d2OMEGA/dy2=gy*dRHO/dx
% Composing matrix of coefficients L()
% and vector (column) of right parts R()
% Boundary conditions: OMEGA=0
% Process all Grid points
for i=1:1:ynum
  for j=1:1:xnum
    % Global index for current node
    k=(j-1)*ynum+i;
    %Boundary nodes OMEGA=0
    if(i==1 || i==ynum || j==1 || j==xnum)
        L(k,k)=1;
        R(k,1)=0;
    %Internal nodes
    else
        %Left part: d2OMEGA/dx2+d2OMEGA/dy2
        L(k,k-ynum)=1/xstp^2;
        L(k,k-1)=1/ystp^2;
        L(k,k)=-2/xstp^2-2/ystp^2;
        L(k,k+1)=1/ystp^2;
        L(k,k+ynum)=1/xstp^2;
        % Right part: gy*dRHO/dx
        R(k,1)=gy/eta*(rho(i,j+1)-rho(i,j-1))/2/xstp;
    end
  end
end

%Obtaining vector of solutions S()
S=L\R;

% Reload solutions S() to 2D vorticity array OMEGA()
OMEGA=zeros(ynum,xnum);
% Process all Grid points
for i=1:1:ynum
  for j=1:1:xnum
    % Global index for current node
    k=(j-1)*ynum+i;
    OMEGA(i,j)=S(k);
  end
end

% Solving Poisson equation for stream function
% d2PSI/dx2+d2PSI/dy2=OMEGA
% Simplified procedure as below is only possible
% when boundary conditions for OMEGA and PSI are the same
% othervise i,j cycle is needed as in the previous case
%L=L; % Left parts remain the same
R=S; % Creates right parts from previous solution

%Obtaining vector of solutions S()
S=L\R;

% Reload solutions S() to 2D stream function array PSI()
PSI=zeros(ynum,xnum);
% Process all Grid points
```

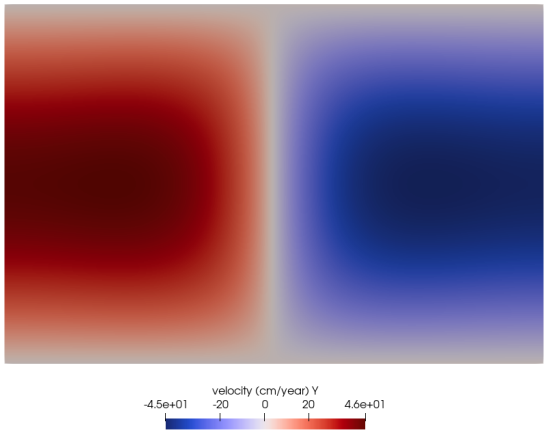
```
for i=1:1:ynum
  for j=1:1:xnum
    % Global index for current node
    k=(j-1)*ynum+i;
    PSI(i,j)=S(k);
  end
end

% Compute vx,vy for internal nodes
vx=zeros(ynum,xnum);
vy=zeros(ynum,xnum);
% Process internal Grid points
for i=2:1:ynum-1
  for j=2:1:xnum-1
    % vx=dPSI/dy
    vx(i,j)=(PSI(i+1,j)-PSI(i-1,j))/2/xstp;
     % vy=-dPSI/dx
    vy(i,j)=-(PSI(i,j+1)-PSI(i,j-1))/2/xstp;
  end
end
```

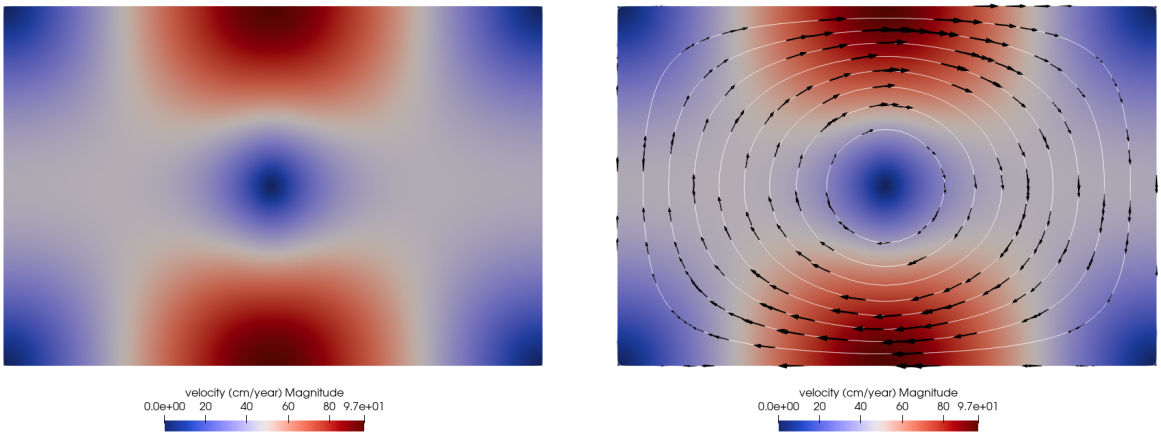
Our code is much more compact but very similar in structure. Results for ω and Ψ are shown here:











Top row: ω and Ψ fields. Middle row: u and v fields. Bottom row: velocity magnitude. Velocity glyphs on Ψ isocontours.

To Do, open questions, future work?

- compute pressure via PPE

Stone 154

Vorticity-Streamfunction - isoviscous 2D case with no slip



python_codes/fieldstone_154/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_154

Purpose: implement and document bla bla

Poiseuille flow benchmark

In what follows we consider a 2D domain of dimensions $L_x \times L_y$ in which a fluid flows in the absence of gravitational forces (the flow is entirely driven by the boundary conditions impose on the left - the 'inlet'). The domain is discretised by means of a regular grid of $nnx \times nny$ points.

The default parameters are $L_x = 5$, $L_y = 1$, $\delta t = 0.002$, $t_{\text{final}} = 5$, $u(y) = y(1-y)$, mesh is 101×21 nodes,$\mathsf{Re} = 400$.

Boundary conditions

In the context of this experiment, we have four boundaries (left, right, bottom and top).

- The left one (the inlet) is characterised by $\vec{\mathbf{v}} = (u_0, 0)$ for $y \in [0, L_y]$, so that

$$\omega(0, y) = \partial_x v - \partial_y u = 0 \tag{154.1}$$

  Likewise $u = \frac{\partial \Psi}{\partial y} = u_0$ so Ψ is a linear function of y on this boundary and since $v = -\frac{\partial \Psi}{\partial x} = 0$ then Ψ does not depend on x.

- The right one is such that a laminar flow exits there. In this case $\vec{\mathbf{v}} = (u(y), 0)$ so that $\omega_{right} = \partial_x v - \partial_y u = -\partial_y u$. This is not particularly helpful since we do not know the exact form of $u(y)$, but we also have

$$\frac{\partial \omega}{\partial x}(L_x, y) = \frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 u}{\partial x \partial y} = -\frac{\partial^2 u}{\partial x \partial y} = 0 \tag{154.2}$$

  since u is only a function of y.

- At the bottom, we have $u = v = 0$ (no-slip boundary conditions) for $x > 0$ (the lower left corner 'belongs' to the left boundary). As such it is not straightforward to establish the value of the vorticity there. However, let us look at a Taylor expansion of $\Psi(x, y)$ at a point with coordinates (x, h) just above the bottom boundary:

$$\Psi(x, h) = \Psi(x, 0) + h\frac{\partial \Psi}{\partial y}(x, 0) + \frac{h^2}{2}\frac{\partial^2 \Psi}{\partial y^2}(x, 0) + ...$$

Since by definition $u = \frac{\partial \Psi}{\partial y}$ and $u = 0$ on this boundary we are left with:

$$\Psi(x, h) = \Psi(x, 0) + \frac{h^2}{2}\frac{\partial^2 \Psi}{\partial y^2}(x, 0) + ...$$

Also, we have the following relationship: $\vec{\nabla}^2\Psi = \frac{\partial^2 \Psi}{\partial x^2} + \frac{\partial^2 \Psi}{\partial y^2} = -\omega$ and since at the wall Ψ is independent of x ($0 = v = -\partial_x \Psi$) the second-order x-derivative is zero, so that $\frac{\partial^2 \Psi}{\partial y^2} = -\omega$ and then

$$\Psi(x, h) = \Psi(x, 0) - \frac{h^2}{2}\omega(x, 0) + ...$$

In discrete form[1]:

$$\Psi_{i,j+1} = \Psi_{i,j} - \frac{h^2}{2}\omega_{i,j} \qquad \text{for} \quad j = 0$$

and then

$$\omega_{i,0} = \frac{2}{h^2}(\Psi_{i,0} - \Psi_{i,1}) \tag{154.3}$$

Also $u = \frac{\partial \Psi}{\partial y} = 0$ and $v = -\frac{\partial \Psi}{\partial x} = 0$ on this boundary so Ψ is constant (but not necessarily zero).

- We can use the same approach at the top boundary (also no-slip boundary conditions, with the upper left corner belonging to the left boundary) by taking a Taylor expansion at a point with coordinates $(x, L_y - h)$:

$$\Psi(x, L_y - h) = \Psi(x, L_y) - h\frac{\partial \Psi}{\partial y}(x, L_y) + \frac{h^2}{2}\frac{\partial^2 \Psi}{\partial y^2}(x, L_y) + ...$$

which becomes (since $u = \partial_y \Psi = 0$)

$$\Psi(x, L_y - h) = \Psi(x, L_y) - \omega(x, L_y) + ...$$

and then

$$\omega_{i,nny-1} = \frac{2}{h^2}(\Psi_{i,nny-1} - \Psi_{i,nny-2}) \tag{154.4}$$

Also $u = \frac{\partial \Psi}{\partial y} = 0$ and $v = -\frac{\partial \Psi}{\partial x} = 0$ on this boundary so Ψ is constant (but not necessarily zero).

The algorithm

Because of the presence of no-slip boundary conditions, the algorithm goes as follows. At the beginning zero all fields, i.e. $u = v = \omega = \Psi = 0$.

```
LTTTTTTTTTTTTTTTTTTTTTTTR
L                        R
L                        R
L                        R
LBBBBBBBBBBBBBBBBBBBBBBBR
```

[1] I here adopt python conventions, $j = 0$ is the row of nodes at the bottom of the domain, while $nny - 1$ is the row of nodes at the top.

1. solve $\Delta\Psi = -\omega$

$$\frac{\Psi_{i-1,j} - 2\Psi_{i,j} + \Psi_{i+1,j}}{h_x^2} + \frac{\Psi_{i,j-1} - 2\Psi_{i,j} + \Psi_{i,j+1}}{h_y^2} = -\omega_{i,j}$$

or,

$$\frac{1}{h_x^2}\Psi_{i-1,j} + \frac{1}{h_y^2}\Psi_{i,j-1} - \left(\frac{2}{h_x^2} + \frac{2}{h_y^2}\right)\Psi_{i,j} + \frac{1}{h_x^2}\Psi_{i+1,j} + \frac{1}{h_y^2}\Psi_{i,j+1} = -\omega_{i,j}$$

so that we obtain a pentadiagonal matrix.

About the boundary conditions: Since Ψ is known up to a constant (and is constant) at the bottom, we arbitrarily fix $\Psi = 0$. Since Ψ is linear and only depends on y on the inlet (left boundary) then $\Psi(y) = u_0 y$ there. This means that at the top Ψ is constant and equal to $u_0 L_y$. At the outlet we have $\partial\Psi/\partial x = 0$. This last condition is compatible with the values on the top and bottom so no need to think about the corners.

2. compute velocity field at interior nodes with centered stencils

$$u_{i,j}^n = \frac{\partial\Psi_{i,j}^n}{\partial y} = \frac{\Psi_{i,j+1}^n - \Psi_{i,j-1}^n}{2h_y} \qquad v_{i,j}^n = -\frac{\partial\Psi_{i,j}^n}{\partial x} = -\frac{\Psi_{i+1,j}^n - \Psi_{i-1,j}^n}{2h_x}$$

Velocity is analytically known on the bottom, left and top boundaries. On the right boundary we have $v = 0$ (that stems from us imposing $\partial\Psi/\partial x = 0$ there) and we can compute $u_{i,j}$ there using the formula above for $j \in [1, nny - 2]$.

3. Having obtained $\vec{\mathbf{v}}^n$ we can solve the vorticity equation stemming for the time-dependent Navier-Stokes equation (see Section 20.1):

$$\frac{\partial\omega}{\partial t} + \vec{\mathbf{v}}\cdot\vec{\nabla}\omega = \frac{1}{\mathsf{Re}}\left(\frac{\partial^2\omega}{\partial x^2} + \frac{\partial^2\omega}{\partial y^2}\right)$$

We discretise the time derivative as follows:

$$\frac{\partial\omega}{\partial t} = \frac{\omega^n - \omega^{n-1}}{\delta t}$$

and then write

$$\frac{\omega_{i,j}^n - \omega_{i,j}^{n-1}}{\delta t} + u_{i,j}\frac{\omega_{i+1,j}^n - \omega_{i-1,j}^n}{2h_x} + v_{i,j}\frac{\omega_{i,j+1}^n - \omega_{i,j-1}^n}{2h_y} - \frac{1}{\mathsf{Re}}\left(\frac{\omega_{i-1,j}^n - 2\omega_{i,j}^n + \omega_{i+1,j}^n}{h_x^2} + \frac{\omega_{i,j-1}^n - 2\omega_{i,j}^n + \omega_{i,j+1}^n}{h_y^2}\right) = 0$$

or,

$$\omega_{i,j}^n + \frac{u_{i,j}\delta t}{2h_x}(\omega_{i+1,j}^n - \omega_{i-1,j}^n) + \frac{v_{i,j}\delta t}{2h_y}(\omega_{i,j+1}^n - \omega_{i,j-1}^n) - \frac{\delta t}{\mathsf{Re}}\left(\frac{\omega_{i-1,j}^n - 2\omega_{i,j}^n + \omega_{i+1,j}^n}{h_x^2} + \frac{\omega_{i,j-1}^n - 2\omega_{i,j}^n + \omega_{i,j+1}^n}{h_y^2}\right) = \omega_{i,j}^{n-1}$$

that is,

$$\begin{aligned}\left(-\frac{u_{i,j}\delta t}{2h_x} - \frac{\delta t}{h_x^2\mathsf{Re}}\right)\omega_{i-1,j}^n + \left(-\frac{v_{i,j}\delta t}{2h_y} - \frac{\delta t}{h_y^2\mathsf{Re}}\right)\omega_{i,j-1}^n \\ + \left(\frac{u_{i,j}\delta t}{2h_x} - \frac{\delta t}{h_x^2\mathsf{Re}}\right)\omega_{i+1,j}^n + \left(\frac{v_{i,j}\delta t}{2h_y} - \frac{\delta t}{h_y^2\mathsf{Re}}\right)\omega_{i,j+1}^n \\ + \left(1 + \frac{2\delta t}{h_x^2\mathsf{Re}} + \frac{2\delta t}{h_y^2\mathsf{Re}}\right)\omega_{i,j}^n &= \omega_{i,j}^{n-1}\end{aligned} \tag{154.5}$$

Again, a pentadiagonal matrix.

Note that if we only consider the steady-state equations, then $\partial\omega/\partial t \to 0$ so that we obtain:

$$\begin{aligned}\left(-\frac{u_{i,j}}{2h_x} - \frac{1}{h_x^2\mathsf{Re}}\right)\omega_{i-1,j}^n + \left(-\frac{v_{i,j}}{2h_y} - \frac{1}{h_y^2\mathsf{Re}}\right)\omega_{i,j-1}^n \\ + \left(\frac{u_{i,j}}{2h_x} - \frac{1}{h_x^2\mathsf{Re}}\right)\omega_{i+1,j}^n + \left(\frac{v_{i,j}}{2h_y} - \frac{1}{h_y^2\mathsf{Re}}\right)\omega_{i,j+1}^n \\ + \left(\frac{2}{h_x^2\mathsf{Re}} + \frac{2}{h_y^2\mathsf{Re}}\right)\omega_{i,j}^n &= 0\end{aligned} \tag{154.6}$$

Finally, if we only wish to solve the Stokes equations, then the advection (inertia) term also disappears and we are left with

$$\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} = 0$$

and we see that the Reynolds number Re is no more a parameter of the PDE. It translates into

$$\left(-\frac{1}{h_x^2}\right)\omega^n_{i-1,j} + \left(-\frac{1}{h_y^2}\right)\omega^n_{i,j-1} + \left(-\frac{1}{h_x^2}\right)\omega^n_{i+1,j} + \left(-\frac{1}{h_y^2}\right)\omega^n_{i,j+1} + \left(\frac{2}{h_x^2} + \frac{2}{h_y^2}\right)\omega^n_{i,j} = 0 \quad (154.7)$$

Concerning the boundary conditions we use Eqs. (154.1),(154.2),(154.3),(154.4) above, i.e.

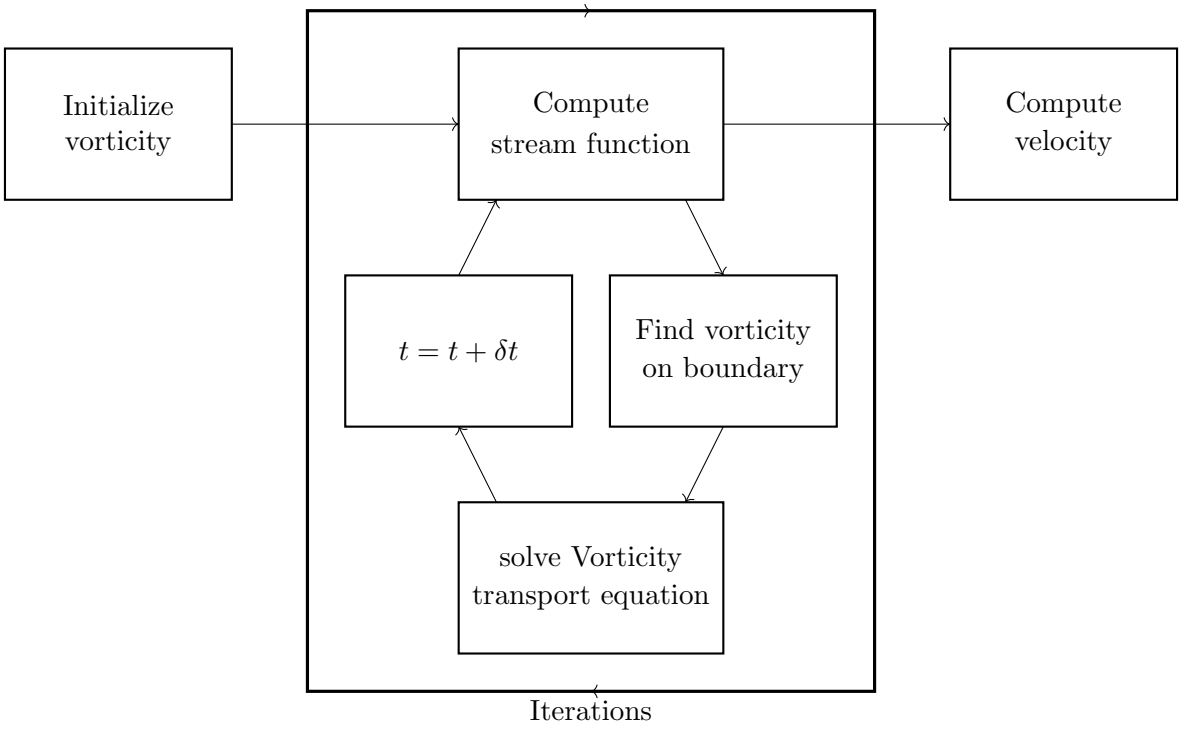
$$\omega_{0,j} = 0 \quad \text{left} \quad (154.8)$$

$$\frac{\partial \omega_{nnx-1,j}}{\partial x} = 0 \quad \text{right} \quad (154.9)$$

$$\omega_{i,0} = \frac{2}{h^2}(\Psi_{i,0} - \Psi_{i,1}) \quad \text{bottom} \quad (154.10)$$

$$\omega_{i,nny-1} = \frac{2}{h^2}(\Psi_{i,nny-1} - \Psi_{i,nny-2}) \quad \text{top} \quad (154.11)$$

4. if u, v, ω and Ψ not converged go to 1



benchmark

Before we solve the problem above we need to make sure that the code is correct. After all, we are solving to PDEs with somewhat complex boundary conditions inside a convergence loop. We here focus on the steady state Navier-Stokes equation and on the Stokes equation.

Let us now prescribe $u(y) = y(1-y)$ on the inlet, i.e. a flow with a parabolic profile. Since the velocity is independent of the x-coordinate this implies that all derivatives with respect to x will vanish. We then expect this velocity everywhere in the domain since it is solution of the steady state Navier-Stokes and Stokes equations (todo), so that $\omega(x,y) = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = -1 + 2y$ anywhere in the domain.

Since $u = y(1-y) = \frac{\partial \Psi}{\partial y}$ on the boundary and Ψ independent of x there too, I can compute $\Psi(y) = \frac{1}{2}y^2 - \frac{1}{3}y^3 + c$ where c is a constant. Since Ψ is known up to a constant (and is constant) at the bottom, we arbitrarily fix $\Psi = 0$ at the bottom, which automatically sets $c = 0$ (continuity of Ψ). This means that

at the top Ψ is constant and equal to $\frac{1}{2} - \frac{1}{3} = \frac{1}{6}$. At the outlet we have $\partial\Psi/\partial x = 0$. This last condition is compatible with the values on the top and bottom so no need to think about the corners.

We then have the analytical solution for this problem:

$$\begin{aligned} u(x,y) &= y(1-y) \\ v(x,y) &= 0 \\ \Psi(x,y) &= \frac{1}{2}y^2 - \frac{1}{3}y^3 \\ \omega(x,y) &= -1 + 2y \end{aligned} \tag{154.12}$$

We can use these expressions to check the results of each Poisson solver: when solving for Ψ we use the analytical expression for ω, and when solving for ω we can use the analytical expression for Ψ. In the end we find that each block of the code is correct.

At this stage the loop goes as follows:

```
omega[:]=0
for iter in range(0,niter):
    solve Delta psi = -omega
    compute u,v from psi
    solve vorticity eq (psi used in bc) -> omega
```

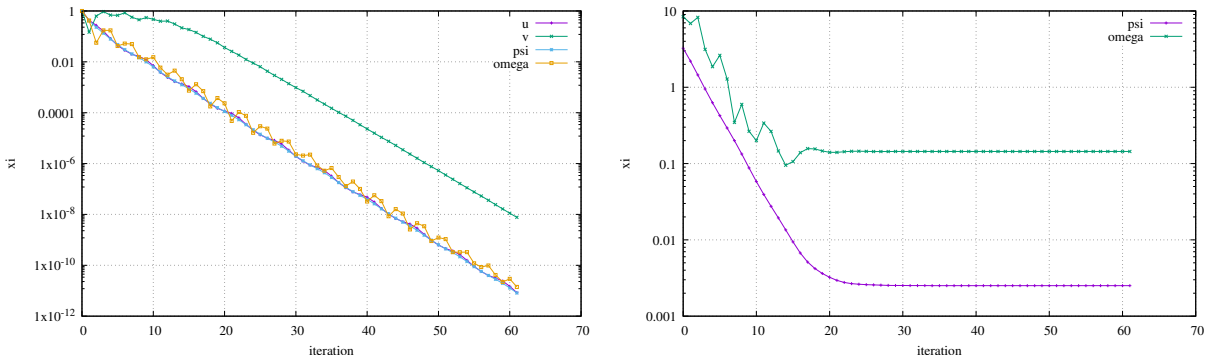
We find that this simply diverges from the 2nd iteration onwards and ultimately 'explodes' with field values absurdly high.

One way to fix this problem is to use relaxation:

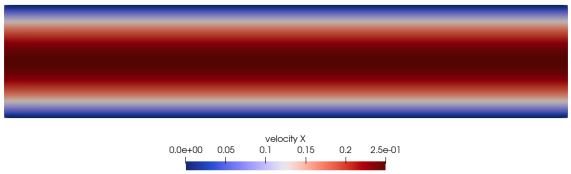
```
omega[:]=0
omegamem[:]=0
psimem[:]=0
for iter in range(0,niter):
    solve Delta psi = -omega
    psi[:]=beta*psi[:]+(1-beta)*psimem[:]
    compute u,v from psi
    solve vorticity eq (psi used in bc) -> omega
    omega[:]=beta*omega[:]+(1-beta)*omegamem[:]
    psimem[:]=psi[:]
    omegamem[:]=omega[:]
```

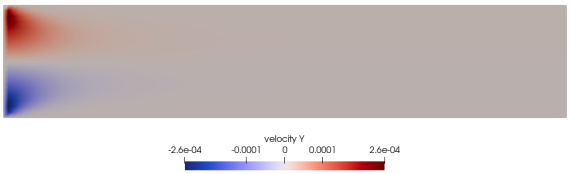
where β controls the relaxation. If equal to one, there is none. If $\beta \to 0$ there is little progress being made in the loop towards convergence. We find that $\beta = 0.5$ still diverges but $\beta = 0.33$ yields a successful iteration.

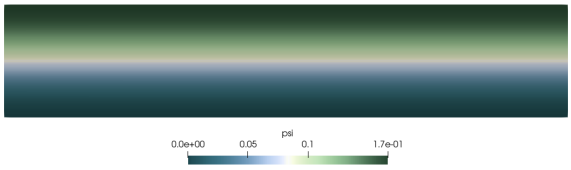
Remark: Another avenue could be to think about the first PDE $\Delta\Psi = -\omega$. Although it is true that we do not know ω everywhere in the domain it could be still beneficial to start with a field ω that reflects the boundary conditions, i.e. apply the b.c. for ω. Of course some of these b.c. depend on Ψ inside the domain which we obviously do not know, but we can nevertheless give it a try. On 2nd thought, it is somewhat stupid: i can only compute ω on the boundary which is precisely where psi b.c. are prescribed so that these values of omega are not used in the end.

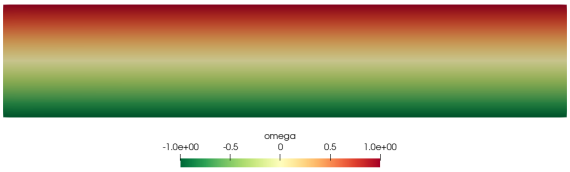


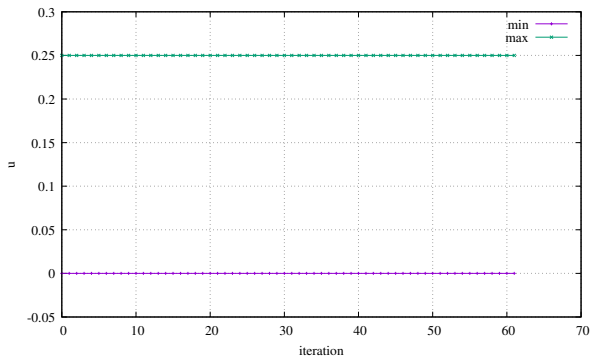
Left: convergence of u, v, ψ and ω during the iterations. Right: 2-norm of the error fields (i.e. computed fields - analytical solution)

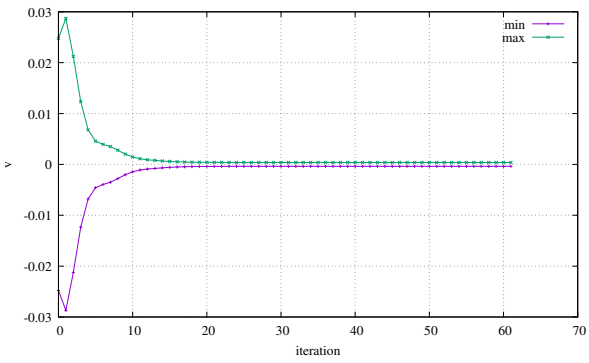


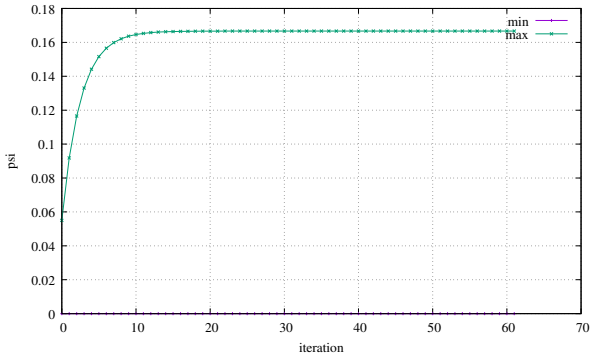


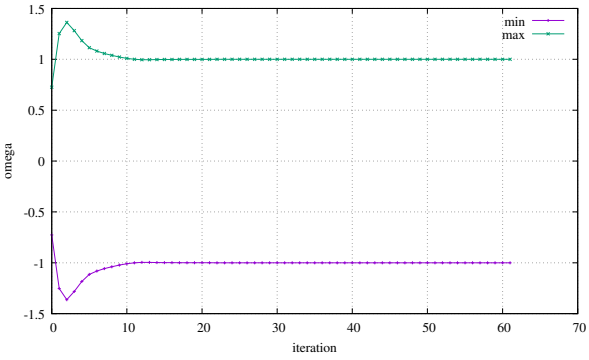












Donea & Huerta manufactured solution

We start from the biharmonic equation

$$\vec{\nabla}^4 \Psi = \frac{1}{\eta_0} \left(-\frac{\partial b_x}{\partial y} + \frac{\partial b_y}{\partial x} \right)$$

In Section **??** we find the stream function defined in the unit square

$$\Psi(x,y) = f(x)\cdot g(y) = x^2(x-1)^2\cdot y^2(y-1)^2,$$

with

$$\begin{aligned}
f'(x) &= 2x(x-1)^2 + 2x^2(x-1)\\
&= 2(x-1)[x(x-1)+x^2]\\
&= 2(x-1)(2x^2-x)\\
&= 2x(x-1)(2x-1)\\
f''(x) &= 2[(x-1)(2x-1)+x(2x-1)+2x(x-1)]\\
&= 2(2x^2-x-2x+1+2x^2-x+2x^2-2x)\\
&= 2(6x^2-6x+1)\\
f'''(x) &= 12(2x-1)\\
f''''(x) &= 24\\
g'(y) &= 2y(y-1)^2 + 2y^2(y-1)\\
&= 2(y-1)[y(y-1)+y^2]\\
&= 2(y-1)(2y^2-y)\\
&= 2y(y-1)(2y-1)\\
g''(y) &= 2[(y-1)(2y-1)+y(2y-1)+2y(y-1)]\\
&= 2[2y^2-3y+1+2y^2-y+2y^2-2y]\\
&= 2(6y^2-6y+1)\\
g'''(y) &= 2(12y-1)\\
g''''(y) &= 24
\end{aligned}$$

In what follows we wish to verify that once we have postulated a pressure field this stream function is compatible with the $\vec{b}$ vector of the Donea & Huerta benchmark of Section **??**.

We then must compute:

$$\begin{aligned}
\vec{\nabla}^2\Psi &= \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2}\\
&= \frac{\partial^2}{\partial x^2}(fg) + \frac{\partial^2}{\partial y^2}(fg)\\
&= f''g + fg''\\
\vec{\nabla}^4\Psi &= \vec{\nabla}^2(f''g + fg'')\\
&= f''''g + 2f''g'' + fg''''\\
&= 24y^2(y-1)^2 + 2\cdot 2(6x^2-6x+1)\cdot 2(6y^2-6y+1) + 24x^2(x-1)^2\\
&= 24(y^4-2y^3+y^2) + 8(6x^2-6x+1)(6y^2-6y+1) + 24(x^4-2x^3+x^2)\\
&= 24(y^4-2y^3+y^2) + (288x^2y^2-288x^2y+48x^2-288xy^2+288xy-48x+48y^2-48y+8) + 24(x^4-2x^3+\\
&= 8-48x-48y+288xy+(48+24)x^2+(24+48)y^2-288x^2y-288xy^2-48x^3-48y^3+288x^2y^2+24x^4+\\
&= 8-48x-48y+288xy+72x^2+72y^2-288x^2y-288xy^2-48x^3-48y^3+288x^2y^2+24x^4+24y^4
\end{aligned}$$

Assuming $\eta_0 = 1$ the stream function above yields the Donea & Huerta velocity field:

$$\begin{aligned}
u(x,y) &= \partial_y\Psi = f(x)g'(y) = x^2(x-1)^2 2y(y-1)(2y-1)\\
v(x,y) &= -\partial_x\Psi = -f'(x)g(y) = -2x(x-1)(2x-1)y^2(y-1)^2
\end{aligned} \tag{154.13}$$

As we have seen when deriving the biharmonic equation the pressure entirely disappears by cancellation so we can postulate the pressure field to be

$$p(x,y) = \frac{1}{2}x^2 - \frac{1}{6}$$

Note that the 1/6 offset is such that $\int p dV = 0$.

The right hand side $\vec{b}$ is to be found in Section **??**:

$$\begin{aligned} b_x &= (12-24y)x^4 + (-24+48y)x^3 + (-48y+72y^2-48y^3+12)x^2 \\ &\quad +(-2+24y-72y^2+48y^3)x + 1 - 4y + 12y^2 - 8y^3 \\ b_y &= (8-48y+48y^2)x^3 + (-12+72y-72y^2)x^2 \\ &\quad +(4-24y+48y^2-48y^3+24y^4)x - 12y^2 + 24y^3 - 12y^4 \end{aligned}$$

so that we can compute

$$\begin{aligned} \frac{\partial b_x}{\partial y} &= -24x^4 + 48x^3 + (-48+144y-144y^2)x^2 + (24-144y+144y^2)x - 4 + 24y - 24y^2 \\ &= -24x^4 + 48x^3 - 48x^2 + 144x^2y - 144x^2y^2 + 24x - 144xy + 144xy^2 - 4 + 24y - 24y^2 \\ \frac{\partial b_y}{\partial x} &= (8-48y+48y^2)3x^2 + (-12+72y-72y^2)2x + (4-24y+48y^2-48y^3+24y^4) \\ &= 24x^2 - 144x^2y + 144x^2y^2 - 24x + 144xy - 144xy^2 + 4 - 24y + 48y^2 - 48y^3 + 24y^4 \end{aligned}$$

and then

$$\begin{aligned} & \mathcal{F}(x,y) \\ = \; & -\frac{\partial b_x}{\partial y} + \frac{\partial b_y}{\partial x} \\ = \; & (4+4) + (-24-24)x + (-24-24)y + \\ & (48+24)x^2 + (24+48)y^2 + (144+144)xy \\ & +(-144-144)x^2y + (-144-144)xy^2 + (-48)x^3 + (-48)y^3 \\ & (144+144)x^2y^2 + (24)x^4 + (24)y^4 \\ = \; & 8 - 48x - 48y + 72x^2 + 72y^2 + 288xy - 288x^2y - 288xy^2 - 48x^3 - 48y^3 + 288x^2y^2 + 24x^4 + 24y^4 \end{aligned}$$

In the end we wish to solve $\vec{\nabla}^2\omega = -\mathcal{F}(x,y)$ since $\eta_0 = 1$.

To Do, open questions, future work?

- N-S case i.e. time dependent ?
- do original bench

Stone 155

Vorticity-Streamfunction - Convection in 2D box



python_codes/fieldstone_155/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_155`

This STONE computes the 2-D thermal convection of a fluid of infinite Prandtl number in a unit square closed cell as described in Blankenbach et al. [23]. The mass, momentum, energy equations are solved by means of the Finite Difference method applied to a vorticity-stream function formulation.

The temperature is fixed to zero on the top and to $\Delta T = 1$ at the bottom, with reflecting symmetry at the sidewalls (i.e. $\partial_x T = 0$) and there are no internal heat sources. Free-slip conditions are implemented on all boundaries. Models are run for $\mathsf{Ra} = 10^{3,4,5,6}$.

The initial temperature field is given by

$$T(x,y) = (1-y) - 0.01\cos(\pi x)\sin(\pi y) \tag{155.1}$$

The perturbation in the initial temperature fields leads to a perturbation of the density field and sets the fluid in motion. Depending on the Rayleigh number, the system ultimately reaches a steady state after some time (the higher the Rayleigh number, the larger this time). Note that at $t = 0$, we have

$$\frac{\partial T}{\partial x} = 0.01\pi\sin(\pi x)$$

which we need in the context of the stream function.

The Nusselt number (i.e. the mean surface temperature gradient over mean bottom temperature) is computed as follows [23]:

$$\mathsf{Nu} = L_y \frac{\int_0^{L_x} \frac{\partial T}{\partial y}(y=L_y)\, dx}{\int_0^{L_x} T(y=0)\, dx} \tag{155.2}$$

Note that in our case the denominator is equal to 1 since $L_x = 1$ and the temperature at the bottom is prescribed to be 1.

If no convection is taking place (i.e. Ra is too low) then only diffusion is taking place and the steady state temperature field is then $T(y) = 1 - y$, so that $\partial_y T = -1$ and we find that the Nusselt number is simply 1.

The equations to be solved are as follows:

$$\vec{\nabla}^2\omega = -\mathsf{Ra}\frac{\partial T}{\partial x} \tag{155.3}$$

$$\vec{\nabla}^2\Psi = -\omega \tag{155.4}$$

$$\frac{\partial T}{\partial t} + \vec{\mathrm{v}}\cdot\vec{\nabla}T - \vec{\nabla}^2 T = 0 \qquad \text{with} \qquad \vec{\mathrm{v}} = (\partial\Psi/\partial y, -\partial\Psi/\partial x) \tag{155.5}$$

Discretisation

The vorticity and stream function PDEs are Poisson equations and we use a 2nd order stencil (see Section **??**). Boundary conditions are specified here after.

When it comes to the energy equation, we have to solve a PDE of the type:

$$\frac{\partial T}{\partial t} = \mathcal{F}(\vec{\mathrm{v}}, \vec{\nabla}T, \Delta T)$$

with $\mathcal{F}(\vec{\mathrm{v}}, \vec{\nabla}T, \Delta T) = (-\vec{\mathrm{v}}\cdot\vec{\nabla}T + \vec{\nabla}\cdot k\vec{\nabla}T)$. The (explicit) forward Euler method is:

$$\frac{T^n - T^{n-1}}{\delta t} = \mathcal{F}^{n-1}(\vec{\mathrm{v}}, \vec{\nabla}T, \Delta T)$$

but we know this results in an unstable scheme. The (implicit) backward Euler method is:

$$\frac{T^n - T^{n-1}}{\delta t} = \mathcal{F}^{n}(\vec{\mathrm{v}}, \vec{\nabla}T, \Delta T)$$

which is better, and the (implicit) Crank-Nicolson algorithm is:

$$\frac{T^n - T^{n-1}}{\delta t} = \frac{1}{2}\left[\mathcal{F}^{n}(\vec{\mathrm{v}}, \vec{\nabla}T, \Delta T) + \mathcal{F}^{n-1}(\vec{\mathrm{v}}, \vec{\nabla}T, \Delta T)\right]$$

where the superscript n indicates the time step. The Crank-Nicolson is obviously based on the trapezoidal rule, with second-order convergence in time.

In the fully implicit case the energy equation is discretised as follows:

$$\frac{T^n_{i,j} - T^{n-1}_{i,j}}{\delta t} = -u^n_{i,j}\frac{T^n_{i+1,j} - T^n_{i-1,j}}{2h_x} - v^n_{i,j}\frac{T^n_{i,j+1} - T^n_{i,j-1}}{2h_y} + \frac{T^n_{i+1,j} - 2T^n_{i,j} + T^n_{i-1,j}}{h_x^2} + \frac{T^n_{i,j+1} - 2T^n_{i,j} + T^n_{i,j-1}}{h_y^2}$$

or,

$$\left(1 + \frac{2\delta t}{h_x^2} + \frac{2\delta t}{h_y^2}\right)T^n_{i,j} - \left(\frac{u^n_{i,j}\delta t}{2h_x} + \frac{\delta t}{h_x^2}\right)T^n_{i-1,j} - \left(\frac{v^n_{i,j}\delta t}{2h_y} + \frac{\delta t}{h_y^2}\right)T^n_{i,j-1} + \left(\frac{u^n_{i,j}\delta t}{2h_x} - \frac{\delta t}{h_x^2}\right)T^n_{i+1,j} + \left(\frac{v^n_{i,j}\delta t}{2h_y} - \frac{\delta t}{h_y^2}\right)T^n_{i,j+1} = T^{n-1}_{i,j}$$

In the Crank-Nicolson case the energy equation is discretised as follows:

$$\begin{aligned}\frac{T^n_{i,j} - T^{n-1}_{i,j}}{\delta t} = &-u^n_{i,j}\frac{T^n_{i+1,j} - T^n_{i-1,j}}{4h_x} - v^n_{i,j}\frac{T^n_{i,j+1} - T^n_{i,j-1}}{4h_y} + \frac{T^n_{i+1,j} - 2T^n_{i,j} + T^n_{i-1,j}}{2h_x^2} + \frac{T^n_{i,j+1} - 2T^n_{i,j} + T^n_{i,j-1}}{2h_y^2}\\ &-u^{n-1}_{i,j}\frac{T^{n-1}_{i+1,j} - T^{n-1}_{i-1,j}}{4h_x} - v^{n-1}_{i,j}\frac{T^{n-1}_{i,j+1} - T^{n-1}_{i,j-1}}{4h_y} + \frac{T^{n-1}_{i+1,j} - 2T^{n-1}_{i,j} + T^{n-1}_{i-1,j}}{2h_x^2} + \frac{T^{n-1}_{i,j+1} - 2T^{n-1}_{i,j} + T^{n-1}_{i,j-1}}{2h_y^2}\end{aligned}$$

$$\begin{aligned}\frac{T^n_{i,j} - T^{n-1}_{i,j}}{\delta t} + &u^n_{i,j}\frac{T^n_{i+1,j} - T^n_{i-1,j}}{4h_x} + v^n_{i,j}\frac{T^n_{i,j+1} - T^n_{i,j-1}}{4h_y} - \frac{T^n_{i+1,j} - 2T^n_{i,j} + T^n_{i-1,j}}{2h_x^2} - \frac{T^n_{i,j+1} - 2T^n_{i,j} + T^n_{i,j-1}}{2h_y^2}\\ = &-u^{n-1}_{i,j}\frac{T^{n-1}_{i+1,j} - T^{n-1}_{i-1,j}}{4h_x} - v^{n-1}_{i,j}\frac{T^{n-1}_{i,j+1} - T^{n-1}_{i,j-1}}{4h_y} + \frac{T^{n-1}_{i+1,j} - 2T^{n-1}_{i,j} + T^{n-1}_{i-1,j}}{2h_x^2} + \frac{T^{n-1}_{i,j+1} - 2T^{n-1}_{i,j} + T^{n-1}_{i,j-1}}{2h_y^2}\end{aligned}$$

$$
\begin{aligned}
T^n_{i,j} \quad &+ \quad u^n_{i,j}\frac{T^n_{i+1,j}-T^n_{i-1,j}}{4h_x}\delta t + v^n_{i,j}\frac{T^n_{i,j+1}-T^n_{i,j-1}}{4h_y}\delta t - \frac{T^n_{i+1,j}-2T^n_{i,j}+T^n_{i-1,j}}{2h_x^2}\delta t - \frac{T^n_{i,j+1}-2T^n_{i,j}+T^n_{i,j-1}}{2h_y^2}\delta t \\
&= \quad T^{n-1}_{i,j} - u^{n-1}_{i,j}\frac{T^{n-1}_{i+1,j}-T^{n-1}_{i-1,j}}{4h_x}\delta t - v^{n-1}_{i,j}\frac{T^{n-1}_{i,j+1}-T^{n-1}_{i,j-1}}{4h_y}\delta t + \frac{T^{n-1}_{i+1,j}-2T^{n-1}_{i,j}+T^{n-1}_{i-1,j}}{2h_x^2}\delta t + \frac{T^{n-1}_{i,j+1}-2T^{n-1}_{i,j}+T^{n-1}_{i,j-1}}{2h_y^2}\delta t
\end{aligned}
$$

We set

$$
\begin{aligned}
i,j &\rightarrow k \\
i-1,j &\rightarrow k_W \\
i+1,j &\rightarrow k_E \\
i,j-1 &\rightarrow k_S \\
i,j+1 &\rightarrow k_N
\end{aligned}
$$

$$
\begin{aligned}
T^n_k \quad &+ \quad u^n_k\frac{T^n_{k_E}-T^n_{k_W}}{4h_x}\delta t + v^n_k\frac{T^n_{k_N}-T^n_{k_S}}{4h_y}\delta t - \frac{T^n_{k_E}-2T^n_k+T^n_{k_W}}{2h_x^2}\delta t - \frac{T^n_{k_N}-2T^n_k+T^n_{k_S}}{2h_y^2}\delta t \\
&= \quad T^{n-1}_k - u^{k-1}_k\frac{T^{n-1}_{k_E}-T^{n-1}_{k_W}}{4h_x}\delta t - v^{k-1}_k\frac{T^{n-1}_{k_N}-T^{n-1}_{k_S}}{4h_y}\delta t + \frac{T^{n-1}_{k_E}-2T^{n-1}_k+T^{n-1}_{k_W}}{2h_x^2}\delta t + \frac{T^{n-1}_{k_N}-2T^{n-1}_k+T^{n-1}_{k_S}}{2h_y^2}\delta t
\end{aligned}
$$

or,

$$
\begin{aligned}
&\left(1+\frac{\delta t}{h_x^2}+\frac{\delta t}{h_y^2}\right)T^n_k - \left(\frac{u^n_k\delta t}{4h_x}+\frac{\delta t}{2h_x^2}\right)T^n_{k_W} - \left(\frac{v^n_k\delta t}{4h_y}+\frac{\delta t}{2h_y^2}\right)T^n_{k_S} + \left(\frac{u^n_k\delta t}{4h_x}-\frac{\delta t}{2h_x^2}\right)T^n_{k_E} + \left(\frac{v^n_k\delta t}{4h_y}-\frac{\delta t}{2h_y^2}\right)T^n_{k_N} \\
= \quad &\left(1-\frac{\delta t}{h_x^2}-\frac{\delta t}{h_y^2}\right)T^{n-1}_k + \left(\frac{u^{n-1}_k\delta t}{4h_x}+\frac{\delta t}{2h_x^2}\right)T^{n-1}_{k_W} + \left(\frac{v^{n-1}_k\delta t}{4h_y}+\frac{\delta t}{2h_y^2}\right)T^{n-1}_{k_S} + \left(-\frac{u^{n-1}_k\delta t}{4h_x}+\frac{\delta t}{2h_x^2}\right)T^{n-1}_{k_E} + \left(-\frac{v^{n-1}_k\delta t}{4h_y}+\frac{\delta t}{2h_y^2}\right)T^{n-1}_{k_N}
\end{aligned}
$$

Remark: in the rhs, should I use velocity at n or $n-1$?

Boundary conditions

We have two Poisson equations to solve. and we wish to prescribe free slip boundary conditions on all four sides of the domain. As documented in STONE 153, We then impose $\omega = 0$ on all sides (first PDE), and $\Psi = 0$ on all sides (second PDE).

Concerning the temperature equation, we prescribe $T = 1$ at the bottom and $T = 0$ at the top. We also prescribe $\partial T/\partial x = 0$ on the sides.

Code structure

Inside the time stepping loop the following five steps are necessary

1. solve vorticity equation
2. solve stream function equation
3. compute velocity field
4. solve temperature equation
5. compute temperature gradient
6. assess convergence

Measurements

At the moment the root mean square velocity is computed using effectively a one-point quadrature: The velocity is computed at the center of each cell, squared, and multiplied by the area of a cell, i.e. $h_x h_y$.

Remark: one could even think of computing the square of the velocity on the nodes and *then* average in the middle. Which is more accurate?

Concerning the Nusselt number, $\partial T/\partial y$ is assumed to be constant along the edge and the integration is here again a one point quadrature.

Both are likely to be less accurate than other approaches. This needs to be investigated at some point.

Arriving to steady state

In the end we are mostly interested in the steady state values of the Nusselt number and the root mean square velocity. Instead of solving the energy equation above, we then cross the $\partial T/\partial t$ term. In this case the value of the time step is irrelevant and we set $\delta t = 1$. Instead of representing steps in time the loop now is an iteration loop (Picard iterations). The same criterion remains to detect steady state. Unsurprisingly this proves to be much faster than the time evolution approach. This is parameterised by the steady_state variable in the code.

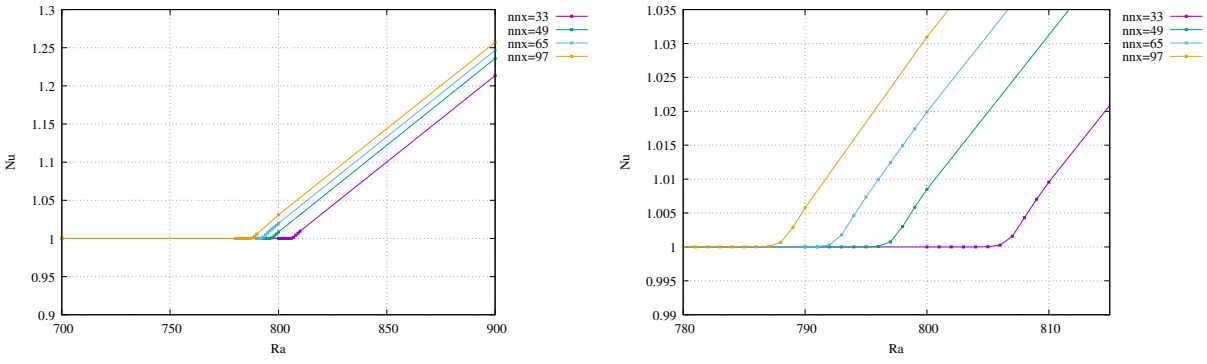
Assessing convergence

Whether models are in steady state mode or run actual time evolution of the system we need to assess whether steady state has been reached. For this we store in temporary arrays the previous values of the u, v, T, ω, Ψ fields and compute the relative change of these quantities at each time step/iteration and store these in $\xi_u, \xi_v, \xi_T, \xi_\omega, \xi_\Psi$. If all five quantities are below a user-defined tolerance then steady state has been reached.

I have found that convergence is extremely slow when close to the critical Rayleigh number. If $\mathsf{Ra} < \mathsf{Ra}_{cr}$ then we expect a conductive geotherm and zero velocities. I have therefore added $\mathrm{v}_{rms} < 10^{-2}$ as a test for steady state convergence.

Results: Onset of convection Steady state approach

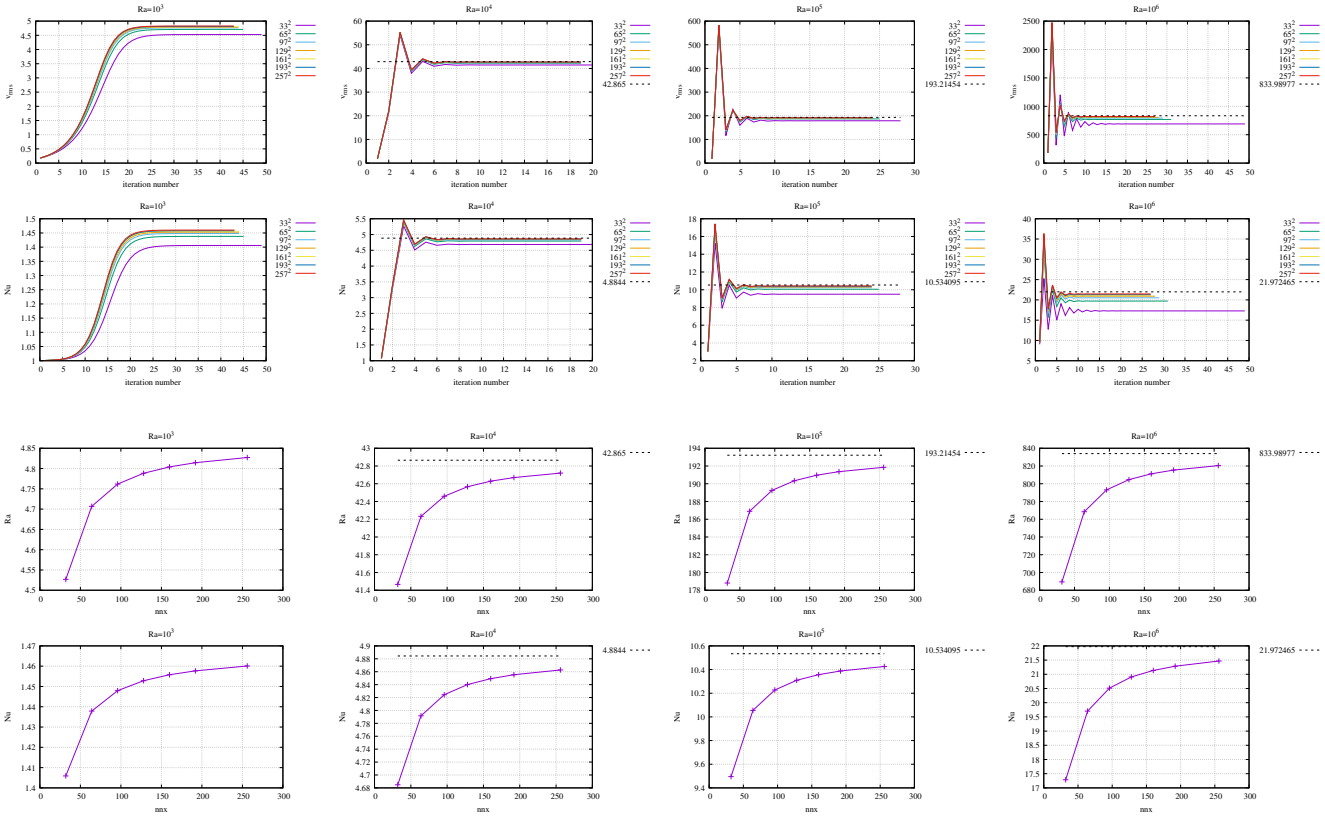
Results are obtained with `script_onset`, having set steady_state=True.



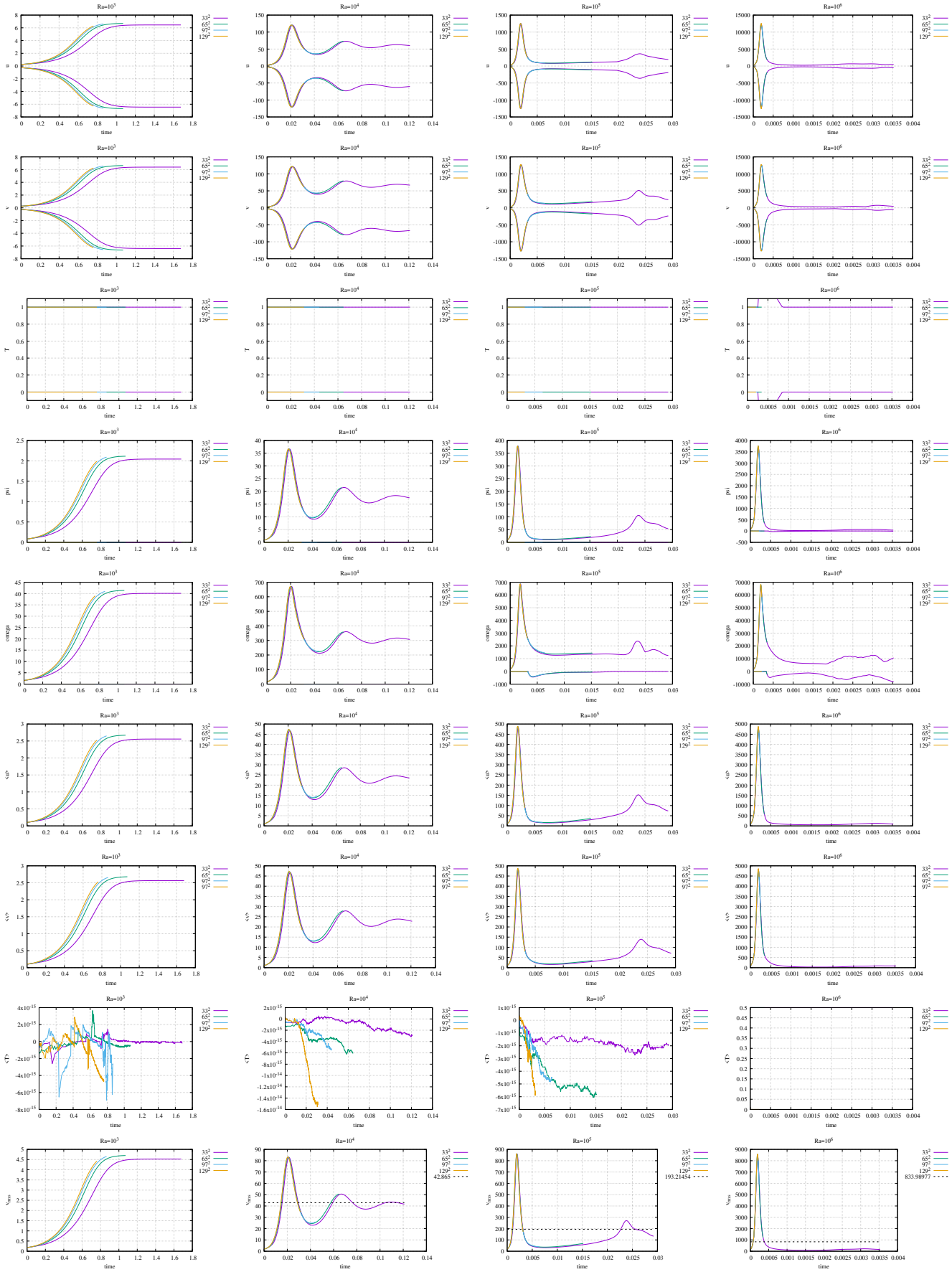
Rather interestingly, we find that the critical Rayleigh number very much depends on resolution. It here looks like a resolution of 97^2 nodes is nowhere near enough to capture resolution-independent values.

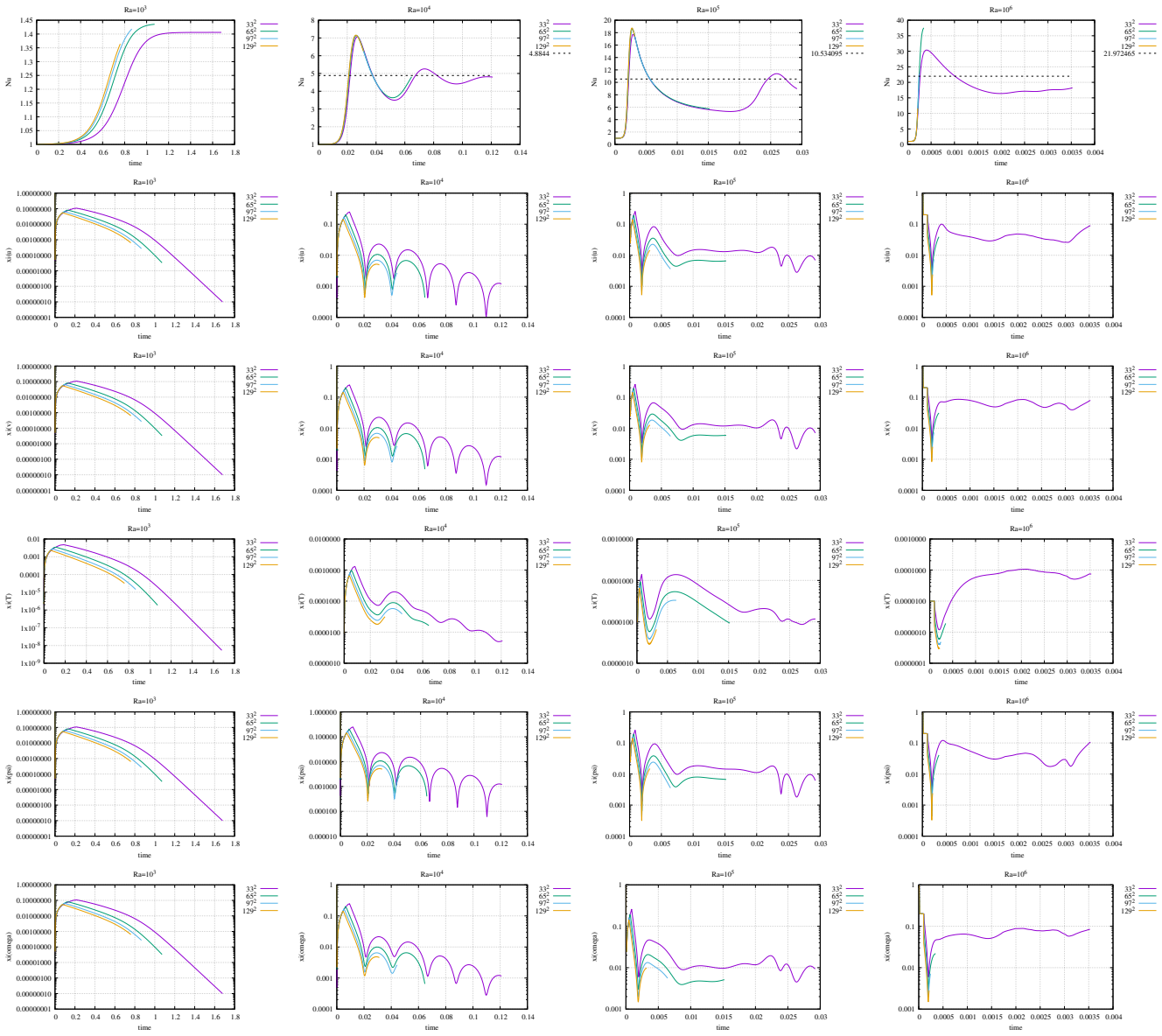
Results: $\mathrm{Ra} = 10^{3,4,5,6}$, Steady state approach

Results are obtained with `script_ss`.



Results: $\mathrm{Ra} = 10^{3,4,5,6}$, CFL=0.5, time evolution





To Do, open questions, future work?

- compute dTdx,dTdy with second order accurate stencil on boundary too
- run all to steady state

Stone 156

Vorticity-Streamfunction & the Lorenz equations - Kellogg & Turcotte (1990)



python_codes/fieldstone_156/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_156

This STONE is inspired by and is meant to be an incomplete replication of Kellogg and Turcotte [185] (1990). Most of the text below is copied verbatim from the paper (but not the codes nor the results). Note that the same equations can be found in Kellogg [186] (1993). It turns out that the Lorenz equations are actually a staple of chaos theory and there is *a lot* of literature about these[1]. Of relevance is the paper: Stewart and Turcotte [310] (1989).

The major goals of this work are to generate a flow in two dimensions that is chaotic in both space and time and to calculate the deformation of a strain marker in that flow. The flow is confined to a box of length w and detph h. The stream function is specified as follows:

$$\Psi = \left(\frac{\lambda\sqrt{2}}{\pi^2} \right) \sin \pi y' \left\{ B(t') \sin \frac{\pi x'}{\lambda} + [C(t') - 27] \sin \frac{2\pi x'}{\lambda} \right\} \tag{156.1}$$

where $x' = x/h$, $y' = y/h$, $\lambda = w/h$ and t' is the nondimensional time. B and C are time-dependent driver of the flow. This solution satisfies conservation of mass and free slip conditions. The flow contains either one or two convection cells within the box, depending on the magnitude of B and C. As B and C vary smoothly but chaotically in time, the flow will oscillate smoothly but chaotically between one and two cells.

The flow is driven using the solutions to the Lorenz equations in the chaotic regime:

$$\frac{\partial A}{\partial t'} = -\sigma A + \sigma B \tag{156.2}$$

$$\frac{\partial B}{\partial t'} = -AC + rA - B \tag{156.3}$$

$$\frac{\partial C}{\partial t'} = AB - bC \tag{156.4}$$

where σ, r, and b are parameters and $t' = \pi^2(1+\lambda^{-2})\kappa t/h^2$ is the nondimensional time with $b = 4/(1+\lambda^{-2})$ and κ the thermal diffusivity.

[1] https://en.wikipedia.org/wiki/Lorenz_system

These are numerically integrated to find the time dependence of the solution. The parameters σ and r represent the Prandtl number and the ratio of the Rayleigh number to the critical Rayleigh number for the onset of convection.

In the study the authors use: $r = 28$, $\sigma = 10$, and $b = 8/3$ ($\lambda = \sqrt{2}$ since $w = \sqrt{2}$ and $h = 1$). The solution to this problem has three fixed points, corresponding to steady flows:

- $A = B = C = 0$ which corresponds to no convection;
- $A = B = 6\sqrt{2}$ and $C = 27$;
- $A = B = -6\sqrt{2}$ and $C = 27$.

These steady solutions are unstable, however, and after an initial transient, the solutions to the equations above approach the strange attractor[2] in $A - B - C$ space. The solution spirals around the fixed points $A = B = 6\sqrt{2}$, $C = 27$ and $A = B = -6\sqrt{2}$, $C = 27$. It is however non repeating, and the solution is highly sensitive to the initial conditions.

In the physical $x - y$ space, the flow obtained from Lorenz's model is chaotic in time but not space; a particle follows a closed path, corresponding to a single streamline, but its position on that path and its velocity are chaotic in time. This is unlikely to occur in real fluids, and the Lorenz model has very limited direct applicability to real thermal flows.

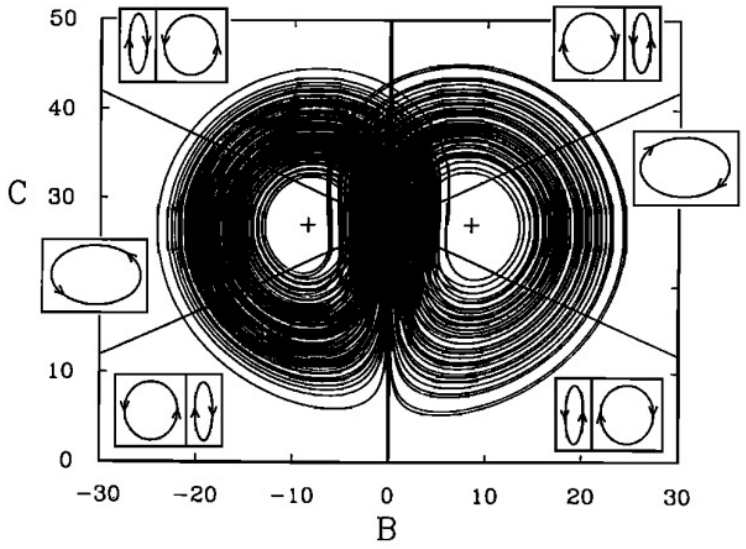
Also, the Lorenz model cannot be directly applied to mantle convection, because at infinite Prandtl number, the chaotic behavior disappears and the Lorenz model reverts to steady convection.

The velocity field is obtained as follows[3]:

$$\begin{aligned} u'(x',y') &= \frac{\partial \Psi}{\partial y'} = \left(\frac{\lambda\sqrt{2}}{\pi}\right) \cos \pi y' \left\{ B(t') \sin \frac{\pi x'}{\lambda} + [C(t') - 27] \sin \frac{2\pi x'}{\lambda} \right\} \\ v'(x',y') &= -\frac{\partial \Psi}{\partial x'} = -\left(\frac{\sqrt{2}}{\pi}\right) \sin \pi y' \left\{ B(t') \cos \frac{\pi x'}{\lambda} + 2[C(t') - 27] \cos \frac{2\pi x'}{\lambda} \right\} \end{aligned} \tag{156.5}$$

We can easily verify that $u'(x' = 0) = u'(x' = w) = 0$ and $v'(x', y' = 0) = v'(x', y' = 1) = 0$.

The evolution of the flow is shown in the figure here under, which contains a projection of the solution of the Lorenz equations onto the $B - C$ plane:



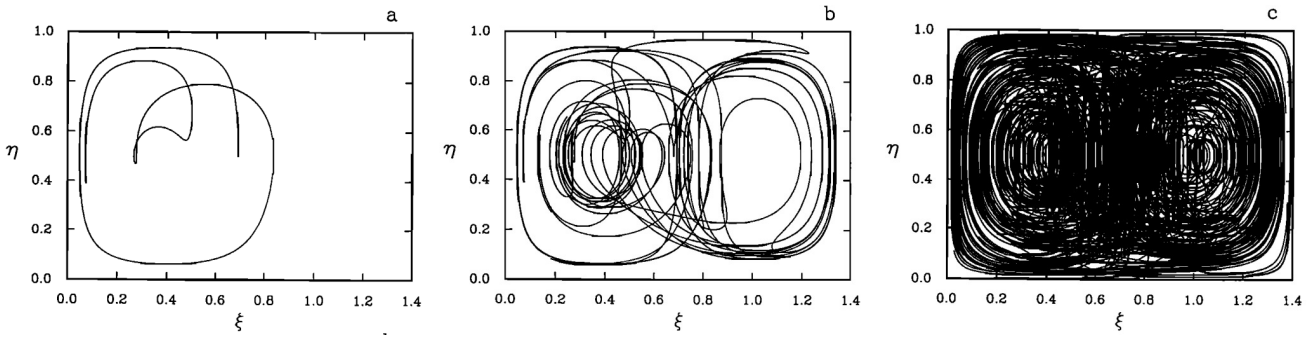
Evolution of the Lorenz variables B and C, which govern the time-dependent flow of equation (156.1). This is the solution to the Lorenz equations (156.2)-(156.4) projected onto the $B - C$ plane. It is calculated using σ=10, r=28, and $b = 8/3$. The inset figures are schematics of the streamlines of Eq. (156.1) corresponding different values of B and C. The flow used in the mixing calculations oscillates smoothly between one and two cells.

[2] `https://en.wikipedia.org/wiki/Attractor`

[3] The authors use a different convention for the minus sign so these velocities are opposites of those in the paper.

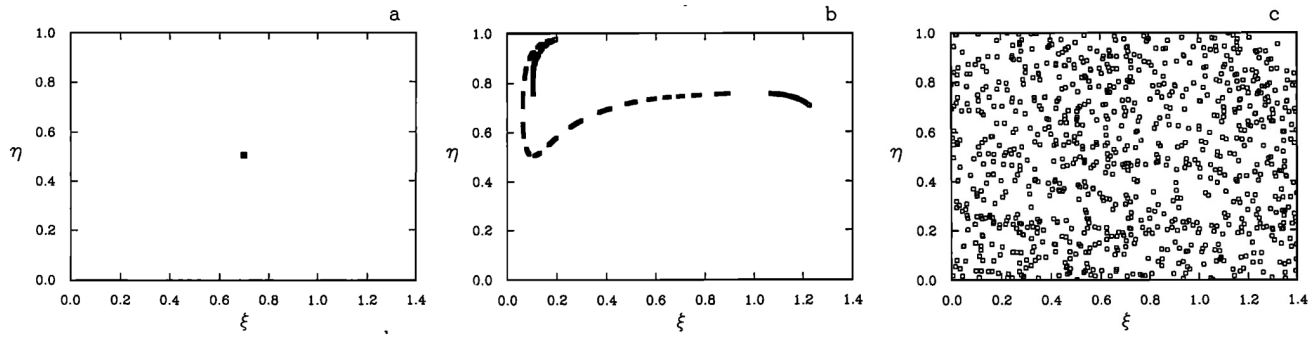
The figure is divided into regions corresponding to the differing flow regimes. The inset figures show schematically the streamlines obtained for different values of B and C. Though the streamlines are closed at all times, particle trajectories are not, and the resulting flow has space-filling particle paths.

A representative particle path is shown in the next figure after times of $t' = 1$ (one overturn time), $t' = 10$, and $t' = 100$. The flow is space-filling in that a particle will eventually migrate to every point in the box.



Particle trajectory in the chaotic flow. The particle path is shown after times of (a) $t' = 1$, (b) $t' = 10$, and (c) $t' = 100$. The particle is initially positioned at the center of the box.

A group of particles which are initially clustered close together will eventually be driven apart and scattered evenly throughout the box:

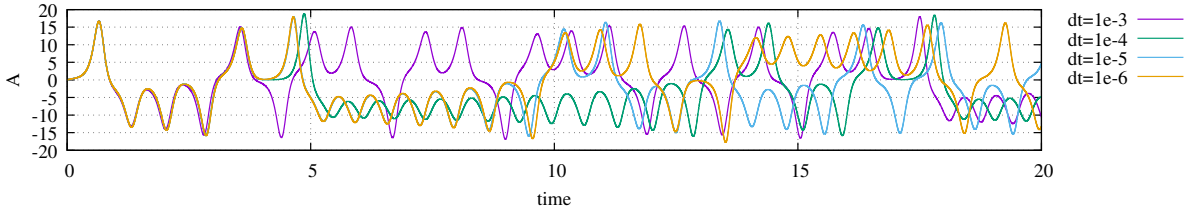


Scattering of a cluster of particles in the chaotic flow. (a) 900 particles are placed near the center of the box, and their positions are tracked and plotted after times of (b) $t' = 1$ and (c) $t' = 10$.

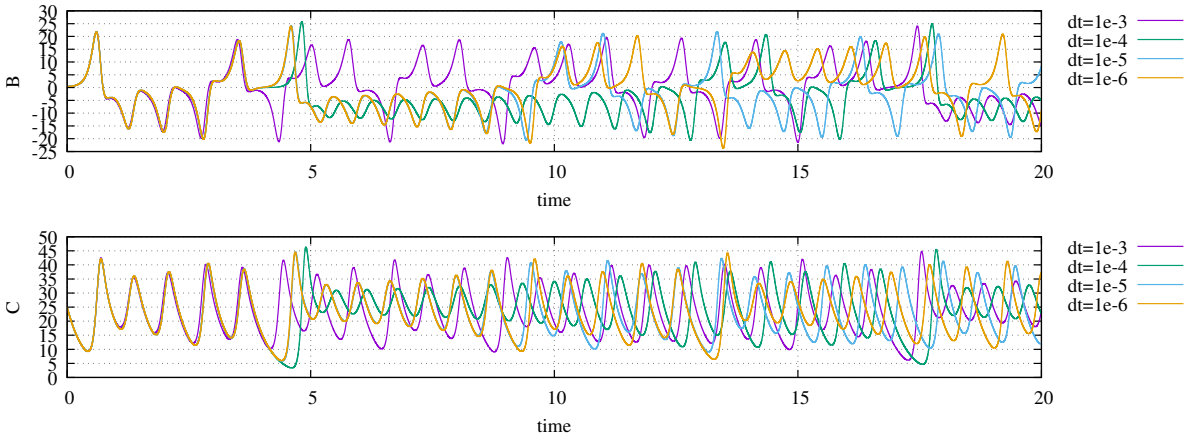
What we find online

Approach 1

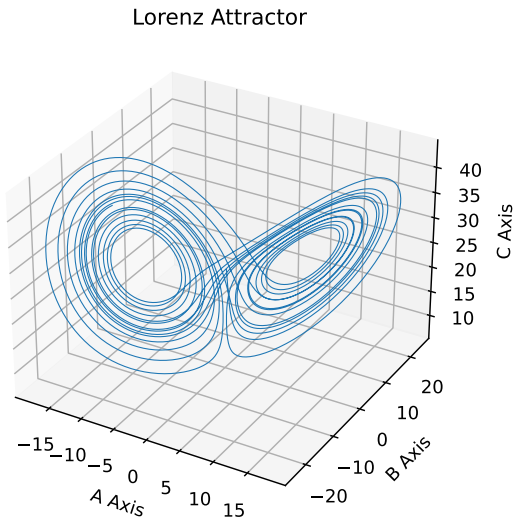
There is a Wikipedia entry[4] for the Lorenz system. As it turns out, it is also run with $s = 10$, $\beta = 8/3$ and $\rho = 28$. There are also a few codes available, in Matlab, Python and Julia on the page. The python one is saved in the folder `wiki` in this STONE . It unfortunately implements a first order time integrator so that it will not be of much use when it comes to accuracy of the results (I have then also modified the starting point to match mine, i.e. 0,0.5.25).



[4] `https://en.wikipedia.org/wiki/Lorenz_system`

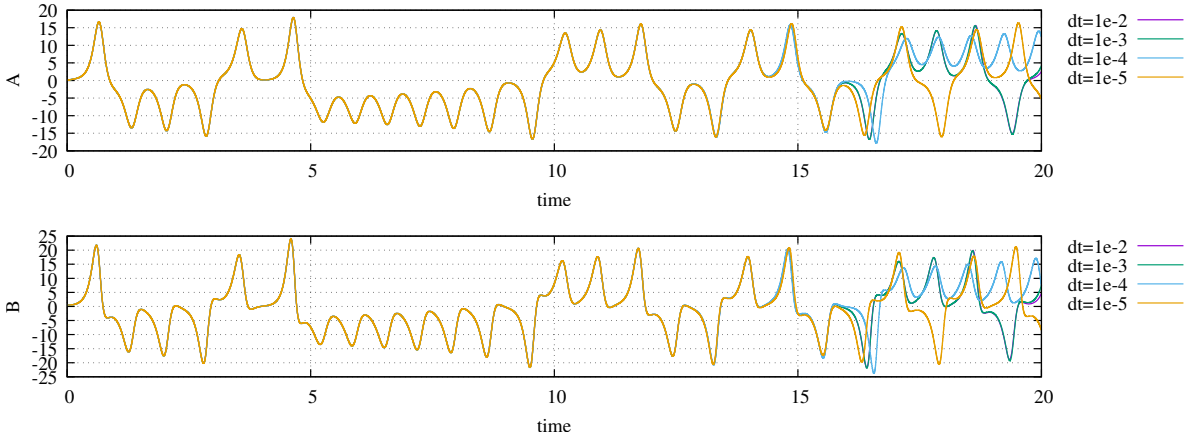


Given the simplistic integration scheme, we find that results diverge after $t' > 5$.

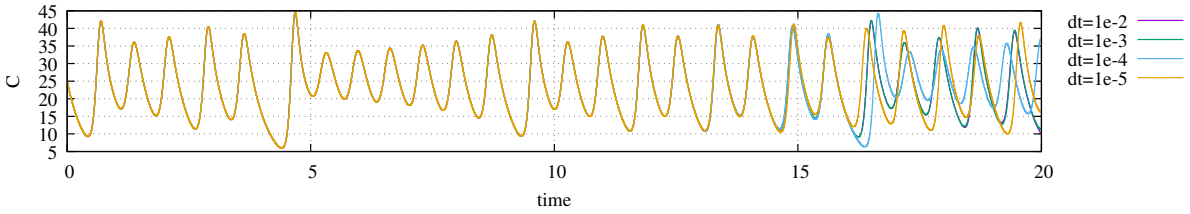


Approach 2

I found another similar code[5], but it now makes use of 'odeint' which seems to be deprecated[6]. I am not sure what the method behind odeint actually is. The code is to be found in the `marksmath` folder. The timestep is set by the user.



[5] `https://marksmath.org/classes/Fall2020DiffEq/demos/Lorenz.html`
[6] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html`



Results are much better and only diverge for $t' > 15$.

Approach 3

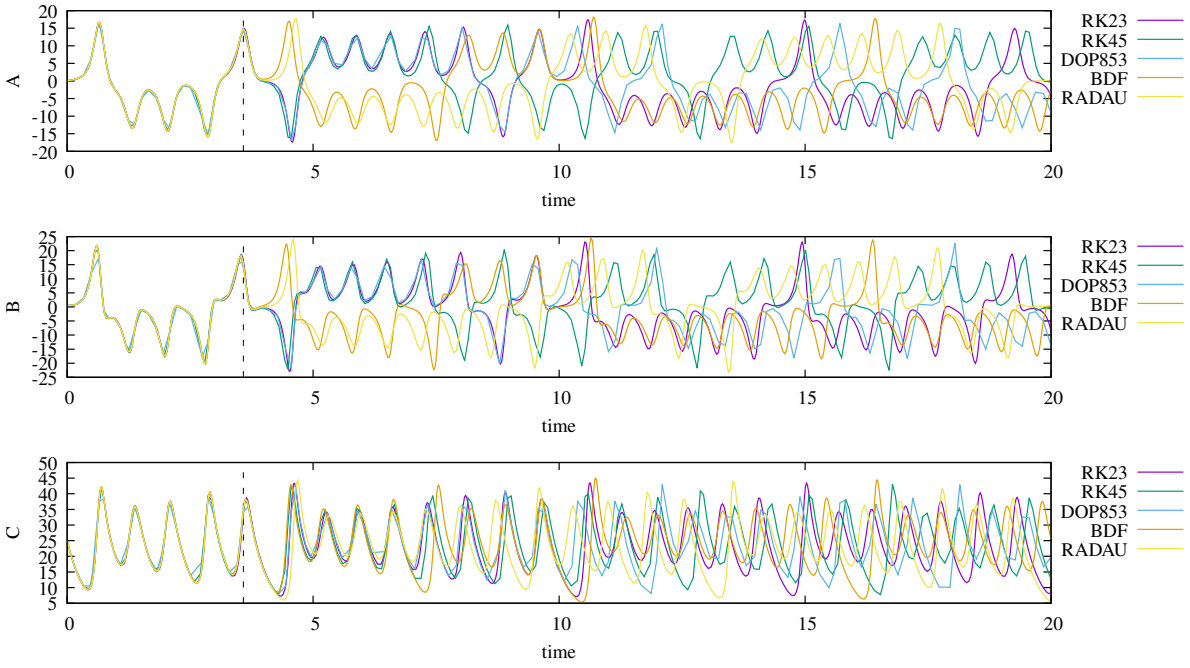
We find elsewhere[7] an even more advanced take on this problem. It uses the solve_ivp function[8]. It uses by default RK45, but can be called with RK23, DOP853 (Dormand & Prince, see hereafter), Radau[9] and BDF (the last two are implicit methods). The Lorenz function is defined as follows:

```
def lorenz(t, X, sigma, beta, rho):
    u, v, w = X
    up = -sigma*(u - v)
    vp = rho*u - v - u*w
    wp = -beta*w + u*v
    return up, vp, wp
```

and a call to solve_ivp goes as follows:

```
soln = solve_ivp(lorenz, (0,tmax), (u0,v0,w0), args=(sigma, beta, rho),
                 method='RK23', max_step=maxdt)
```

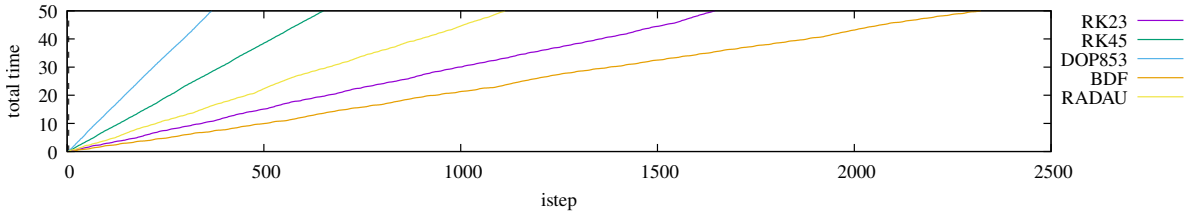
For now the step size is not bounded (maxdt is infinite) and determined solely by the integration scheme.



Looking at $A(t')$, we find that at $t' \simeq 4$ the RK-like methods 'go up' while the Radau & BDF methods go down. This is raher disappointing.

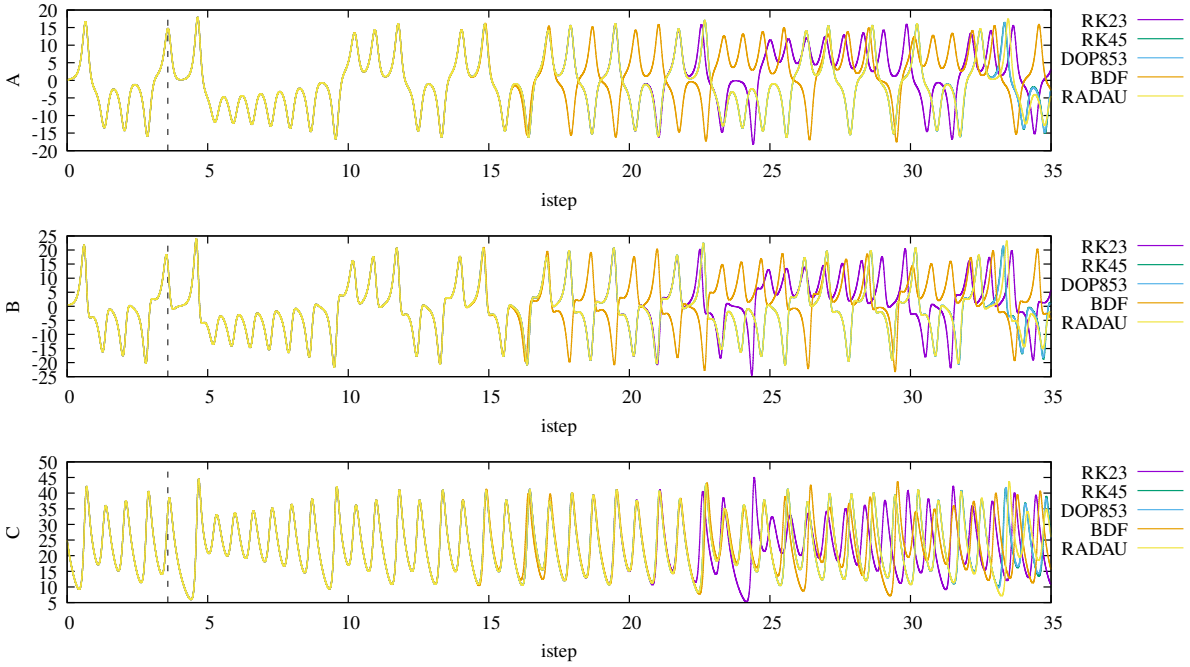
Looking closer I found out that the total number of time steps used to reach $t' = 20$ for each method varies quite a lot:

[7] https://scipython.com/blog/the-lorenz-attractor/
[8] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html
[9] https://en.wikipedia.org/wiki/List_of_Runge-Kutta_methods



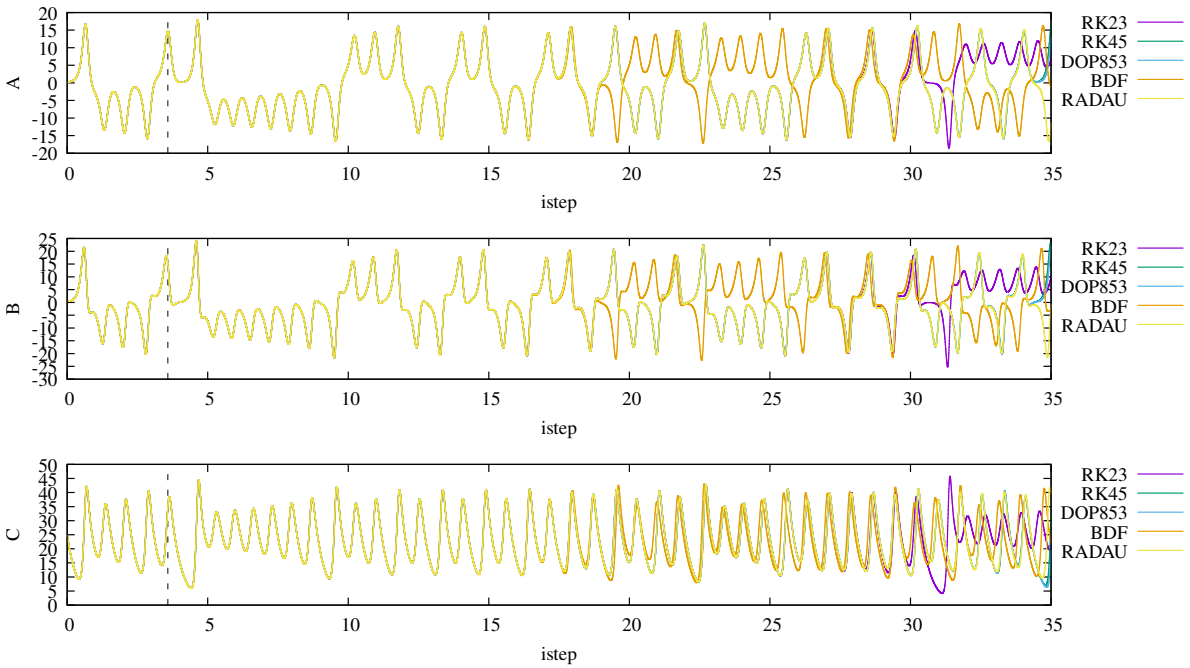
RK23: 1647 steps; RK45: 652 steps $\delta t \simeq 0.077$; DOP853: 369 steps; Radau: 1114 steps; BDF: 2323 $\delta t \simeq 0.0215$.

I feel that we need to try these methods again with much smaller time step. I therefore set $\delta t = 10^{-4}$ for all five methods:



My hunch was correct and a smaller time step allows all the methods to agree up to $t' \simeq 15$. RK23 unsurprinsigly does not perform as well as the others.

This is very encouraging so I now set $\delta t = 10^{-5}$ for all five methods:



Surprisingly BDF refuses to yield the same results as all other methods.

In the end, no matter what I try, I cannot seem to get all methods to agree past $t' = 30$!

Let us now try to solve these equations ourselves...

About the code

The domain is a rectangle of size L_x, L_y with $L_x = w = \sqrt{2}$ and $L_y = h = 1$ so that $\lambda = w/h = \sqrt{2}$ as in the publication (see figures above). We set $\sigma = 10$, $r = 28$ and $b = 8/3$. What we ultimately miss are the initial values for A, B, C. We cannot start with all three equal to zero, otherwise we obtain the solution $A = B = C = 0$ solution. Looking at the first figure in $B - C$ space we see that the values are constrained in $[-30, 30] \times [0, 50]$ so we can set $B = 0.5$ and $C = 25$ at $t' = 0$. For now we also set $A = 0$ at startup.

At the moment markers are set on a 40×40 grid in a square of size 0.025×0.025 placed in the middle of the domain, and they are advected by means of a simple Euler step[10].

The code then solve Eqs. (156.2),(156.3),(156.4). These are coupled first-order nonlinear equations. From our experience with codes found online, this is no good news and specific numerical techniques should be used in this case.

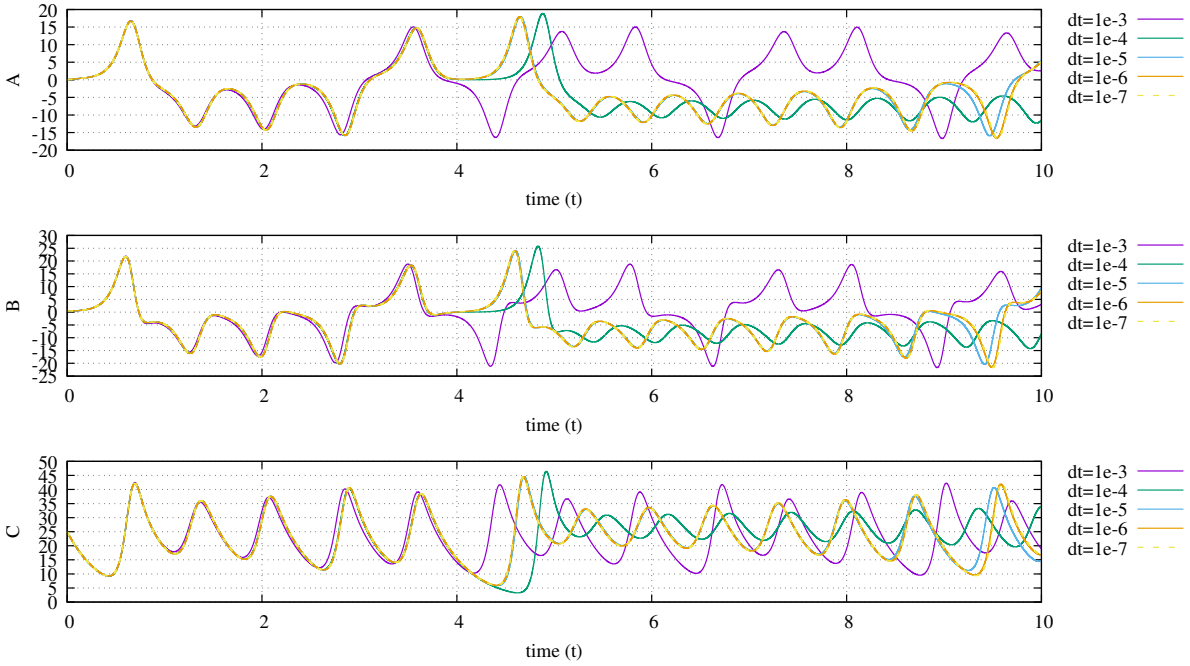
scheme 1

To get things started I nevertheless opt for a simplistic approach. The time derivatives are discretised by means of 1st order explicit formulas (`scheme 1`):

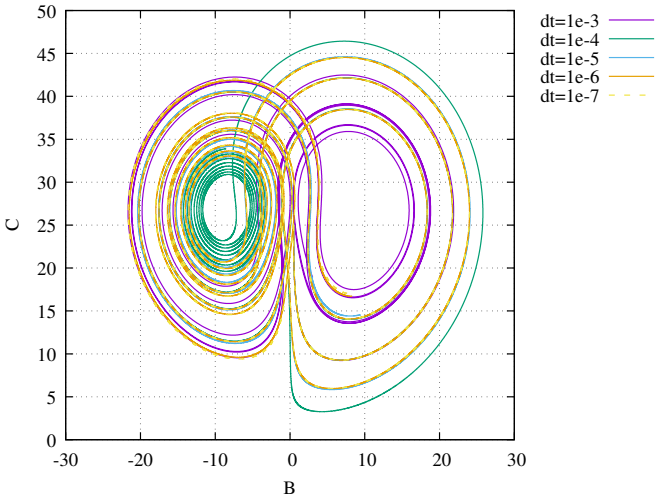
$$\begin{aligned}
\frac{A^{n+1} - A^n}{\delta t} &= -\sigma A^n + \sigma B^n \\
\frac{B^{n+1} - B^n}{\delta t} &= -A^n C^n + r A^n - B^n \\
\frac{C^{n+1} - C^n}{\delta t} &= A^n B^n - b C^n
\end{aligned}$$

where the upperscript denotes the timestep. We will use tiny time steps and hope for the best.

Every time that new B and C values are obtained we can use these to compute the velocity field and to drive the flow.



[10] This should be improved!



Clearly $\delta t = 1e-3$ is too large, and given the simplistic integration scheme, we find that even using very small time steps results diverge after $t' = 10$.

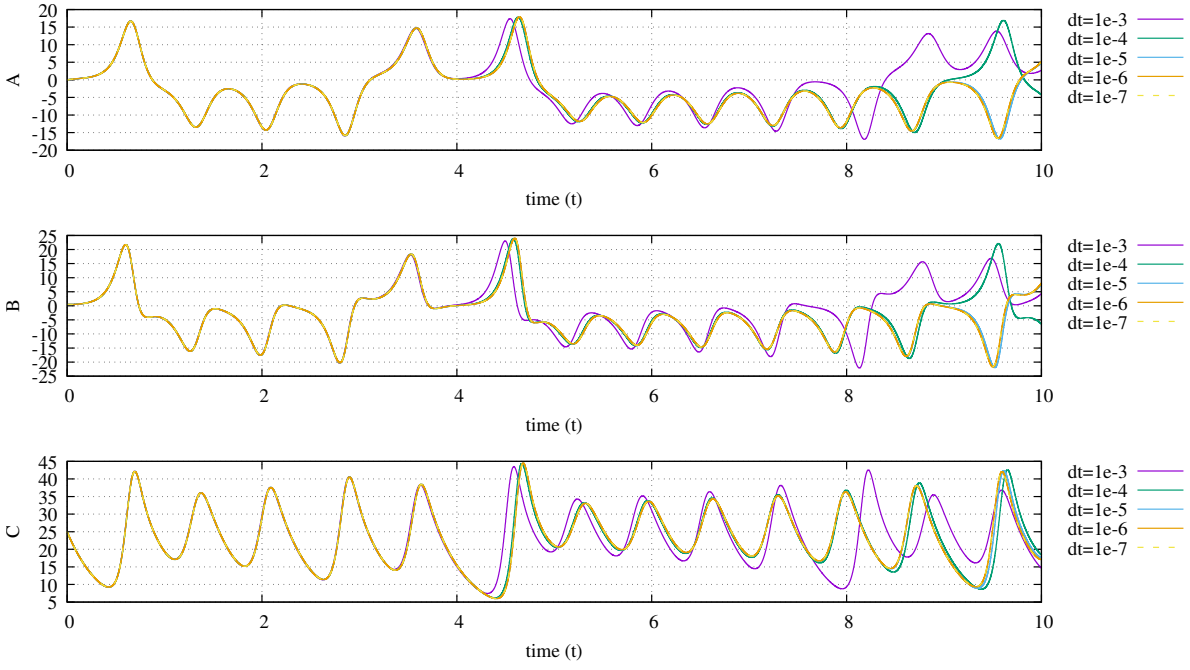
scheme 2

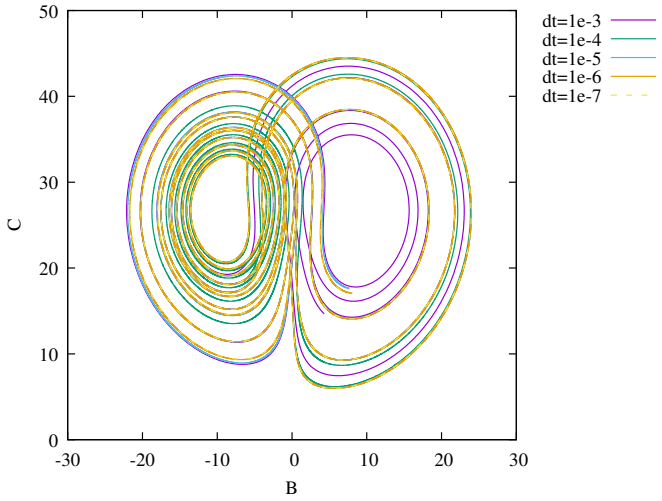
Based on these encouraging results, it is now time to look at a more appropriate scheme to solve the equations. We can try something very simple, i.e. use updated values of B and C when possible (`scheme 2`):

$$\begin{aligned}
\frac{A^{n+1} - A^n}{\delta t} &= -\sigma A^n + \sigma B^n \\
\frac{B^{n+1} - B^n}{\delta t} &= -A^{n+1} C^n + r A^{n+1} - B^n \\
\frac{C^{n+1} - C^n}{\delta t} &= A^{n+1} B^{n+1} - b C^n
\end{aligned}$$

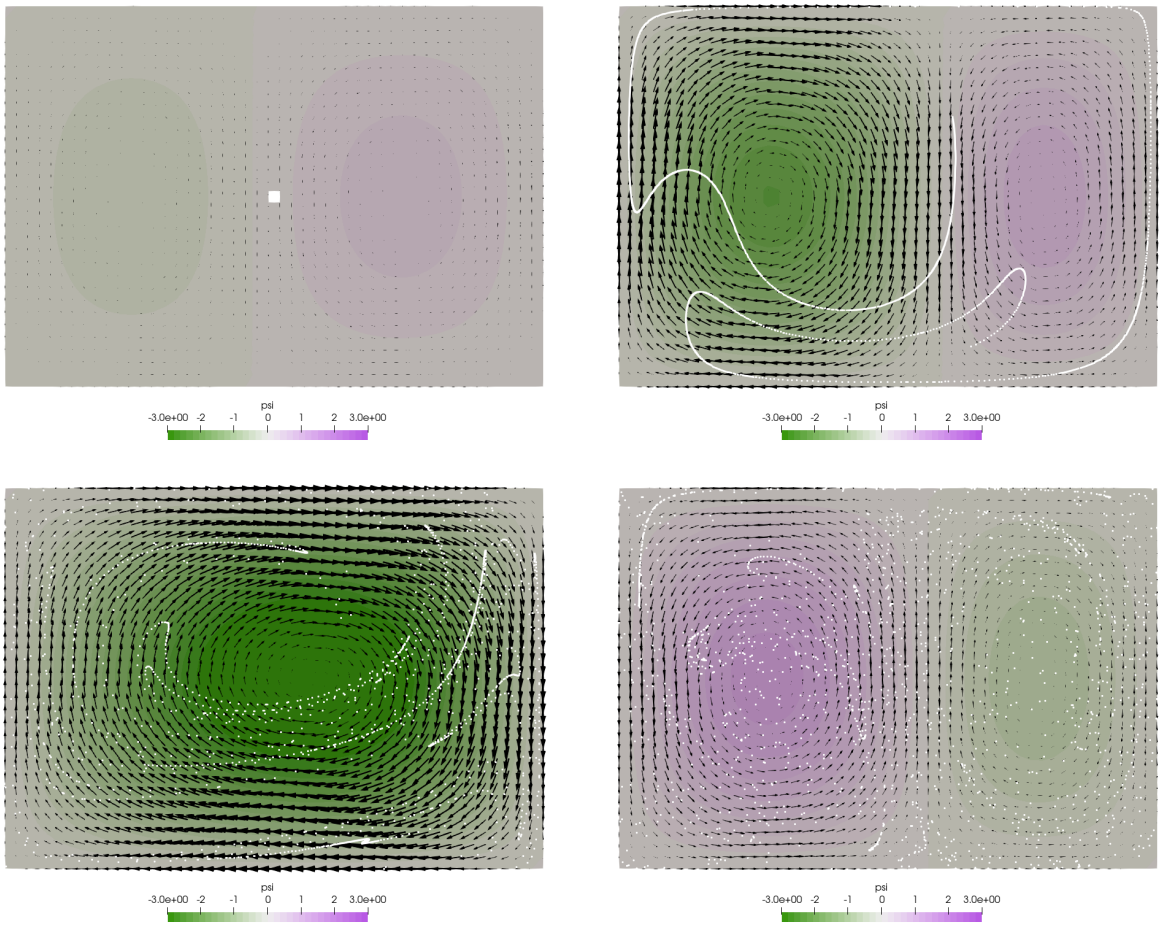
i.e.

$$\begin{aligned}
A^{n+1} &= (-\sigma A^n + \sigma B^n)\delta t + A^n \\
B^{n+1} &= (-A^{n+1} C^n + r A^{n+1} - B^n)\delta t + B^n \\
C^{n+1} &= (A^{n+1} B^{n+1} - b C^n)\delta t + C^n
\end{aligned}$$





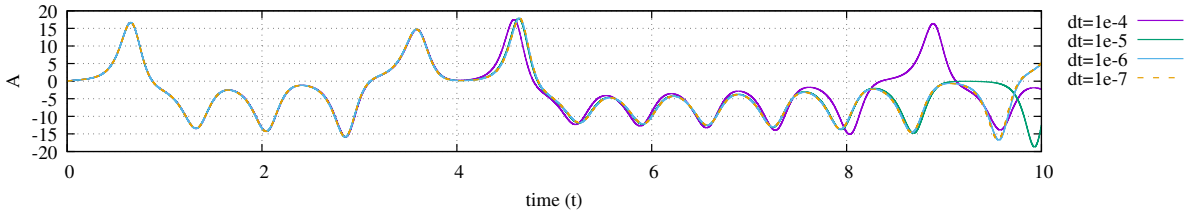
This is somewhat better, lines with 'large' δt are closer to the reference $\delta t = 5 \cdot 10^{-5}$ line. I have also made a nice video of what happens: `https://www.youtube.com/watch?v=Bd9rRSSbH1o`.

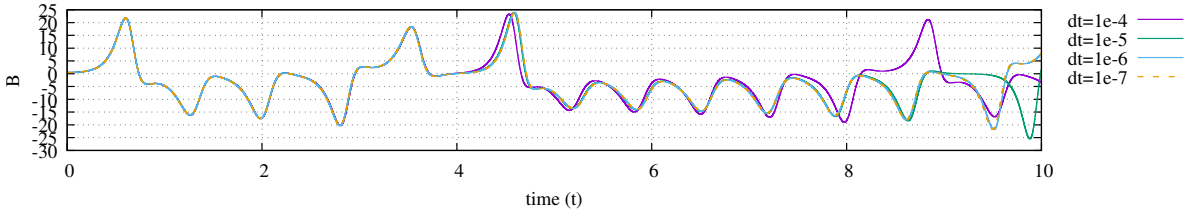


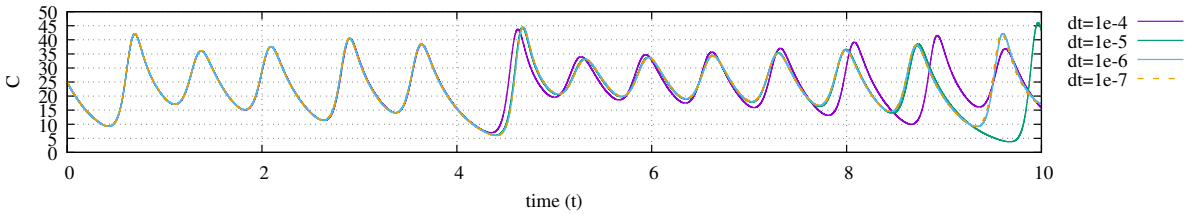
Markers, stream function field and velocity arrows. Obtained with $\delta t = 10^{-4}$.

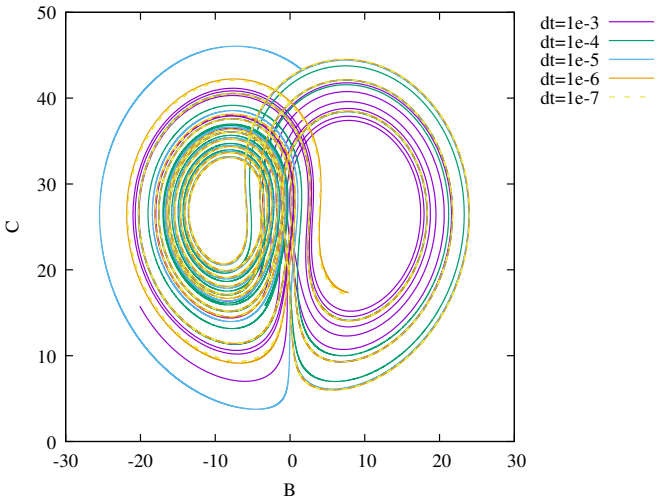
scheme 3

`scheme=3` does a few (fixed point) iterations at each time step:





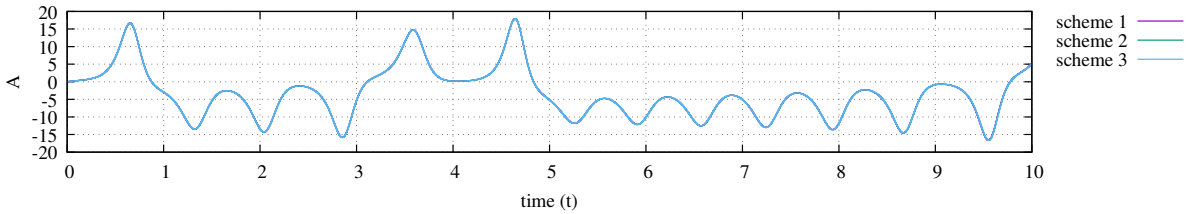


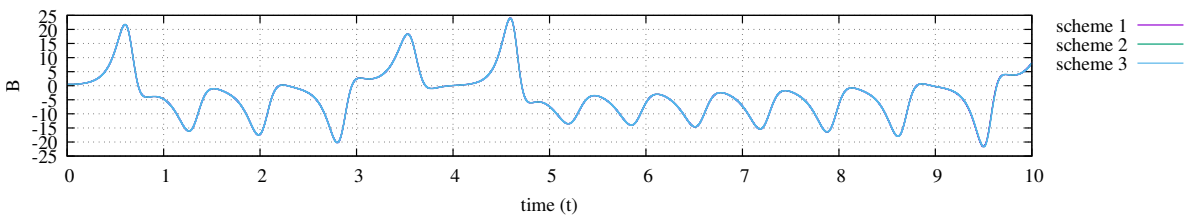


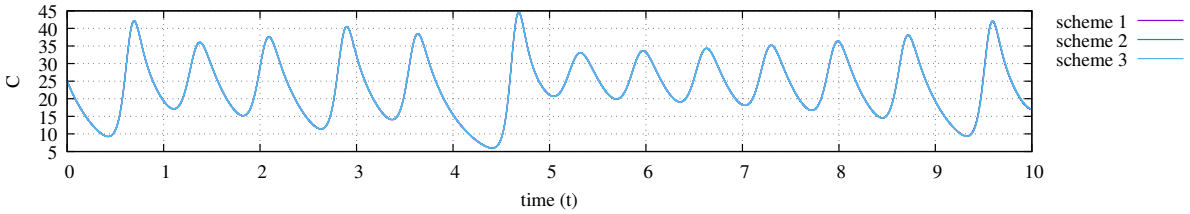
Not sure what to make of it... looks different than the other two...

Comparing results from schemes 1, 2 and 3

We can compare the three schemes (1,2,3) when the smallest time step $\delta t = 10^{-7}$ is used. We find that up to $t' = 10$ they agree very well.







scheme 4

Let us now switch to the standard 4th order Runge-Kutta method[11], that we denote by `scheme=4`. It is often described as follows. Let an initial value problem be specified as follows:

$$dy/dt = f(t, y), y(t_0) = y_0$$

Here y is an unknown function (scalar or vector) of time t, which we would like to approximate; we are told that dy/dt, the rate at which y changes, is a function of t and of y itself. At the initial time t_0 the corresponding y value is y_0. The function f and the initial conditions t_0, y_0 are given. Now we pick a step-size $h > 0$ and define:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad\qquad t_{n+1} = t_n + h$$

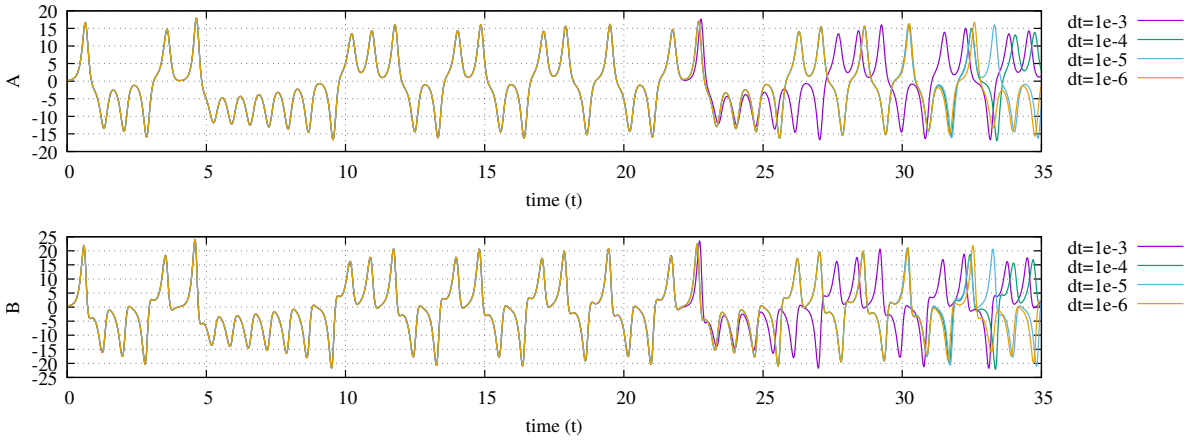
for $n = 0, 1, 2, 3, ...$, using

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 &= f(t_n + h, y_n + hk_3) \end{aligned}$$

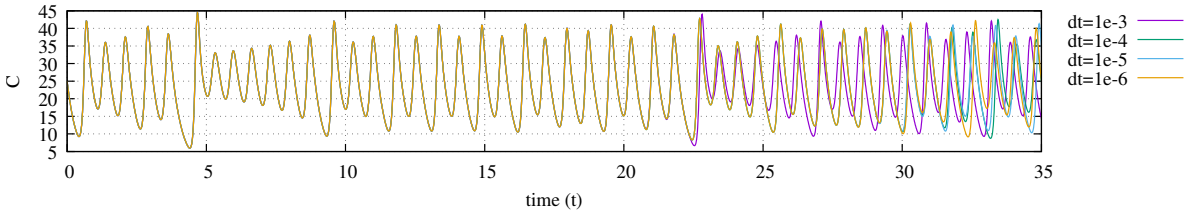
In this case, we have

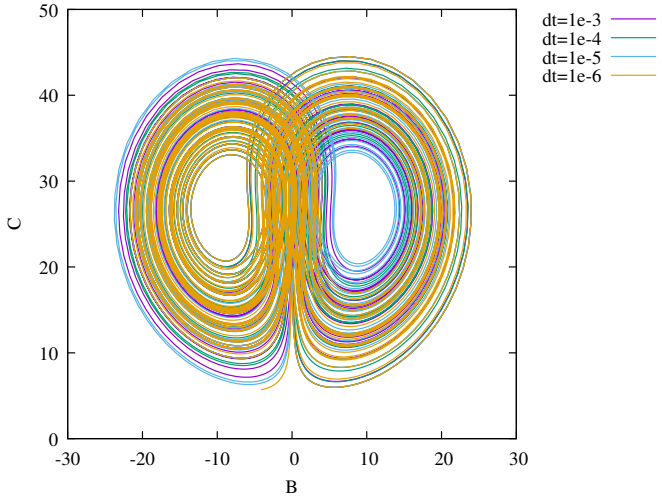
$$\frac{d}{dt}\begin{pmatrix} A \\ B \\ C \end{pmatrix} = \vec{f}(A, B, C)$$

This is probably a somewhat naive attempt at using RK4 for coupled PDEs. Note that Runge-Kutta methods are discussed in Section 14.10. We find that even when the time step is 'large' the solutions are identical to our most accurate solutions so far. I therefore increase the final time to $t' = 35$. We find that differences start to show up at the $t' \simeq 22$ mark.



[11] `https://en.wikipedia.org/wiki/Runge-Kutta_methods`

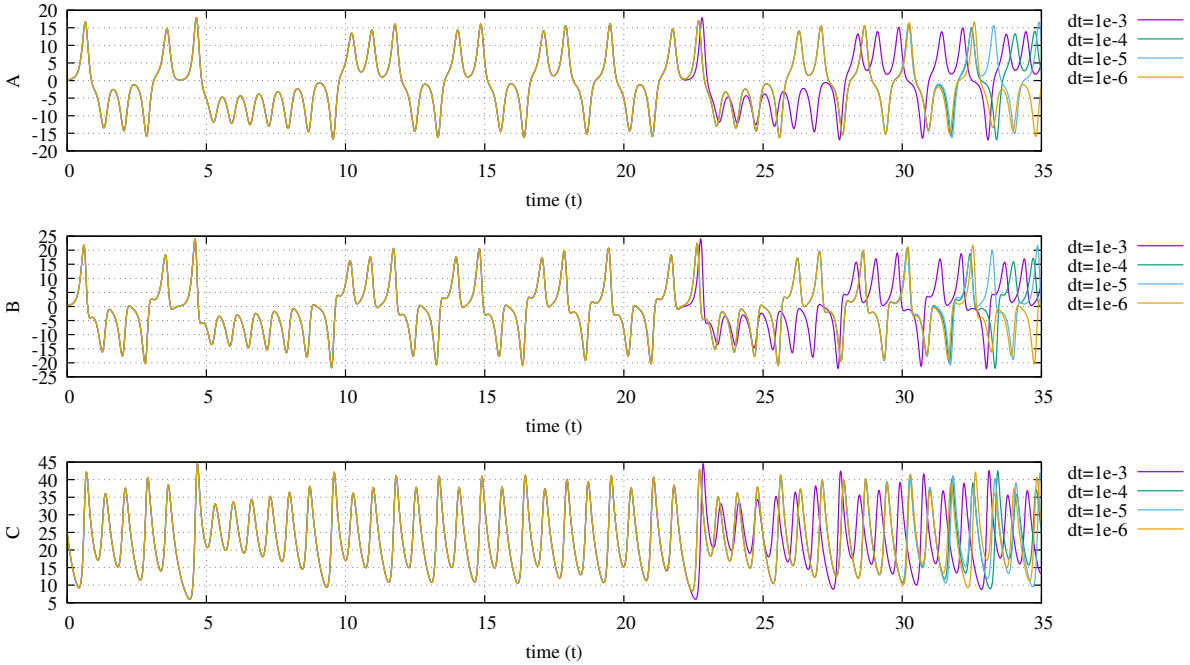


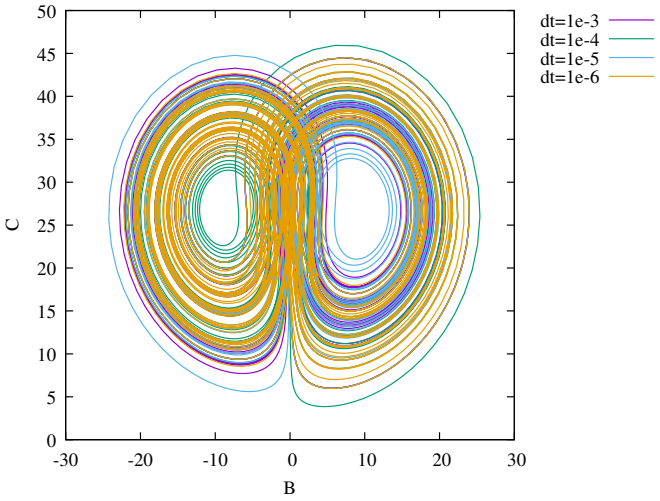


This approach is documented in this video `https://www.youtube.com/watch?v=EXvLju3DLMY` or this one `https://www.youtube.com/watch?v=5CXhHx56COo`.

scheme 5

The Runge-Kutta method of scheme 4 is only one of many, and several other ones which are more accurate exist. Let us try the so-called 3/8-rule one .





scheme 6

We can also try the Runge-Kutta-Fehlberg Method (RKF45, [95]), which is 5th order. The particularity of this method is that from the same Butcher Tableau one can produce a 4th-order approximation $\tilde{A}$ and a 5th-order approximation A:

$$\begin{aligned}\tilde{A}_{n+1} &= A_n + \frac{25}{216}A_1 + \frac{1408}{2565}A_3 + \frac{2197}{4101}A_4 - \frac{1}{5}A_5 \\ A_{n+1} &= A_n + \frac{16}{135}A_1 + \frac{6656}{12825}A_3 + \frac{28561}{56430}A_4 - \frac{9}{50}A_5 + \frac{2}{55}A_6\end{aligned}$$

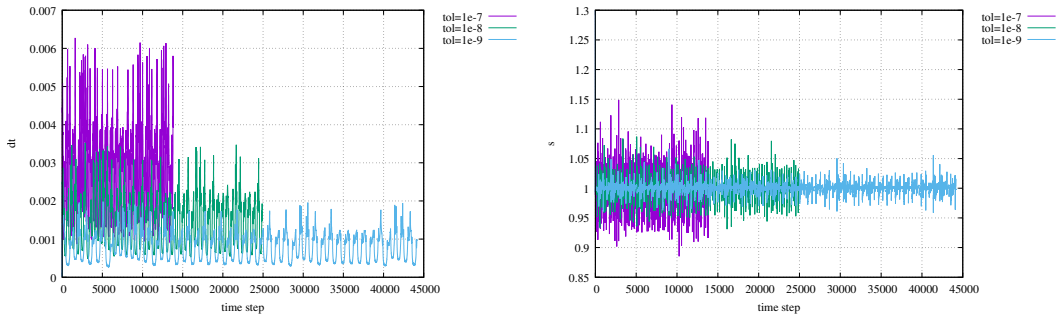
I then define

$$\begin{aligned}R_A &= \frac{1}{\delta t}|\tilde{A}_{n+1} - A_{n+1}| \\ R_B &= \frac{1}{\delta t}|\tilde{B}_{n+1} - B_{n+1}| \\ R_C &= \frac{1}{\delta t}|\tilde{C}_{n+1} - C_{n+1}|\end{aligned}$$

and the new time step is $s \cdot \delta t$, with $s = min(s_A, s_B, s_C)$ and[12]

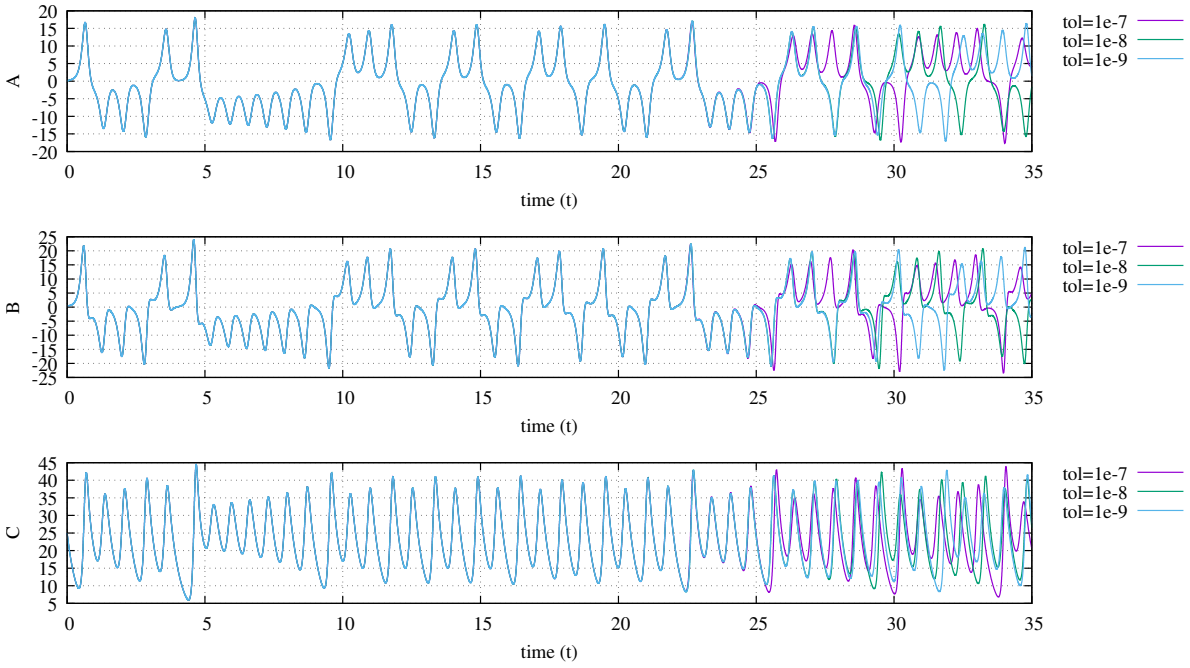
$$s_A = \left(\frac{tol}{2R_A}\right)^{\frac{1}{4}} \qquad s_B = \left(\frac{tol}{2R_B}\right)^{\frac{1}{4}} \qquad s_C = \left(\frac{tol}{2R_C}\right)^{\frac{1}{4}}$$

This means that in this case we do not control the value of the timestep directly, but rather indirectly by means of the tolerance value. We find that for $tol = 1e-7, 8, 9$ we obtain a timestep around 10^{-4}.



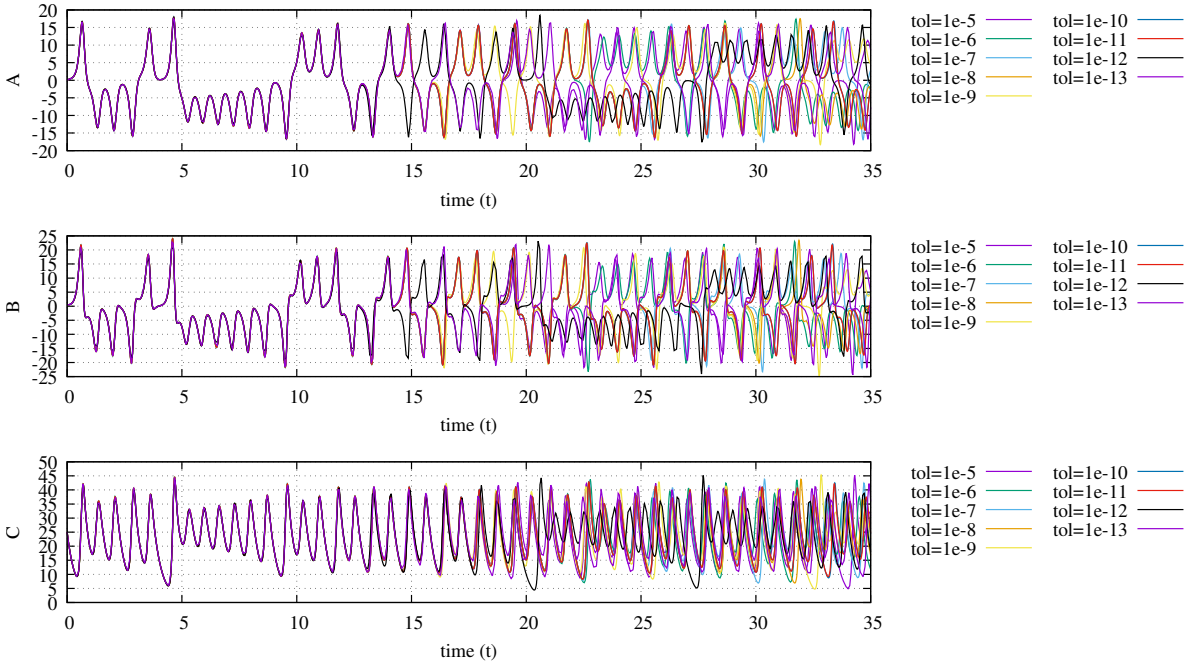
and the results are as follows:

[12] I have a doubt about the exponent: 1/4 or 1/5?



scheme 7

I then went 'all in' and decided to implement a 9th-order Dormand-Prince (often called 'ode87'), see Section 14.10 or also `https://nl.mathworks.com/matlabcentral/fileexchange/3616-ode87-integrator`.



I have also tried to set δt to small values, instead of the tolerance. I recovered similar results as when DOP853 was used in Approach 3 (see earlier).

The conclusion so far is that using (very) high order methods does not yield stable results past $t' > 30$. Based on all this I am actually wondering whether it is actually impossible to have results overlap past $t' > 30$ because of the chaotic nature of the equations themselves.

Stone 157

Solving the 1D diffusion equation with the Method of Lines



python_codes/fieldstone_157/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_157

We start from the 1D diffusion equation

$$\frac{\partial T}{\partial t}=\kappa\frac{\partial^2 T}{\partial x^2}$$

We then discretise the space derivative by means of a centered stencil:

$$\frac{\partial T_i}{\partial t}=\kappa\frac{T_{i+1}-2T_i+T_{i-1}}{h^2}$$

where $i=0,nnx-1$.

Traditionallly we would continue by discretising the time derivative and end up with an explicit, implit or hybrid scheme (e.g. Crank-Nicolson). However in this case we stop here and in essence we have replaced a PDE by a set of ODEs, one per node in the domain, which can be written as follows:

$$\frac{\partial \vec{T}}{\partial t}=\vec{\mathcal{F}}(\vec{T})$$

where $\vec{T}$ contains all the temperature values at the nodes.

In Boudreau [31] (from which this idea is borrowed, see p328) this method is called the Method of Lines (MoL)[1] and we read: " There are significant advantages in solving diagenetic equations[2] by the MoL: i) stiff problems caused by fast reactions are handled readily; ii) highly nonlinear problems can be solved with no fuss; iii) the initial guess is not all that important when solving for the steady state; iv) this procedure is usually very stable and convergent; v) the number of space dimensions is irrelevant; and vi) both transient and steady-state problems can be solved. " The method lends itself to using any kind of ODE integrator (RK methods and more, see Section 14.10).

The domain is a segment of length $L_x=1$. There are $nnx=101$ equidistant nodes. The initial temperature is set to $T(x,t=0)=0.123$ everywhere except at the ends where it is set to $T_{left}=1$ and

[1] http://var.scholarpedia.org/article/Method_of_lines

[2] These are advection-diffusion-reaction equations.

$T_{right} = 0$. The heat diffusivity is set to $\kappa = 0.01$. The model is run until $t_{final} = 100$. The time step is computed as $\delta t = h^2/2\kappa$ where $h = L_x/(nnx - 1)$. To start with the time derivative will be dealt with by means of a 1st order Euler step:

$$\frac{\partial T}{\partial t} \simeq \frac{T^{n+1} - T^n}{\delta t}$$

In a second step, we may wish to use the solve_ivp function[3] which gives access to 5 methods: RK23, RK45 (default) DOP853, BDF and Radau.

```
soln = solve_ivp(F,(0,tfinal),T0,args=(kappa,),method=RKmethod ,max_step=maxdt)
```

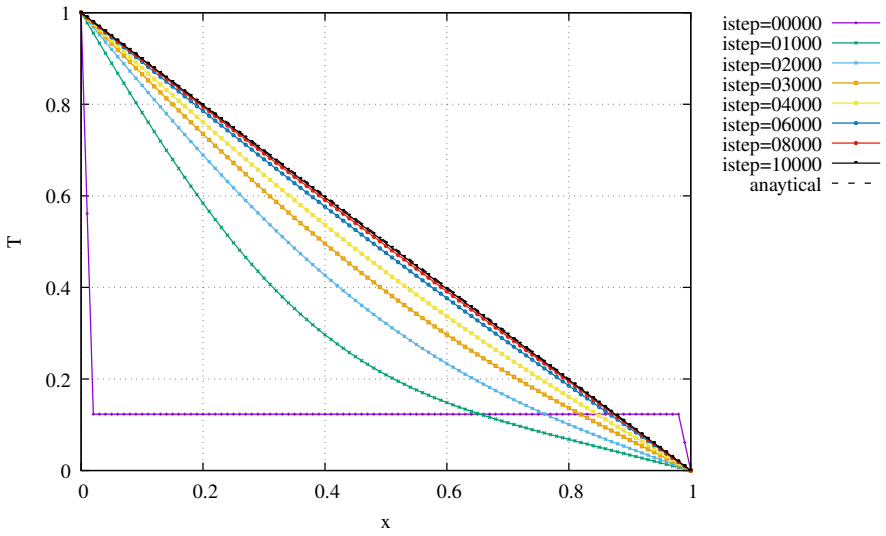
In our case the ODEs at hand are linear and not stiff, so that there is in fact no need for such high-order methods. I have verified that this works but since it yields virtually identical results to the 1st order scheme I am not showing any results obtained with this function in what follows.

option #1

If Dirichlet boundary conditions are prescribed at both ends of the domain, then the temperature is constant there in time and then $\frac{\partial T_i}{\partial t} = 0$ there. This translates as follows:

```
def F(t,T,kappa):
    dT_dt=np.zeros(nnx,dtype=np.float64)
    for i in range(0,nnx):
        if i==0:
            dT_dt[i]=0
        elif i==nnx-1:
            dT_dt[i]=0
        else:
            dT_dt[i]=kappa*(T[i+1]-2*T[i]+T[i-1])/h**2
    return dT_dt
```

The time evolution of the temperature is shown hereunder and we find that it converges to the expected analytical solution $t(x) = 1 - x$.



option #2

If we now assume a Dirichlet boundary condition at $x = 0$ and a Neumann boundary condition on the right (i.e. $\partial T/\partial x = 0$, or no heat flux), we must look closer at what is going on on this node. We start by writing the second-order derivative as follows (fully backward, since a standard centered stencil would involve a node to the right of the node under consideration but there is none):

$$\frac{\partial T_i}{\partial t} = \kappa \frac{T_i - 2T_{i-1} + T_{i-2}}{h^2}$$

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

On this node the boundary condition writes (again, a backward formulation is necessary)

$$\frac{T_i - T_{i-1}}{h} = 0$$

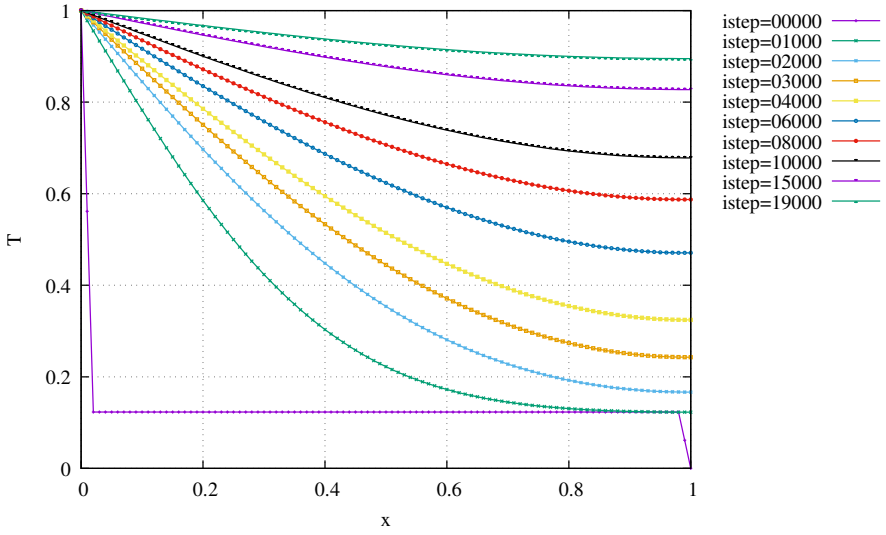
which, when incorporated in the equation above, leads to:

$$\frac{\partial T_i}{\partial t} = \kappa \frac{(T_i - T_{i-1}) - T_{i-1} + T_{i-2}}{h^2} = \kappa \frac{-T_{i-1} + T_{i-2}}{h^2}$$

The function $\vec{\mathcal{F}}$ then writes

```
def F(t,T,kappa):
    dT_dt=np.zeros(nnx,dtype=np.float64)
    for i in range(0,nnx):
        if i==0:
            dT_dt[i]=0
        elif i==nnx-1:
            dT_dt[i]=kappa*(-T[i-1]+T[i-2])/h**2
        else:
            dT_dt[i]=kappa*(T[i+1]-2*T[i]+T[i-1])/h**2
    return dT_dt
```

The steady-state solution is actually $T(x) = 1$ and we find that our solution slowly converges towards it while maintaining a zero derivative on the right side of the domain.



Stone 158

Solving the Stokes Eqs. with FDM in 2D



python_codes/fieldstone_158/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_158`

Before reading any further please read carefully Section 6.12.

Implementation

The domain is a Cartesian box of dimensions $L_x \times L_y$. It is discretised by means of $nnx \times nny$ nodes (so-called background mesh forming the cells). We then define:

```
ncellx=nnx-1
ncelly=nny-1
ncell=ncellx*ncelly
Nb=nnx*nny          # background mesh
Nu=nnx*ncelly       # u-nodes
Nv=ncellx*nny       # v-nodes
Np=ncellx*ncelly    # p-nodes
N=Nu+Nv+Np          # total nb of unknowns
```

For the 4×3 cell mesh of Section 6.12 we have

```
nnx= 6
nny= 5
Nb= 30
Nu= 24
Nv= 25
Np= 20
N= 69
ncell= 20
```

Also, because we will need the cell dimensions and their inverse a lot thoughout the code we define:

```
hx=Lx/ncellx
hy=Ly/ncelly
hhx=1/hx
hhy=1/hy
```

We must then define the δ_{bc} parameter that controls whether boundaries are free slip or no slip (see Section 6.12.8). We here assign one δ_{bc} parameter per side of the domain and arbitrarily choose that free slip is default:

```
bottom_no_slip=False
top_no_slip=False
left_no_slip=False
right_no_slip=False

if bottom_no_slip:
    delta_bc_bottom=-1
else:
    delta_bc_bottom=+1

if top_no_slip:
    delta_bc_top=-1
else:
    delta_bc_top=+1

if left_no_slip:
    delta_bc_left=-1
else:
    delta_bc_left=+1

if right_no_slip:
    delta_bc_right=-1
else:
    delta_bc_right=+1
```

We store the coordinates of the background nodes (where density and viscosity live) in arrays xb,yb, u nodes in arrays xu,yu, the coordinates of the v nodes in arrays xv,yv, and the coordinates of the p nodes in arrays xp,yp.

```
xu=np.zeros(Nu,dtype=np.float64)
yu=np.zeros(Nu,dtype=np.float64)
u=np.zeros(Nu,dtype=np.float64)
...
xv=np.zeros(Nv,dtype=np.float64)
yv=np.zeros(Nv,dtype=np.float64)
v=np.zeros(Nv,dtype=np.float64)
...
xp=np.zeros(Np,dtype=np.float64)
yp=np.zeros(Np,dtype=np.float64)
p=np.zeros(Np,dtype=np.float64)
```

The code first loops over all u-nodes, then all v-nodes and then p-nodes. For each node it establishes the index of all the required stencil neighbouring points. These are printed when the code is ran in debug mode. For example we will read:

```
u node # 4
ii,jj = 4 0
index_eta_nw= 9
index_eta_n = 10
index_eta_ne= 11
index_eta_sw= 3
index_eta_s = 4
index_eta_se= 5
index_p_w = 3
index_p_e = 4
index_rho_n = 10
index_rho_s = 4
index_v_sw = 3
index_v_se = 4
index_v_nw = 8
index_v_ne = 9
index_u_n = 10
index_u_s = -2
index_u_w = 3
index_u_e = 5
```

```
v node # 7
ii,jj = 2 1
index_eta_sw= 2
index_eta_w = 8
index_eta_nw= 14
index_eta_se= 3
index_eta_e = 9
index_eta_ne= 15
index_p_s = 2
index_p_n = 7
index_rho_w = 8
index_rho_e = 9
index_v_w = 6
index_v_e = 8
index_v_s = 2
index_v_n = 12
index_u_sw = 2
index_u_se = 3
index_u_nw = 8
index_u_ne = 9
```

```
p node # 10
ii,jj = 0 2
index_u_w 12
index_u_e 13
index_v_s 10
index_v_n 15
```

These printouts should obviously match the numbering of the 4×3 cell mesh of Section 6.12. This step is crucial because these indices will be used to know where to add coefficients in the final assembled matrix. If any of them is wrong the generated matrix is incorrect and therefore the linear system might not even have a solution.

We then start from Eqs. (6.131),(6.132) and (6.133) (the modified versions of these equations containing the δ_{bc} term do not pose any kind of additional difficulty so we focus on the standard equations only):

$$
\begin{aligned}
&\left(\frac{\eta_{\mathtt{n}}}{h_y^2}\right) u_{\mathtt{n}} + \left(\frac{2\eta_{\mathtt{e}}}{h_x^2}\right) u_{\mathtt{e}} + \left(\frac{2\eta_{\mathtt{w}}}{h_x^2}\right) u_{\mathtt{w}} + \left(\frac{\eta_{\mathtt{s}}}{h_y^2}\right) u_{\mathtt{s}} + \left(-\frac{2\eta_{\mathtt{e}}}{h_x^2} - \frac{2\eta_{\mathtt{w}}}{h_x^2} - \frac{\eta_{\mathtt{n}}}{h_y^2} - \frac{\eta_{\mathtt{s}}}{h_y^2}\right) u_{\otimes} \\
&+ \left(\frac{\eta_{\mathtt{n}}}{h_x h_y}\right) v_{\mathtt{ne}} + \left(-\frac{\eta_{\mathtt{n}}}{h_x h_y}\right) v_{\mathtt{nw}} + \left(-\frac{\eta_{\mathtt{s}}}{h_x h_y}\right) v_{\mathtt{se}} + \left(\frac{\eta_{\mathtt{s}}}{h_x h_y}\right) v_{\mathtt{sw}} - \frac{1}{h_x} p_{\mathtt{e}} + \frac{1}{h_x} p_{\mathtt{w}} = -\frac{\rho_{\mathtt{n}} + \rho_{\mathtt{s}}}{2} g_x \\
&\left(\frac{2\eta_{\mathtt{n}}}{h_y^2}\right) v_{\mathtt{n}} + \left(\frac{\eta_{\mathtt{e}}}{h_x^2}\right) v_{\mathtt{e}} + \left(\frac{\eta_{\mathtt{w}}}{h_x^2}\right) v_{\mathtt{w}} + \left(\frac{2\eta_{\mathtt{s}}}{h_y^2}\right) v_{\mathtt{s}} + \left(-\frac{\eta_{\mathtt{e}}}{h_x^2} - \frac{\eta_{\mathtt{w}}}{h_x^2} - \frac{2\eta_{\mathtt{n}}}{h_y^2} - \frac{2\eta_{\mathtt{s}}}{h_y^2}\right) v_{\otimes} \\
&+ \left(\frac{\eta_{\mathtt{e}}}{h_x h_y}\right) u_{\mathtt{ne}} + \left(-\frac{\eta_{\mathtt{e}}}{h_x h_y}\right) u_{\mathtt{se}} + \left(-\frac{\eta_{\mathtt{w}}}{h_x h_y}\right) u_{\mathtt{nw}} + \left(\frac{\eta_{\mathtt{w}}}{h_x h_y}\right) u_{\mathtt{sw}} - \frac{1}{h_y} p_{\mathtt{n}} + \frac{1}{h_y} p_{\mathtt{s}} = -\frac{\rho_{\mathtt{e}} + \rho_{\mathtt{w}}}{2} g_y \\
&\frac{u_{\mathtt{e}} - u_{\mathtt{w}}}{h_x} + \frac{v_{\mathtt{n}} - v_{\mathtt{s}}}{h_y} = 0
\end{aligned}
$$

We can define

$$
\begin{aligned}
\eta_{\mathtt{n},yy} &= \frac{\eta_{\mathtt{n}}}{h_y^2} & \eta_{\mathtt{n},xy} &= \frac{\eta_{\mathtt{n}}}{h_x h_y} \\
\eta_{\mathtt{s},yy} &= \frac{\eta_{\mathtt{s}}}{h_y^2} & \eta_{\mathtt{s},xy} &= \frac{\eta_{\mathtt{s}}}{h_x h_y} \\
\eta_{\mathtt{e},xx} &= \frac{\eta_{\mathtt{e}}}{h_x^2} & \eta_{\mathtt{e},xy} &= \frac{\eta_{\mathtt{e}}}{h_x h_y} \\
\eta_{\mathtt{w},xx} &= \frac{\eta_{\mathtt{w}}}{h_x^2} & \eta_{\mathtt{w},xy} &= \frac{\eta_{\mathtt{w}}}{h_x h_y} \\
\rho_{\mathtt{ns}} &= \frac{1}{2}(\rho_{\mathtt{n}} + \rho_{\mathtt{s}}) & \rho_{\mathtt{ew}} &= \frac{1}{2}(\rho_{\mathtt{e}} + \rho_{\mathtt{w}}) \\
\tilde{h}_x &= h_x^{-1} & \tilde{h}_y &= h_y^{-1}
\end{aligned}
$$

which translates as follows in the code:

```
eta_n_yy=eta_n/hy**2  ;  eta_n_xy=eta_n/hx/hy
eta_e_xx=eta_e/hx**2
eta_w_xx=eta_w/hx**2
eta_s_yy=eta_s/hy**2  ;  eta_s_xy=eta_s/hx/hy
...
eta_n_yy=eta_n/hy**2
eta_s_yy=eta_s/hy**2
eta_e_xx=eta_e/hx**2  ;  eta_e_xy=eta_e/hx/hy
eta_w_xx=eta_w/hx**2  ;  eta_w_xy=eta_w/hx/hy
```

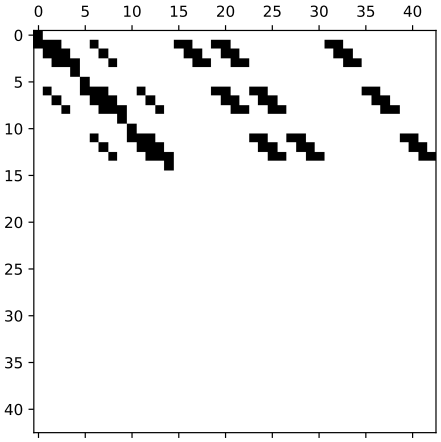
so that we can write the three stencil equations as follows:

$$
\begin{aligned}
&(\eta_{\mathtt{n},yy})\, u_{\mathtt{n}} + (2\eta_{\mathtt{e},xx})\, u_{\mathtt{e}} + (2\eta_{\mathtt{w},xx})\, u_{\mathtt{w}} + (\eta_{\mathtt{s},yy})\, u_{\mathtt{s}} + (-2\eta_{\mathtt{e},xx} - 2\eta_{\mathtt{w},xx} - \eta_{\mathtt{n},yy} - \eta_{\mathtt{s},yy})\, u_{\otimes} \\
&+ (\eta_{\mathtt{n},xy})\, v_{\mathtt{ne}} + (-\eta_{\mathtt{n},xy})\, v_{\mathtt{nw}} + (-\eta_{\mathtt{s},xy})\, v_{\mathtt{se}} + (\eta_{\mathtt{s},xy})\, v_{\mathtt{sw}} + (-\tilde{h}_x) p_{\mathtt{e}} + (\tilde{h}_x) p_{\mathtt{w}} = -\rho_{\mathtt{ns}} g_x \\
&(2\eta_{\mathtt{n},yy})\, v_{\mathtt{n}} + (\eta_{\mathtt{e},xx})\, v_{\mathtt{e}} + (\eta_{\mathtt{w},xx})\, v_{\mathtt{w}} + (2\eta_{\mathtt{s},yy})\, v_{\mathtt{s}} + (-\eta_{\mathtt{e},xx} - \eta_{\mathtt{w},xx} - 2\eta_{\mathtt{n},yy} - 2\eta_{\mathtt{s},yy})\, v_{\otimes} \\
&+ (\eta_{\mathtt{e},xy})\, u_{\mathtt{ne}} + (-\eta_{\mathtt{e},xy})\, u_{\mathtt{se}} + (-\eta_{\mathtt{w},xy})\, u_{\mathtt{nw}} + (\eta_{\mathtt{w},xy})\, u_{\mathtt{sw}} + (-\tilde{h}_y) p_{\mathtt{n}} + (\tilde{h}_y) p_{\mathtt{s}} = -\rho_{\mathtt{ew}} g_y \\
&(\tilde{h}_x) u_{\mathtt{e}} - (\tilde{h}_x) u_{\mathtt{w}} + (\tilde{h}_y) v_{\mathtt{n}} - (\tilde{h}_y) v_{\mathtt{s}} = 0
\end{aligned}
$$

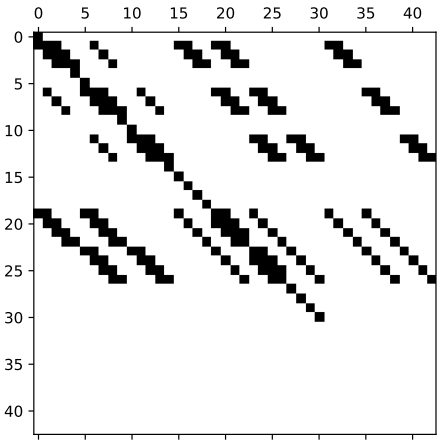
The logic of the code with regards to matrix and rhs assembly is as follows:

```
#############################################################
# loop over all u nodes
#############################################################
for i in range(0,Nu):
    if left[i]: # u node on left boundary
       A[...,...]=...
       b[...]=...
    elif right[i]: # u node on right boundary
       A[...,...]=...
       b[...]=...
    else:
       ...
       if jj==0: # bottom row, ghosts nodes used
          A[...,...]=...
          b[...]=...
       elif jj==ncelly-1: # top row, ghosts nodes used
          A[...,...]=...
          b[...]=...
       else:
          A[...,...]=...
          b[...]=...

#############################################################
# loop over all v nodes
#############################################################
for i in range(0,Nv):
    if bottom[i]: # v node on bottom boundary
       A[...,...]=...
       b[...]=...
    elif top[i]: # v node on top boundary
       A[...,...]=...
       b[...]=...
    else:
       if ii==0: # left column, ghosts nodes used
          A[...,...]=...
          b[...]=...
       elif ii==ncellx-1: # right column, ghosts nodes used
          A[...,...]=...
          b[...]=...
       else:
          A[...,...]=...
          b[...]=...

#############################################################
# loop over all p nodes
#############################################################
for i in range(0,Np):
    A[...,...]=...
    b[...]=...
```

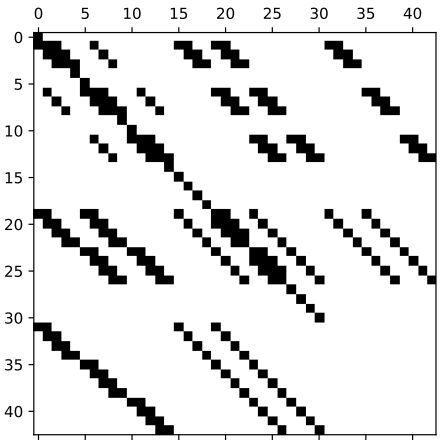
After the loop on u nodes, the first N_u lines have been filled and the matrix non-zero pattern is as follows:



After the loop on v nodes, the matrix non-zero pattern is as follows:



And finally after the loop over cells/pressure nodes:



As predicted in Eq. (6.134) we recover a matrix that is composed of 9 blocks, one being entirely zero.

Viscosity averaging

A single parameter allows to control the type of viscosity averaging that is used (see Section 6.12.7):

```
avrg=2
```

with 1: arithmeric, 2: geometric, 3: harmonic.

Matrix block scaling

Following Section 6.12.10 we define:

```
eta_ref=1
L_ref=min(hx,hy)
```

These parameters will be used while building the matrix and also after solving the linear system to scale back the pressure.

Matrix storage

At first one could define the matrix and rhs arrays as follows:

```
A=np.zeros((N,N),dtype=np.float64)
b=np.zeros(N,dtype=np.float64)
```

However, this is definitely not optimal since we know that the matrix is *very* sparse and potentially very large, so that this would lead to large memory requirements for no good reason. Instead the matrix is declared as follows[1] (the rhs is unchanged):

```
A=sps.lil_matrix((N,N),dtype=np.float64)
```

Before solving the system the matrix storage needs to be converted to CSR format (see Section 15.6) as follows:

```
sol = sps.linalg.spsolve(sps.csr_matrix(A),b)
```

Pressure normalisation

For all experiments here we impose a zero average pressure after solving the linear system. The average pressure is then computed as follows:

$$< p >= \frac{1}{L_x L_y} \sum_e p_e V_e = \frac{1}{L_x L_y} \sum_e p_e h_x h_y = \frac{1}{L_x L_y} \sum_e p_e \frac{L_x}{ncellx} \frac{L_y}{ncelly} = \sum_e p_e \frac{1}{ncellx} \frac{1}{ncelly} = \frac{1}{ncell} \sum_e p_e$$

i.e.,

```
p_avrg=np.sum(p)/ncell
```

and then

```
p-=p_avrg
```

Benchmarking of strain rate calculations

If we set

```
u[:]=1
v[:]=1
```

we indeed recover all components of the strain rate being zero.

If we set

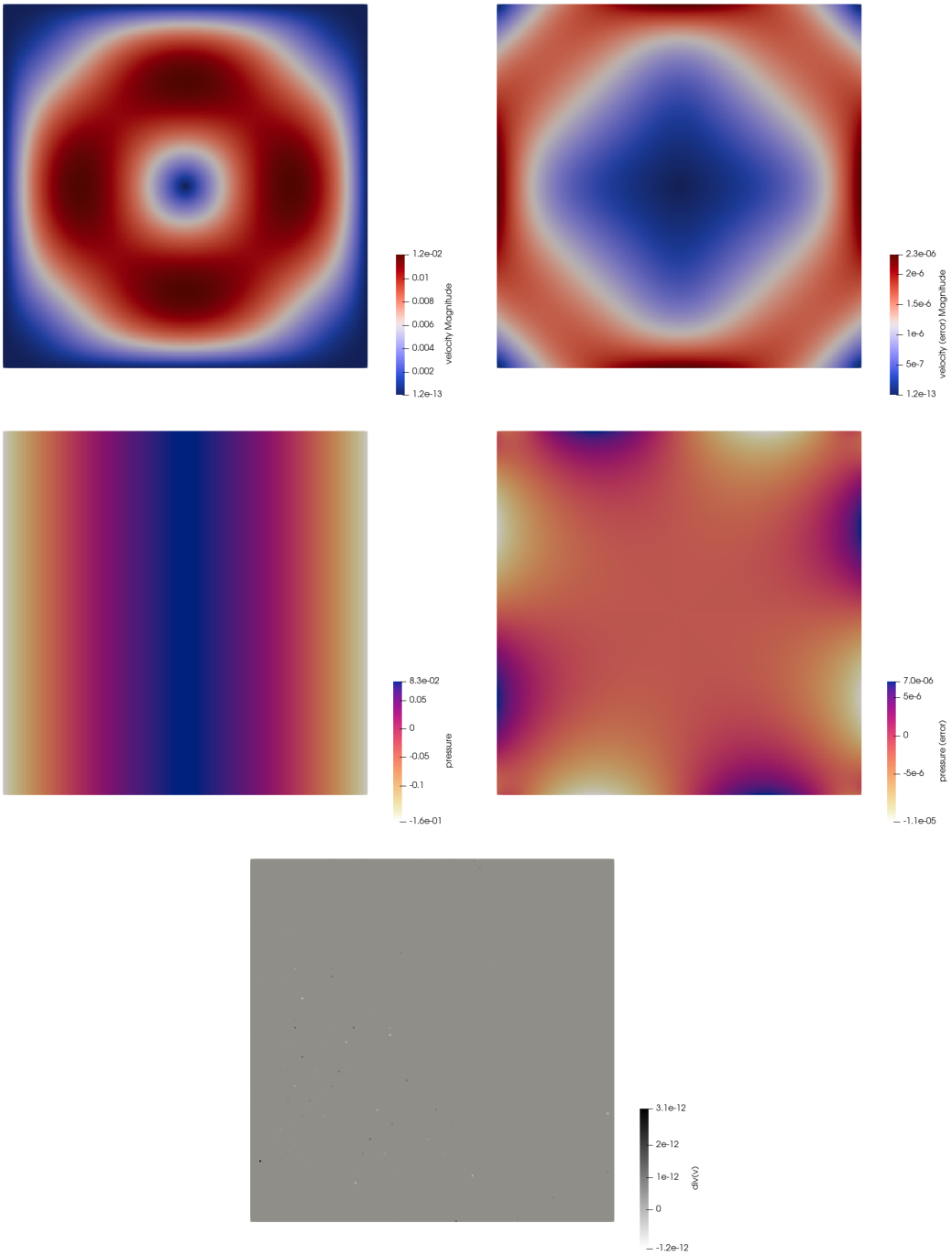
```
u[:]=xu[:]
v[:]=yv[:]
```

then we recover $\dot{\varepsilon}_{xx} = \dot{\varepsilon}_{yy} = 1$ and $\dot{\varepsilon}_{xy} = 0$ everywhere in the domain.

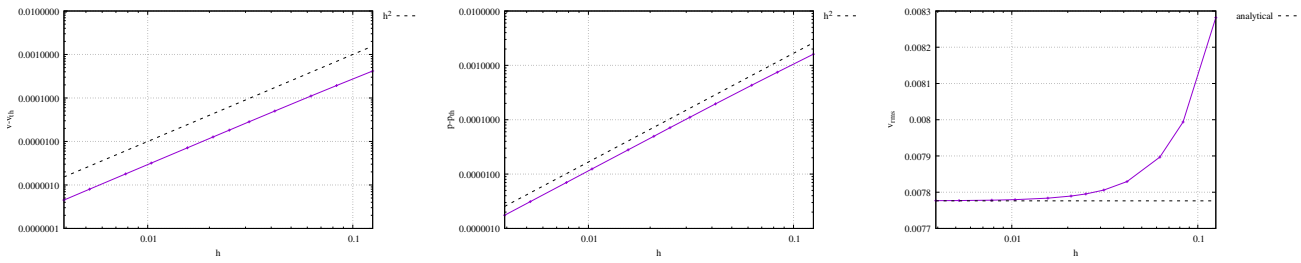
[1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.lil_matrix.html

Experiment 4: Donea & Huerta manufacture solution

This manufactured solution is described in Section 18.2.



Obtained on 200 × 200 mesh.

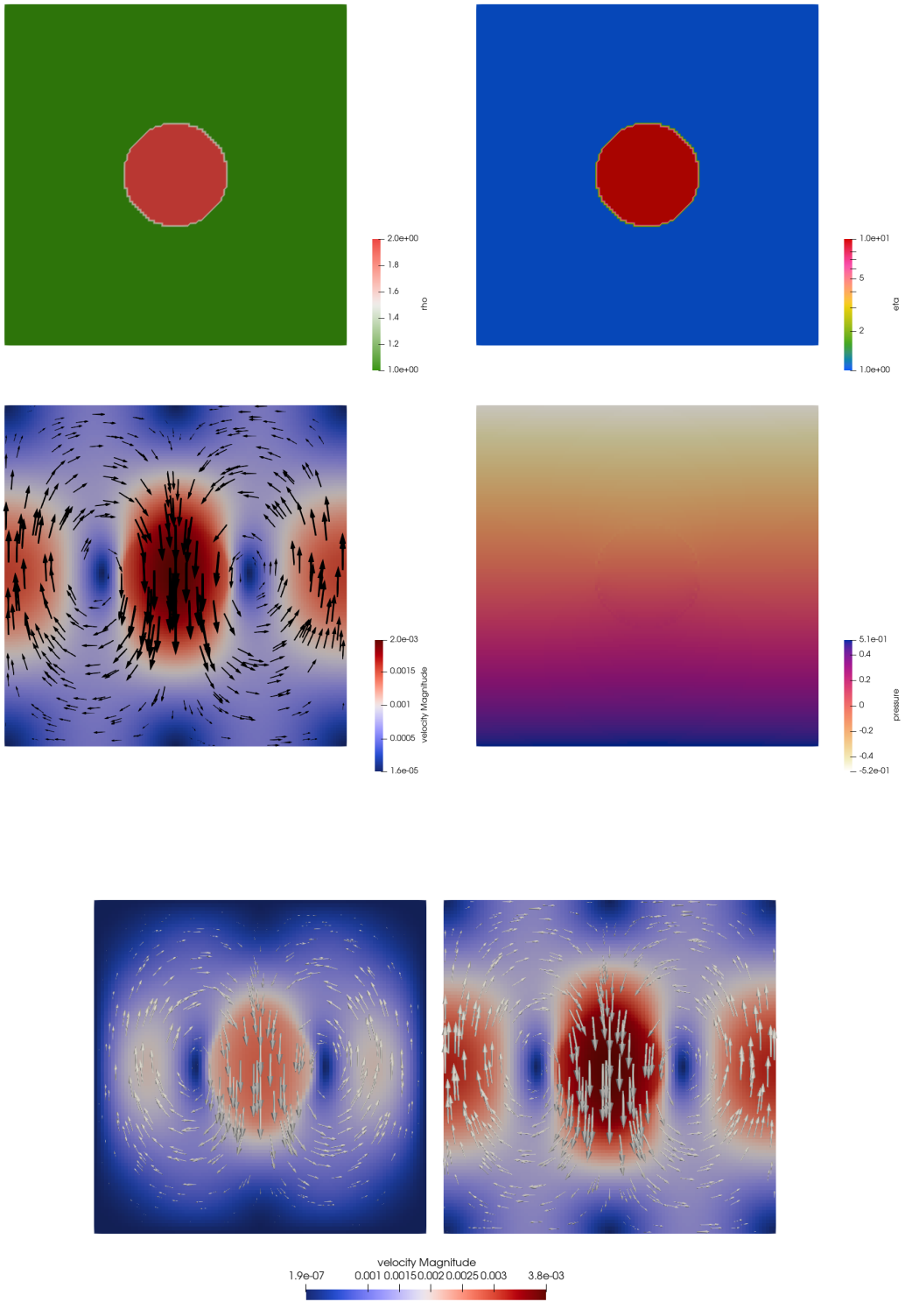


Velocity error, pressure error and v_{rms} as a function of resolution.

We recover a second-order convergence for both velocity and pressure errors.

Experiment 1: the sinker

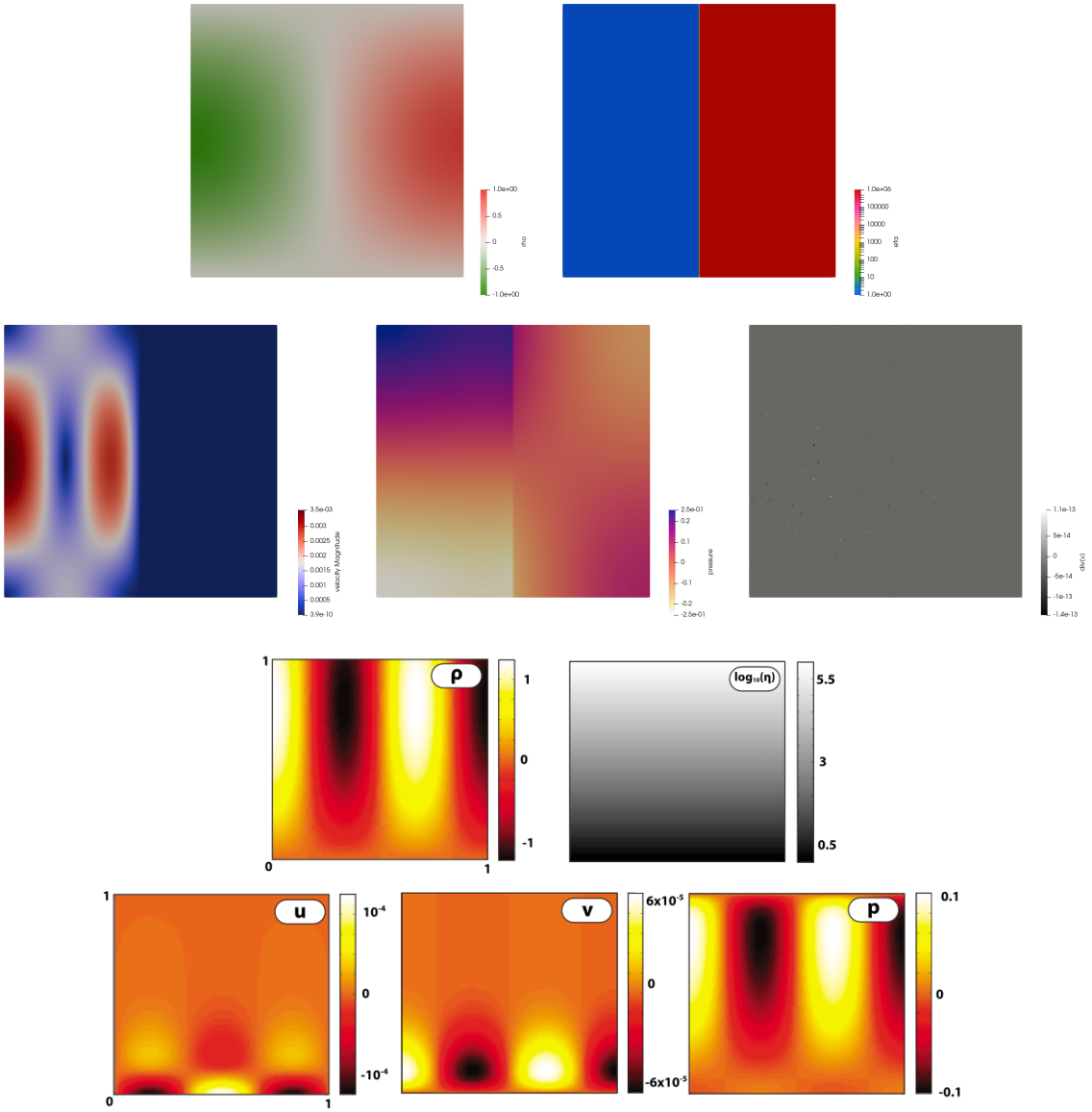
This is a basic sanity check. Boundary conditions are either free slip or no slip. The domain is the unit square filled with a Newtonian fluid characterised by $\rho = 1$ and $\eta = 1$. There is a sphere in the middle of radius 0.15 with $\rho = 2$ and $\eta = 10$.



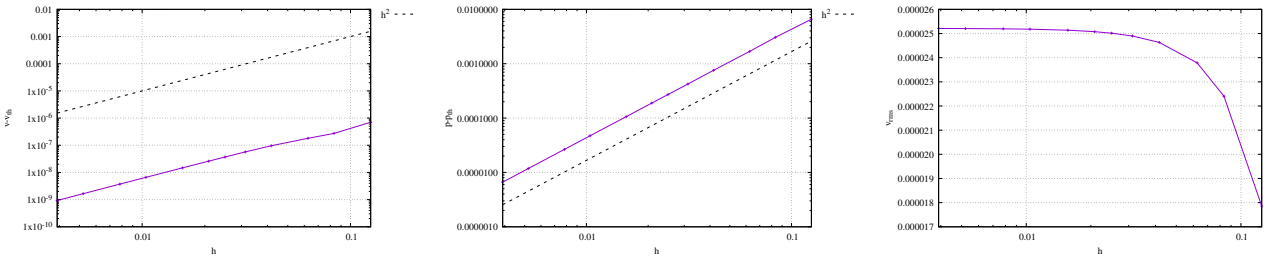
Velocity field: no slip b.c. (left) and free slip b.c. (right).

Experiment 2: SolKz

The experiment is described in Section 18.4.



Taken from Duretz, May, Gerya, and Tackley [97] (2011).

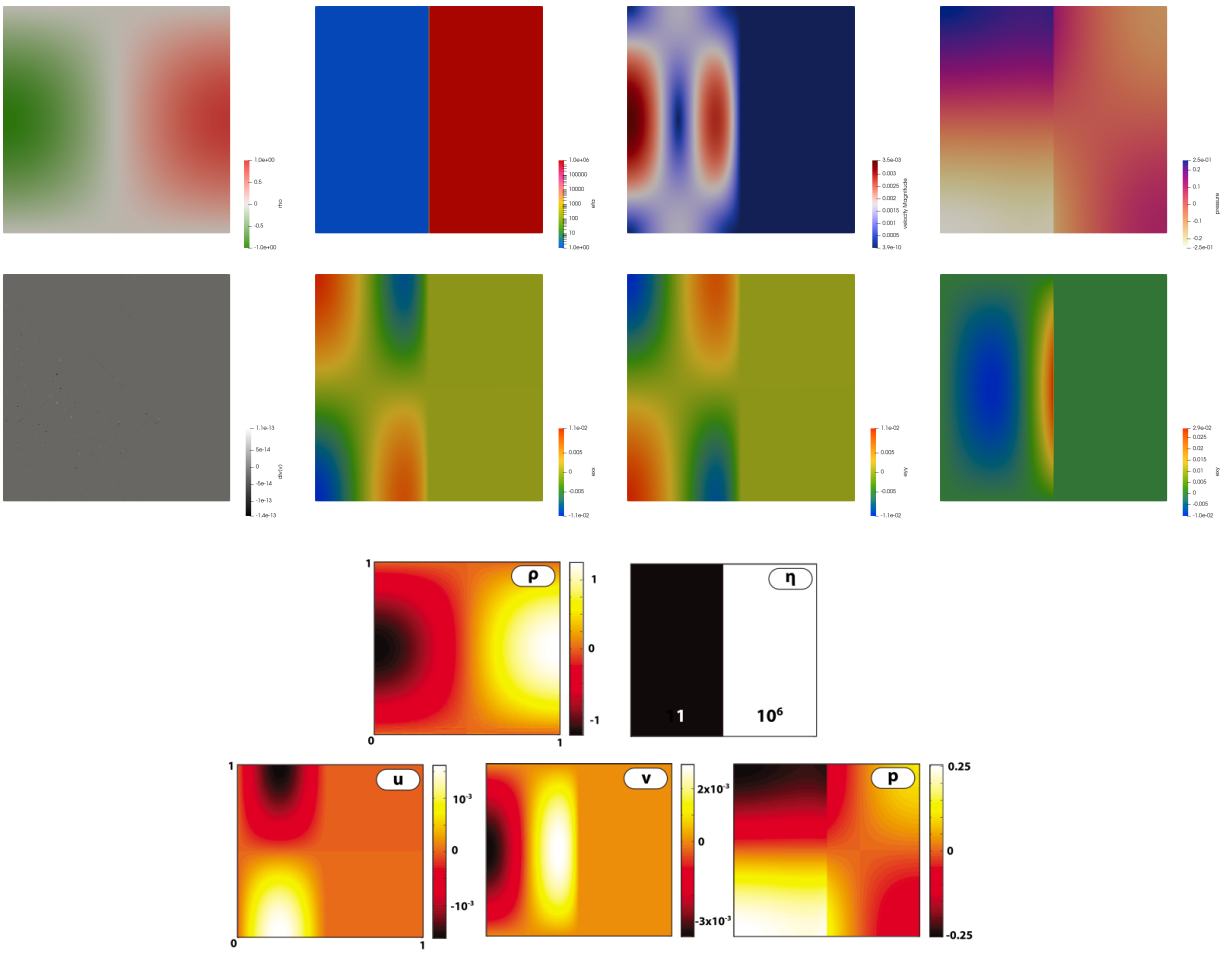


Velocity error, pressure error and ν_{rms} as a function of resolution.

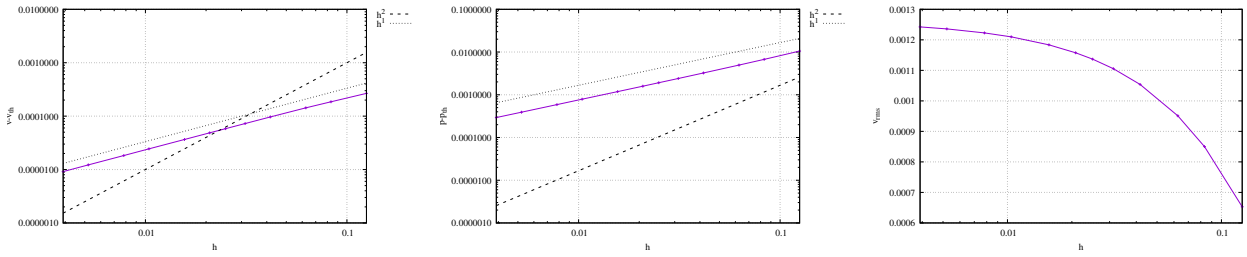
We recover a second-order convergence for both velocity and pressure errors.

Experiment 3: SolCx

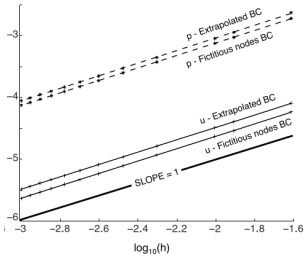
The experiment is described in Section 18.3.



Taken from Duretz, May, Gerya, and Tackley [97] (2011).



Velocity error, pressure error and v_{rms} as a function of resolution.



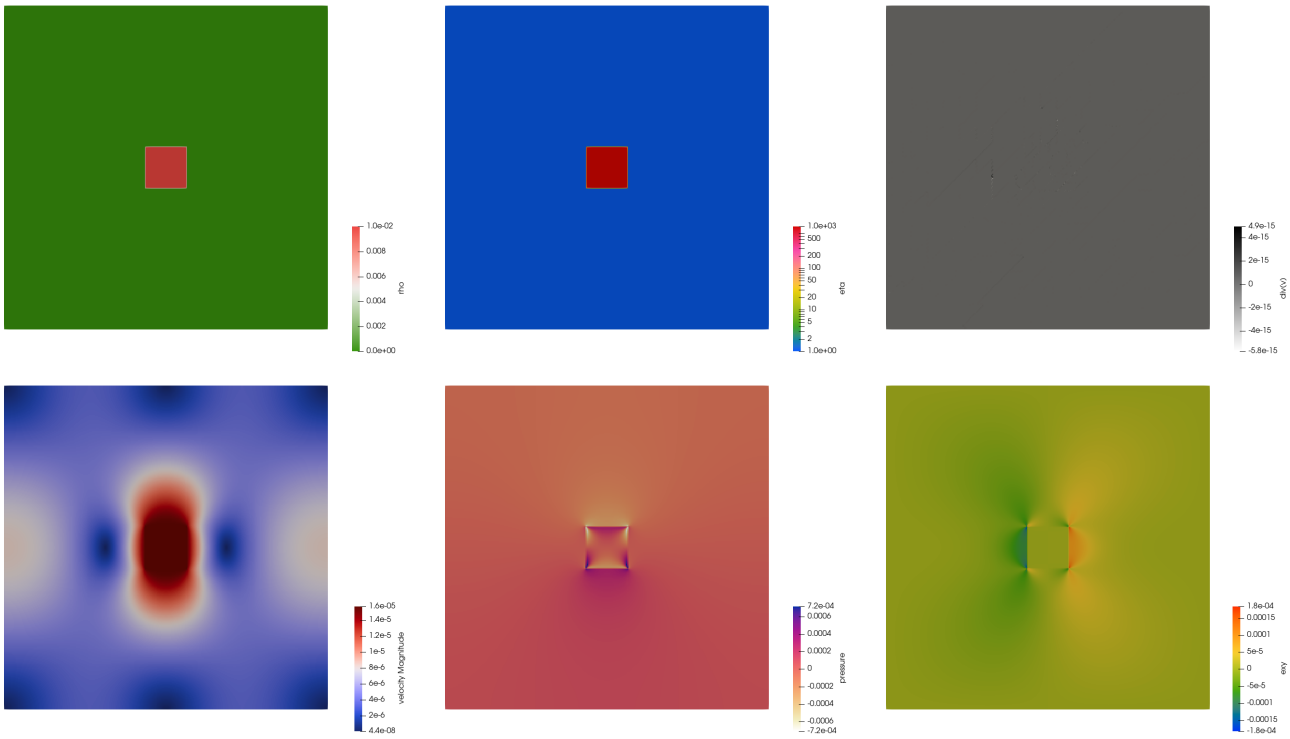
Taken from Fig. 7 of Duretz, May, Gerya, and Tackley [97] (2011).

Due to the viscosity jump we only recover a first-order convergence for both velocity and pressure errors.

Experiment 5: sinking block

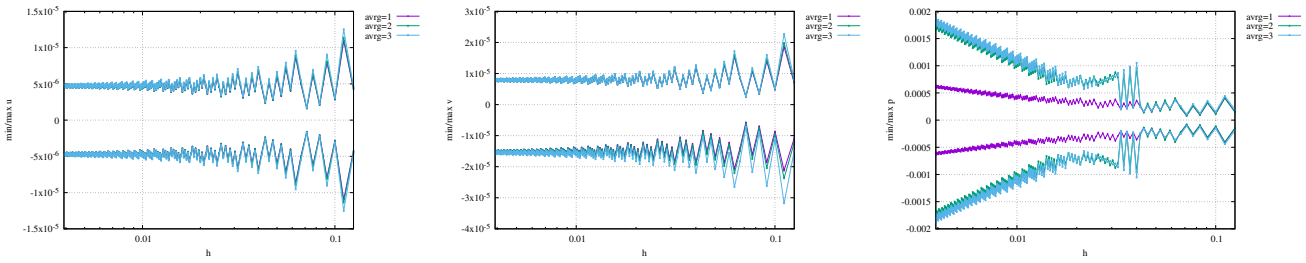
This experiment is described in Section 19.22. The domain is a unit square. Fluids are such that $\rho_1 = 1$, $\eta_1 = 1$ and $\rho_2 = 1.01$, $\eta_2 = 1000$. Boundary conditions are either free slip or no slip on all sides. Pressure is normalised so that the volume average is zero. Gravity points downwards with $|\vec{g}| = 1$. Profile measurements are carried out on a vertical line passing through the middle of the block.

The following results are actually obtained for a case with reduced densities, i.e. the backgroud density profile has been removed so that $\rho_1 = 0$ and $\rho_2 = 0.01$. The pressure is then the dynamic pressure. Arithmetic averaging is used.



Obtained on 320×320 mesh.

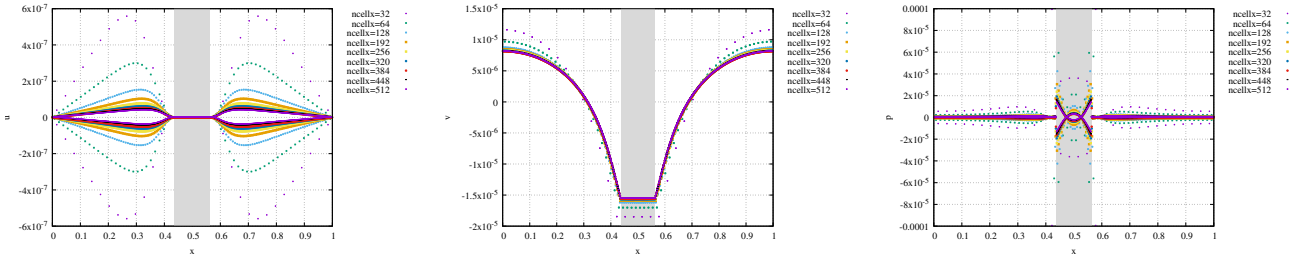
We can look at the evolution of the min/max statistics as a function of resolution. Using the `script_exp5` we can run the code from resolution 9^2 to 257^2 nodes. In many cases the sides of the block do not align with the cell boundaries and this explains why the line is not monotonous in convergence:



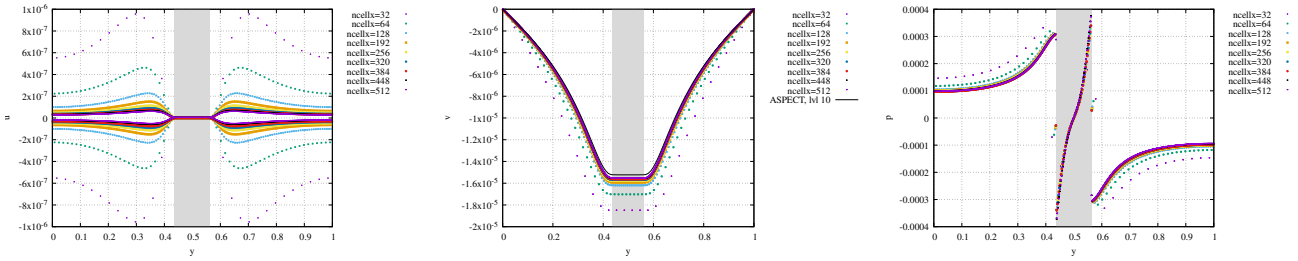
min/max values of the u, v, p fields with the three different viscosity averagings.

We now wish to plot the u, v, p fields on two lines passing through the middle of the block (one vertical, one horizontal). This poses a problem since not all three variables are available on such lines. I have therefore chosen to consider the mid-cell projected fields and export the values on each side of each line.

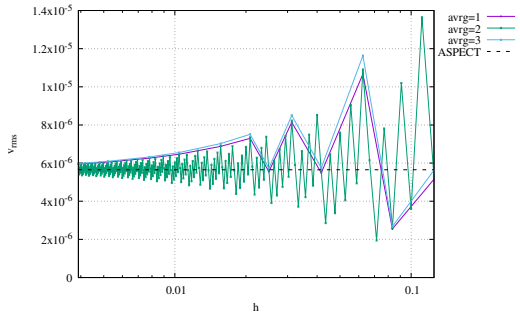
Horizontal line $y = 1/2$:



Vertical line $x = 1/2$:



Finally we can plot the root mean square velocity as a function of the mesh size:



Stone 159

Computing plastic yield envelopes



python_codes/fieldstone_159/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_159`

The goal of this STONE is to explore, check and plot the yield envelopes of the main plastic yield criteria. We will here focus on:

- von Mises
- Tresca
- Mohr-Coulomb
- Drucker-Prager (all three variants)
- Griffith-Murrell

All the models are parameterized by a cohesion c and an angle of friction ϕ.

The code works as follows: We will explore a stress space $\{\sigma_1, \sigma_2, \sigma_3\} \in [-\sigma_{max}, \sigma_{max}]^3$. For each point inside this domain, we compute the 1st, 2nd and 3rd moment invariants, i.e.

$$\mathcal{I}_1(\boldsymbol{\sigma}), \qquad \mathcal{I}_2(\boldsymbol{\tau}), \qquad \mathcal{I}_3(\boldsymbol{\tau}), \qquad \theta_{\rm L} \in [-\pi/6, \pi/6]$$

At each point we will then evaluate the yield function $\mathcal{F}$. If `make_vtu` is True, a (very large) vtu file will be produced (in this case nnx should not exceed 200). The code exports the point that fulfill $\mathcal{F} = 0$ within 10 kPa. which allows to trace the yield envelopes. The same can be achieved by opening the vtu file and looking at zero isocontours of each $\mathcal{F}$ field.

We set $c = 20$ MPa and $\phi = 20°$. After trial an error, we set $\sigma_{max} = 60 MPa$.

The first invariant of the stress tensor is given by

$$\mathcal{I}_1(\boldsymbol{\sigma}) = \sigma_{xx} + \sigma_{yy} + \sigma_{zz} = \sigma_1 + \sigma_2 + \sigma_3$$

which translates into

```
I1sig=x+y+z
```

The second invariant of the deviatoric stress tensor is given by

$$\mathcal{I}_2(\boldsymbol{\tau}) = \frac{1}{6}\left[(\sigma_{xx}-\sigma_{yy})^2+(\sigma_{yy}-\sigma_{zz})^2+(\sigma_{xx}-\sigma_{zz})^2\right]+\sigma_{xy}^2+\sigma_{xz}^2+\sigma_{yz}^2$$

In the case of a diagonal stress tensor then

$$\mathcal{I}_2(\boldsymbol{\tau}) = \frac{1}{6}\left[(\sigma_1-\sigma_2)^2+(\sigma_2-\sigma_3)^2+(\sigma_1-\sigma_3)^2\right]$$

which translates into

```
I2tau=((x-y)**2+(x-z)**2+(y-z)**2)/6.
sqrtI2tau=np.sqrt(I2tau)
```

The third invariant is given by

$$\mathcal{I}_3(\boldsymbol{\tau}) = \frac{1}{81}\left[(2\sigma_1-\sigma_2-\sigma_3)^3+(2\sigma_2-\sigma_1-\sigma_3)^3+(2\sigma_3-\sigma_1-\sigma_2)^3\right]$$

which translates into

```
I3tau=((2*x-y-z)**3+(2*y-x-z)**3+(2*z-x-y)**3)/81.
```

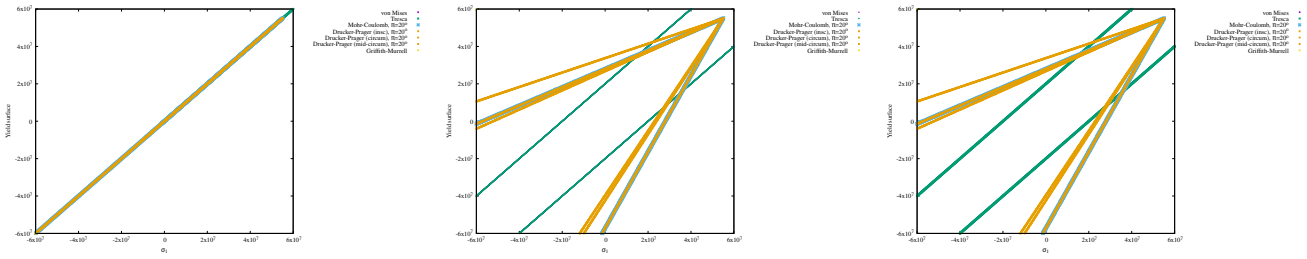
Finally the Lode angle is given by

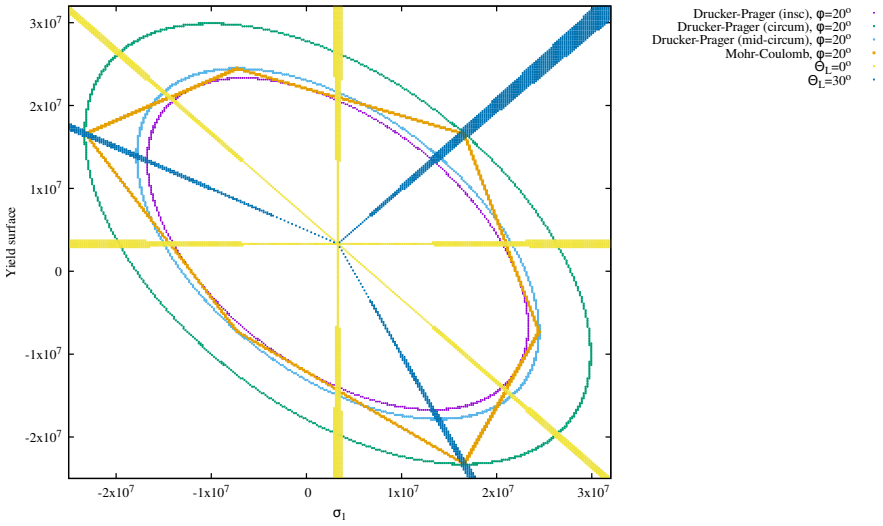
$$\theta_{\rm L} = \frac{1}{3}\sin^{-1}\left(-\frac{3\sqrt{3}}{2}\frac{\mathcal{I}_3(\boldsymbol{\tau})}{\mathcal{I}_2(\boldsymbol{\tau})^{3/2}}\right)$$

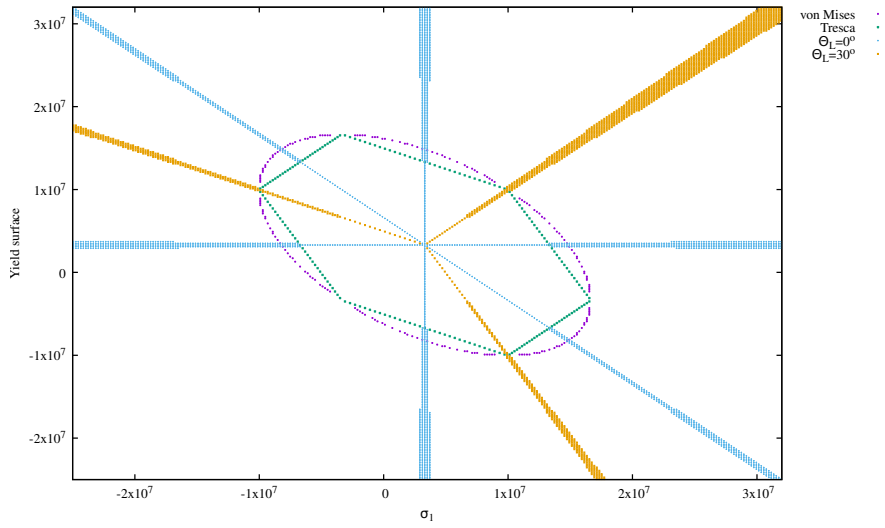
and is implemented as follows:

```
theta=-3*np.sqrt(3)/2*I3tau/sqrtI2tau**3
theta=min(theta,0.99999999)
theta=max(-0.99999999,theta)
theta=np.arcsin(theta)/3.
```

When ran at high resolution, one can extract the following plots:

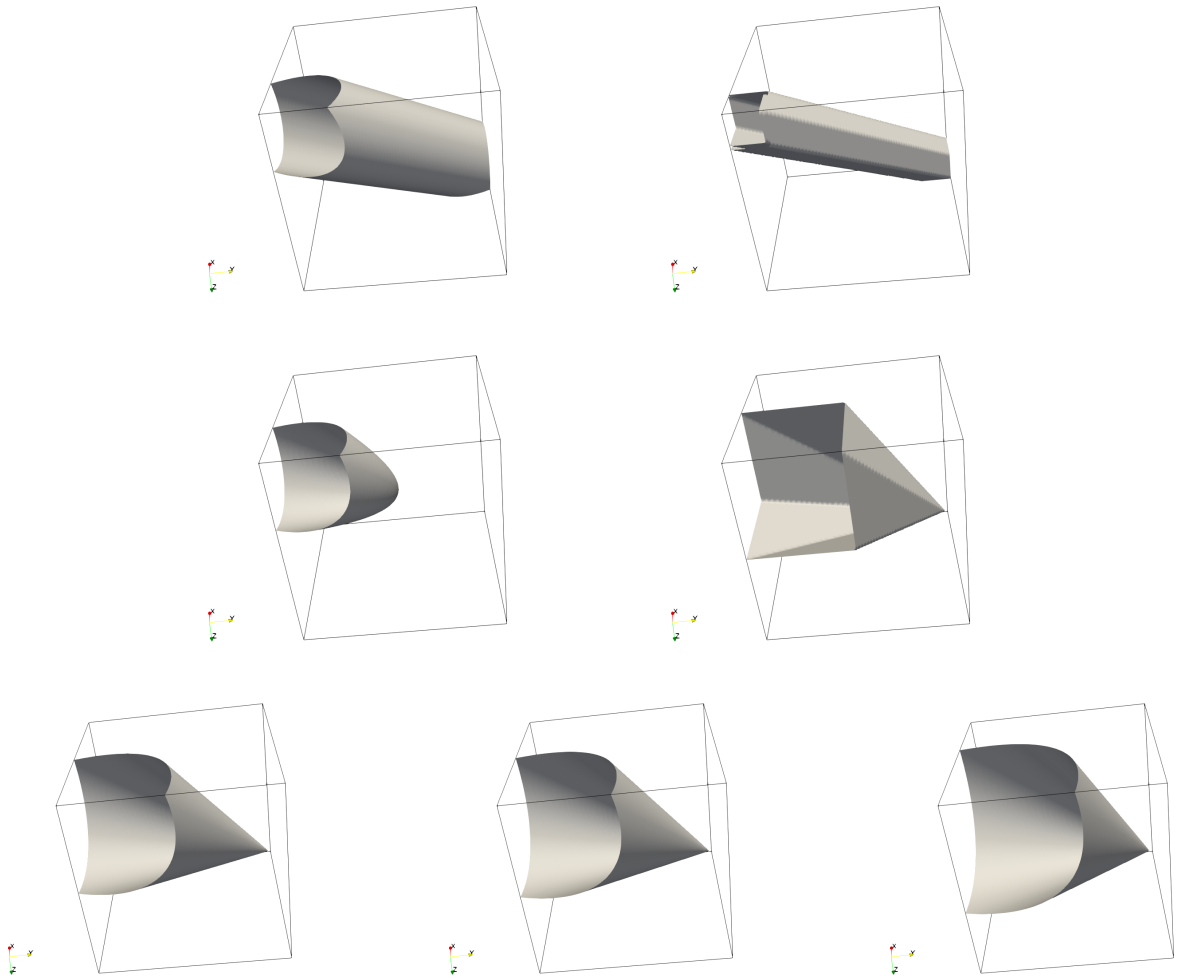






Obtained with nnx=512.

Using paraview on the solution.vtu file, and plotting zero-isocontours of each yield function:



Obtained with nnx=80.

not sure how to choose the parameter for GM, so after trial and error I took c/20. cGM influences the 'diameter' of the envelope PLOT!!!

Need to run high res again with GM, and add to gnuplot

export axis and p line to vtu !

Stone 160

A simple look at dynamic topography



python_codes/fieldstone_160/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_160

This stone was developed in collaboration with J.C. Afonso and C. Conrad.

Analytical exercise

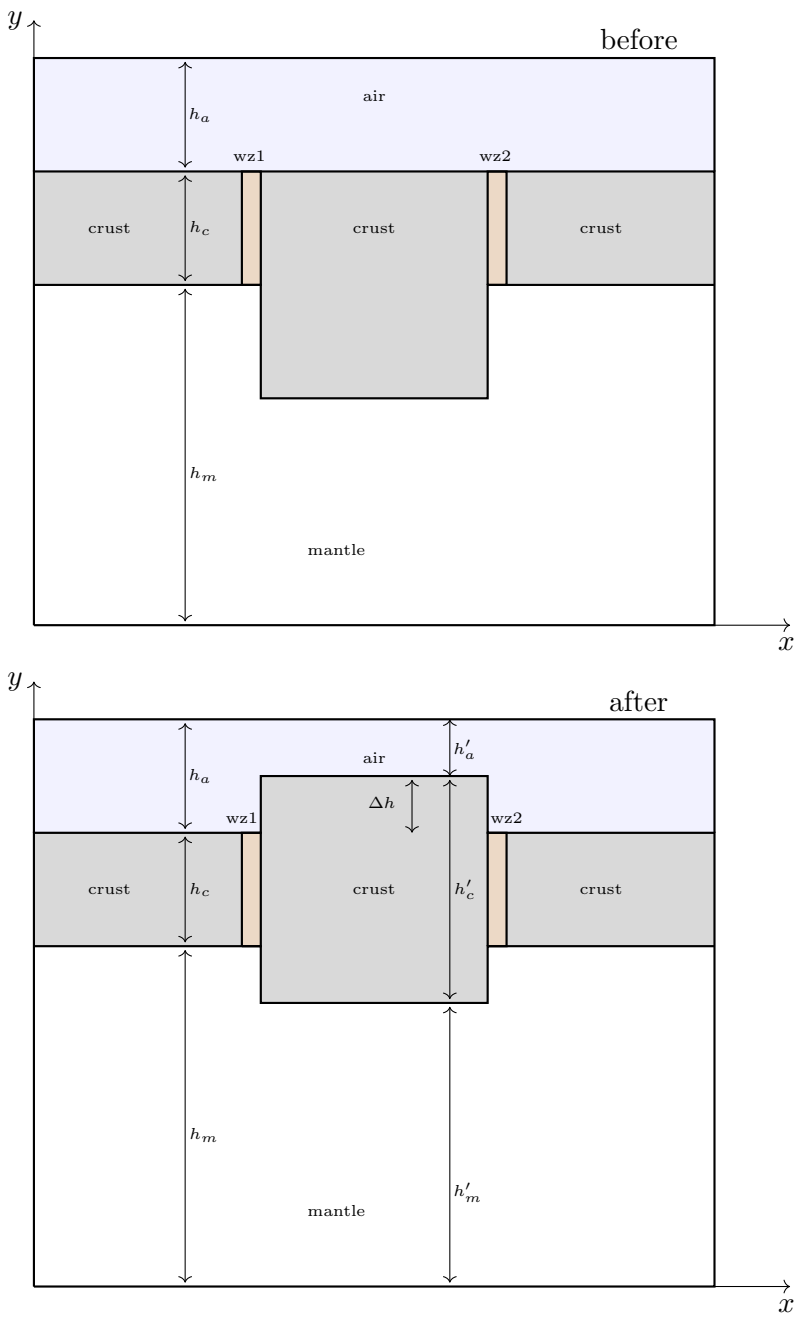
What follows is inspired by an experiment from an unpublished paper provided by J.C.

The domain is a Cartesian box of size $L_x = 120$ km, $L_y = 120$ km. There are four materials in the domain:

- crust: $\rho_c = 2300$ kg m^{-3} and $\eta_c = 10^{23}$ Pa s;
- mantle: $\rho_m = 3300$ kg m^{-3} and $\eta_m = 10^{21}$ Pa s;
- weak zones: $\rho_{wz} = \rho_c$ and $\eta_{wz} = 10^{-3}\eta_c$.
- air or water: $\rho_a = 0$ and $\eta_a << \eta_m$

The crust is 20 km thick but is thickened (40 km depth) in the middle 30 km. On each side we find weak zones as thick as the crust and 1 km wide. The air layer is also 20 km thick.

The thickened crust has a positive buoyancy with respect to the normal crust due to density difference, whose analytical value is found via a standard isostatic balance (i.e. equating the mass per unit area of two columns at the compensation level, here taken at the base of the domain).



h_c and h_m are the initial thicknesses of the crust and mantle. h'_c is the thickness of the thickened crust, and h'_m is the thickness of the mantle below the thickened crust at isostatic equilibrium. Drawing not to scale. wz1 and wz2 stand for 'weak zone' 1 & 2.

Note that these drawings are very conceptual: for example one would expect the weak zones to deform while the thickned crust is moving up. Also, between the two figures mantle volume and air volume are not conserved.

Clint commented:

> I would call this isostatic topography, not dynamic topography. The definition of dynamic topography is debated, but generally the debate is whether temperature-induced variations in density within the mantle lithosphere should be called dynamic topography or isostatic topography. I think pretty much universally crustal variations are referred to isostatic topography - so calling this dynamic topography here might be confusing to some. But I can see where you are coming from and this is an interesting and illustrative test. I would just suggest adding a note that this topography is well-understood, and it is usually considered isostatic topography.

In lecture notes by S. Zhong, I found:

Dynamic topography is that due to non-isostatic (i.e., non-crustal and non-lithospheric) effects, i.e., residual topography after correcting for isostatic effects or crustal and lithospheric contributions to the topography.

Dynamic topography is important because it tells us about the dynamics. It is also more difficult to get because some tricky "corrections" are needed.

Air layer case

Let us now assume $\rho_a = 0$.

On the left or right sides (i.e. $x = 0$ or $x = L_x$), the lithostatic pressure at the base of the crust is then $\rho_c g h_c = 2300 \cdot 10 \cdot 20000 = 460$ MPa. At the base of the model, then the pressure is $460e6 + \rho_m g h_m = 460e6 + 3300 \cdot 10 \cdot 80000 = 460e6 + 2640e6 = 3100$ MPa. In a more abstract way, $P_{bottom} = \rho_c g h_c + \rho_m g h_m$.

In the middle, after the block has moved up and is now in isostatic equilibrium, we should have $P_{bottom} = \rho_c g h'_c + \rho_m g h'_m$. In the end:

$$P_{bottom} = \rho_c g h_c + \rho_m g h_m = \rho_c g h'_c + \rho_m g h'_m$$

We can divide both sides by g:

$$\begin{aligned} \rho_c h_c + \rho_m h_m &= \rho_c h'_c + \rho_m h'_m \\ \Rightarrow \rho_m h'_m &= \rho_c (h_c - h'_c) + \rho_m h_m \\ \Rightarrow h'_m &= h_m - (h'_c - h_c)\frac{\rho_c}{\rho_m} \end{aligned}$$

Then the dynamic topography is computed as follows:

$$\begin{aligned} \Delta h &= h'_c + h'_m - h_c - h_m \\ &= h'_c + h_m - (h'_c - h_c)\frac{\rho_c}{\rho_m} - h_c - h_m \\ &= h'_c - (h'_c - h_c)\frac{\rho_c}{\rho_m} - h_c \\ &= (h'_c - h_c) - (h'_c - h_c)\frac{\rho_c}{\rho_m} \\ &= (h'_c - h_c)\left(1 - \frac{\rho_c}{\rho_m}\right) \end{aligned} \tag{160.1}$$

Substituting for the parameters of our example, the predicted topography uplift is

$$\Delta h = (40000 - 20000) \cdot (1 - 2300/3300) \simeq 6060 \text{ m}$$

Water layer case

In this case we set $\rho_a = 1000$.

On the left side we have $P_{bottom} = \rho_a g h_a + \rho_c g h_c + \rho_m g h_m$, which yields $P_{bottom} = (1000 * 20e3 + 2300 * 20e3 + 3300 * 80e3) * 10 = 33e8 = 3300$ MPa. In the middle: $P_{bottom} = \rho_a g h'_a + \rho_c g h'_c + \rho_m g h'_m$.

$$\begin{aligned} P_{bottom} &= \rho_a g h_a + \rho_c g h_c + \rho_m g h_m \\ &= \rho_a g h'_a + \rho_c g h'_c + \rho_m g h'_m \end{aligned}$$

or, after diving by g:

$$\rho_a h_a + \rho_c h_c + \rho_m h_m = \rho_a (h_a - \Delta h) + \rho_c h'_c + \rho_m h'_m$$

which we write:

$$\begin{aligned} \rho_m h'_m &= \rho_a h_a + \rho_c h_c + \rho_m h_m - [\rho_a (h_a - \Delta h) + \rho_c h'_c] \\ &= \rho_a h_a + \rho_c h_c + \rho_m h_m - \rho_a (h_a - \Delta h) - \rho_c h'_c \\ &= \rho_c h_c + \rho_m h_m + \rho_a \Delta h - \rho_c h'_c \end{aligned} \tag{160.2}$$

Looking at the figure we also have

$$\begin{aligned}
\Delta h &= h'_c + h'_m - h_c - h_m \\
&= h'_c + \frac{1}{\rho_m}[\rho_c h_c + \rho_m h_m + \rho_a \Delta h - \rho_c h'_c] - h_c - h_m \\
&= h'_c + \frac{\rho_c}{\rho_m} h_c + h_m + \frac{\rho_a}{\rho_m}\Delta h - \frac{\rho_c}{\rho_m} h'_c - h_c - h_m \\
&= h'_c + \frac{\rho_c}{\rho_m} h_c + \frac{\rho_a}{\rho_m}\Delta h - \frac{\rho_c}{\rho_m} h'_c - h_c \\
(1 - \frac{\rho_a}{\rho_m})\Delta h &= h'_c + \frac{\rho_c}{\rho_m} h_c - \frac{\rho_c}{\rho_m} h'_c - h_c \\
(1 - \frac{\rho_a}{\rho_m})\Delta h &= (1 - \frac{\rho_c}{\rho_m}) h'_c - (1 - \frac{\rho_c}{\rho_m}) h_c \\
(1 - \frac{\rho_a}{\rho_m})\Delta h &= (1 - \frac{\rho_c}{\rho_m})(h'_c - h_c) \\
\Delta h &= (1 - \frac{\rho_a}{\rho_m})^{-1}(1 - \frac{\rho_c}{\rho_m})(h'_c - h_c) \\
\Delta h &= \frac{\rho_m}{\rho_m - \rho_a}\frac{\rho_m - \rho_c}{\rho_m}(h'_c - h_c) \\
\Delta h &= \frac{\rho_m - \rho_c}{\rho_m - \rho_a}(h'_c - h_c) \qquad (160.3)
\end{aligned}$$

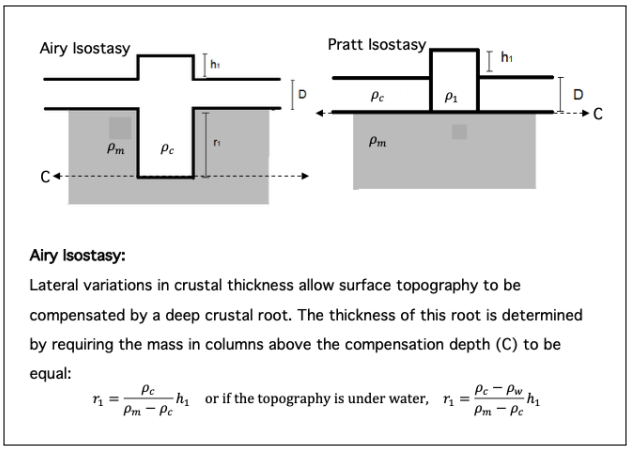
In this case we get

$$\Delta h = \frac{3300 - 2300}{3300 - 1000}(40e3 - 20e3) = \frac{1000}{2300} 20e3 \simeq 8695 \text{ m}$$

Obviously if we set $\rho_a = 0$ we find the value obtained for air.

Clint's comments

Your predicted topography is basically isostatic topography: I get the same 6.06 km when I compute Airy topography using the equations from my Geodynamics notes:



However, this is topography above the surrounding topography - h_1 in the left figure. I got this by considering that the topography is under water in this case, and $r_1 + h_1 = 20$ km, then I use the right hand equation to find:

$$20km - h_1 = (\rho_c - \rho_w)/(\rho_m - \rho_c) * h_1$$

$$20km = h_1 + (1300/1000) * h_1 = 2.3 * h_1$$

$$h_1 = 8.69km$$

How to compute DT in geodynamical codes?

In the ASPECT manual we find the following documentation for the 'dynamic topography' post-processor:

> A postprocessor that computes a measure of dynamic topography based on the stress at the boundary. The exact approach works as follows: At each selected boundary, we compute the traction that acts normal to the boundary faces using the consistent boundary flux method as described in Gresho, Lee, Sani, Maslanik, and Eaton [141] (1987). From this traction, the dynamic topography is computed using the formula $h = \sigma_n/\rho g$ where g is the norm of the gravity and ρ is the density [of the fluid right below the surface].

In practice the dynamic topography is then computed as follows:

$$h = \frac{\vec{n} \cdot \boldsymbol{\sigma} \cdot \vec{n}}{\rho_c g} = \frac{\sigma_{yy}}{\rho_c g} = \frac{-p + \tau_{yy}}{\rho_c g}$$

where $\vec{n}$ is the normal vector to the surface[1].

Clint adds:

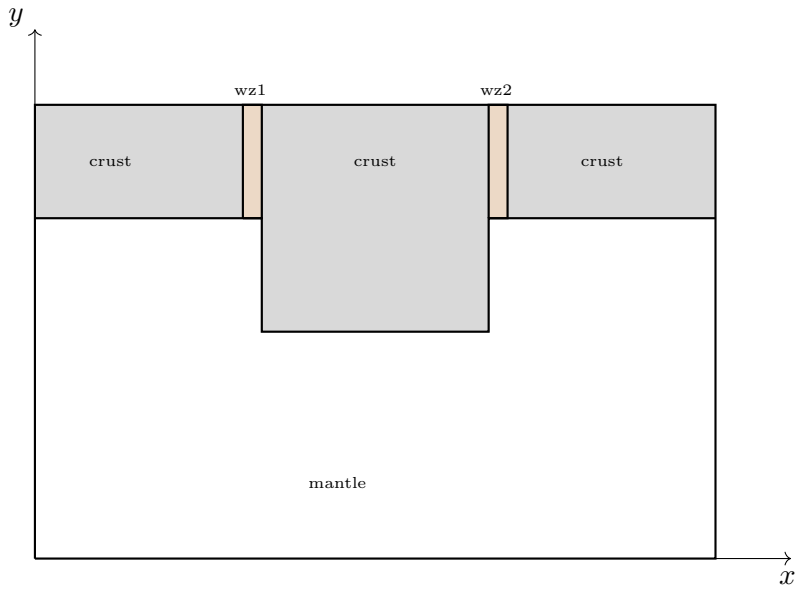
> We usually use the full normal stress σ_n to compute dynamic topography as $h = \sigma_n/(\delta\rho * g)$ It is important to keep the pressure term (and not omit it as with deviatoric stress) because pressure variations also contribute to dynamic topography. The idea is that if the boundary cannot move vertically (no free surface), then the stress on that boundary tells us what the topography *would be* if the surface could move. (I guess this can be tested with a code that can handle a free surface).
> Setting the surface average equal to zero would mean that the average dynamic topography h is also zero (if $\delta\rho$ is constant) - we kind of used this in this paper (based on CitComS): Conrad and Husson [79] (2009). The zero-average dynamic topography means that if dynamic topography on the continents is on-average increasing, then it is decreasing on-average over the oceans, leading to sea level drop.
> One interesting thing that we explored in our 2009 paper: Let's say you have a dense blob in the mantle - it produces negative dynamic topography at the surface that gets smaller with time as the blob descends. Thus, the sense of motion is uplift. Now consider a low-density blob - this produces positive dynamic topography that gets bigger with time as the blob gets closer to the surface - also uplift! So you get uplift above both positive and negative density anomalies in the mantle. By the same logic, you would always get CMB subsidence below positive and negative density anomalies. But I guess you get the opposite behavior everywhere else away from above/below the density anomaly, so everything evens out.

[1] $\vec{t} = \boldsymbol{\sigma} \cdot \vec{n}$ is the traction on the boundary and then $\vec{n} \cdot \vec{t}$ is the component of this traction along the normal.

Results

Let us now investigate this by means of numerical modelling. For many decades (and still now for large 3D spherical models) so-called sticky-air, sticky-water or true free surface was not an option and people relied on free surface boundary conditions at the top of the crust, only to use the algorithm above to arrive at an estimate of the equivalent topography.

The setup/geometry for the calculations is as follows

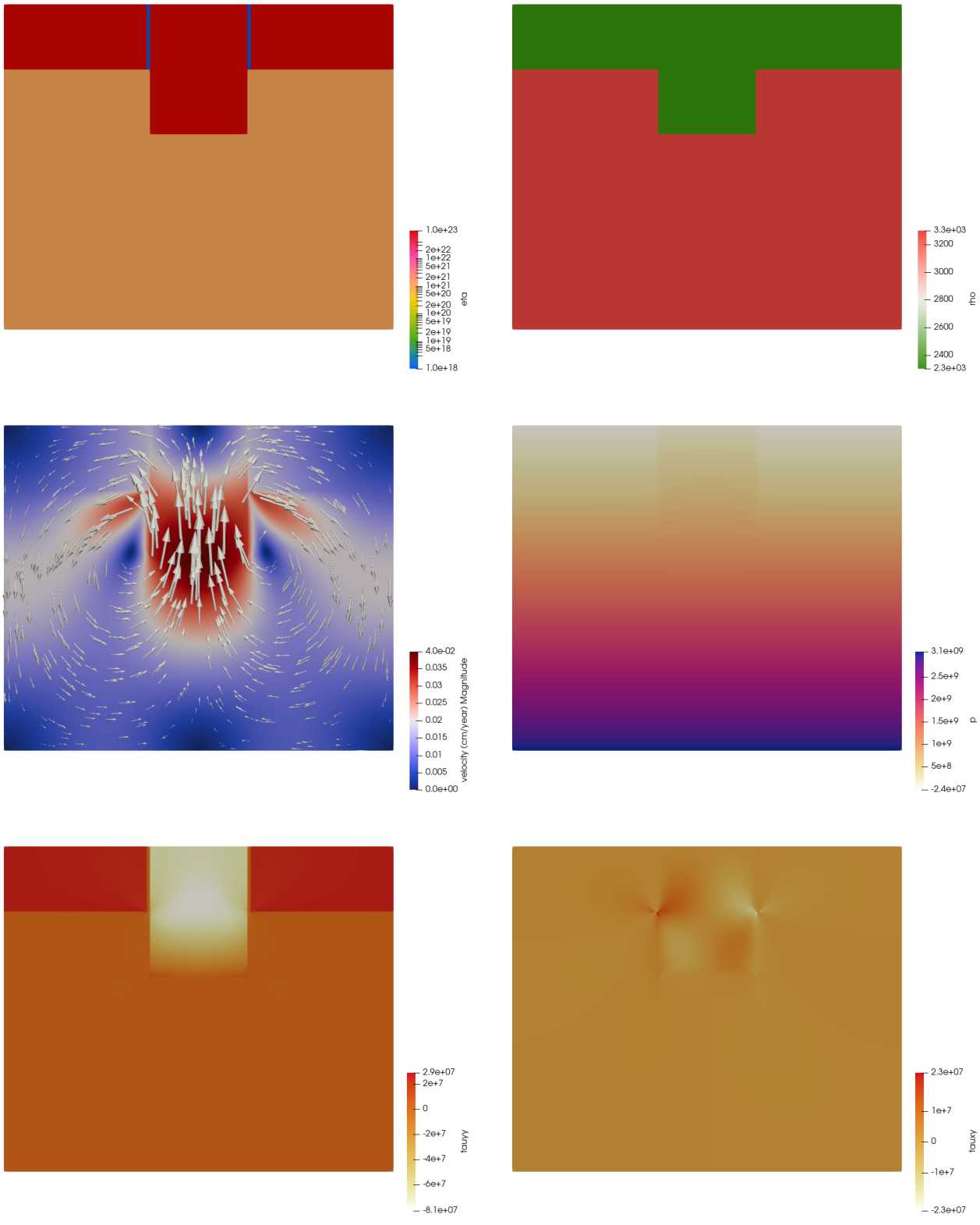


Drawing not to scale.

Free slip boundary conditions are imposed on all four sides. To be very clear, the air/water layer is absent so that the domain is 120×100 km. $\boldsymbol{Q}_2 \times Q_1$ (continuous pressure) or $\boldsymbol{Q}_2 \times P_{-1}$ (discontinuous pressure) elements are used. The boundary conditions are such that the pressure can only be computed up to a constant (null space) and it is therefore normalised by imposing that the average pressure at the surface is zero (the implicit assumption being that the air above it is massless). Concerning this Clint states:

> Removing the average pressure to me makes sense. If there were net positive pressure on the boundary, then we could think about this as a net shift of the entire boundary upwards, meaning that we could choose a slightly thicker box (or larger earth radius) to make the average pressure zero - in some sense this is an arbitrarily chosen parameter. I guess one challenging thing to think about would be how to handle changes in the pressure that develop over time - if this even happens in the models? I guess these would correspond to changes in volume that happen over time - for example based on changes in average temperature (heating or cooling earth). For an incompressible calculation this would not result in a change in volume, but for compressible convection I think it might be - a warming earth would be less dense, meaning that its radius should expand, resulting in dynamic topography that grows with time. But I think this component would be missed if the pressure is always removed from the stress at the surface. This paper discusses this, and seems to suggest (in the abstract) that the earth radius cannot change with time (even for compressible convection?) [109].

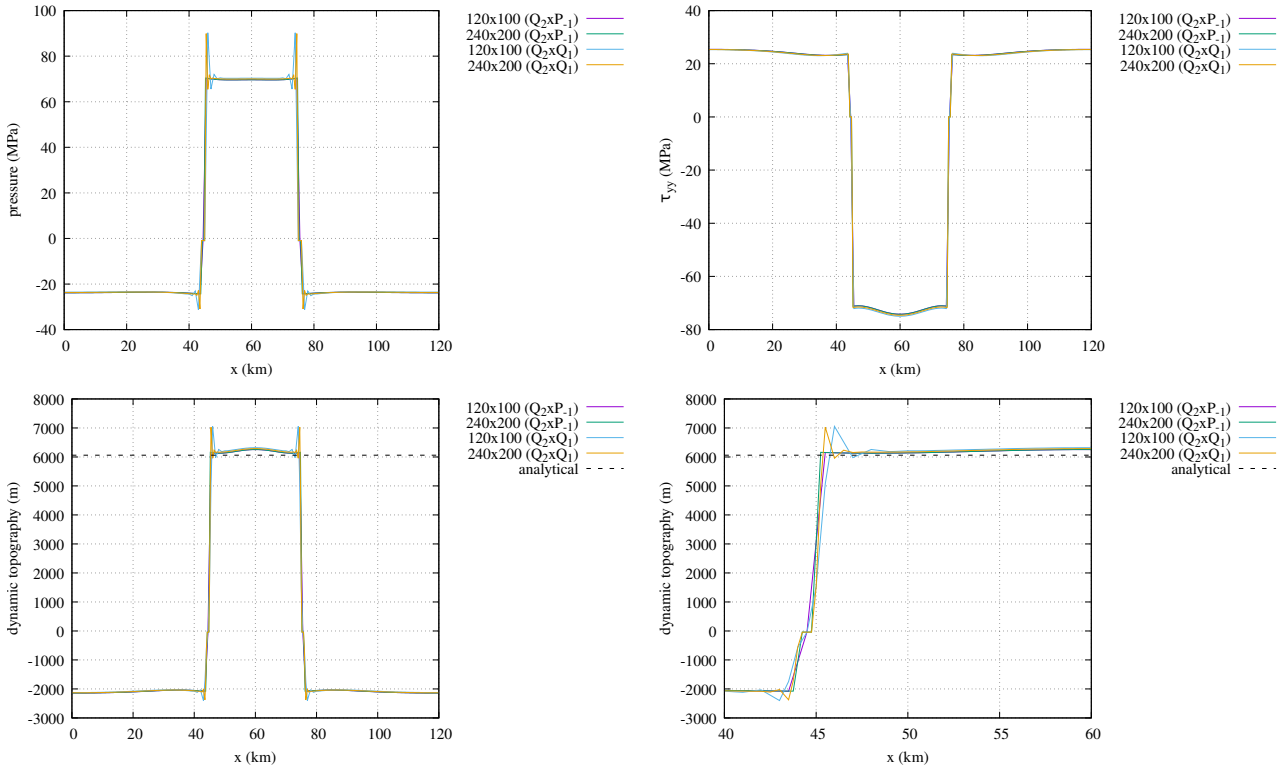
In all what follows temperature effects are discarded. A single Stokes solve is carried out, without any time stepping nor advection of the materials. The default resolution is set to 120×100 elements.



Results obtained on 240x200 mesh. Top row: viscosity and density fields; Middle row: velocity and pressure fields; Bottom row: τ_{yy} and τ_{xy} fields.

Reference case

The surface pressure, yy component of deviatoric stress $\boldsymbol{\tau}$ and dynamic topography are shown in the following plots:



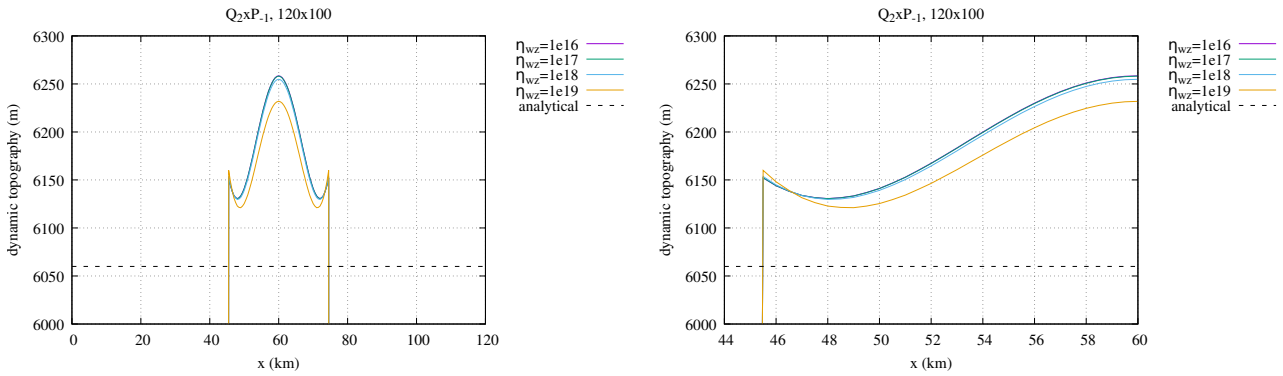
We find that the differences between low and high resolution are really small. The recovered dynamic topography is very close to the expected analytical value, but since its accuracy is not improving with resolution this probably means that the weak zones and/or the coupling of the keel of the crust with the mantle is playing a prominent role.

We also find that the discontinuous pressure element unsurprisingly copes better with the element boundary-aligned viscosity jumps, while the continuous pressure one yields overshoots on the pressure field.

Note that the topography is about 6000 m relative to the zero, but since the crust left and right has a topography of about -2000 m, then the 'real' topography is more akin to 8000 m.

Influence of the weak zones viscosity

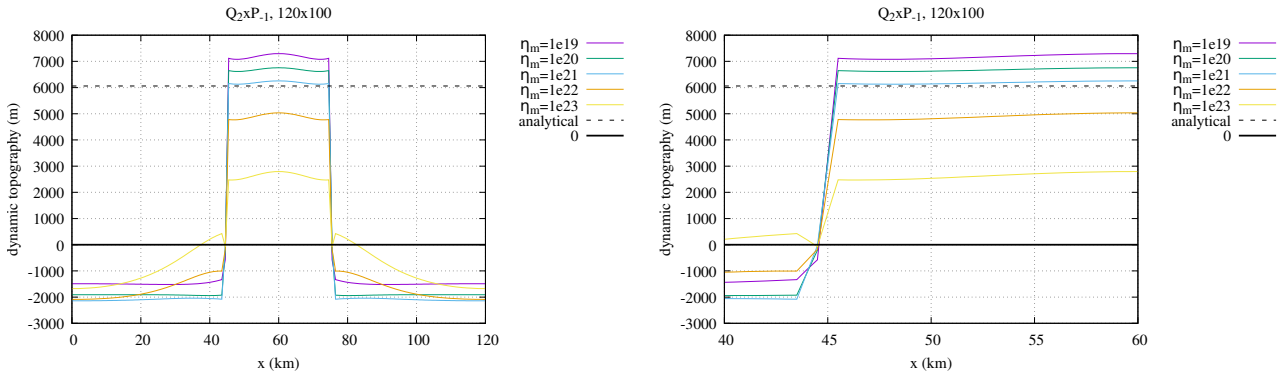
Let us now investigate the influence of the weak zones viscosity on the dynamic topography (zoomed on the important part in the right figure):



The default value of $\eta_{wz} = 10^{18}$ Pa s is clearly low enough since lower values do not yield any meaningful change. This means that the signal we observe is entirely due to the geometry of the experiment and/or the coupling crust keel-mantle.

Influence of the mantle viscosity

We can now look at the influence of the mantle viscosity (while keeping the weak zones at $\eta_{wz} = 10^{18}$ Pa s):



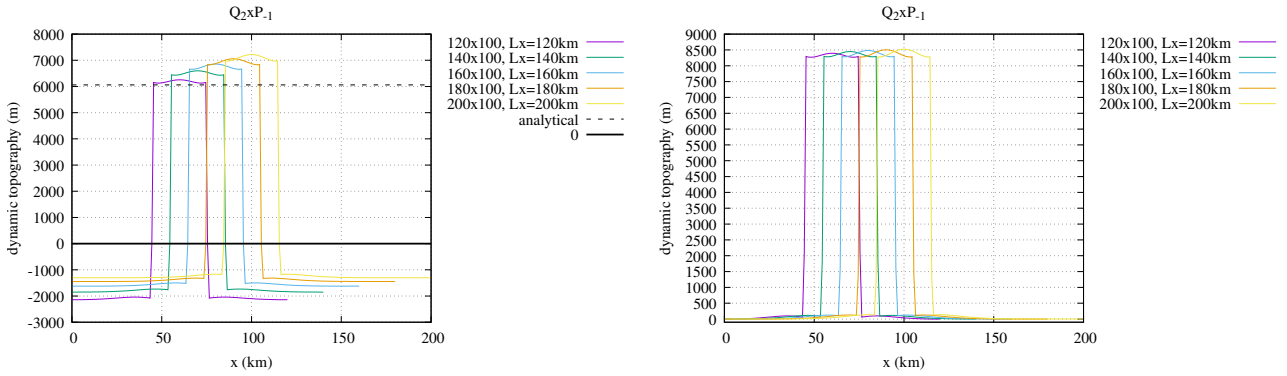
The higher the viscosity of the mantle, the lower the dynamic topography. It appears that $\eta_m = 10^{21}$ Pa s yields a dynamic topography closest to the expected theoretical value. It is also worth noting that on each side of the block the dynamic topography is about -2000m. Looking at the velocity field on the previous page figure, we see that the domain is such that the positive buoyancy of the block generates a convection cell which therefore pulls down the crust on each side of the block.

Clint comments:

> In this figure you get increasingly smaller topography as you increase the mantle viscosity. I think what you are seeing here is only partially compensated topography (not yet in isostatic equilibrium) - kind of a GIA process that has not run to completion. The timescale for this to happen is longer for higher mantle viscosity, so the topography has relaxed less for the higher viscosity cases. For the 1e19 Pa s case (purple line), then it is actually almost fully compensated already. My prediction is that if you let these systems run for many timesteps then the region of thick crust would pop up more and more, until it reaches they all reach theoretical value of 8.69 km above the surrounding topography [if water was indeed present above the crust]. This might take a long time for the 1e23 Pa s case. The timescale for relaxation also depends on wavelength, so you could make these run faster if you make the thick crust region wider.

Influence of the box geometry

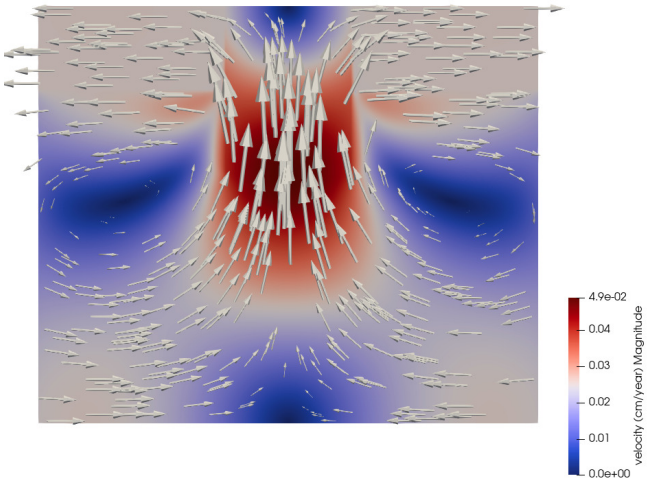
Let us now investigate the influence of the box geometry (and in particular its lateral extent L_x while keeping a 1×1 km resolution).



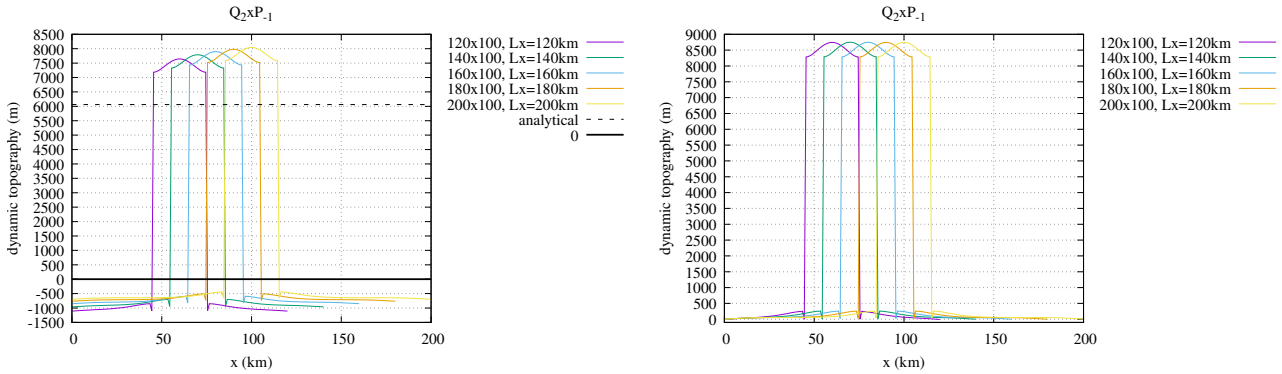
Left: raw signal; Right: signal offset so that DT is zero on the sides of the domain.

We find that the dynamic topography depends mildly on this parameter (all other things equal), which is evident when the signals are offset so that the topograhy is zero on the sides (right figure above).

In light of all this, let us now prescribe the lithostatic pressure (Neumann boundary conditions) on the left and right boundaries so as to allow for free in/outflow and hopefully tame the return flow. A single node in the middle of the bottom boundary is assigned $u = 0$ so as to remove the translational nullspace. We obtain the following velocity field for the default setup:



and the dynamic topography varies with L_x as follows:



Left: raw signal; Right: signal offset so that DT is zero on the sides of the domain.

When looking at the raw signal, we find that the dynamic topography is less dependent on horizontal extent than before but still increases by about 1000m. However, we also must consider the negative topography on each side of the block. When offseting the topography signal so that it is zero on the sides of the domain (right figure) we find that the DT is constant across the range of L_x values!

In light of all this, I am a bit lost as to what this theoretical value stands for. I guess an infinitely long and deep domain? My plan is to revisit this with ASPECT asap to make sure that these results are correct.

Stone 161

Implementing the $Q_{2,1} \times Q_{1,2} \times Q'_1$ element



python_codes/fieldstone_161/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_161

We here consider the $Q_{2,1} \times Q_{1,2} \times Q'_1$ element introduced by Huang and Zhang [159] (2011) which follows Zhang [367] (2009). The shape functions in 2D are in derived in Section 10.4.16.

The question we wish to answer is whether this element could be used in the context of computational geodynamics. The element has been proven stable and verified on a simple isoviscous manfufactured solution with trivial boundary conditions, but quid of more complex boundary conditions ? free surface ? sharp viscosity transitions? buoyancy-driven flow?

The list of velocity degrees of freedom per element is as follows:

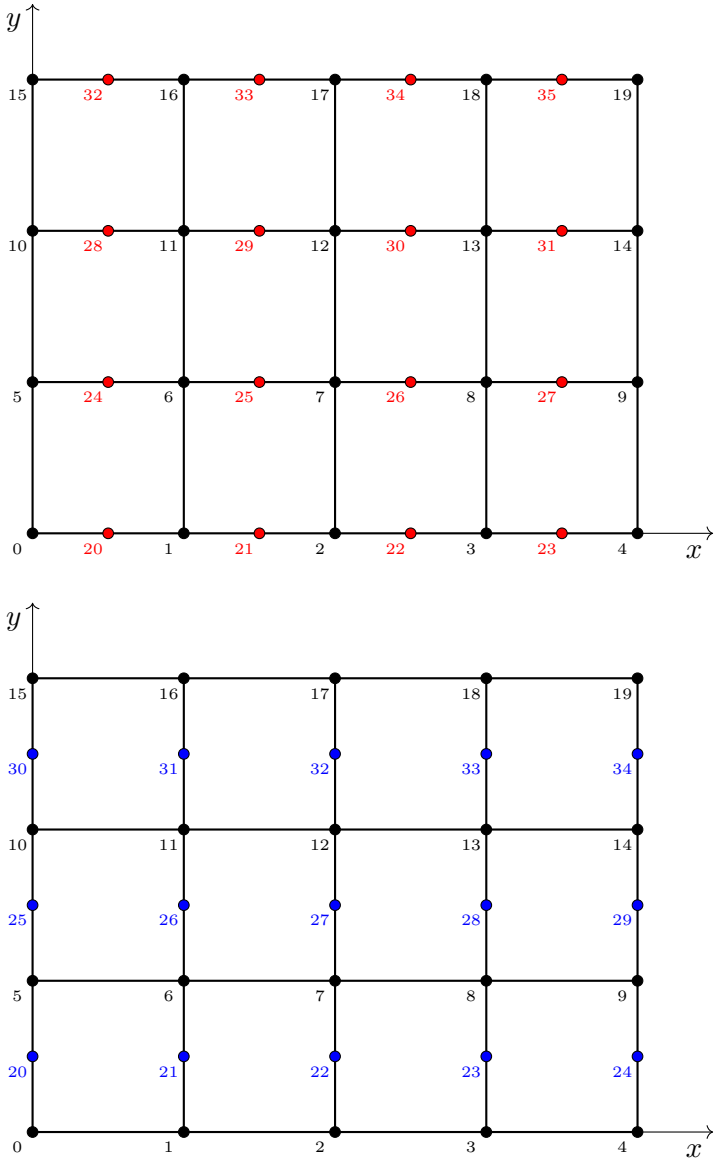
$$\vec{\mathcal{V}} = (u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4, u_5, v_5, u_6, v_6)$$

where u is the x-component of the velocity field and v its y-component, with the following internal numbering

```
u dofs      v dofs

4--6--3     4-----3
|     |     |     |
|     |     5     6
|     |     |     |
1--5--2     1-----2
```

Remark: the location of the 'extra' nodes for u and v are opposite of those for the Fortin $Q_1^+ \times P_0$ element, see Section 10.4.6 (see also STONE 80).

We start with a simple 4×3 element mesh. u dofs are represented in red and v dofs are shown in blue:



For this mesh we have

```
nelx=4
nely=3
nnx=nelx+1
nny=nely+1
nel=nelx*nely  (=12)
Nu=nnx*nny+nelx*nny  (=20+16=36)
Nv=nnx*nny+nnx*nely  (=20+15=35)
```

The total number of velocity dofs is

```
NfemV=Nu+Nv  (=36+35=71)
```

What makes this element pair rather awkward to implement is the fact that there are two connectivity arrays iconu and iconv, both of size mV × nel, where mV=6 is the number of nodes linked to an element for each velocity component.

- content of iconu

```
elt udofs
0 | [ 0  1  6  5 20 24]
1 | [ 1  2  7  6 21 25]
2 | [ 2  3  8  7 22 26]
3 | [ 3  4  9  8 23 27]
```

```
4 | [ 5  6 11 10 24 28]
5 | [ 6  7 12 11 25 29]
6 | [ 7  8 13 12 26 30]
7 | [ 8  9 14 13 27 31]
8 | [10 11 16 15 28 32]
9 | [11 12 17 16 29 33]
10 | [12 13 18 17 30 34]
11 | [13 14 19 18 31 35]
```

- content of iconv

```
elt vdofs
0 | [ 0  1  6  5 20 21]
1 | [ 1  2  7  6 21 22]
2 | [ 2  3  8  7 22 23]
3 | [ 3  4  9  8 23 24]
4 | [ 5  6 11 10 25 26]
5 | [ 6  7 12 11 26 27]
6 | [ 7  8 13 12 27 28]
7 | [ 8  9 14 13 28 29]
8 | [10 11 16 15 30 31]
9 | [11 12 17 16 31 32]
10 | [12 13 18 17 32 33]
11 | [13 14 19 18 33 34]
```

Unlike many other codes in this book the global solution vector $\vec{\mathcal{V}}$ is organised as follows:

$$\vec{\mathcal{V}} = (u_1, u_2, ...u_{Nu}, v_1, v_2, ...v_{Nv})$$

As mentioned in the articles[1]

> It is difficult to find a local basis for P_h. But on the other side, it is the special interest of the method that the space P_h can be omitted in computation and the discrete solutions approximating the pressure function in the Stokes equations can be obtained as byproducts, as we shall see next.

so I will leave the pressure dofs alone for now, and will come back to it a bit later.

Remark 1: given the weird placement of nodes for u and v, it makes sense not to use an isoparameteric mapping[2], and instead choose a simple Q_1 mapping (this is not very important here since all elements are rectangles anyways).

Remark 2: the paper does not mention quadrature at all. I choose a 3×3 quadrature as for $\boldsymbol{Q}_2 \times Q_1$.

Remark 3: the papers also do not mention 3D, nor non-rectangular elements.

Iterated penalty method

Following Zhang [367] and Huang and Zhang [159] we implement this method since

> it is the special interest of the divergence-free finite element method that the space P_h can be omitted in computation and the discrete solutions approximating the pressure function in the Stokes equations can be obtained as byproducts, if an iterated penalty method is adopted to solve system (2.5).

Also in Huang and Zhang [159]:

[1] See also Section 10.4.16

[2] Would it even be well defined?

We do enough iterated penalty iterations until the iterative error is smaller than the truncation error each time.

In Zhang [367] we find the following definition:

> DEFINITION 2.1 (The iterated penalty method). *Let the initial iterate* $\mathbf{u}_h^0 = \mathbf{0}$ *for the finite element Stokes equation* (2.5). *The rest iterates* $\mathbf{u}_h^n$ *are defined iteratively to be the unique solution of*
>
> $$(2.9)\quad a(\mathbf{u}_h^n, \mathbf{v}_h) + \alpha(\operatorname{div}\mathbf{u}_h^n, \operatorname{div}\mathbf{v}_h) = (\mathbf{f}, \mathbf{v}_h) - \left(\operatorname{div}\sum_{j=0}^{n-1}\alpha\mathbf{u}_h^j, \operatorname{div}\mathbf{v}_h\right) \quad \forall \mathbf{v}_h \in \mathbf{V}_h,$$
>
> $n = 1, 2, \ldots$. *Here* α *is positive constant. At the end of iteration, we let*
>
> $$p_h^n = -\operatorname{div}\sum_{j=0}^{n}\alpha\mathbf{u}_h^j.$$

Let us then write the first steps of this algorithm[3].

- For $n = 1$:

$$\begin{aligned} a(\boldsymbol{u}_h^1, \boldsymbol{v}_h) + \lambda(\operatorname{div}\boldsymbol{u}_h^1, \operatorname{div}\boldsymbol{v}_h) &= (\boldsymbol{f}, \boldsymbol{v}_h) - \operatorname{div}\left(\lambda\boldsymbol{u}_h^0, \operatorname{div}\boldsymbol{v}_h\right) \\ p_h^1 &= -\operatorname{div}\left[\lambda(\boldsymbol{u}_h^0 + \boldsymbol{u}_h^1)\right] \end{aligned} \tag{161.1}$$

- For $n = 2$:

$$\begin{aligned} a(\boldsymbol{u}_h^2, \boldsymbol{v}_h) + \lambda(\operatorname{div}\boldsymbol{u}_h^2, \operatorname{div}\boldsymbol{v}_h) &= (\boldsymbol{f}, \boldsymbol{v}_h) - \operatorname{div}\left(\lambda(\boldsymbol{u}_h^0 + \boldsymbol{u}_h^1), \operatorname{div}\boldsymbol{v}_h\right) \\ p_h^2 &= -\operatorname{div}\left[\lambda(\boldsymbol{u}_h^0 + \boldsymbol{u}_h^1 + \boldsymbol{u}_h^2)\right] \end{aligned} \tag{161.2}$$

Concretely, we start with $\boldsymbol{u}_h^0 = \mathbf{0}$ and we compute the pressure only when the iterations have converged, i.e. the inf norm of the difference between two consecutive velocity fields is less than a user chosen tolerance:

```
xi_u=np.max(abs(u_prev-u))/np.max(abs(u))
xi_v=np.max(abs(v_prev-v))/np.max(abs(v))
if xi_u<tol and xi_v<tol:
   break
```

These values are written at every iteration in the `conv.ascii` file.

Once the velocity field has been obtained the pressure is computed at each corner of each element (Q_{-1} representation).

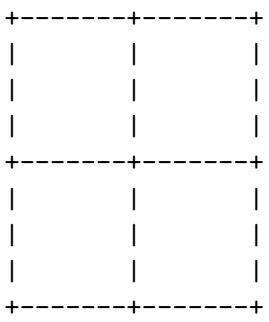
[3] For personal reasons and to establish a parallel with the penalty method, I replace α by λ.

Grids and dofs

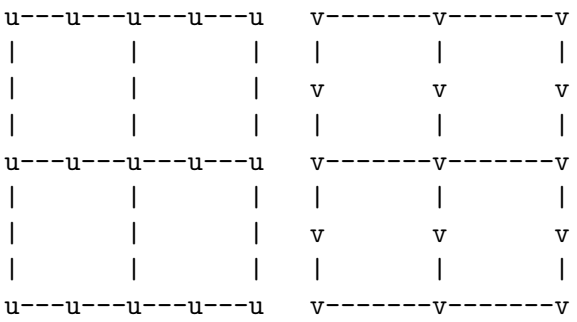
In Huang and Zhang [159] we find the following table:

level	dim V_h	dim P_h
2	6	6
3	42	38
4	210	174
5	930	734
6	3906	3006

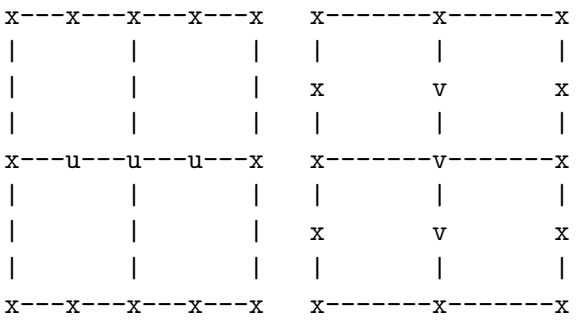
The authors state that "The initial grid, level one grid, is simply one unit square" so that the level 2 grid is a 2×2 element grid:



with the following velocity dofs:

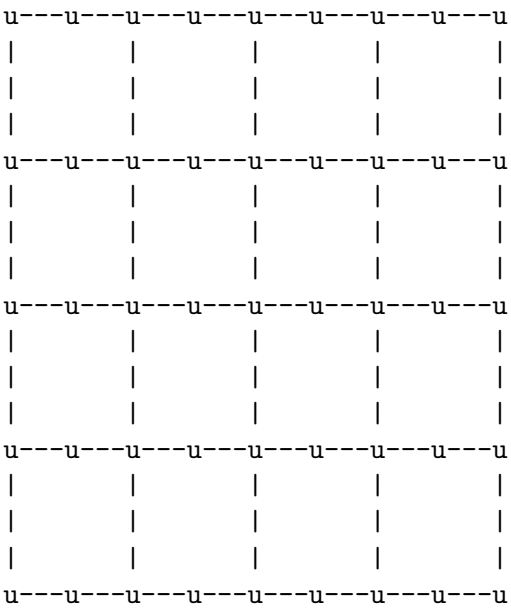


Since $u = v = 0$ on all boundaries we replace the u, v dofs by $\times$ to indicate that these are no longer degrees of freedom:

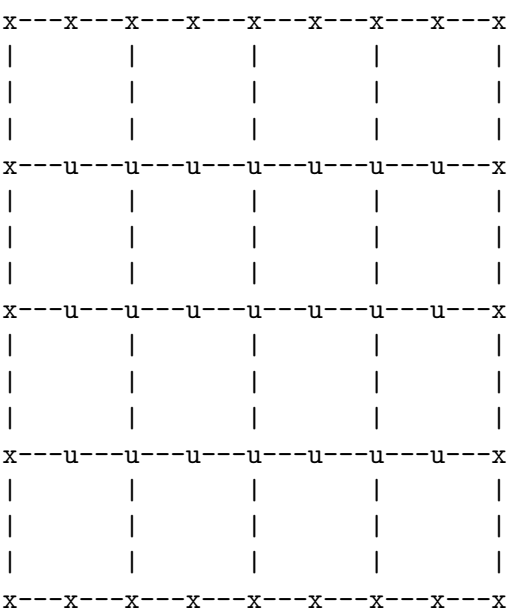


and we indeed find dim $V_h = 6$.

We consider now a level 3 grid, i.e. 4×4 elements. The u dofs are placed as follows:



After boundary conditions are imposed:



Only 21 u are left, and we will find as many v dofs, so in this case $\dim(V_h) = 21 + 21 = 42$, in accordance with the table.

Let us now turn to the pressure dofs. Warning: it is not as simple as the velocity dofs. In Huang and Zhang [159] we read: "we decrease the space Q_1^{dc} for the pressure to Q_1', by removing all spurious modes, i.e., eliminating one degree of freedom at each vertex. We have to emphasize that the discrete pressure space is introduced for the analysis, but not in the computation. By an iterated penalty method, we obtain the discrete solutions for the pressure without coding the pressure element" and in Zhang [367]: " We note that, by choosing $\mathrm{div}(Q_{k+1,k} \times Q_{k,k+1})$ as the discrete finite element space for the pressure, the spurious modes in discontinuous Q_k space are filtered out automatically. [...] P_h is a subspace of discontinuous, piecewise polynomials of separate-degree k or less. As singular vertices are present (see [15, 16]), P_h is a proper subset of the discontinuous piecewise Q_k polynomials. It is difficult to find a local basis for P_h. [...] We note that the spurious (checkerboard) modes, generated at each singular vertex (cf. (3.8)), in the discrete pressure space are filtered out naturally. "

In Zhang [368] the definition for a singular vertex is given:

> A vertex is called singular if all edges meeting at the vertex form two cross lines.

In order to understand this and arrive at the correct number of pressure dofs presented in the table above it is useful to look at this part of Huang and Zhang [159]:

Here, a_i^M, $1 \leqslant i \leqslant 4$, are the four vertices of macro-element M, cf. Figure 4.

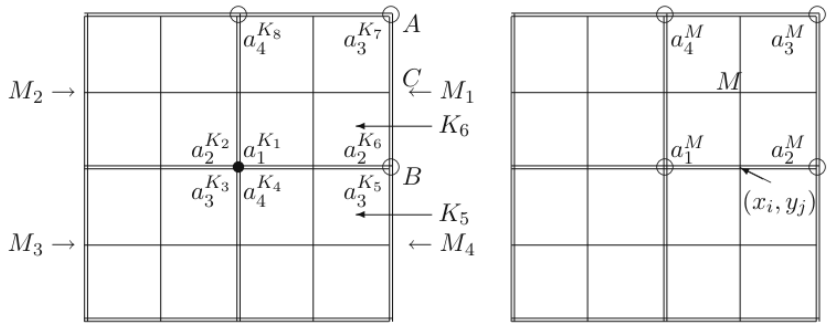


Fig. 4 Four macro-elements meeting at a singular vertex, cf. (3.3) and (3.7)

Proof According to Scott and Vogelius [23], all vertices in $\mathscr{T}_h$ are singular, which means a function $q \in P_h$, because $q = \nabla \cdot \boldsymbol{w}_h$ for some $\boldsymbol{w}_h \in \boldsymbol{V}_h$, is subject to a minor continuity constraint:

$$\sum_{i=1}^{4} (-1)^i q(a_i^{K_i}) = 0, \tag{3.6}$$

where $a_i^{K_i}$ are the vertices of four elements K_i meeting at an internal vertex of a macro-mesh $\mathscr{M}_h$, shown in Fig. 4. However, at a boundary vertex A or B, where less than four squares meet, see Fig. 4, we have

$$q(a_3^{K_7}) = 0, \quad q(a_2^{K_6}) = q(a_3^{K_5}). \tag{3.7}$$

All nodes are subjected to a constraint. After the mesh is divided in 2×2 macro-elements we see that there are 3 type of vertices: the corner ones, the mid-edge ones, and the interior ones. Let us start with the simplest case: a 2×2 element mesh, i.e. a single macro element. If the pressure field was a 'true' Q_{-1} field[4], we would have 4 pdofs per element, so $4 \cdot 4 = 16$ pdofs in total:

```
+-------+-------+
|4     3|4     3|
|       |       |
|1     2|1     2|
+-------+-------+
|4     3|4     3|
|       |       |
|1     2|1     2|
+-------+-------+
```

However it is clear from the Eq. (3.7) above that the corner nodes are constrained to 0, so we have 4 constraints, i.e. 4 pdofs removed. The same equation tells us that mid-edge vertices are constrained (i.e. the pressure is continuous at those vertices) so we have again 4 more constraints. Finally there is one constraint for the vertex in the middle, and one should not forget that the pressure space is such that $\int_\Omega p \, dV = 0$, which introduces a last constraint. In total, 4+4+1+1=10 constraints, so we are left with 16-10=6 pdofs, which is the number in the table.

We also find in Zhang and Mu [369][5] a rewording of the constraints:

[4] Same as Q_1^{dc}, but I prefer Q_{-1} to remain consistent with the rest of the book.

[5] That paper is about a different FE pair, that is not divergence-free, but it presents the pressure space of our FE element here.

Recall that for the divergence-free elements in [26–28], the velocity space is defined as $\mathbf{V}_h' = C^0\text{-}Q_{k,k-1} \times Q_{k-1,k}$ and the pressure space is defined as

$$P_h' = \{p_h \in L_0^2(\Omega) \mid p_h|_K \in Q_{k-1},\ T_{\mathbf{x}_j} v_h = 0\ \mathbf{x}_j \text{ is a vertex of } \mathcal{T}_h\}, \tag{2.3}$$

where operator $T_{\mathbf{x}_j}$ is defined for each vertex $\mathbf{x}_j$ of $\mathcal{T}_h$, see Fig. 2.1,

$$T_{\mathbf{x}_j} p_h = \begin{cases} p_h|_{K_1}(\mathbf{x}_j) & \mathbf{x}_j \text{ in Fig. 2.1(a)}, \\ p_h|_{K_1}(\mathbf{x}_j) - p_h|_{K_2}(\mathbf{x}_j) & \mathbf{x}_j \text{ in Fig. 2.1(b)}, \\ \sum_{i=1}^{4} (-1)^{i-1} p_h|_{K_i}(\mathbf{x}_j) & \mathbf{x}_j \text{ in Fig. 2.1(c)}. \end{cases} \tag{2.4}$$

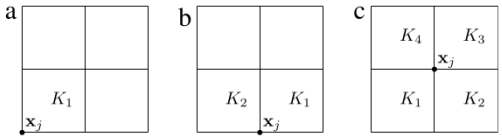


Fig. 2.1. The constraints on pressure p_h by the operator $T_{\mathbf{x}_j}$ in (2.4).

In this case, the finite element solution is divergence-free pointwise. Then in its computation the pressure space is not coded but defined implicitly as the divergence space of $\mathbf{V}_h'$, and the finite element solution of pressure is computed as a byproduct.

Let us look at a 4×4 element mesh, i.e. a 2×2 macro element as in the figure above.

```
C-------+-------+-------+-------C
|4     3|4     3|4     3|4     3|
|       |       |       |       |
|1     2|1     2|1     2|1     2|
+-------+-------+-------+-------+
|4     3|4     3|4     3|4     3|
|       |       |       |       |
|1     2|1     2|1     2|1     2|
+-------+-------+-------+-------+
|4     3|4     3|4     3|4     3|
|       |       |       |       |
|1     2|1     2|1     2|1     2|
+-------+-------+-------+-------+
|4     3|4     3|4     3|4     3|
|       |       |       |       |
|1     2|1     2|1     2|1     2|
C-------+-------+-------+-------C
```

We start from potentially $16 \cdot 4 = 64$ pdofs. We find that *all* vertices are subjected to a constraint, either because located on a corner, an edge or in the middle of four elements. This explains the sentence "we decrease the space Q_1^{dc} for the pressure to Q_1', by removing all spurious modes, i.e., eliminating one degree of freedom at each vertex." In the end each vertex is assigned a constraint, which removes as many pdofs. In this case, $5 \cdot 5 = 25$ constraints. Taking into account the zero-average constraint, we are left with 64-25-1=38 pdofs, as in the table!

Concretely if the mesh is composed of $n \times n$ elements, then it counts $4n^2 - (n+1)^2 - 1$ pdofs.

- $n = 2$: $4n^2 - (n+1)^2 - 1 = 16 - 9 - 1 = 6$ (we covered that already)
- $n = 4$: $4n^2 - (n+1)^2 - 1 = 64 - 25 - 1 = 38$ (we covered that already)
- $n = 8$: $4n^2 - (n+1)^2 - 1 = 256 - 81 - 1 = 174$ (value in the table)

- $n = 16$: $4n^2 - (n+1)^2 - 1 = 1024 - 289 - 1 = 734$ (value in the table)

Having understood this, it is clear that it would be very difficult to come up with analytical expressions for the pressure basis functions.

At this stage, we can ask ourselves the question of the consequences of eq(3.7) (i.e. p=0 at corners) for generic cases? We shall come back to this later.

Remark: Although the constraints are explained by means of 2x2 macro-elements, one could wonder whether using off number of elements per side could yield problems. I have tried this, and in practice, this makes no difference in the error values nor their convergence rates.

code structure

```
build mesh arrays, xu, yu, xv, yv
build connectivity arrays iconu, iconv
define boundary conditions
for iter in range(0,niter):
    build FE matrix and rhs
    solve linear system
    check convergence of u and v fields
compute Q_{-1} pressure and div(v)
compute int p dV
compute divv on markers
compute L2 errors v,p,div(v)
export to paraview
```

The chosen format for the vtu file is Q_{-1}: each element is exported separately, and 4 corner values for each u, v, p, $div(v)$ field.

The bench parameter allows to choose which benchmark is carried out.

All functions are jitted.

Because all elements are rectangles the mapping is trivial and analytical.

The following results are obtained by running the basch script `script_errors`.

The code relies on a direct solver as provided by scipy.

Manufactured solutions & experiments

Huang & Zhang manufactured solution

In Huang and Zhang [159] (2011) the authors propose the following manufactured solution:

$$g(x,y) = 2^8(x-x^2)^2(y-y^2)^2 = 2^8 f(x)h(y) = 256 f(x)h(y)$$

The velocity is given by

$$\vec{\mathrm{v}} = \begin{pmatrix} g_y \\ -g_x \end{pmatrix} = \begin{pmatrix} 256 f h' \\ -256 f' h \end{pmatrix}$$

and the pressure is defined as

$$p = -g_{xx} = -256 f'' h$$

We have (assuming viscosity is 1)

$$\vec{f} = -\Delta\vec{\mathrm{v}} + \vec{\nabla} p = 256 \begin{pmatrix} -\Delta(fh') - f'''h \\ -\Delta(-f'h) - f''h' \end{pmatrix} = 256 \begin{pmatrix} -f''h' - fh''' - f'''h \\ f'''h + f'h'' - f''h' \end{pmatrix}$$

with

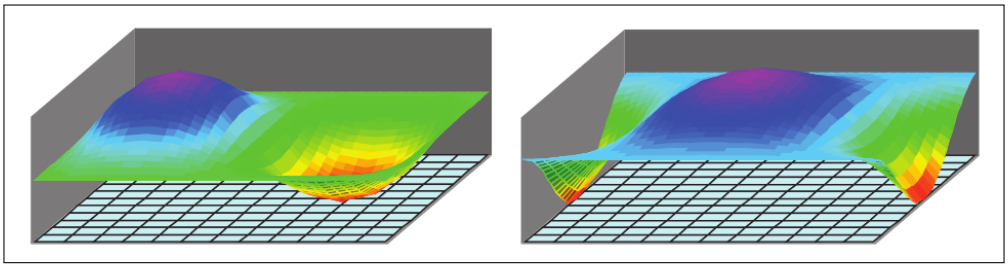
$$\begin{aligned} f(x) &= (x-x^2)^2 \\ f'(x) &= 2(1-2x)(x-x^2) \\ f''(x) &= 2(1-6x+6x^2) \\ f'''(x) &= 24x - 12 \end{aligned}$$

We have

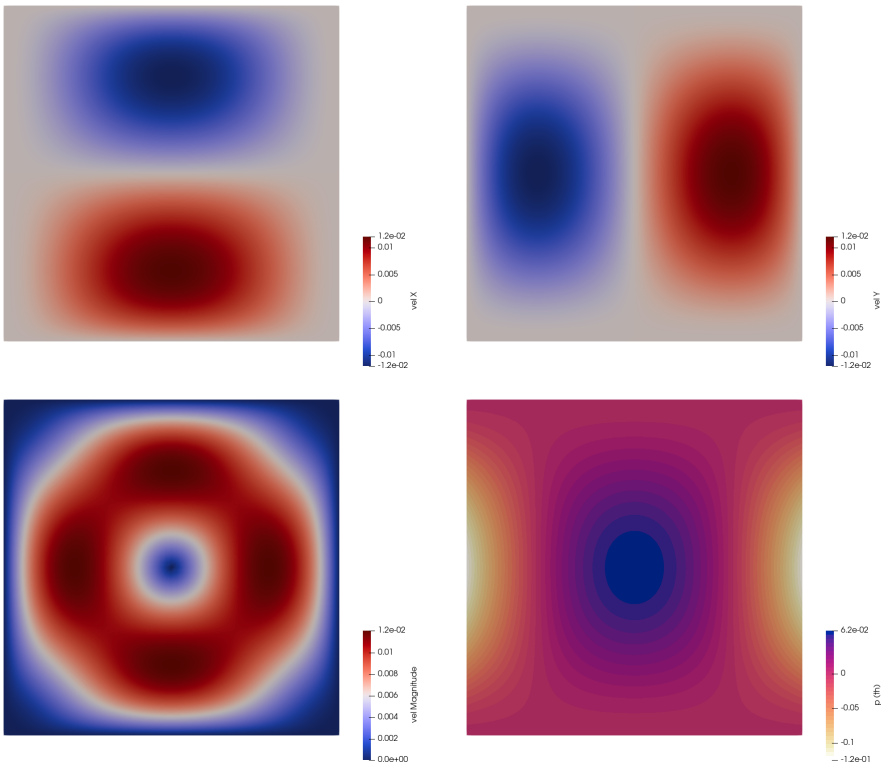
$$\begin{aligned} \iint_\Omega p(x,y)dxdy &= -256 \int_0^1 \int_0^1 f''(x)h(y)dxdy \\ &= -256 \int_0^1 f''(x)dx \cdot \int_0^1 h(y)dy \\ &= -256 \int_0^1 2(1-6x+6x^2)dx \cdot \int_0^1 (y-y^2)^2 dy \\ &= -256 \cdot 0 \cdot \frac{1}{30} \\ &= 0 \end{aligned} \tag{161.3}$$

and the pressure is zero at all four corners:

$$p(0,0) = p(1,1) = p(0,1) = p(1,0) = 0$$



Taken from [367]. Second component of velocity and pressure fields.



Figures obtained with Paraview. Visualisation of the analytical solution. factor 2^8 missing.

They also report the following rates:

Table 2 Errors $\boldsymbol{e}_h = \boldsymbol{u} - \boldsymbol{u}_I$ and $\varepsilon_h = p - p_I$ for $Q_{2,1;1,2}$ div-free element

	$\lvert\boldsymbol{e}_h\rvert_{H^1}$	h^n	$\lvert\boldsymbol{e}_h\rvert_{L^2}$	h^n	$\lvert\boldsymbol{e}_h\rvert_{l_\infty}$	h^n	$\lVert\varepsilon_h\rVert_{L^2}$	h^n	$\lVert\varepsilon_h\rVert_{l_\infty}$	h^n
2	0.0000		0.00000		0.00000		2.9212		7.200	
3	1.2110		0.10712		0.19947		1.8436	0.7	6.666	0.1
4	0.3119	2.0	0.02724	2.0	0.04576	2.1	0.4541	2.0	1.446	2.2
5	0.0783	2.0	0.00682	2.0	0.01100	2.1	0.1129	2.0	0.372	2.0
6	0.0196	2.0	0.00170	2.0	0.00270	2.0	0.0282	2.0	0.092	2.0
7	0.0049	2.0	0.00042	2.0	0.00067	2.0	0.0070	2.0	0.023	2.0
8	0.0012	2.0	0.00010	2.0	0.00016	2.0	0.0018	2.0	0.005	2.0
9	0.0003	2.0	0.00002	2.0	0.00004	2.0	0.0004	2.0	0.001	2.0

Note that the error convergence rates (in the L_2 norm) are identical for both velocity and pressure.

Donea & Huerta manufactured solution

Taken from the Donea & Huerta book [94]. We consider a two-dimensional problem in the square domain $\Omega = [0,1]\times[0,1]$, which possesses a closed-form analytical solution. The fluid viscosity is $\eta = 1$. The velocity and pressure fields are given by:

$$\begin{aligned} u(x,y) &= x^2(1-x)^2(2y-6y^2+4y^3) \\ &= x^2(1-x)^2 2y(1-3y+2y^2) \\ &= x^2(1-x)^2 2y(y-1)(2y-1) \\ v(x,y) &= -y^2(1-y)^2(2x-6x^2+4x^3) \\ &= -y^2(1-y)^2 2x(1-3x+2x^2) \\ &= -y^2(1-y)^2 2x(x-1)(2x-1) \\ p(x,y) &= x(1-x)-1/6 \end{aligned}$$

Note that the pressure obeys $\int_\Omega p\, dV = 0$.

The components of the body force $\vec{b}$ are then

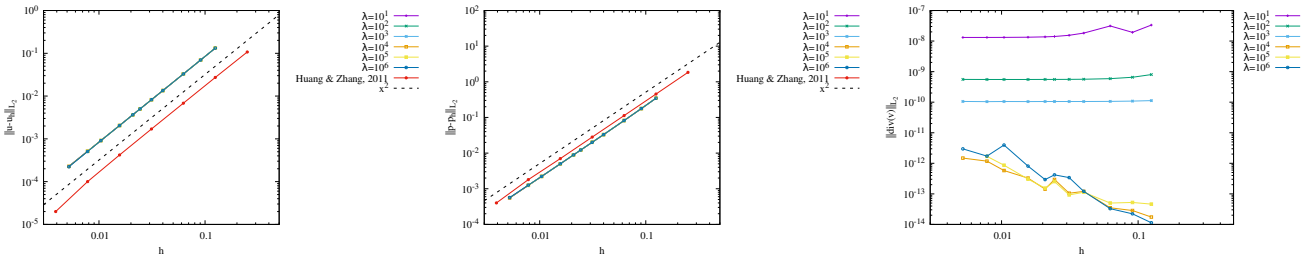
$$\begin{aligned} b_x &= (12-24y)x^4+(-24+48y)x^3+(-48y+72y^2-48y^3+12)x^2 \\ &\quad +(-2+24y-72y^2+48y^3)x+1-4y+12y^2-8y^3 \\ b_y &= (8-48y+48y^2)x^3+(-12+72y-72y^2)x^2 \\ &\quad +(4-24y+48y^2-48y^3+24y^4)x-12y^2+24y^3-12y^4 \end{aligned}$$

We find that the velocity fields are identical across both benchmarks (aside from 2^8 factor). Only the pressure fields differ!

Results

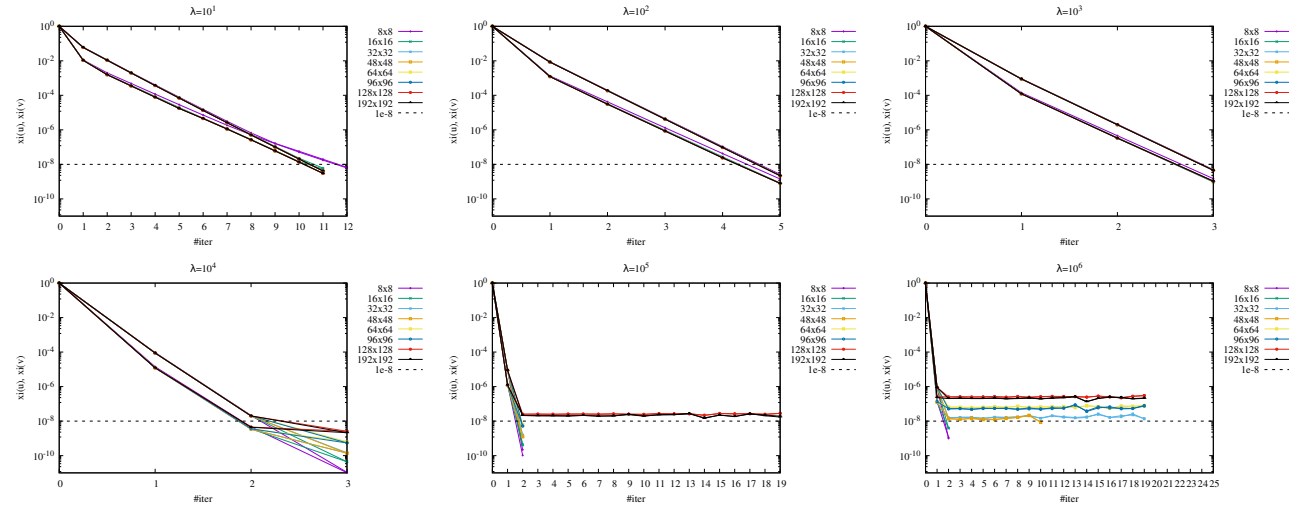
Huang & Zhang manufactured solution (`bench=2`)

The velocity error, pressure error and velocity divergence (in the L_2 norm) are measured as follows[6]:

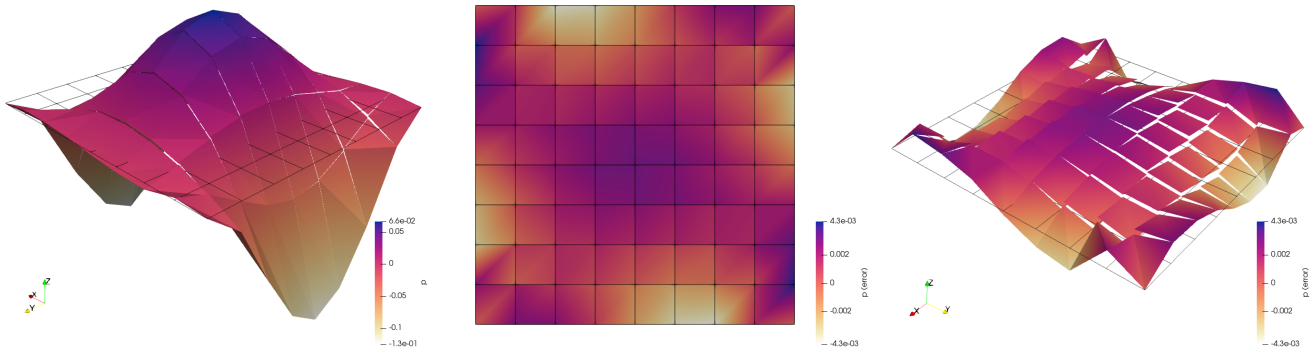


As in the publication both velocity and pressure errors converge quadratically. However there seems to be a factor 5 between my results and theirs for velocity, and a factor 1.414 between my results and theirs for pressure. I could explain the second by how h is computed (element edge vs. element diagonal), but the first one?

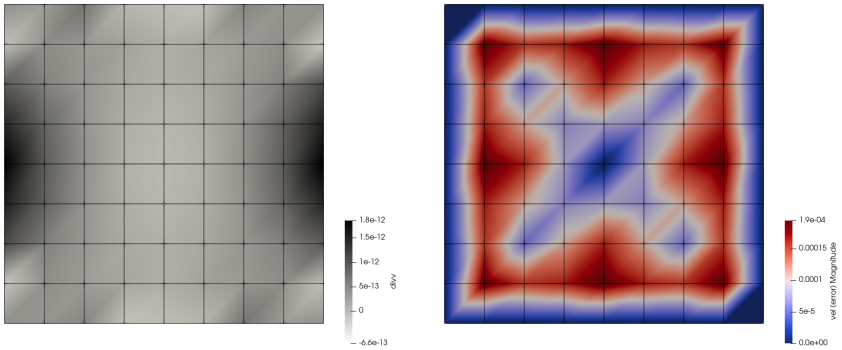
On the following plots I show the convergence of the iterated penalty method as a function of resolution for different values of the penalty coefficient λ:



Note that at high resolution the cases $\lambda = 10^5$ and $\lambda = 10^6$ fail to converge to the desired relative tolerance 10^{-8}.



[6] λ is the penalty parameter.



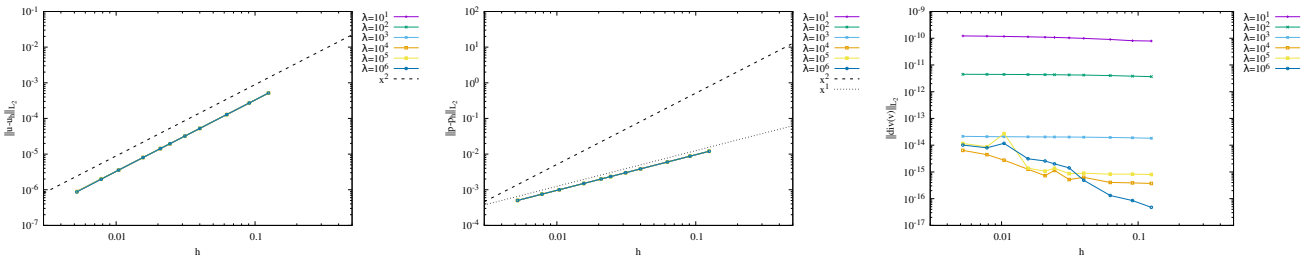
Obtained on 8×8 grid (factor 2^8 is absent). We find that the divergence is indeed very low. Note that Paraview is misleading: it looks like the squares are divived into 2 triangles but in reality it is a visualisation artefact!

It looks like pressure is continuous on the edges of the domain, which is indeed what we expect.

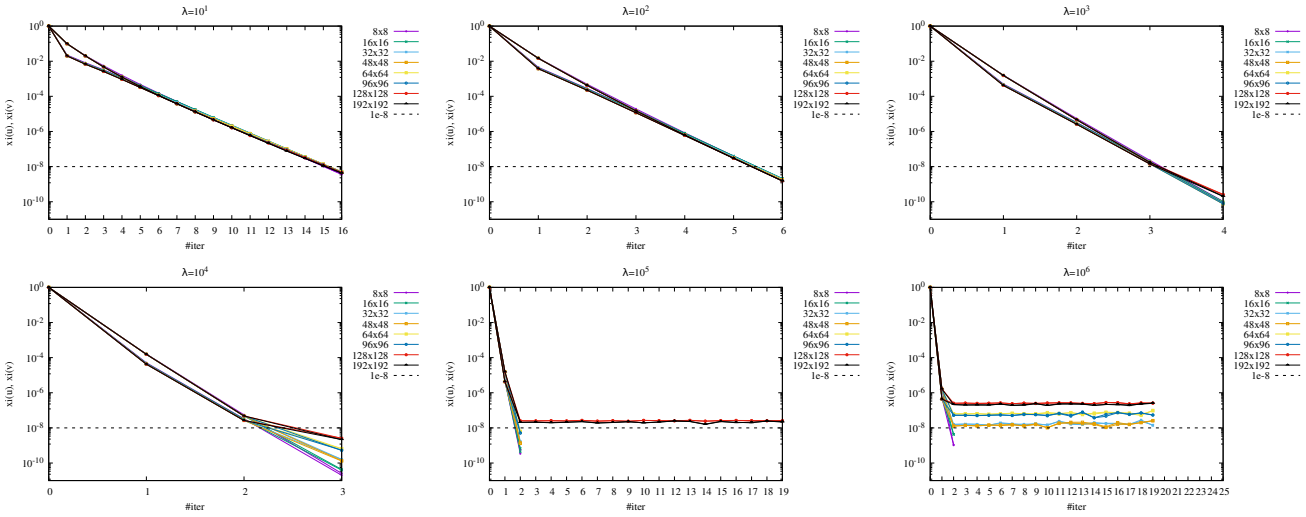
After having successfully reproduced the results of Huang and Zhang [159] I now carry out the other manufactured solution.

Donea & Huerta manufactured solution (`bench=1`)

On the following figures I show the velocity error, pressure error and velocity divergence in the L_2 norm. This time the pressure only convergences linearly:

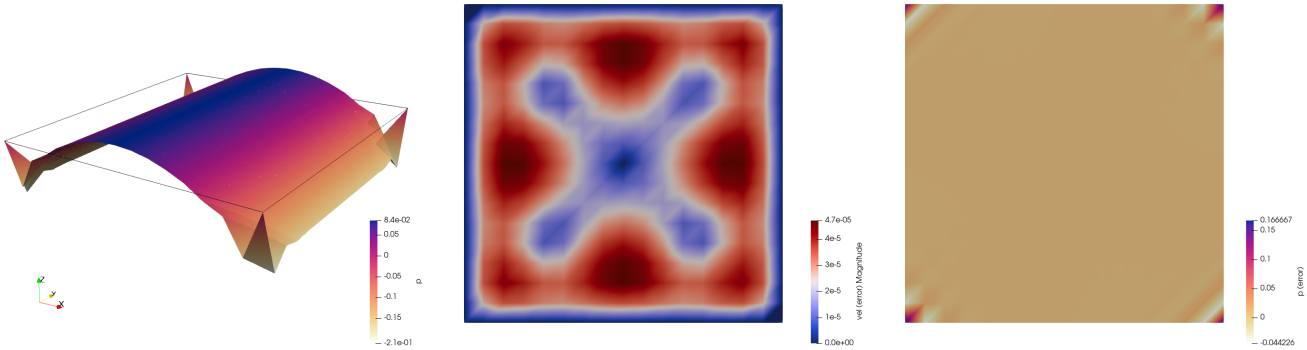


We find that the L_2 norm of velocity divergence is very low and the higher λ the lower the divergence.



We find that for $\lambda \geq 10^2$ the number of iterations becomes really small. Also the number of iterations to reach convergence decreases when λ increases. Any value higher than 10^4 does not improve the convergence, and even worse at high resolution $\lambda = 10^6$ does not converge anymore.

Let us use a 16×16 mesh with $\lambda = 10^3$ and let us look at the error fields, i.e. $v_h - v$ and $p_h - p$:



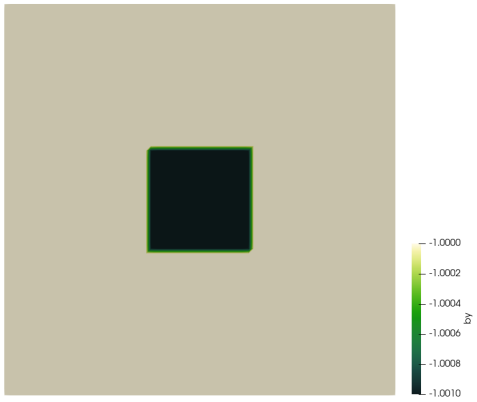
Left: pressure field; Middle: velocity error; Right: pressure error.

The analytical pressure field is $p(x, y) = x(1 - x) - 1/6$. The 1/6=0.16667 term insures that $\int pdV = 0$. We see that the constraints $p = 0$ in the corners yield an error that is exactly 1/6 there (Everywhere else the error is small, the continuity requirements on the edges do not interfere with the true solution). This is a big problem that is not discussed in the original paper: is there a way around this?

Remark: after having computed the pressure the code computes the integral of the pressure on the domain and the recovered values are extremely small ($< 10^{-15}$).

The sinking block (`bench=3`)

This is a typical test designed to check whether a particular FE pair is suitable for buoyancy driven flow (e.g. Thieulot and Bangerth [328], 2022). In this version I keep the viscosity constant in the entire domain with $\eta = 1$. The body force is as follows: in a square of size 0.25×0.25 in the middle of the domain $\rho\vec{g} = -1.001\ \vec{e}_y$ while it is $-1\ \vec{e}_y$ outside of it:



Vertical component of the body force. It corresponds to a gravity $|\vec{g}| = 1$ and a density difference of 0.1%.

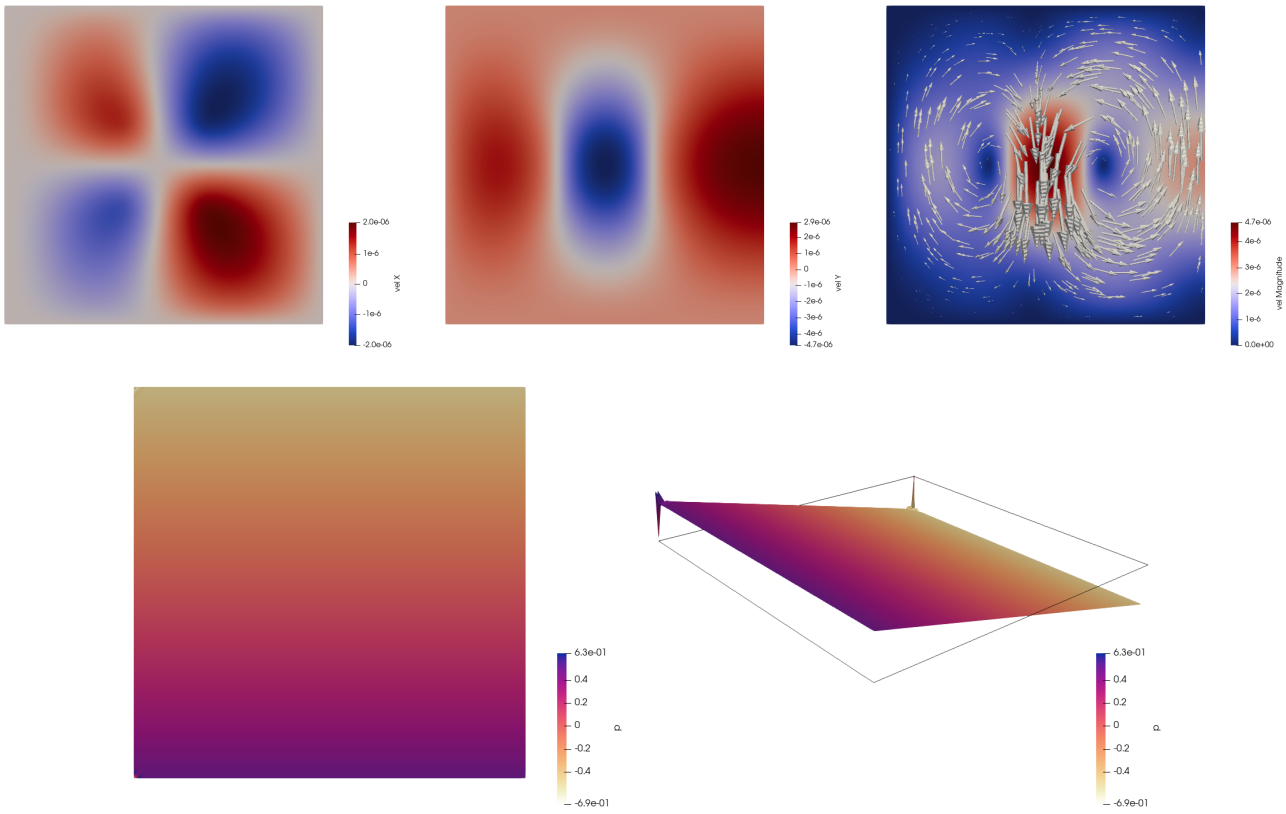
The reason for choosing this very small density difference is justified by our experiments in Thieulot and Bangerth [328] where it revealed that some FE pairs could not yield an accurate velocity solution in the presence of a background hydrostatic pressure dominating the dynamic pressure.

The boundary conditions are as follows[7]:

- Left: no slip
- Right: free slip
- Bottom: no slip
- Top: free slip

[7] The reasons for this peculiar choice will be clear very soon

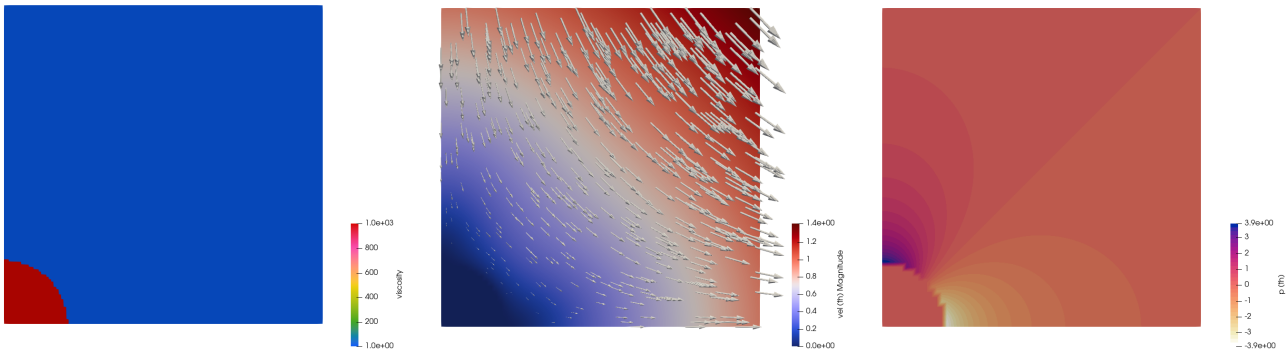
The solution is as follows (note that this is not a benchmark *stricto sensu* since there is no analytical solution available):



We find that the velocity field conforms to the boundary conditions and that the pressure is (visually) hydrostatic, *but* it shows two spikes with abnormal values on the top left and bottom left corners (better seen on the right '3d' plot): its value is there zero! Rather interestingly this happens on the left side, where no-slip boundary conditions are prescribed, but not on the right side with free-slip. This shows the complex interplay between velocity space and the constraint $p = 0$ at the corners, which casts a shadow on the practical use of this element in geodynamics where it is common to have no-slip and free-slip boundary conditions on various sides of the domain.

The viscous inclusion (`bench=4`)

This benchmark originates in Schmid and Podlachikov [283] (2003) in which the authors derive a simple analytic solution for the pressure and velocity fields for a circular inclusion under simple shear and it was used in [88, 315, 97, 200, 126].



Left: viscosity field on 128 × 128 mesh; Middle: analytical velocity field; Right: analytical pressure field.

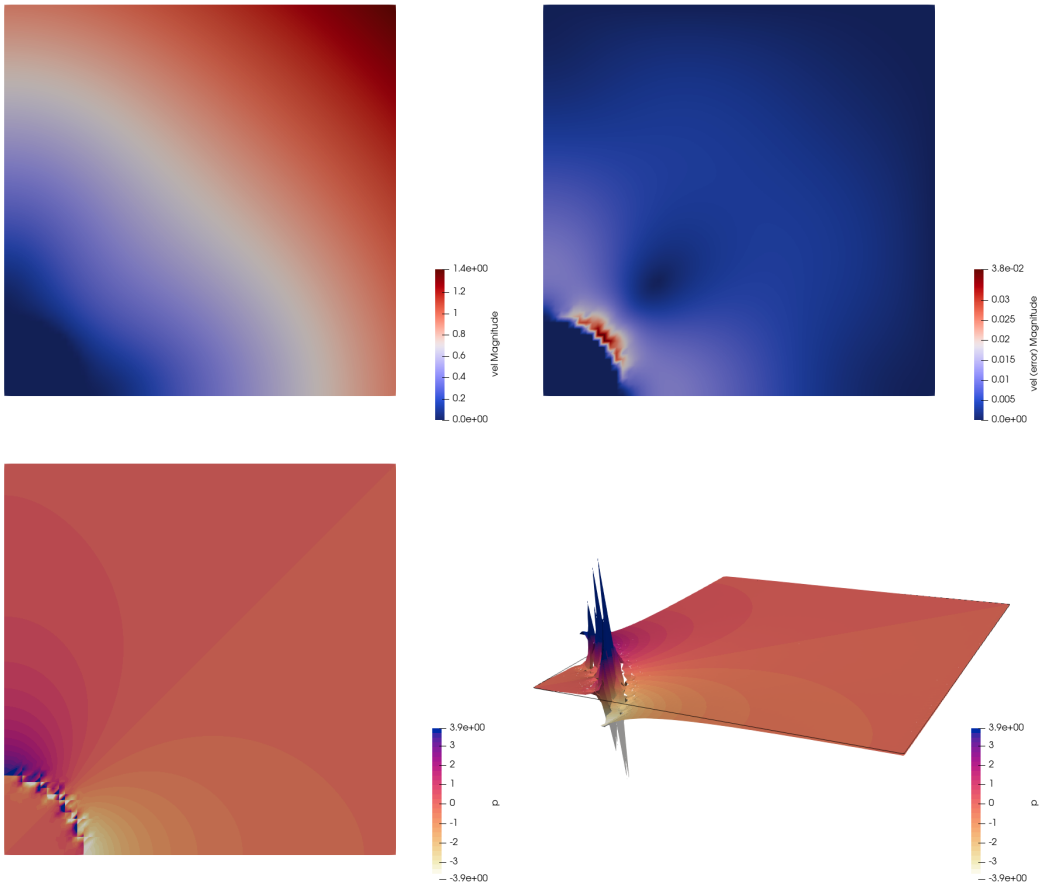
A characteristic of the analytical solution is that the pressure is zero inside the inclusion (a circle centered on the origin), while outside it follows the relation

$$p_m = 4\dot{\epsilon}\frac{\eta_m(\eta_i - \eta_m)}{\eta_i + \eta_m}\frac{r_i^2}{r^2}\cos(2\theta)$$

where $\eta_i = 10^3$ is the viscosity of the inclusion and $\eta_m = 1$ is the viscosity of the background media, $\theta = \tan^{-1}(y/x)$, and $\dot{\epsilon} = 1$ is the applied strain rate. The velocity expression is too complex to write here (see solution_solvi function in the code).

One important observation with this benchmark is the fact that the velocity is not zero even far away from the inclusion, so that the analytical (velocity) solution must be imposed on the sides. Also, because of symmetry, it is often run on the top quadrant $x > 0$, $y > 0$.

Boundary conditions are free slip on left and bottom boundaries (axes of symmetry) and the analytical solution is prescribed on the right and top boundaries. Viscosity is prescribed at the elemental level (i.e. all quadrature points inside an element are assigned the same viscosity).

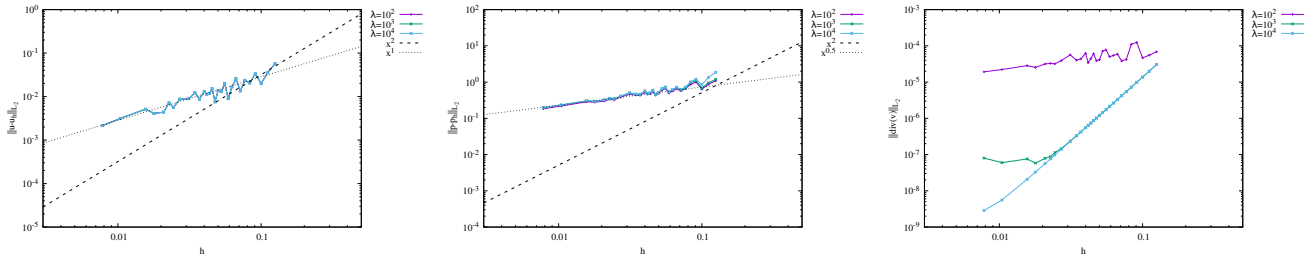


Results obtained on 64 × 64 grid, $\lambda = 10^3$. Pressure has been cut off at the analytical min/max for comparison. Measured overshoot undershoot are ±16.

In order for this benchmark to be usable in the context of this element implementation (which automatically enforces $\int p\, dV = 0$), the analytical solution should verify $\int p\, dV = 0$. In the inclusion $p = 0$ so we look at the following integral (the constants have been removed for clarity):

$$\begin{aligned}\int p\, dV \propto \iint \frac{1}{r^2}\cos(2\theta)\, dV &= \iint \frac{1}{r^2}(\cos^2\theta - \sin^2\theta)dV \\ &= \iint \frac{1}{r^2}\left(\frac{x^2}{x^2+y^2} - \frac{y^2}{x^2+y^2}\right) \\ &= 0 \end{aligned} \tag{161.4}$$

by symmetry along the diagonal (see also analytical pressure field figure at the beginning of this section). It then makes sense to compute the velocity and pressure error rates:



We recover $\mathcal{O}(h^1)$ convergence for the velocity error and $\mathcal{O}(h^{1/2})$ pressure error convergence, as in Thieulot and Bangerth [328] (2022). Unsurprisingly this element does not (cannot) perform any better than the $\boldsymbol{Q}_2 \times P_{-1}$ element in terms of error rates, since these are controlled by the fact that the viscosity discontinuity does not align with element edges.

Discussion/Conclusions/Open questions

- The grid/vertex layout is somewhat awkward to build. In practice any naive ordering would need re-ordering (reverse Cuthill-Mckee, Sloan, or other) to get a matrix with a reasonable skyline.
- The pressure space is too complex to build so we *have* to use iterative penalty method. It seems to converge reasonably nice for $\lambda = 10 - 100$ so this should not be too much of a problem for an iterative solver.
- $Q_{21} \times Q_{12}$ velocity space smaller than than $Q_2 \times Q_2$ so the velocity block smaller for a given number of elements (although it is not spectacular). Also, the linear system is as big as the viscosity block, so in case one uses a direct solver, cheaper than solving matrix coming from mixed formulation
- bonus: divergence free element!
- with Q2Q1, we must iterate (outer solver to arrive at solution, FGMRES, or PCG. With this element, outer iterations (iterated penalty method) also necessary. What is best? Zhang [368] states: " . So it is not possible to find a local basis for Ph in general, and it is not possible to derive a linear system of equations for (2.6), different from all other mixed finite methods. It seems that the definition of Ph (2.5) is not practical to implement th mixed element. But on the other side, it is the special interest of the new formulation where the space Ph is not implemented at all, and the discrete solutions approximating the pressure function in the Stokes equations will be obtained as byproducts. So there will be only one finite element space for the velocity, and the mixed element pair becomes a single element. This does not only greatly simplify the coding work, but also avoids the difficulty of solving non-positive definite systems of linear equations encountered in the classic mixed element method. "
- quid of pressure not zero at corners? write Zhang ? try different mms
- matrix does not change during the iterations, this can be exploited in case a direct solver like MUMPS is used (analysis and factorisation only needed once)!
- can free surface be envisaged?
- quid of viscosity constrasts?

Upon my contacting Prof. Zhang, his answer was:

> As p is the Lagrange multiplier for enforcing div u=0, then p should be in the divergence space of u.
>
> If $u = 0$ on the boundary, then $divu = 0$ at boundary corners. So, we have to choose the exact solution $p = 0$ at corners.
>
> This way the weak solution in $H^2 \times H^1$ space is the strong solution.
>
> Now if $p \neq 0$ at corners, one can still find weak solutions in the $H^1 \times L^2$ space. But the current element does not produce an optimal order solution for p, (still the optimal order convergence for u). p_h converges at 1/2 order or so.
>
> To get an optimal order convergent p_h in this case, we need to add piecewise polynomial bubbles at the corner square elements so that div u is a two piece non-zero constants at each corner.

Based on this answer, this concludes my study.

Stone 162

Testing the $Q_1^\star \times P_0$ element [WIP]



python_codes/fieldstone_162/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_162`

This stone is not finished. It does not work. There is something fundamental I do not understand and I do not know how to fix it yet.

According to Boffi, Brezzi, and Fortin [27] (p.90) one can make the $\boldsymbol{Q}_1 \times P_0$ stable by adding *one* internal degree of freedom. This then translates as follows:

$$u_h(r,s) = \overset{x}{\mathcal{N}}_1(r,s)u_1 + \overset{x}{\mathcal{N}}_2(r,s)u_2 + \overset{x}{\mathcal{N}}_3(r,s)u_3 + \overset{x}{\mathcal{N}}_4(r,s)u_4 + \overset{x}{\mathcal{N}}_5(r,s)\alpha \tag{162.1}$$

$$v_h(r,s) = \overset{y}{\mathcal{N}}_1(r,s)v_1 + \overset{y}{\mathcal{N}}_2(r,s)v_2 + \overset{y}{\mathcal{N}}_3(r,s)v_3 + \overset{y}{\mathcal{N}}_4(r,s)v_4 + \overset{y}{\mathcal{N}}_5(r,s)\alpha \tag{162.2}$$

where $\overset{x}{\mathcal{N}}_i$ and $\overset{y}{\mathcal{N}}_i$ are the standard Q_1 basis functions, α is the additional internal dof and

$$\overset{x}{\mathcal{N}}_5(r,s) = r(1-r^2)(1-s^2) \tag{162.3}$$

$$\overset{y}{\mathcal{N}}_5(r,s) = s(1-r^2)(1-s^2) \tag{162.4}$$

with

$$\frac{\partial \overset{x}{\mathcal{N}}_5}{\partial r}(r,s) = (1-3r^2)(1-s^2) \tag{162.5}$$

$$\frac{\partial \overset{x}{\mathcal{N}}_5}{\partial s}(r,s) = r(1-r^2)(-2s) \tag{162.6}$$

$$\frac{\partial \overset{y}{\mathcal{N}}_5}{\partial r}(r,s) = s(-2r)(1-s^2) \tag{162.7}$$

$$\frac{\partial \overset{y}{\mathcal{N}}_5}{\partial s}(r,s) = (1-r^2)(1-3s^2) \tag{162.8}$$

Note that in this form the basis functions are not a partition of unity.

If we write the basis functions as follows

$$\begin{aligned}
\overset{x}{\mathcal{N}}_1(r,s) &= \frac{1}{4}(1-r)(1-s) && (162.9)\\
\overset{x}{\mathcal{N}}_2(r,s) &= \frac{1}{4}(1+r)(1-s) && (162.10)\\
\overset{x}{\mathcal{N}}_3(r,s) &= \frac{1}{4}(1+r)(1+s) && (162.11)\\
\overset{x}{\mathcal{N}}_4(r,s) &= \frac{1}{4}(1-r)(1+s) && (162.12)\\
\overset{x}{\mathcal{N}}_5(r,s) &= r(1-r^2)(1-s^2) && (162.13)
\end{aligned}$$

then they fail the zeroth-order consistency check, i.e. for $f(r,s)=C$:

$$f^h(r,s)=\sum_{i=1}^{5}\overset{x}{\mathcal{N}}_i(r,s)f_i=C\sum_{i=1}^{5}\overset{x}{\mathcal{N}}_i(r,s)\neq C \tag{162.14}$$

Looking at the $Q_1^+\times Q_1$ of Section 8.3.8, which also showcases a bubble in the center of the element of the form $b(r,s)=(1-r^2)(1-s^2)\phi(r,s)$ where ϕ is a bilinear function, we can set

$$\begin{aligned}
\overset{x}{\mathcal{N}}_1(r,s) &= \frac{1}{4}(1-r)(1-s)-\frac{1}{4}b(r,s) && (162.15)\\
\overset{x}{\mathcal{N}}_2(r,s) &= \frac{1}{4}(1+r)(1-s)-\frac{1}{4}b(r,s) && (162.16)\\
\overset{x}{\mathcal{N}}_3(r,s) &= \frac{1}{4}(1+r)(1+s)-\frac{1}{4}b(r,s) && (162.17)\\
\overset{x}{\mathcal{N}}_4(r,s) &= \frac{1}{4}(1-r)(1+s)-\frac{1}{4}b(r,s) && (162.18)\\
\overset{x}{\mathcal{N}}_5(r,s) &= b(r,s) && (162.19)\\
b(r,s) &= r(1-r^2)(1-s^2) && (162.20)
\end{aligned}$$

Then these basis functions automatically pass the zeroth-order consistency check, i.e. for $f(r,s)=C$:

$$f^h(r,s)=\sum_{i=1}^{5}\overset{x}{\mathcal{N}}_i(r,s)f_i=C\sum_{i=1}^{5}\overset{x}{\mathcal{N}}_i(r,s)=C \tag{162.21}$$

First-order consistency check $f(r,s)=r$ (or $f(r,s)=s$):

$$\begin{aligned}
f^h(r,s) &= \sum_{i=1}^{5}\mathcal{N}_i(r,s)f_i\\
&= \sum_{i=1}^{5}\mathcal{N}_i(r,s)r_i\\
&= \overset{x}{\mathcal{N}}_1(r,s)(-1)+\overset{x}{\mathcal{N}}_2(r,s)(+1)+\overset{x}{\mathcal{N}}_3(r,s)(+1)+\overset{x}{\mathcal{N}}_4(r,s)(-1)+\overset{x}{\mathcal{N}}_5(r,s)(0)\\
&= -\overset{x}{\mathcal{N}}_1(r,s)+\overset{x}{\mathcal{N}}_2(r,s)+\overset{x}{\mathcal{N}}_3(r,s)-\overset{x}{\mathcal{N}}_4(r,s)\\
&= r && (162.22)
\end{aligned}$$

Second-order consistency check $f(r,s) = rs$ ($f_1 = (-1)(-1) = 1$, $f_2 = (+1)(-1) = -1$, etc ...)

$$\begin{aligned}
f^h(r,s) &= \sum_{i=1}^{5} \overset{x}{\mathcal{N}}_i(r,s) f_i \\
&= \sum_{i=1}^{5} \overset{x}{\mathcal{N}}_i(r,s) r_i s_i \\
&= \overset{x}{\mathcal{N}}_1(r,s)(+1) + \overset{x}{\mathcal{N}}_2(r,s)(-1) + \overset{x}{\mathcal{N}}_3(r,s)(+1) + \overset{x}{\mathcal{N}}_4(r,s)(-1) + \overset{x}{\mathcal{N}}_5(r,s)(0) \\
&= \overset{x}{\mathcal{N}}_1 - \overset{x}{\mathcal{N}}_2 + \overset{x}{\mathcal{N}}_3 - \overset{x}{\mathcal{N}}_4 \\
&= \left(\frac{1}{4}(1-r)(1-s) - \frac{1}{4}b(r,s)\right) - \left(\frac{1}{4}(1+r)(1-s) - \frac{1}{4}b(r,s)\right) \\
&+ \left(\frac{1}{4}(1+r)(1+s) - \frac{1}{4}b(r,s)\right) - \left(\frac{1}{4}(1-r)(1+s) - \frac{1}{4}b(r,s)\right) \\
&= \frac{1}{4}(1-r)(1-s) - \frac{1}{4}(1+r)(1-s) + \frac{1}{4}(1+r)(1+s) - \frac{1}{4}(1-r)(1+s) \\
&= \frac{1}{2}(-r)(1-s) + \frac{1}{2}(+r)(1+s) \\
&= rs
\end{aligned} \tag{162.23}$$

We find that the basis functions can represent a bilinear field exactly.

Let us now turn to the derivatives. For $f(r,s) = C$, we have

$$\begin{aligned}
\frac{\partial f}{\partial r}|_h(r,s) &= \sum_{i=1}^{5} \frac{\partial \overset{x}{\mathcal{N}}_i}{\partial r} f_i && (162.24\\
&= C \sum_{i=1}^{5} \frac{\partial \overset{x}{\mathcal{N}}_i}{\partial r} && (162.25\\
&= C\left[\frac{1}{4}(-1)(1-s) - \frac{1}{4}\frac{\partial b}{\partial r} + \frac{1}{4}(+1)(1-s) - \frac{1}{4}\frac{\partial b}{\partial r} + \frac{1}{4}(+1)(1+s) - \frac{1}{4}\frac{\partial b}{\partial r} + \frac{1}{4}(-1)(1+s) - \frac{1}{4}\frac{\partial b}{\partial r}\right] && (162.26\\
&= C\left[\frac{1}{4}(-1)(1-s) + \frac{1}{4}(+1)(1-s) + \frac{1}{4}(+1)(1+s) + \frac{1}{4}(-1)(1+s)\right] && (162.27\\
&= 0 && (162.28
\end{aligned}$$

Implementation

This is a rather awkward situation where there is only one dof in the center and not two. This means that each element has 4x2+1=9 dofs. I then logically organise the unknowns/dofs inside an element as follows:

$$\vec{\mathcal{V}} = (u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4, \alpha)$$

I then need to carefully go through the derivations of Section 10.8.3:

$$\int_\Omega \vec{\nabla}\vec{\omega} : \boldsymbol{\sigma}\, d\Omega = \int_\Omega \vec{\omega} \cdot \vec{b}\, d\Omega + \int_\Gamma \vec{\omega} \cdot \vec{t}\, d\Gamma \tag{162.29}$$

$$\int_\Omega q \vec{\nabla} \cdot \vec{\mathrm{v}}\, d\Omega = 0 \tag{162.30}$$

where $\vec{\omega}$ and q are the velocity and pressure test functions respectively.

Since we are going to prescribe the velocity on all sides there is no precribed traction on the boundary so that $\int_\Gamma \vec{\omega} \cdot \vec{t}\, d\Gamma = 0$.

Let us first look at the gradient operator $\boldsymbol{B}$:

$$
\vec{\dot{\varepsilon}}_h = \begin{pmatrix} \frac{\partial u_h}{\partial r} \\ \frac{\partial v_h}{\partial s} \\ \frac{\partial u_h}{\partial s} + \frac{\partial v_h}{\partial r} \end{pmatrix} = \begin{pmatrix} \sum_i \frac{\partial \overset{x}{\mathcal{N}}_i}{\partial r} u_i + \frac{\partial \overset{x}{\mathcal{N}}_5}{\partial r}\alpha \\ \sum_i \frac{\partial \overset{y}{\mathcal{N}}_i}{\partial s} v_i + \frac{\partial \overset{y}{\mathcal{N}}_5}{\partial s}\alpha \\ \sum_i (\frac{\partial \overset{x}{\mathcal{N}}_i}{\partial s} u_i + \frac{\partial \overset{y}{\mathcal{N}}_i}{\partial r} v_i) + \frac{\partial \overset{x}{\mathcal{N}}_5}{\partial s}\alpha + \frac{\partial \overset{y}{\mathcal{N}}_5}{\partial r}\alpha \end{pmatrix}
$$

$$
= \underbrace{\begin{pmatrix} \frac{\partial \overset{x}{\mathcal{N}}_1}{\partial x} & 0 & \frac{\partial \overset{x}{\mathcal{N}}_2}{\partial x} & 0 & \frac{\partial \overset{x}{\mathcal{N}}_3}{\partial x} & 0 & \frac{\partial \overset{x}{\mathcal{N}}_4}{\partial x} & 0 & \frac{\partial \overset{x}{\mathcal{N}}_5}{\partial x} \\ 0 & \frac{\partial \overset{y}{\mathcal{N}}_1}{\partial y} & 0 & \frac{\partial \overset{y}{\mathcal{N}}_2}{\partial y} & 0 & \frac{\partial \overset{y}{\mathcal{N}}_3}{\partial y} & 0 & \frac{\partial \overset{y}{\mathcal{N}}_4}{\partial y} & \frac{\partial \overset{y}{\mathcal{N}}_5}{\partial y} \\ \frac{\partial \overset{x}{\mathcal{N}}_1}{\partial y} & \frac{\partial \overset{y}{\mathcal{N}}_1}{\partial x} & \frac{\partial \overset{x}{\mathcal{N}}_2}{\partial y} & \frac{\partial \overset{y}{\mathcal{N}}_2}{\partial x} & \frac{\partial \overset{x}{\mathcal{N}}_3}{\partial y} & \frac{\partial \overset{y}{\mathcal{N}}_3}{\partial x} & \frac{\partial \overset{x}{\mathcal{N}}_4}{\partial y} & \frac{\partial \overset{y}{\mathcal{N}}_4}{\partial x} & \frac{\partial \overset{x}{\mathcal{N}}_5}{\partial y} + \frac{\partial \overset{y}{\mathcal{N}}_5}{\partial x} \end{pmatrix}}_{\boldsymbol{B}} \cdot \underbrace{\begin{pmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_3 \\ v_3 \\ \alpha \end{pmatrix}}_{\vec{\nu}}
$$

I will then focus on the rhs term

$$
\int_\Omega \vec{\omega} \cdot \vec{b}\, d\Omega = \int_\Omega (\omega_x\ \omega_y) \cdot \begin{pmatrix} b_x \\ b_y \end{pmatrix}\, d\Omega
$$

In the case of the $\boldsymbol{Q}_1^\star \times P_0$ element we can write the components of the test function as follows:

$$
\omega_x(x,y) = \overset{x}{\mathcal{N}}_1(x,y)\overset{x}{\omega}_1 + \overset{x}{\mathcal{N}}_2(x,y)\overset{x}{\omega}_2 + \overset{x}{\mathcal{N}}_3(x,y)\overset{x}{\omega}_3 + \overset{x}{\mathcal{N}}_4(x,y)\overset{x}{\omega}_4 + \overset{x}{\mathcal{N}}_5(x,y)\beta \qquad (162.31)
$$

$$
\omega_y(x,y) = \overset{y}{\mathcal{N}}_1(x,y)\overset{y}{\omega}_1 + \overset{y}{\mathcal{N}}_2(x,y)\overset{y}{\omega}_2 + \overset{y}{\mathcal{N}}_3(x,y)\overset{y}{\omega}_3 + \overset{y}{\mathcal{N}}_4(x,y)\overset{y}{\omega}_4 + \overset{y}{\mathcal{N}}_5(x,y)\beta \qquad (162.32)
$$

$$
\vec{\mathcal{W}}^T = (\overset{x}{\omega}_1, \overset{y}{\omega}_1, \overset{x}{\omega}_2, \overset{y}{\omega}_2, \overset{x}{\omega}_3, \overset{y}{\omega}_3, \overset{x}{\omega}_4, \overset{y}{\omega}_4, \beta)
$$

One can then write

$$
(\omega_x\ \omega_y) = (\overset{x}{\omega}_1, \overset{y}{\omega}_1, \overset{x}{\omega}_2, \overset{y}{\omega}_2, \overset{x}{\omega}_3, \overset{y}{\omega}_3, \overset{x}{\omega}_4, \overset{y}{\omega}_4, \beta) \cdot \begin{pmatrix} \overset{x}{\mathcal{N}}_1(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_1(x,y) \\ \overset{x}{\mathcal{N}}_2(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_2(x,y) \\ \overset{x}{\mathcal{N}}_3(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_3(x,y) \\ \overset{x}{\mathcal{N}}_4(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_4(x,y) \\ \overset{x}{\mathcal{N}}_5(x,y) & \overset{y}{\mathcal{N}}_5(x,y) \end{pmatrix}
$$

so that

$$(\omega_x\ \omega_y)\cdot\begin{pmatrix} b_x \\ b_y \end{pmatrix} = \vec{\mathcal{W}}^T\cdot\begin{pmatrix} \overset{x}{\mathcal{N}}_1(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_1(x,y) \\ \overset{x}{\mathcal{N}}_2(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_2(x,y) \\ \overset{x}{\mathcal{N}}_3(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_3(x,y) \\ \overset{x}{\mathcal{N}}_4(x,y) & 0 \\ 0 & \overset{y}{\mathcal{N}}_4(x,y) \\ \overset{x}{\mathcal{N}}_5(x,y) & \overset{y}{\mathcal{N}}_5(x,y) \end{pmatrix}\cdot\begin{pmatrix} b_x \\ b_y \end{pmatrix} = \vec{\mathcal{W}}^T\cdot\underbrace{\begin{pmatrix} b_x\,\overset{x}{\mathcal{N}}_1(x,y) \\ b_y\,\overset{y}{\mathcal{N}}_1(x,y) \\ b_x\,\overset{x}{\mathcal{N}}_2(x,y) \\ b_y\,\overset{y}{\mathcal{N}}_2(x,y) \\ b_x\,\overset{x}{\mathcal{N}}_3(x,y) \\ b_y\,\overset{y}{\mathcal{N}}_3(x,y) \\ b_x\,\overset{x}{\mathcal{N}}_4(x,y) \\ b_y\,\overset{y}{\mathcal{N}}_4(x,y) \\ b_x\,\overset{x}{\mathcal{N}}_5(x,y)+b_y\,\overset{y}{\mathcal{N}}_5(x,y) \end{pmatrix}}_{\vec{f}}$$

Finally let us turn to the $\mathbb{G}$ block. We have in the general case:

$$q = \vec{\mathcal{N}}_p^T\cdot\vec{\mathcal{Q}}$$

where $\vec{\mathcal{N}}_p$ is the vector[1] (of size $(m_P,1)$) of pressure basis functions and $\vec{\mathcal{Q}}$ is the vector (of size $(m_P,1)$) of test pressure dofs.

Let us now turn to the mass conservation equation. Since $\vec{\mathcal{N}}_p^T\cdot\vec{\mathcal{Q}} = \vec{\mathcal{Q}}^T\cdot\vec{\mathcal{N}}_p$ then

$$\int_\Omega q\vec{\nabla}\cdot\vec{\mathrm{v}}\,d\Omega = \int_\Omega \vec{\mathcal{Q}}^T\cdot\vec{\mathcal{N}}^p\vec{\nabla}\cdot\vec{\mathrm{v}}\,d\Omega = \vec{\mathcal{Q}}^T\cdot\int_\Omega\vec{\mathcal{N}}^p\vec{\nabla}\cdot\vec{\mathrm{v}}\,d\Omega$$

We have

$$\begin{aligned}
\vec{\nabla}\cdot\vec{\mathrm{v}} &= \dot{\varepsilon}_{xx}(\vec{\mathrm{v}})+\dot{\varepsilon}_{yy}(\vec{\mathrm{v}}) && (162.33)\\
&= \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}\cdot\begin{pmatrix} \dot{\varepsilon}_{xx}(\vec{\mathrm{v}}) \\ \dot{\varepsilon}_{yy}(\vec{\mathrm{v}}) \\ 2\dot{\varepsilon}_{xy}(\vec{\mathrm{v}}) \end{pmatrix}\\
&= \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}\cdot\vec{\dot{\varepsilon}}(\vec{\mathrm{v}})\\
&= \begin{pmatrix} 1 & 1 & 0 \end{pmatrix}\cdot\boldsymbol{B}\cdot\vec{\mathcal{V}} && (162.34)
\end{aligned}$$

so that

$$\begin{aligned}
\vec{\mathcal{N}}^p\vec{\nabla}\cdot\vec{\mathrm{v}} &= \vec{\mathcal{N}}^p\begin{pmatrix} 1 & 1 & 0 \end{pmatrix}\cdot\boldsymbol{B}\cdot\vec{\mathcal{V}}\\
&= \underbrace{\begin{pmatrix} \vec{\mathcal{N}}^p & \vec{\mathcal{N}}^p & 0 \end{pmatrix}}_{\boldsymbol{\mathcal{N}}^T}\cdot\boldsymbol{B}\cdot\vec{\mathcal{V}}\\
&= \boldsymbol{\mathcal{N}}^T\cdot\boldsymbol{B}\cdot\vec{\mathcal{V}} && (162.35)
\end{aligned}$$

where $\boldsymbol{N}_p$ is a $(m_p\times 3)$ matrix. Finally

$$\begin{aligned}
\int_\Omega q\vec{\nabla}\cdot\vec{\mathrm{v}}\,d\Omega &= \vec{\mathcal{Q}}^T\cdot\int_\Omega\boldsymbol{\mathcal{N}}^T\cdot\boldsymbol{B}\,d\Omega\cdot\vec{\mathcal{V}}\\
&= -\vec{\mathcal{Q}}^T\cdot\mathbb{G}^T\cdot\vec{\mathcal{V}} && (162.36)
\end{aligned}$$

where

$$\mathbb{G} = -\int_\Omega\boldsymbol{B}^T\cdot\boldsymbol{\mathcal{N}}\,d\Omega$$

[1] Remember that by default all vectors are column vectors.

In the case of the $\boldsymbol{Q}_1^\star \times P_0$ element there is only one pressure dof per element so that the corresponding pressure basis function is simply constant and equal to 1. Then

$$\boldsymbol{\mathcal{N}} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

and

$$\boldsymbol{B}^T \cdot \boldsymbol{\mathcal{N}} = \begin{pmatrix} \overset{x}{\partial\mathcal{N}_1} & 0 & \overset{x}{\partial\mathcal{N}_2} & 0 & \overset{x}{\partial\mathcal{N}_3} & 0 & \overset{x}{\partial\mathcal{N}_4} & 0 & \overset{x}{\partial\mathcal{N}_5} \\ 0 & \overset{y}{\partial\mathcal{N}_1} & 0 & \overset{y}{\partial\mathcal{N}_2} & 0 & \overset{y}{\partial\mathcal{N}_3} & 0 & \overset{y}{\partial\mathcal{N}_4} & \overset{y}{\partial\mathcal{N}_5} \\ \overset{x}{\partial\mathcal{N}_1} & \overset{y}{\partial\mathcal{N}_1} & \overset{x}{\partial\mathcal{N}_2} & \overset{y}{\partial\mathcal{N}_2} & \overset{x}{\partial\mathcal{N}_3} & \overset{y}{\partial\mathcal{N}_3} & \overset{x}{\partial\mathcal{N}_4} & \overset{y}{\partial\mathcal{N}_4} & \overset{x}{\partial\mathcal{N}_5} + \overset{y}{\partial\mathcal{N}_5} \end{pmatrix}^T \cdot \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \overset{x}{\partial\mathcal{N}_1} \\ \overset{y}{\partial\mathcal{N}_1} \\ \overset{x}{\partial\mathcal{N}_2} \\ \overset{y}{\partial\mathcal{N}_2} \\ \overset{x}{\partial\mathcal{N}_3} \\ \overset{y}{\partial\mathcal{N}_3} \\ \overset{x}{\partial\mathcal{N}_4} \\ \overset{y}{\partial\mathcal{N}_4} \\ \overset{x}{\partial\mathcal{N}_5} + \overset{y}{\partial\mathcal{N}_5} \end{pmatrix}$$

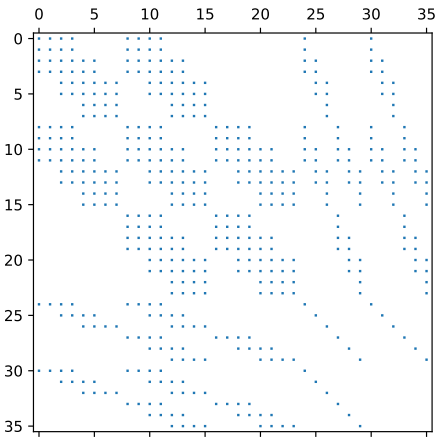
There are in total $NfemV = NV * ndfoV + nel$ velocity degrees of freedom and $NfemP = nel$ pressure degrees of freedom.

Imposing boundary conditions is rather straightforward since the additional dofs are never on the boundary.

When it comes to quadrature, the velocity basis functions contain quadratic terms so we'll use a 3×3 quadrature for the $\mathbb{K}$ block. However it is made clear by the authors that the bilinear form responsible for $\mathbb{G}$ must be underintegrated (1 point quadrature).

The code for this stone is based on STONE 14.

Remark: Since the additional dofs are internal one could resort to static condensation so as to reduce the size of the FEM system $\mathbb{K}$ block.



nonzero pattern for a 3×2 mesh.

Note that stone1.py is the code for which there is only 1 dof in the middle of each element while stone2.py is the code with two dofs in the middle.

Results

No matter what I try stone1 or stone2: if I use 3x3 quad points on all terms, or 3x3 $\mathbb{K}$ + 2x2 $\mathbb{G}$, velocity is ok, but chequerboard on pressure. If I use 1 quad pt for $\mathbb{G}$ velocity not ok, and pressure is wrong. If I use 2x2 quad pt for $\mathbb{K}$ I get nonsense or nan!

I wrote Prof Boffi 16 july 2024.

Stone 163

A false good idea doing parallel pic



F Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_163`

This stone was developed in collaboration with Arie van den Berg.

The first version of this code considers a mesh of $nelx \times nely$ elements spanning the unit square. The coordinates of the nodes are stored in the `x,y` arrays and the connectivity is `icon`. At each node of the mesh the velocity $(u,v) = (-0.5+y, 0.5-x)$ is prescribed:

```
do i=1,np
   u(i)=-(0.5-y(i))
   v(i)=(0.5-x(i))
end do
```

Inside the domain `swarm_n` markers are placed randomly:

```
do i=1,swarm_n
   call random_number(eta)
   call random_number(chi)
   swarm_x(i)=eta*Lx
   swarm_y(i)=chi*Ly
end do
```

The markers are then localised, i.e. one must find for each the element it resides in. The velocity is then computed on the marker by means of Q_1 basis functions and it is advected with a first order Euler step for simplicity:

```
do i=1,swarm_n

   ! find cell
   ielx=swarm_x(i)/Lx*nelx+1
   iely=swarm_y(i)/Ly*nely+1
   iel(i)=nelx*(iely-1)+ielx

   ! find local coordinates in element
```

```
   xmin=x(icon(1,iel(i)))
   xmax=x(icon(3,iel(i)))
   ymin=y(icon(1,iel(i)))
   ymax=y(icon(3,iel(i)))
   r=((swarm_x(i)-xmin)/(xmax-xmin)-0.5d0)*2.d0
   s=((swarm_y(i)-ymin)/(ymax-ymin)-0.5d0)*2.d0

   ! evaluate Q1 basis functions
   N(1)=0.25*(1-r)*(1-s)
   N(2)=0.25*(1+r)*(1-s)
   N(3)=0.25*(1+r)*(1+s)
   N(4)=0.25*(1-r)*(1+s)

   ! compute velocity
   swarm_u(i)=sum(N*u(icon(:,iel(i))))
   swarm_v(i)=sum(N*v(icon(:,iel(i))))

   ! advect marker
   swarm_x(i)=swarm_x(i)+swarm_u(i)*dt
   swarm_y(i)=swarm_y(i)+swarm_v(i)*dt

end do
```

Looking at this code we find that the background resolution of the mesh plays virtually no role, and that this loop is executed for each marker but localising and advecting one marker can be done completely independently from another marker. Assuming that a large number of markers is used, say $\mathcal{O}(10^7)$, one could be tempted to think of parallelisation to improve performance.

There are usually 2 main paradigms when it comes to parallelization, see for example Zelst, Crameri, Pusok, Glerum, Dannberg, and Thieulot [361] (2022): openmp and mpi. We fill now focus on mpi and therefore initialise mpi in the code as follows:

```
program opla
use mpi
[...]
call mpi_init(ierr)
call mpi_comm_size (mpi_comm_world,nproc,ierr)
call mpi_comm_rank (mpi_comm_world,iproc,ierr)
```

The code is then compiled with

```
> mpif90 -O3 stone.f90
```

and can be run in parallel as follows:

```
> mpirun -np 4 ./a.out
```

First approach

Although it is often a bad idea, one could wish to use the code above and modify it as little as possible while inserting the required mpi commands. Let us adopt here this (doomed[1]) approach for the sake of learning.

We then proceed to change

```
do i=1,swarm_n
   [...]
end do
```

[1] spoiler alert

into

```
do i=1+iproc,swarm_n,nproc
   print *,iproc,i
   [...]
end do
```

The reason for this is as follows: let us set `swarm_n`=15 and run the code with nproc=3. The print statement returns the following lines[2]:

```
 iproc=           0 takes care of marker           1
 iproc=           0 takes care of marker           4
 iproc=           0 takes care of marker           7
 iproc=           0 takes care of marker          10
 iproc=           0 takes care of marker          13
 iproc=           1 takes care of marker           2
 iproc=           1 takes care of marker           5
 iproc=           1 takes care of marker           8
 iproc=           1 takes care of marker          11
 iproc=           1 takes care of marker          14
 iproc=           2 takes care of marker           3
 iproc=           2 takes care of marker           6
 iproc=           2 takes care of marker           9
 iproc=           2 takes care of marker          12
 iproc=           2 takes care of marker          15
```

We see that each thread only deals with a subset of the markers. We then proceed to modify the last two lines of the big loop as follows:

```
   xtemp(i)=swarm_x(i)+swarm_u(i)*dt
   ytemp(i)=swarm_y(i)+swarm_v(i)*dt
```

where `xtemp` and `ytemp` are `swarm_n`-long arrays that have been set to zero beforehand. In this case the `xtemp` array would look like this at the end on each thread (the empty spaces indicate that nothing was written there):

```
         -------------------------------------------------
iproc=0  | X |   |   | X |   |   | X |   |   | X |   |   | X |   |   |
         -------------------------------------------------

         -------------------------------------------------
iproc=1  |   | X |   |   | X |   |   | X |   |   | X |   |   | X |   |
         -------------------------------------------------

         -------------------------------------------------
iproc=2  |   |   | X |   |   | X |   |   | X |   |   | X |   |   | X |
         -------------------------------------------------
```

All we have to do is to do a global summation of these arrays into `swarm_x` across all threads (same for `swarm_y`):

```
call mpi_allreduce(xtemp,swarm_x,swarm_n,mpi_double_precision,mpi_sum,mpi_comm_world,ierr)
call mpi_allreduce(ytemp,swarm_y,swarm_n,mpi_double_precision,mpi_sum,mpi_comm_world,ierr)
```

In the end we have parallelized this loop. We finish by adding mpi commands that allow for the timing of sections of code:

[2] I have reordered the lines for better readibility

```
start  = MPI_Wtime()
[...]
finish  = MPI_Wtime()
print *,'time=',finish-start,'s'
```

In practice one would use a RK4 algorithm to advect the markers. In order to approximate the cost of RK4 without actually implementing it since we are not interested in the advection itself we can wrap the inside of the loop inside an additional loop that enforces that localisation and advection happen 4 times for each marker (similar to RK4):

```
do i=1+iproc,swarm_n,nproc
   do k=1,4 ! to mimic cost of RK4

   end do
end do
```

Timings

We then proceed to run the code on 1,2,3,4 threads[3]:

```
advect:  0.19242236700000004             0
allreduce:   1.4708635000000081E-002     0

advect:  0.13948245200000001             1
advect:  0.14018181199999999             0
allreduce:   9.7357410000000005E-002     0
allreduce:   9.7526858999999994E-002     1

advect:  0.12392915000000000             1
advect:  0.12359175000000000             0
advect:  0.12411840600000000             2
allreduce:  0.19212144400000003          2
allreduce:  0.20365041300000000          0
allreduce:  0.20382717899999997          1

advect:  0.13753345000000000             1
advect:  0.13849821400000001             2
advect:  0.13913681200000000             3
advect:  0.13707424700000001             0
allreduce  0.20147564300000001           3
allreduce  0.20159588100000003           0
allreduce  0.20169826699999999           2
allreduce  0.20180693799999999           1
```

We immediately see the problem: when using only 1 thread (i.e. no parallelisation at all) the entire loop over all 10 million markers only takes 0.2s! This raises the question whether we should even have tried to parallelise this bit of code! Second, as soon as multiple threads are used the cost of the allreduce becomes as large as the cost to carry out the big loop. After all 0.2s to for all threads to send each other 2x 10 million double precision numbers is not too surprising[4].

This approach is doomed, unless the tasks inside the loop over markers would be very costly.

This explains why the above approach was not used in Thieulot [326] (2011):

[3] This was run on my Thinkpad laptop with Intel Core i7-7820HQ CPUi at 2.90GHz.

[4] 10^7 double precision take $8 \cdot 10^7$ byes in memory, i.e. approx 76Mb.

In the case of parallel computations, the (grid) domain decomposition leads to a partitioning of the cloud across the processes. Each process only stores the part of the cloud necessary to compute the part of the FE matrix it needs to build. This allows for subsequent memory savings (per process): a 128x128x32 grid with an average of 27 points per element contains over 14 millions points, whose storage in double precision (x, y, z and ϵ fields) occupies about 430Mb in memory. Using only 16 processes reduces the memory requirement to less than 27Mb per process. Also, most of the algorithms dealing with the cloud (advection, strain increment) are in $\mathcal{O}(N)$, so that the cloud decomposition also improves the overall efficiency.

[...]

Also, when a given point is advected and enters an element which belongs to two processes, its position must be communicated to the other process. A very similar procedure must take place when a point leaves such an element: it must be 'deleted' from the list of the other process. The intrinsic MPI routine `mpi_alltoall` is extensively used to this purpose.

Stone 164

Solving the 1D wave equation with FEM



python_codes/fieldstone_164/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_164

This STONE solves the 1d wave equation using 1st order finite elements. The theory is presented in Chapter 12.

The mesh is built simply

```
x=np.linspace(0,Lx,nnx)
```

The connectivity array is also very simple in 1d:

```
icon=np.zeros((m,nelx),dtype=np.int32)
for iel in range(0,nelx):
    icon[0,iel]=iel
    icon[1,iel]=iel+1
```

where m is the number of nodes per element and $nelx$ the number of elements.

Depending on the method we will need some or all of these arrays: $\vec{\mathcal{U}}(t)$ (uprev array), $\vec{\mathcal{U}}(t-dt)$ (uprevprev array), $\vec{\mathcal{U}}(t+dt)$ (u array), $\dot{\vec{\mathcal{U}}}(t)$ (udotprev array), $\dot{\vec{\mathcal{U}}}(t+dt)$ (udot array). All these are $nnx = nelx+1$ long. In the code we declare and initialise them as follows

```
u=np.zeros(nnx,dtype=np.float64)
uprevprev=np.zeros(nnx,dtype=np.float64)
uprev=np.zeros(nnx,dtype=np.float64)
udot=np.zeros(nnx,dtype=np.float64)
udotprev=np.zeros(nnx,dtype=np.float64)
if method==1 or method==2:
   t=2*dt
   for i in range(0,nnx):
       uprevprev[i]=u_th(x[i],0)
       uprev[i]=u_th(x[i],0)+dt*udot_th(x[i],0)
if method==3:
   t=dt
   for i in range(0,nnx):
       uprev[i]=u_th(x[i],0)
       udotprev[i]=udot_th(x[i],0)
```

Note that methods 1 and 2 require two previous solutions to start, so the first solution we obtain is at time $t=2\,dt$, while method 3 only needs one previous solution so it starts at time $t=dt$.

This code implements all three methods dealing with the second-order time derivative. I recall these here briefly hereafter.

- method # 1a

$$\mathbb{M} \cdot \vec{\mathcal{U}}(t+dt) = [2\mathbb{M} - c^2 dt^2 \mathbb{K}] \cdot \vec{\mathcal{U}}(t) - \mathbb{M} \cdot \vec{\mathcal{U}}(t-dt)$$

The code structure is then as follows:

```
for istep in range(0,nstep):
    for iel in range(0,nelx):
        for k in range(0,m):
            up[k]=uprev[icon[k,iel]]
            upp[k]=uprevprev[icon[k,iel]]
        a_el[:,:]=Me[:,:]
        b_el[:]=(2*Me-c**2*dt**2*Ke).dot(up)-Me.dot(upp)
        #apply b.c.
        #assemble
    # end for
    u=sps.linalg.spsolve(sps.csr_matrix(A_mat),rhs)
    ...
    ...
    uprevprev[:]=uprev[:]
    uprev[:]=u[:]
# end for
```

- method # 1b

$$(\mathbb{M} + c^2 \delta t^2 \mathbb{K}) \cdot \vec{\mathcal{U}}(t+\delta t) = \underbrace{\mathbb{M} \cdot (2\vec{\mathcal{U}}(t) - \vec{\mathcal{U}}(t-\delta t))}_{\vec{b}}$$

The code structure is then as follows:

```
for istep in range(0,nstep):
    for iel in range(0,nelx):
        for k in range(0,m):
            up[k]=uprev[icon[k,iel]]
            upp[k]=uprevprev[icon[k,iel]]
        a_el[:,:]=Me[:,:]+c**2*dt**2*Ke
        b_el[:]=Me.dot(2*up-upp)
        #apply b.c.
        #assemble
    # end for
    u=sps.linalg.spsolve(sps.csr_matrix(A_mat),rhs)
    ...
    ...
    uprevprev[:]=uprev[:]
    uprev[:]=u[:]
# end for
```

- method # 2

$$\mathbb{M} \cdot \vec{\mathcal{W}} = -c^2 \mathbb{K} \cdot \vec{\mathcal{U}}(t)$$

$$\vec{\mathcal{U}}(t+dt) = dt^2 \, \vec{\mathcal{W}} + 2\vec{\mathcal{U}}(t) - \vec{\mathcal{U}}(t-dt)$$

```
for istep in range(0,nstep):
    for iel in range(0,nelx):
        for k in range(0,m):
            up[k]=uprev[icon[k,iel]]
        a_el[:,:]=Me[:,:]
        b_el=-c**2*Ke.dot(up)
        #apply b.c.
        #assemble
    # end for
    uu=sps.linalg.spsolve(sps.csr_matrix(A_mat),rhs)
    u=np.zeros(nnx,dtype=np.float64)
    u[:]=dt**2*uu[:]+2*uprev[:]-uprevprev[:]
    ...
    ...
    uprevprev[:]=uprev[:]
    uprev[:]=u[:]
# end for
```

- method # 3

$$\vec{\mathcal{U}}(t+dt) = \vec{\mathcal{U}}(t) + dt\,\dot{\vec{\mathcal{U}}}(t) \tag{164.1}$$

$$\mathbb{M}\cdot\frac{\dot{\vec{\mathcal{U}}}(t+dt)-\dot{\vec{\mathcal{U}}}(t)}{dt} = -c^2\mathbb{K}\cdot\vec{\mathcal{U}}(t) \tag{164.2}$$

```
for istep in range(0,nstep):
    for iel in range(0,nelx):
        for k in range(0,m):
            up[k]=uprev[icon[k,iel]]
        a_el[:,:]=Me[:,:]
        b_el=-c**2*Ke.dot(up)
        #apply b.c.
        #assemble
    # end for
    u[:]=uprev[:]+udotprev[:]*dt
    R=sps.linalg.spsolve(sps.csr_matrix(A_mat),rhs)
    udot[:]=udotprev[:]+dt*R[:]
    ...
    ...
    uprev[:]=u[:]
    udotprev[:]=udot[:]
# end for
```

At this stage we notice that the FE matrix is simply the mass matrix for all three methods (unless the implicit variant of method 1 is used).

As shown in Section XXX the mass matrix $\mathbb{M}$ and Laplace matrix $\mathbb{K}$ are easily analytically computed for linear 1D elements:

$$\mathbb{M}_e = h_x\frac{1}{3}\begin{pmatrix} 1 & 1/2 \\ 1/2 & 1 \end{pmatrix} = \begin{pmatrix} h_x/3 & h_x/6 \\ h_x/6 & h_x/3 \end{pmatrix} \qquad \mathbb{K}_e = \frac{1}{h_x}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

This translates as follows in the code:

```
Me=np.array([[hx/3,hx/6],[hx/6,hx/3]],dtype=np.float64)
Ke=np.array([[1/hx,-1/hx],[-1/hx,1/hx]],dtype=np.float64)
```

Let us consider a test case with three elements. The first elemental matrix $\mathbb{M}_e$ gets modify by the application of the boundary conditions:

```
[[0.11111111 0.        ]
 [0.         0.11111111]]
```

The second elemental matrix is simply $\mathbb{M}_e$

```
[[0.11111111 0.05555556]
 [0.05555556 0.11111111]]
```

and the third elemental matrix is also modified bc of boundary conditions:

```
[[0.11111111 0.        ]
 [0.         0.11111111]]
```

When assembled we obtain the following FE matrix:

```
[[0.11111111 0.         0.         0.        ]
 [0.         0.22222222 0.05555556 0.        ]
 [0.         0.05555556 0.22222222 0.        ]
 [0.         0.         0.         0.11111111]]
```

The code consists of a time loop, and inside each iteration the FE matrix and rhs are re-built (technically the FE matrix is the same every time), the system solved, and the solution exported to an ascii file for later processing.

Results - Experiment # 1

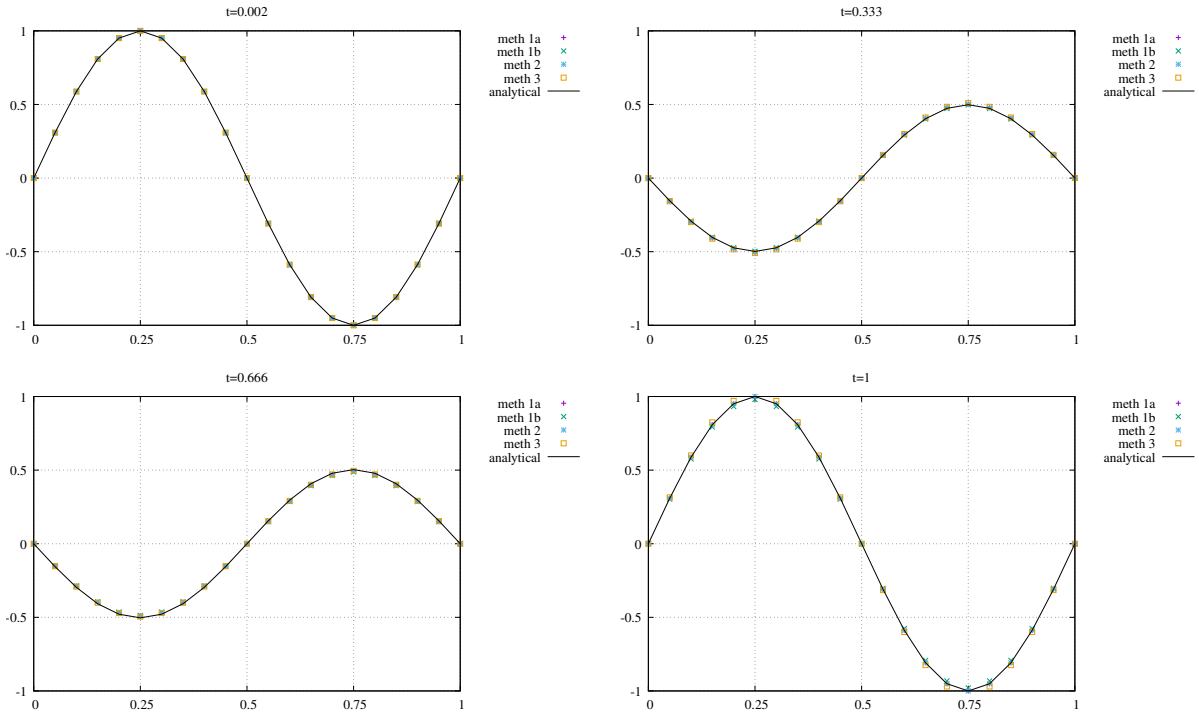
The problem at hand is described in Section 12.2.2. As such we first create two functions:

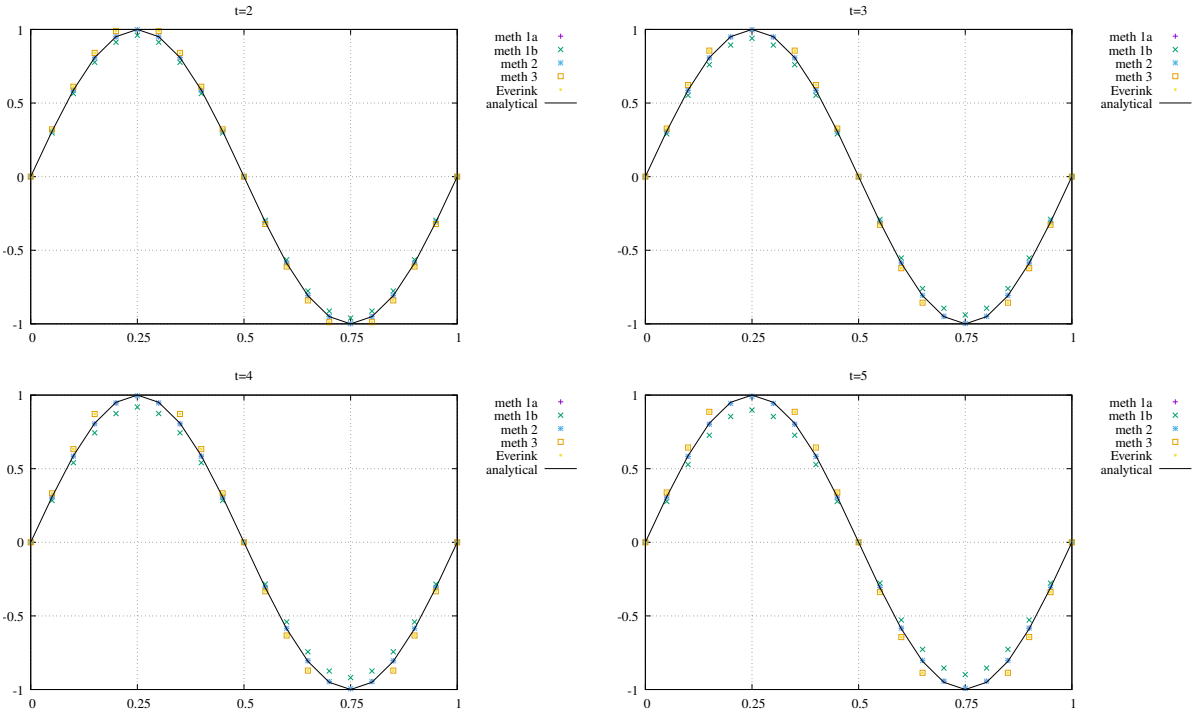
```
def u_th(x,t):
    return np.cos(2*np.pi*t)*np.sin(2*np.pi*x)

def udot_th(x,t):
    return -2*np.pi*np.sin(2*np.pi*t)*np.sin(2*np.pi*x)
```

The total energy is then

$$\begin{aligned}
\left\|\frac{\partial u}{\partial t}(\cdot,t)\right\|^2 + c^2\left\|\frac{\partial u}{\partial x}(\cdot,t)\right\|^2 &= \|-2\pi\sin(2\pi t)\sin(2\pi x)\|^2 + c^2\|-2\pi\cos(2\pi t)\sin(2\pi x)\|^2 \\
&= \int_0^1 (-2\pi\sin(2\pi t)\sin(2\pi x))^2\,dx + \int_0^1 (-2\pi\cos(2\pi t)\sin(2\pi x))^2\,dx \\
&= 4\pi^2\sin^2(2\pi t)\int_0^1 \sin^2(2\pi x)dx + 4\pi^2\cos^2(2\pi t)\int_0^1 \cos^2(2\pi x)dx \\
&= 4\pi^2\sin^2(2\pi t)\frac{1}{2} + 4\pi^2\cos^2(2\pi t)\frac{1}{2} \\
&= 2\pi^2(\sin^2(2\pi t)+\cos^2(2\pi t)) \\
&= 2\pi^2 \\
&\simeq 19.7392 \qquad (164.3)
\end{aligned}$$

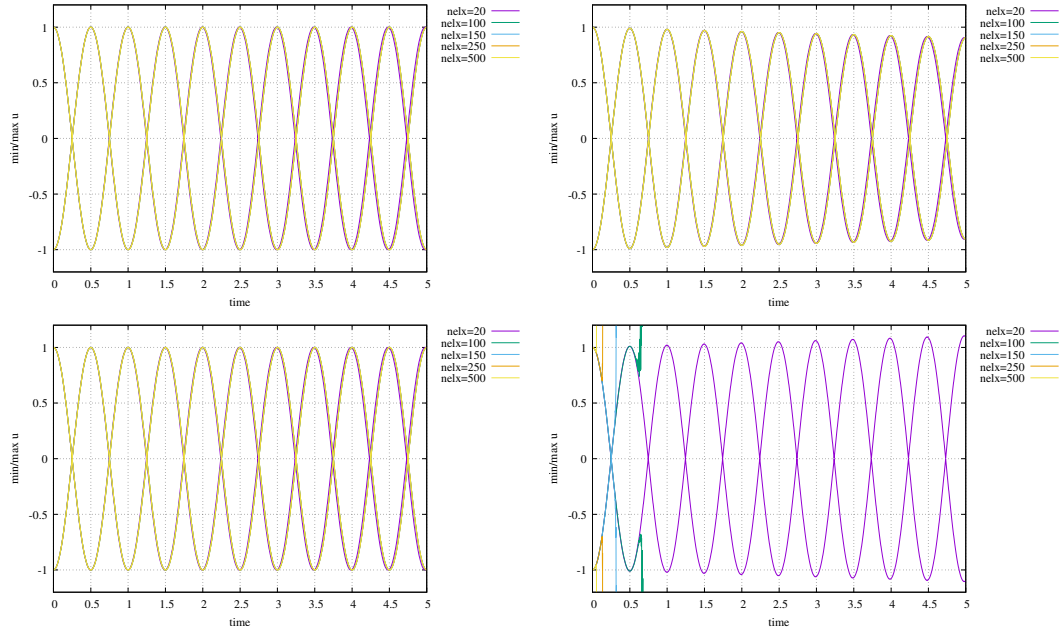
The time step is set to $dt = 10^{-3}$. The following results are obtained for $nelx = 20$ with all three methods.





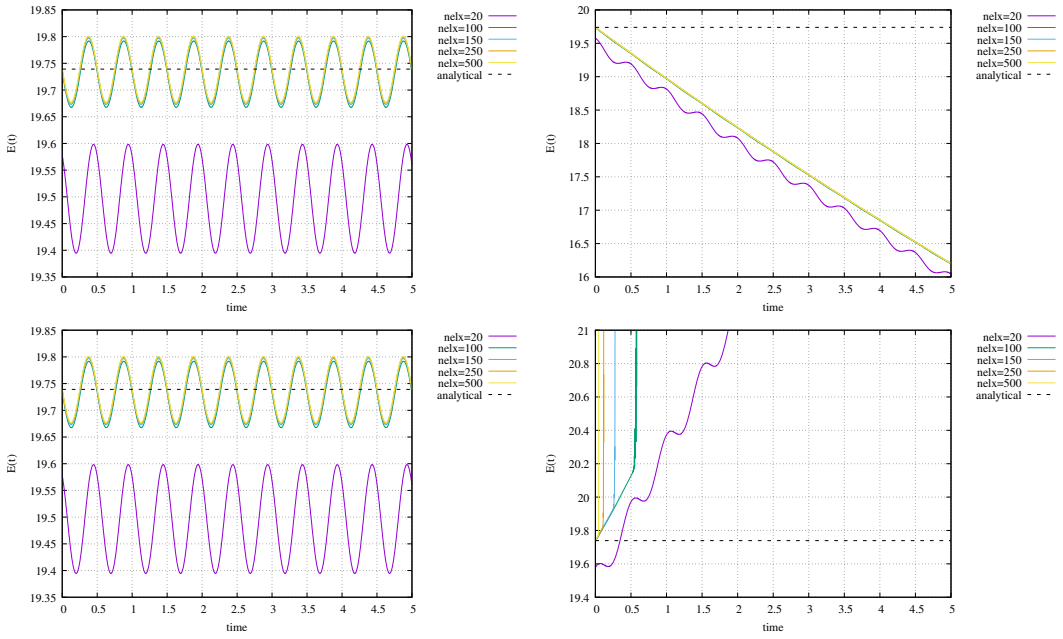
We find that methods 1a and 2 seem to be more accurate than method 3. To make sure I plot the results of Everink obtained with method 3 against mine and recover identical values. Method 2b seems diffusive, i.e. min/max values decrease over time.

One can also look at the min/max values of u for all three methods and for various resolutions:



We find that method 3 is prone to numerical explosion, and method 1b is prone to diffusion so they should probably be avoided.

Finally we can also monitor the total energy in the system over time, for all methods and for various resolutions:



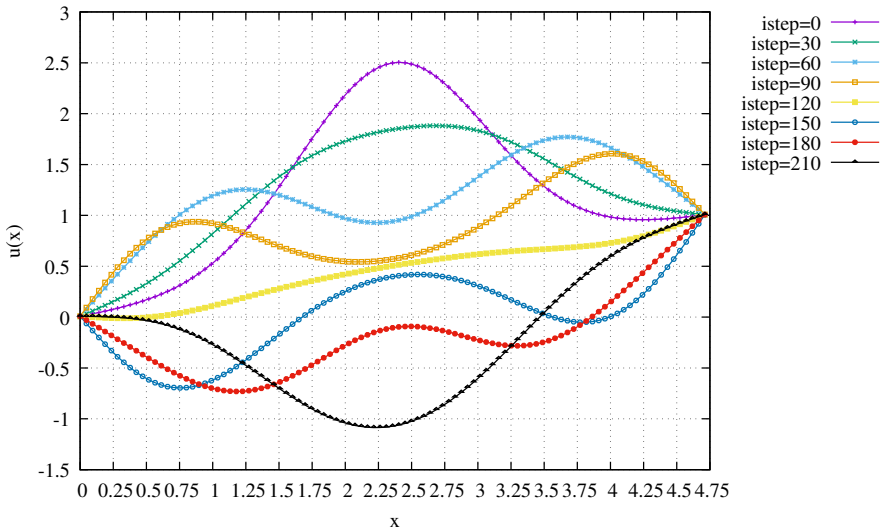
The conclusion is very clear. Again, only methods 1a and 2 seem to conserve the total energy.

Results - Experiment # 2

In this case $L_x = 1.5\pi \simeq 4.712$, $t_{final} = 240$,

$$u(x, t = 0) = 2\exp\left(-(x - L_x/2)^2\right) + x/L_x \qquad \dot{u}(x, t = 0) = 0$$

Boundary conditions are given by $u(0)$ on the left and $u(L)$ on the right. This experiment is borrowed from there[1].



[1] https://juliateachingctu.github.io/Julia-for-Optimization-and-Learning/stable/lecture_13/ode/

Stone 165

Solving the 2D wave equation with FEM



python_codes/fieldstone_165/text.tex

P Code: https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_165

This STONE solves the 2d wave equation using 1st- and 2nd-order finite elements. The theory is presented in Chapter 12. The implementation follows STONE 164 although in this case a more generic approach is taken: the elemental matrices and rhs vectors for each element are computed by means of numerical quadrature. The code is quite close to STONE 43 which solves the advection-diffusion equation in 2D. Note that visualisation now relies on vtu files for Paraview.

Various experiments are carried out:

- Experiment # 1: The domain is a square of size 2, the time step is $dt = 0.01$ and the wave speed is set to $c = 1$. We have
$$u(x,y,t=0) = \cos\left[(x-1)\frac{\pi}{2}\right] \cos\left[(y-1)\frac{\pi}{2}\right]$$
and
$$\dot{u}(x,y,0) = 0$$

- Experiment #2: Same as experiment #1, but with
$$u(x,y,t=0) = (\cos\left[(x-0.25)2\pi\right] \cos\left[(y-0.25)2\pi\right])^2$$
if $x,y \leq 0.5$ and zero otherwise.

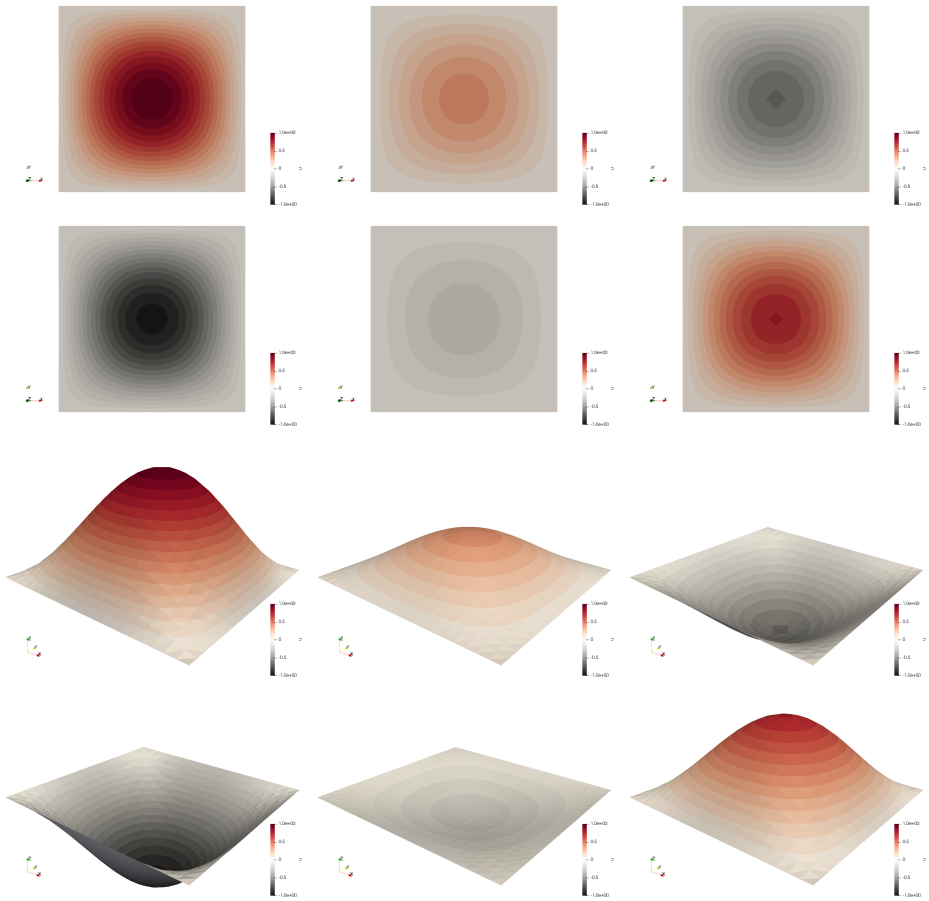
- Experiment #3: Same as Experiment #2, but an additional condition $u = 0$ is set for the nodes with $x = L_x/2$ and $y < L_y/2$.

- Experiment #4: The domain is the unit square. This time the initial wave field is given by
$$u(x,y,t=0) = \left[\cos\left((x-0.25)4\pi\right)\right]^4$$
for $|x - 0.25| < 0.125$ and zero otherwise. Boundary conditions are $u = 0$ on the left and right boundaries. Additional constraints $u = 0$ are placed on nodes $(x = 0.5, y < 0.43)$ and $(x = 0.5, y > 0.57)$.

Note that although in what follows the values of u is set to zero on all four boundaries it can also not be fixed on the boundary(ies) and therefore yields an implicit Neumann boundary condition $du/dn = 0$.

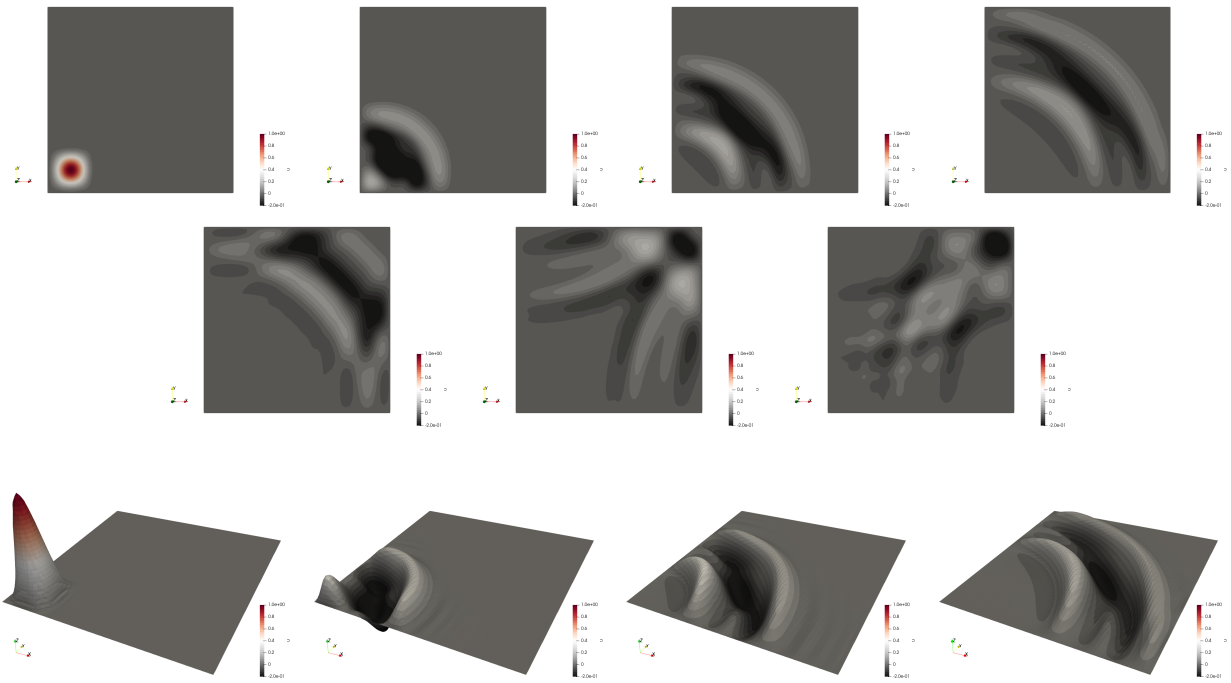
todo: try with zero displacement and initial velocity

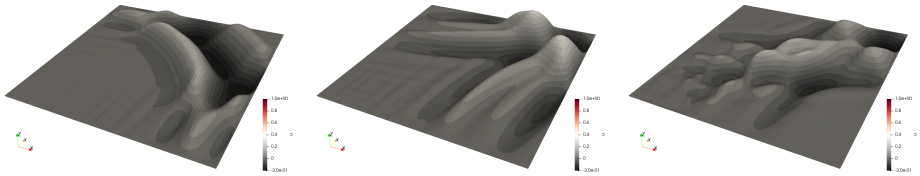
Results (Experiment # 1)



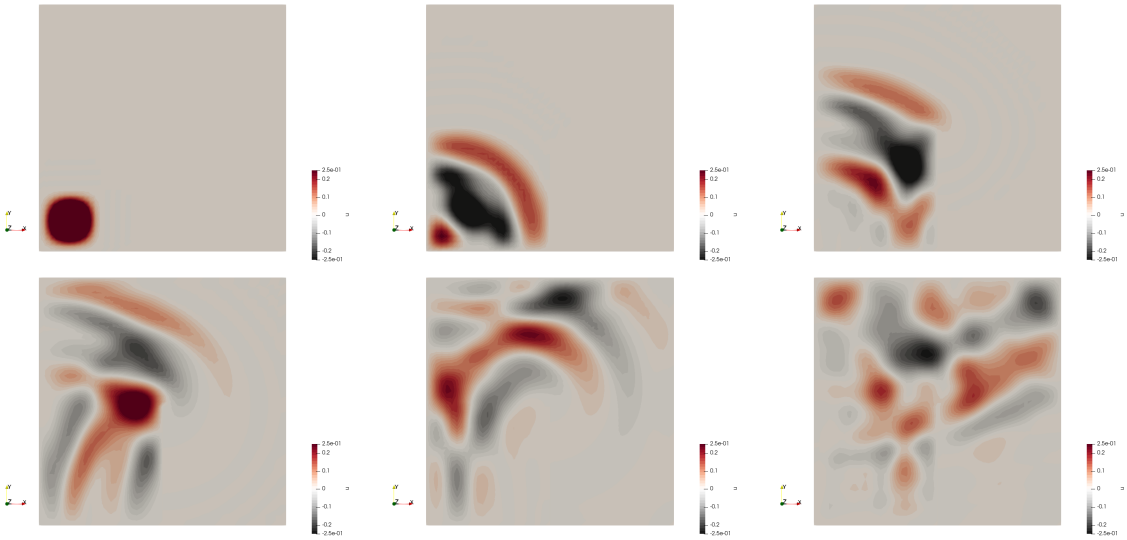
Results (Experiment # 2)

The resolution is set to $64x64$ elements and the simulation is ran for 300 time steps. Obtained with 1st order elements.

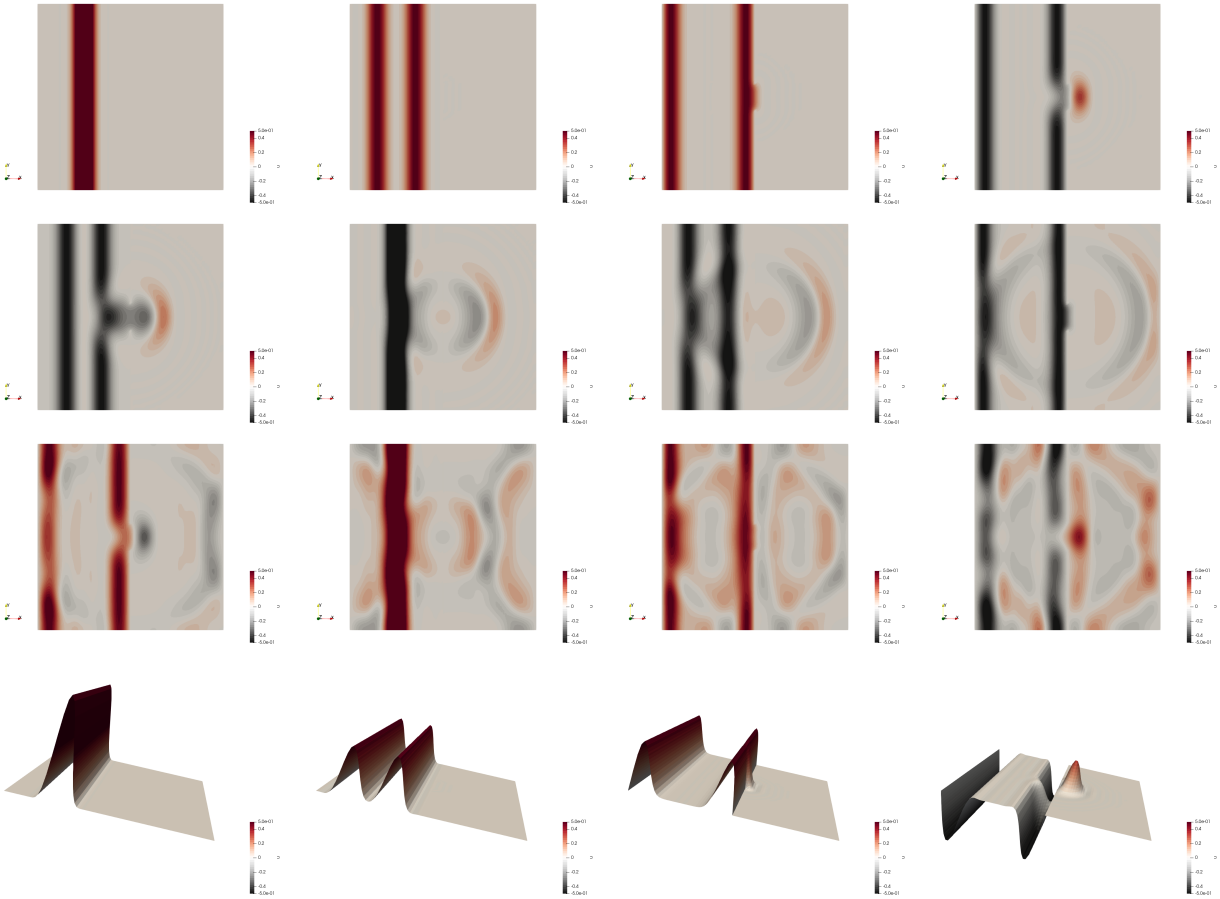


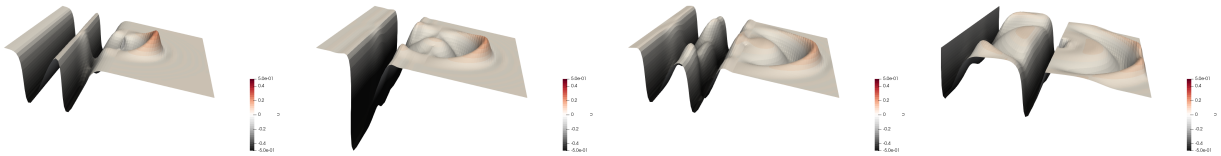


Results (Experiment # 3)



Results (Experiment # 4)





Stone 166

Solving the 1D wave equation with FDM



python_codes/fieldstone_166/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_166`

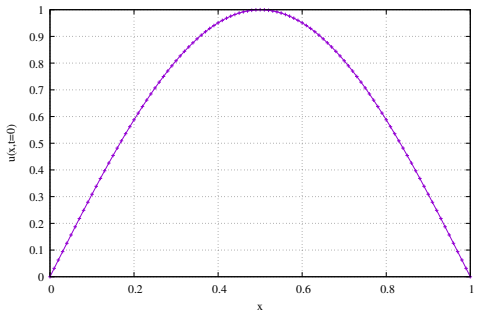
The theory and discretisation are described in Section **??**.

Experiment # 1

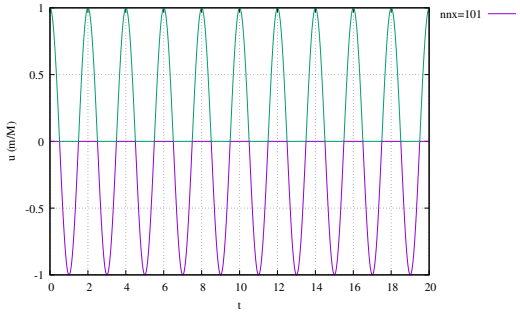
The domain is $[0,1]$, $C_{CFL}=0.5$, $nnx=101$ and $c=1$.

$$u(x,t)=\sin(\pi x)\cos(\pi ct)$$

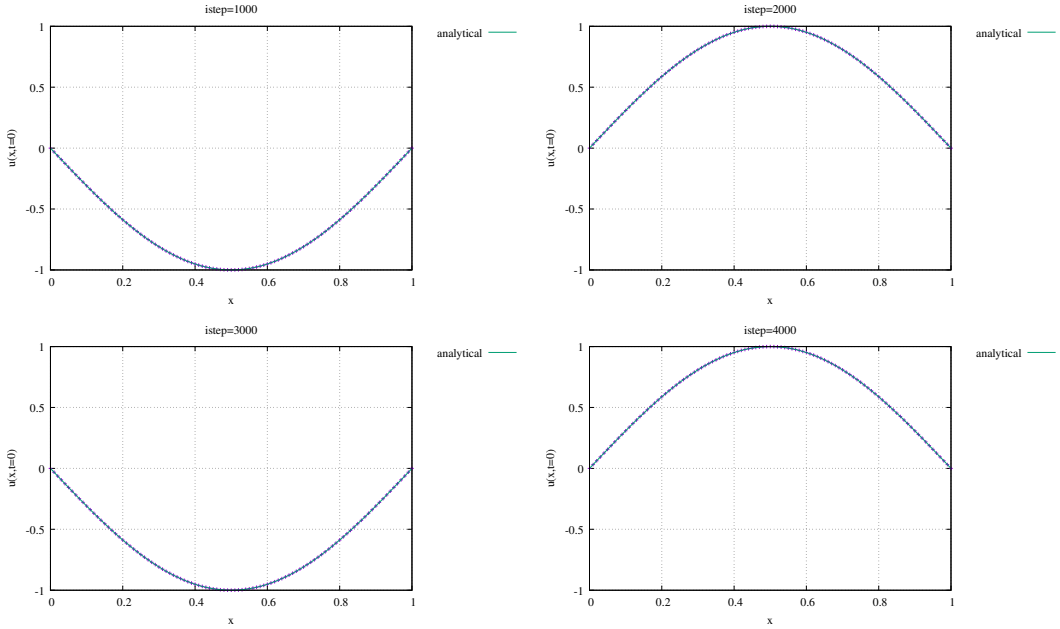
The initial wave field u is as follows at $t=0$:



The min/max values of u as a function are as follows:



The u field is shown hereunder at three different times $t=5,10,15,20$:


istep=1000
analytical
u(x,t=0)
x
istep=2000
analytical
u(x,t=0)
x
istep=3000
analytical
u(x,t=0)
x
istep=4000
analytical
u(x,t=0)
x

Stone 167

Shear heating as a heat source, replicating Stüwe (1998)



python_codes/fieldstone_167/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_167`

This stone was developed in collaboration with Daniel Cayuela-Barcham and Alissa Kotowski.

This STONE takes its origin in the MSc work of Daniel about the subduction interface. Stüwe [313] (1998) proposes a simple mathematical model for the influence of shear heating on the temperature there, so it was decided to reproduce the results of the paper (esp. Fig. 6) and use it as a basis for discussion.

Theory

We start with the heat transport equation where the advection and diffusion terms are assumed to be negligible and the only source of heat would come from the shear heating:

$$\rho C_p \frac{dT}{dt} = H_s = \sum_{ij} \sigma_{ij} \dot{\varepsilon}_{ij} = \boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}}$$

where H_s is the volumetric rate of shear heat production, σ_{ij} is the stress tensor and $\dot{\varepsilon}_{ij}$ the strain rate tensor.

In order to do our estimates on the simplest level, a deformation geometry of biaxial horizontal compression is assumed where the off-diagonal terms of the strain rate tensor as well as $\dot{\varepsilon}_{yy}$ are zero and $\dot{\varepsilon}_{xx} = -\dot{\varepsilon}_{zz} = \dot{\epsilon}$ (thereby implying that the trace of the tensor is zero, i.e. the flow is incompressible).

Concretely, the strain rate tensor (in the right coordinate system) can then be written:

$$\dot{\boldsymbol{\varepsilon}} = \begin{pmatrix} \dot{\epsilon} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \dot{\epsilon} \end{pmatrix}$$

with the second invariant thereof given by

$$\mathcal{I}_2(\dot{\boldsymbol{\varepsilon}}) = \frac{1}{2}\dot{\boldsymbol{\varepsilon}} : \dot{\boldsymbol{\varepsilon}} = \frac{1}{2}(\dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{zz}^2) + \dot{\varepsilon}_{xy}^2 + \dot{\varepsilon}_{xz}^2 + \dot{\varepsilon}_{yz}^2 = \frac{1}{2}(\dot{\epsilon}^2 + \dot{\epsilon}^2) = \dot{\epsilon}^2$$

In this deformation geometry, the principal stresses ($\sigma 1$ and σ_3 correspond to the horizontal and vertical stresses and there is no shear stress so that the mean stress can be expressed as $\tau = \frac{1}{2}(\sigma_1 - \sigma_3)$.

For this, we assume the form suggested by Brace and Kohlstedt [32] (1980) (Stüwe probably refers to their Eq. 4) which includes the exponential dependence of stress on temperature:

$$\tau = \frac{1}{2}(\sigma_1 - \sigma_3) = \frac{1}{2}\left(\frac{\dot{\varepsilon}}{A}\right)^{1/n} \exp\left(\frac{Q}{nRT}\right)$$

which leads to write :

$$\boldsymbol{\tau} = \frac{1}{2}\left(\frac{\dot{\epsilon}}{A}\right)^{-1+1/n} \exp\left(\frac{Q}{nRT}\right)\dot{\boldsymbol{\varepsilon}}$$

where A and Q are the pre-exponent constant and activation energy, n is the power law exponent, which is assumed to be 3, and R is the gas constant.

We have $\boldsymbol{\sigma} = -p\mathbf{1} + \boldsymbol{\tau}$, where $\boldsymbol{\tau}$ is the deviatoric stress tensor so that the shear heating term writes

$$\boldsymbol{\sigma} : \dot{\boldsymbol{\varepsilon}} \;=\; -p \underbrace{\mathbf{1} : \dot{\boldsymbol{\varepsilon}}}_{=Tr(\dot{\varepsilon})=0} + \boldsymbol{\tau} : \dot{\boldsymbol{\varepsilon}} \;=\; \frac{1}{2}\left(\frac{\dot{\epsilon}}{A}\right)^{-1+1/n} \exp\left(\frac{Q}{nRT}\right) \underbrace{\dot{\boldsymbol{\varepsilon}} : \dot{\boldsymbol{\varepsilon}}}_{2\dot{\epsilon}^2} \tag{167.1}$$

so that in the end:

$$\boxed{\frac{dT}{dt} = \frac{\dot{\epsilon}}{\rho C_p}\left(\frac{\dot{\epsilon}}{A}\right)^{1/n} \exp\left(\frac{Q}{nRT}\right)}$$

which is Eq. (6) of Stüwe [313] (1978)[1]

In order to solve this ODE we follow the same procedure as in the appendix of the paper. Assuming the strain rate $\dot{\epsilon}$ to be constant, we define

$$C = \frac{\rho C_p}{\dot{\epsilon}}\left(\frac{\dot{\epsilon}}{A}\right)^{-1/n} \qquad \text{and} \qquad D = \frac{Q}{nR}$$

so that now

$$\begin{aligned} \frac{dT}{dt} &= C^{-1}e^{D/T} \\ \Rightarrow Ce^{-D/T}dT &= dt \\ \Rightarrow \int_{T_0}^{T_f} Ce^{-D/T}dT &= \int_0^t dt \end{aligned} \tag{167.2}$$

where we have integrated between the limits of temperature at the start of deformation, T_0, and temperature at the end of deformation T_f. We then obtain

$$t = C\int_{T_0}^{T_f} e^{-D/T}dT$$

We carry out a simple change of variable: $x = D/T$, so that $dx = -\frac{D}{T^2}dT$ and also $dT = -\frac{D}{x^2}dx$. Then

$$t = C\int_{x_0}^{x_f} e^{-x}\left(-\frac{D}{x^2}\right)dx = CD\int_{x_0}^{x_f}\left(-\frac{1}{x^2}\right)e^{-x}dx$$

where $x_0 = D/T_0$ and $x_f = D/T_f$. There probably is a minus sign missing in Eq. 9.

The integral in the rhs can be integrated by parts[2]:

$$\int_{x_0}^{x_f}\left(-\frac{1}{x^2}\right)e^{-x}dx = \left[\frac{1}{x}e^{-x}\right]_{x_0}^{x_f} - \int_{x_0}^{x_f}\frac{1}{x}(-e^{-x})dx = \frac{e^{-x_f}}{x_f} - \frac{e^{-x_0}}{x_0} + \int_{x_0}^{x_f}\frac{1}{x}e^{-x}dx$$

[1] Save for factor 2. However note that this factor 2 disappears in Eq. 7 so which one is it?

[2] $\int u'v = uv - \int uv'$

The integral in the rhs is close to the so-called exponential integral[3] Ei defined as follows:

$$\mathrm{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t} dt \qquad \text{or,} \quad \mathrm{Ei}(-x) = -\int_{x}^{\infty} \frac{e^{-t}}{t} dt$$

The second expression will prove useful in what follows. Using Eq. 12 of Stüwe [313]:

$$\begin{aligned} \int_{x_0}^{x_f} \frac{e^{-x}}{x} dx &= \int_{x_0}^{\infty} \frac{e^{-x}}{x} dx + \int_{\infty}^{x_f} \frac{e^{-x}}{x} dx \\ &= \int_{x_0}^{\infty} \frac{e^{-x}}{x} dx - \int_{x_f}^{\infty} \frac{e^{-x}}{x} dx \\ &= -\mathrm{Ei}(-x_0) + \mathrm{Ei}(-x_f) \end{aligned} \tag{167.3}$$

we arrive at

$$\boxed{t = CD \left[\frac{e^{-x_f}}{x_f} - \frac{e^{-x_0}}{x_0} + \mathrm{Ei}(-x_f) - \mathrm{Ei}(-x_0) \right]}$$

There remains a problem: this equation is not identical to Eq. 13 of the paper:

$$t = CD \left(\frac{\mathrm{e}^{-x_\mathrm{f}}}{x_\mathrm{f}} - \frac{\mathrm{e}^{-x_0}}{x_0} + Ei(x_0) - Ei(x_\mathrm{f}) \right)$$

Finally, if we replace time by $t = \epsilon/\dot{\epsilon}$, assume we set a desired value for the strain ϵ, the equation above can be solved numerically for T_f:

$$\frac{\epsilon}{\dot{\epsilon}} = CD \left[\frac{e^{-D/T_f}}{D/T_f} - \frac{e^{-D/T_0}}{D/T_0} + \mathrm{Ei}(-D/T_f) - \mathrm{Ei}(-D/T_0) \right]$$

but for simplicity we will solve it for x_f in that form:

$$\frac{t}{CD} - \left[\frac{e^{-x_f}}{x_f} - \frac{e^{-x_0}}{x_0} + \mathrm{Ei}(-x_f) - \mathrm{Ei}(-x_0) \right] = 0 \tag{167.4}$$

This is a nonlinear equation of the form $f(x_f) = 0$. We assume a starting temperature T_0, compute x_0, assign material values ρ, C_p, A, n, choose a strain rate value $\dot{\epsilon}$, a total strain ϵ and can then proceed to solve the equation for x_f.

When it comes to Ei these must be evaluated numerically, for instance with `https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.expi.html` where we see that the exponential integral is defined as

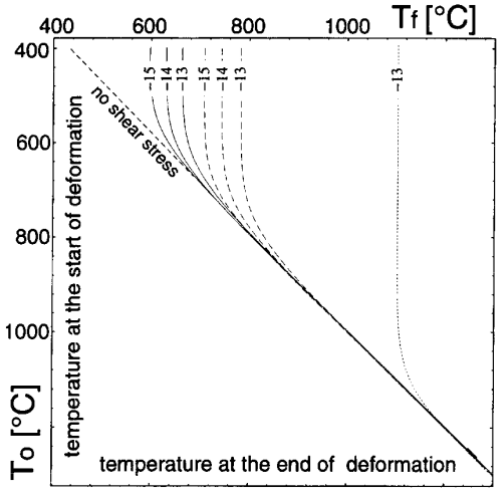
$$Ei(x) = \int_{-\infty}^{x} \frac{e^t}{t} dt = \int_{\infty}^{-x} \frac{e^{-t}}{-t} (-dt) = \int_{\infty}^{-x} \frac{e^{-t}}{t} dt = -\int_{-x}^{\infty} \frac{e^{-t}}{t} dt$$

which is the standard definition that we have used above.

[3] `https://en.wikipedia.org/wiki/Exponential_integral`

About Fig. 6 of the paper

Let us consider Fig. 6 of Stüwe [313], where three different sets rheological constants are used for A and Q.



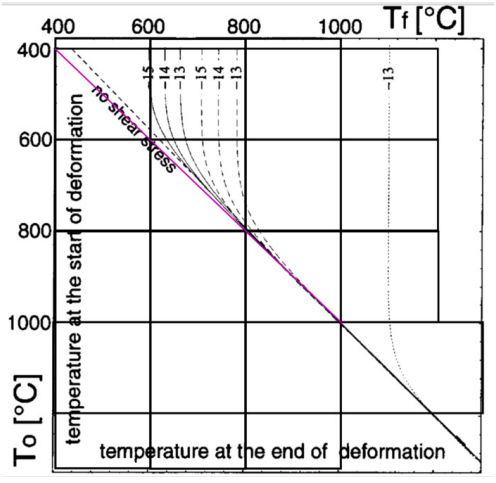
Taken from [313]. The curves are labelled for $\log_{10} \dot{\epsilon}$. The rheological data used for the three solid curves are the quartz data of [32]: $A = 5 \cdot 10^{-6} \mathrm{MPa}^{-3}\,\mathrm{s}^{-1}$, $Q = 1.9 \cdot 10^5 \mathrm{J\,mol}^{-1}$ The rheological data used for the three dashed curves are the average quartz date used by [312]: $A = 2 \cdot 10^{-4} \mathrm{MPa}^{-3}\,\mathrm{s}^{-1}$, $Q = 2.5 \cdot 10^5 \mathrm{J\,mol}^{-1}$. The data for the dotted curve are the olivine data of [32]: $A = 7 \cdot 10^4 \mathrm{MPa}^{-3}\,\mathrm{s}^{-1}$, $Q = 5.2 \cdot 10^5 \mathrm{J\,mol}^{-1}$. All curves are calculated for a total strain of one.

Obviously the final temperature T_f can not be lower than the initial temperature T_0 ($H_s \geq 0$) so that the lower left of the figure is not accessible. Also this means that we should look for the root of Eq. (167.4) in the range $T_f \geq T_0$.

We read:

> Fig. 6 shows that for reasonable strain rates and strains the temperatures of rocks at 600°C may be increased by 100-200°C. Colder rocks can heat more, because they can sustain higher shear stresses until strain heating weakens them (Pavlis, 1986). Importantly, rocks colder than about 500-600°C will reach similar temperatures by the effects of strain heating, independently of their temperature prior to deformation (Pavlis, 1986). Thus, the effects of shear heating on a large vertical section of the crust may be that this section is made isothermal.

Coming back to the figure in the paper, we can add lines to it and we then quickly realise that the diagonal line labeled 'no shear stress' has likely been drawn by hand since it does not follow the $T_0 = T_f$ line indicated in purple:



The python code

First, we notice that the A values are provided in $\mathrm{MPa}^{-3}\,\mathrm{s}^{-1}$. We have

$$1\ \mathrm{MPa}^{-3} = (10^6\mathrm{Pa})^{-3} = 10^{-18}\ \mathrm{Pa}^{-3}$$

so that for example $5\cdot 10^{-6}\ \mathrm{MPa}^{-3}\,\mathrm{s}^{-1} = 5\cdot 10^{-24}\ \mathrm{Pa}^{-3}\,\mathrm{s}^{-1}$.

Second, the gas constant R and the heat capacity C_p are expressed in $\mathrm{J\,mol^{-1}\,K^{-1}}$ and $\mathrm{J\,kg^{-1}\,K^{-1}}$ so that all temperatures in the equations above must be in K.

We first declare all the variables needed:

```
TKelvin=273.
R=8.314
Cp=1000
n=3.
rho=2750
e=1          # total strain
A=5e-24
Q=1.9e5
```

We then create an array that contains temperature values in °C between 400 and 1400:

```
Trange=np.arange(400,1400,1)
```

We then create the following function which corresponds to Eq. (167.4). Note that the conversion from °C to K occurs in the function so that its arguments T,T0 are in °C:

```
def f(T,T0,D,t):
    TT=T+TKelvin
    x=D/TT
    TT0=T0+TKelvin
    x0=D/TT0
    return  t/C/D - ( np.exp(-x)/x -np.exp(-x0)/x0 -scp.special.expi(-x0) +scp.special.expi(-x) )
```

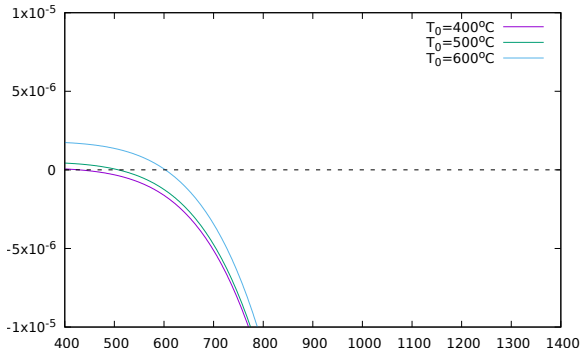
We then proceed to loop over three values of strain rate $10^{-13,-14,-15}$ $\mathrm{s}^{-1}$and for each we compute C, D and t:

```
for sr in (1e-13,1e-14,1e-15):
    D=Q/n/R
    C=rho*Cp/sr*(sr/A)**(-1./n)
    t=e/sr
```

Inside this loop we then loop over all values of T0 and evaluate the function for all temperatures of the Trange array:

```
    for T0 in range (400,1400,1):
        res=f(Trange,T0,D,t)
```

The res array then contains the values of the function for a given T0. We must then proceed to find its zero(es), if any. Before implementing any kind of algorithm, we may wish to visually confirm that the data contained in res does indeed cross the horizontal axis. We therefore set $\dot\epsilon = 10^{-14}\ \mathrm{s}^{-1}$ and plot the array for a few values of T0:



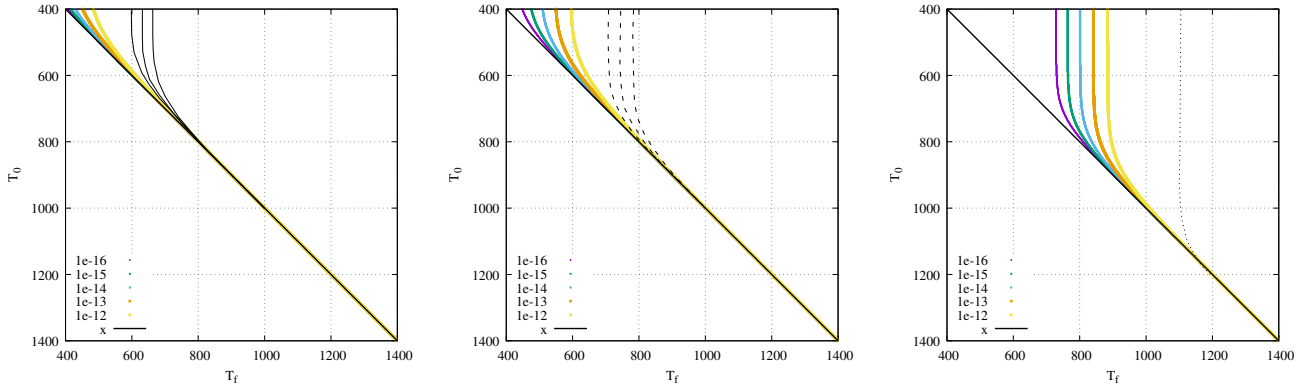
Let us here not dwell on the actual locations of the zeros (see next section), but rather rejoice: The function is monotonous and it looks like there will be only a single zero value between 400 °C and 1400 °C. At this stage, we could then decide to use a ready-made python function that would allow us to find the zero with high accuracy via a Newton-Raphson algorithm, but given the first-order nature of the whole exercise, locating the zero within one degree is more than sufficient. We have therefore implemented a very simple algorithm: if two consecutive values inside the res array are of opposite sign then the zero lies there and the corresponding temperature value is written to file next to the T0 value. This translates as follows:

```
for sr in (1e-13,1e-14,1e-15):
    ...
    srfile=open('sr'+str(sr)+'.ascii',"w")
    ...
    for T0 in range (400,1400,1):
        ...
        for i in range(0,len(Trange)-1):
            if res[i]*res[i+1]<0:
                srfile.write("%e %e \n" %(Trange[i],T0))
```

In the end the code produces three files `sr1e-13.ascii`, `sr1e-14.ascii`, and `sr1e-15.ascii`, which we can plot in a similar fashion as in Fig. 6 of the paper. Actually, as we will see, it will be beneficial to also run $\dot{\epsilon} = 10^{-12,-16}$ s^{-1}.

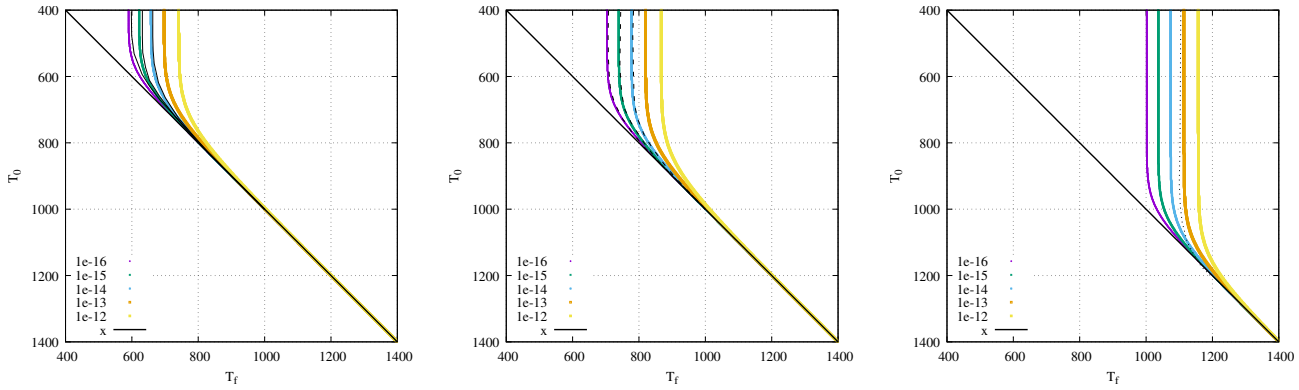
Results

Running our code for temperatures between 400 °C and 1400 °C we arrive at the following temperature profiles for three values of strain rate $10^{-13,-14,-15}$ s^{-1}:



From left to right: Quartz [32], quartz [312], olivine [32].

I have digitized[4] the lines of Fig. 6 and these are also plotted on the figures. This is rather puzzling because our results do not come close to those of Fig 6 at all. However, although it is obviously wrong, we could run our code and choose not to convert temperatures to K (simply by setting TKelvin=0 in the code) and we obtain the following figures:



From left to right: Quartz [32], quartz [312], olivine [32].

[4] `https://plotdigitizer.com/app`

This is rather worrying because these plots do resemble the Fig. 6 of the paper a lot. Also, we find that the lines in the paper are probably for strainrates $\dot{\epsilon} = 10^{-16,-15,-14}$ s^{-1} unlike stated on the figure of the paper.

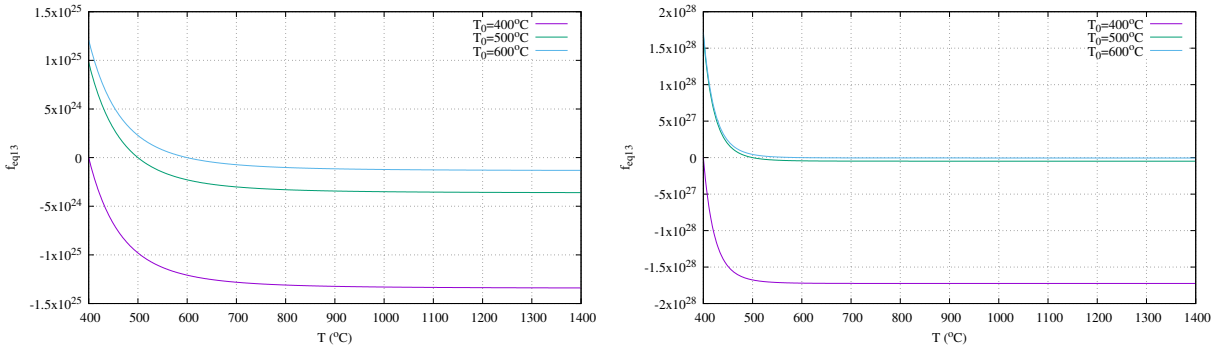
We can also try to use Eq. 13 of the paper,

$$t = CD\left(\frac{e^{-x_f}}{x_f} - \frac{e^{-x_0}}{x_0} + Ei(x_0) - Ei(x_f)\right)$$

implemented as follows:

```
def feq13(T,T0,D,t):
    TT=T+TKelvin
    x=D/TT
    TT0=T0+TKelvin
    x0=D/TT0
    return  t - C*D*( np.exp(-x)/x -np.exp(-x0)/x0 +scp.special.expi(x0) -scp.special.expi(x) )
```

Let us consider three values of T_0, i.e. $T_0 = 400, 500, 600$ °C and plot the function over the range 400 °C-1400 °C for $\dot{\epsilon} = 10^{-14}$ s^{-1}.



Left: function using temperatures in Kelvin; Right: function using temperatures in Celsius.

When Kelvin units are used, we find that the zero of the function is approxmately equal to T_0 which is not what is shown in Fig. 6 of the paper. When Celsius units are used, it is less visually obvious, but the same observation can be made. This reinforces the conclusion that Eq. (13) of the paper is probably wrong.

Stone 168

Edge-driven convection: a simple study by Kim and So [190]



python_codes/fieldstone_168/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_168`

This is an attempt at replicating Dae-Hee Kim and Byung-Dal So. "Effects of rheology and mantle temperature structure on edge-driven convection: Implications for partial melting and dynamic topography". In: *Phys. Earth. Planet. Inter.* 303 (2020), p. 106487. DOI: `10.1016/j.pepi.2020.106487` with both a self-made Python code and the community code ASPECT. When replicating a study I will always reproduce the abstract:

> Edge-driven convection, which affects partial melting, intraplate volcanism, and dynamic topography, is small- scale convection that occurs along a lithospheric keel with a sharp contrast in lithospheric thickness. Various factors, including Rayleigh number, lateral mantle temperature heterogeneity, and geometry of the keel, influence the edge-driven convection, and the correlation between edge-driven convection and surface expressions (dynamic topography and volcanism) is complicated. We performed a finite element study to quantify the effects of these factors on dynamic topography and partial melting. We found that the dynamic topography is more prominent when a strong edge-driven convection cell develops, which corresponds to homogeneous mantle temperatures and the absence of mantle wind. In contrast, the development of edge-driven convection cells and dynamic topography near the lithospheric keel are hindered when the mantle temperature is strongly heterogeneous (laterally varying ∼280 K). This indicates that a large lateral contrast in mantle temperature results in a strong mantle wind that may prevent the development of edge-driven convection cells. An increase in the Rayleigh number results in more vigorous convection and enhances partial melting. Our study shows that the location of volcanic activity at craton edges and passive margins can be reproduced in models with weakly heterogeneous mantle temperature for given mantle viscosity. The existence of a strong mantle wind (e.g., related to subducting slabs or mantle plumes) may inhibit the formation of an edge-driven convection cell and its related partial melt near a lithospheric keel. However, mantle conditions with weak temperature heterogeneity ($<\sim 14$ K) or high mantle viscosity ($> 17 \times 10^{19}$ Pa s), which corresponds to the Rayleigh number of 1.8×10^6, do not induce partial melts despite the development of edge-driven convection cells. Our model parametrized the condition and location of edge-driven convection cells and partial melts, which can contribute to understanding anomalous intraplate volcanisms, such as in Jeju Island south of the Korean Peninsula and the Tanzania Craton near the East African Rift.

The setup

I then proceed to thoroughly read the paper in order to extract all what is needed to setup the experiment. The authors present the following equations after explaining that all quantities in these are dimensionless:

$$u_{i,i} = 0 \tag{2}$$

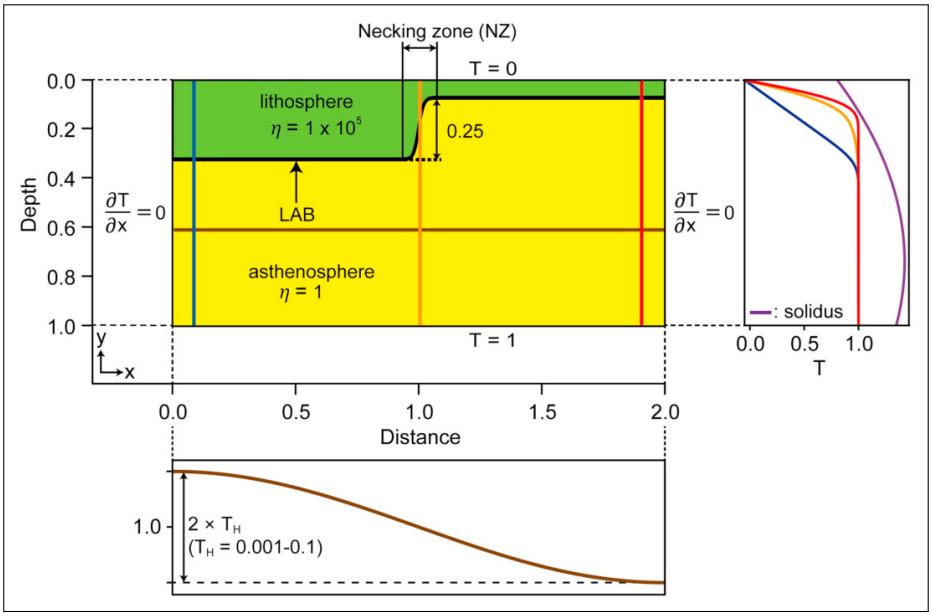
$$-P_{,i} + \eta(u_{i,j} + u_{j,i})_{,j} + RaT\delta_{i2} = 0 \text{ where } Ra = \frac{\rho_0 \alpha g \Delta T D_0^3}{\kappa \eta_0} \tag{3}$$

$$T_{,t} + u_i T_{,i} = T_{,ii} \tag{4}$$

with the following table of parameters:

Descriptions	Symbols	Values
Reference density	ρ_0	3300 [kg/m^3]
Reference viscosity	η_0	3–300 × 10^{19} [Pa·s]
Surface temperature	T_0	273 [K]
Bottom temperature	T_b	1673 [K]
Temperature contrast between bottom and top	ΔT	$T_b - T_0$
Thermal expansion coefficient	α	3.1 × 10^{-5} [K^{-1}]
Thermal diffusivity	κ	10^{-6} [m^2/s]
Reference depth	D_0	600 [km]
Gravity acceleration	g	9.8 [m/s^2]
Solidus coefficient[a]	a	−5.315 [°C/GPa^2]
	b	135.675 [°C/GPa^2]
	c	1107.113 [°C]

We also find a setup figure that shows the geometry, boundary conditions, initial temperature, etc ...



In the text we further find: $L_x = 1200$ km and $L_y = 600$ km. The lithospheric thickness decreases from 0.333 (200 km) to 0.083 (50 km) through the necking zone with a finite width (i.e., NZ). Most of the numerical models in the paper use a lithospheric geometry with NZ=30 km. However, other lithospheric geometries with NZ=1200 km and 300 km were tested to explore the effect of NZ on EDC (edge-driven convection) cell generation.

The viscosity of the lithosphere is 10^5 times that of the asthenosphere. All viscosities are linear and temperature independent (although later on in the paper the authors explore temperature-dependent viscosities). The authors use Ra values from 10^5 to 10^7 by controlling mantle viscosity from $300 \cdot 10^{19}$ Pa s to $3 \cdot 10^{19}$ Pa s, respectively. Radiogenic and adiabatic heating within the crust are ignored.

The thermal boundary conditions of the model are $T_t = 273$ K and $T_b = 1673$ K for the top and bottom temperatures and from the figure we see that no heat-flux boundary conditions are prescribed on the sides. The initial temperature was set to 1673 K along and below the LAB (thick black line on the figure above). In

order to explore the effect of long-wavelength thermal heterogeneity on EDC the authors add a background temperature heterogeneity so that the temperature in the asthenosphere is given by:

$$T(x,y) = T_b + T_H \sin\left(\frac{\pi y}{L_y}\right)\cos\left(\frac{\pi x}{2L_y}\right)$$

where $T_H \in \{0.1, 0.01, 0.001\} \cdot 1400$ K refers to the magnitude of temperature heterogeneity.

The temperature field inside the lithosphere is shown in the right insert of the figure above but its exact expression is not specified, nor how it is prescribed inside the necking zone. All we know is that "A thicker lithosphere has a more gradual geothermal gradient than a thinner lithosphere".

Based on the equations, and in the absence of a Rb number in the rhs (e.g. [68]) that accounts for the different buoyancy of the materials in the domain, we will assume that both lithosphere and asthenosphere have the same density ρ_0 and thermal expansion coefficient α. From the equations above we can infer that the authors use the Boussinesq approximation, with $\rho(T) = \rho_0(1 - \alpha T)$ in the rhs of the momentum equation.

When it comes to the boundary conditions for the Stokes equations, we read "We did not impose any kinematic boundary condition for driving mantle convection", which (also based on the figures) probably means that free slip boundary conditions are imposed on all sides.

The authors solve the equations with the commercial code COMSOL. The finite element method is used with $Q_2 \times Q_1$ elements for velocity and pressure respectively and Q_2 elements are used for temperature. They use a resolution of 5 km, so that there are 240×120 elements in the domain[1].

On the back of the observations above I contacted Prof. So, and asked him the following questions. His answers (to the best of his ability, since the study was already 5 years old upon my contacting him, the original COMSOL input file was lost and he was not the first author) came fast and were nicely detailed. They are reproduced below in dark gray:

1. Do both lithosphere and asthenosphere have the same density?
   yes

2. What is the exact shape of the LAB. How is it parameterised?
   Lab was very simply parameterized. I think that it follows the tanh function[2].

3. How is the initial temperature inside the lithosphere prescribed?
   I remember that we made an initial temperature profile using steady-state heat conduction. (top temperature: 0, LAB temperature: 1, and bottom temperature: 1).
   This assertion is refuted by the right insert of Fig. 1 of the paper where temperature profiles seem to indicate a half cooling space of some sort. This does not matter that much since the lithosphere does not deform.

4. Is the perturbation with the T_H term applied *everywhere* in the asthenosphere?
   Yes. All over the mantle. I think the brown line above has confused you. I am sorry.

5. The temperature perturbation is not zero on the LAB, which renders the initial temperature field discontinuous there. Is this not a problem?
   It is important. I didn't recognize it. But the reference model was similar to King and Anderson (1998). So, I thought it was not a problem.

6. What is not clear from the setup in the paper is whether or not the lithosphere deforms with time. We know that time stepping is carried out (temperature is advected, the v_{rms} evolves with time), but if the lithosphere also deforms with the flow, what is the technique employed? particle-in-cell?

[1] We read in the paper: "The domain for numerical simulation was divided by 241×121 grids in the horizontal and vertical directions, respectively". This obviously makes little sense. Also the mesh would then consist of 481×241 velocity nodes, not 241×121.

[2] `https://www.comsol.com/support/learning-center/article/Modeling-Step-Transitions-50151`

compositional fields?

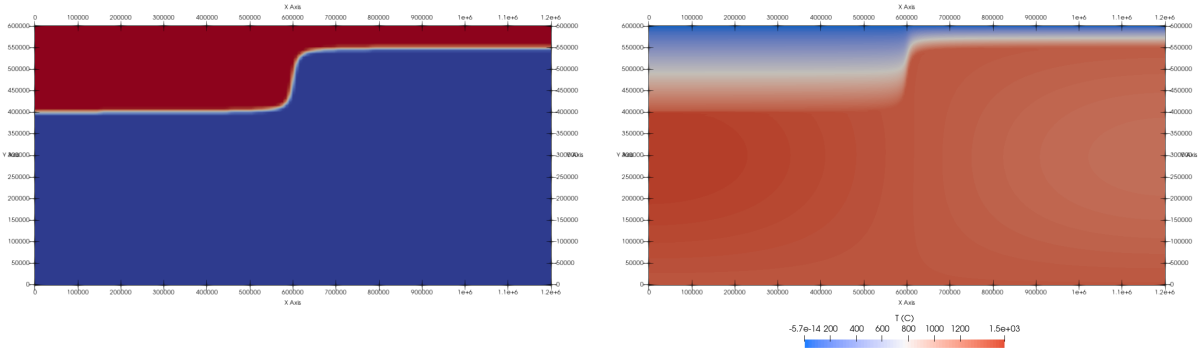
True. It is a very simple model like King and Anderson (1998). I think that the viscosity of the lithosphere has no meaning, only with much higher viscosity with viscosity. Also, there is no temperature dependence on viscosity. So the velocity field of the lithosphere is almost zero compared to the mantle. Above the LAB, the viscosity is simply fixed to large viscosity. Then the velocity is very small. The only advection quantity is temperature in our simple modeling. If we include the temperature dependence, the deformation of the mantle is very complicated.

My implementation

The code also relies on $Q_2 \times Q_1$ elements for the Stokes equations and Q_2 elements for the temperature equation. The LAB is parameterized as follows:

```
def yLAB(x,NZ):
    return np.arctan((x-Lx/2)/(NZ*4))/(np.pi/2)*75e3+475e3
```

In the absence of more information in the article it is a bit difficult to tune the term in the arc tangent function so as to obtain a necking zone of the right dimension. I found that NZ*4 does the job.



Mesh 150×75 elements, $T_H = 0.1\Delta T = 140$ °C.

Results

In Eq. 1 of the paper we see that the characteristic time is given by $D_0^2/\kappa = (6 \cdot 10^5)^2/10^{-6} = 36 \cdot 10^{16}$ s $\simeq$ 11408 Myr.

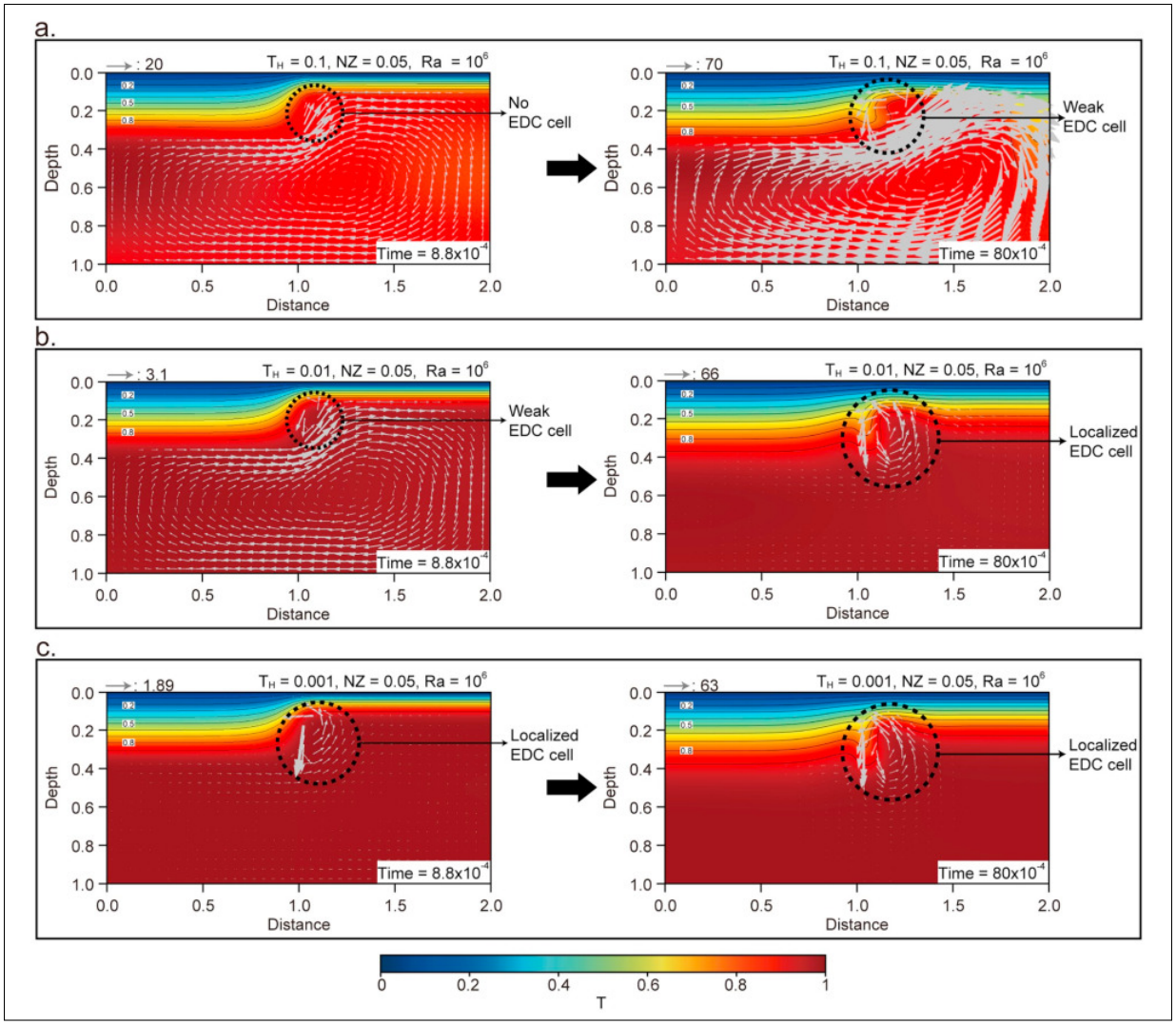
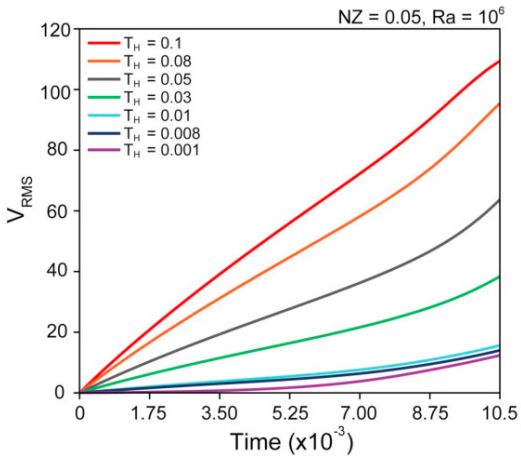


Fig. 2 of the paper. NZ=0.05 indicates that $NZ = 0.05 * L_y = 30$ km, Ra $= 10^6$ indicates that $\eta =$ and Time=$8.8 \cdot 10^{-4}$ means Time=$8.8 \cdot 10^{-4} \cdot 11408$ Myr $\simeq 10$ Myr and Time=$8.8 \cdot 10^{-4} \cdot 11408$ Myr $\simeq 91$ Myr (this value is mentioned in the text of section 3).

Looking at the figure above we must acknowledge the poor choice of color scale. Nevertheless, in order to ease comparison, the code also exports the dimensionless temperature and it will be plotted with the same color scale.

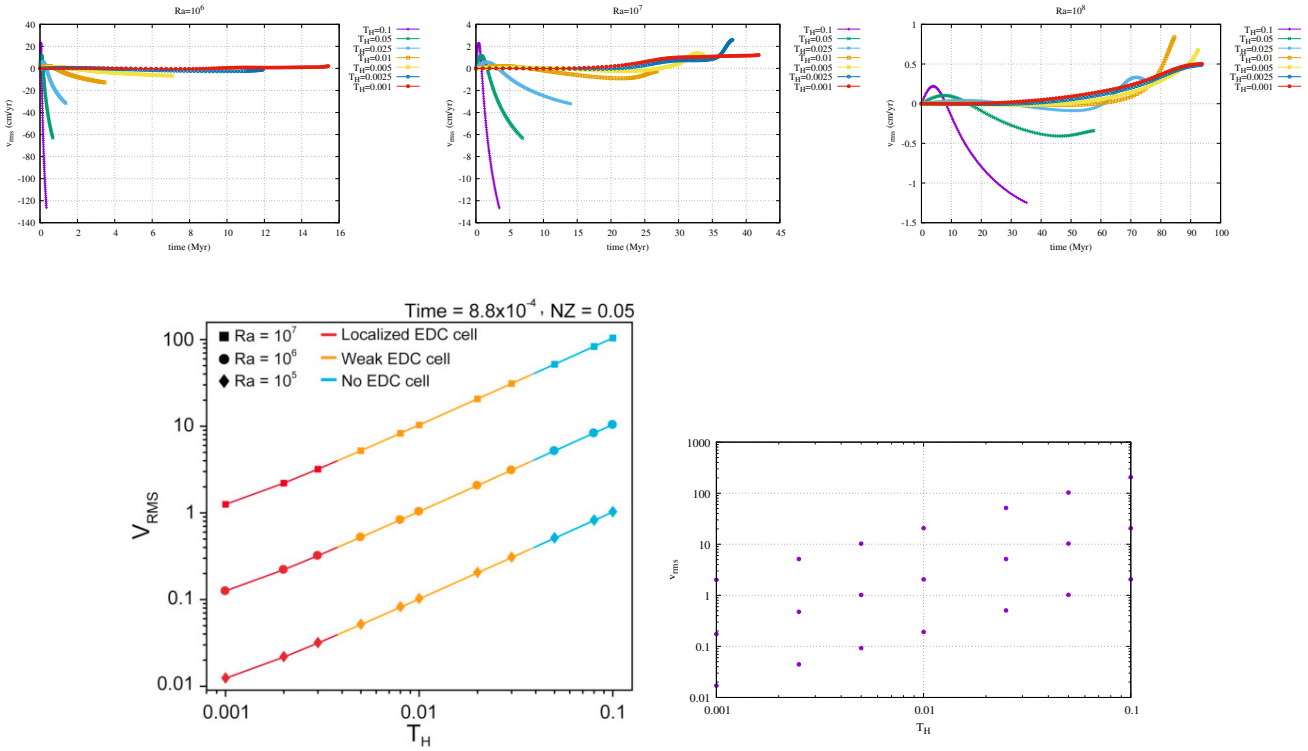
In the text of section 3 we read that the fastest mantle flow velocity in each model was 43, 4 and 2.85 as shown in Fig. 2a,b,c. In Eq. 1 we find that the caharacteristic velocity is given by $\kappa/D_0 = 10^{-6}/6 \cdot 10^5 \simeq 1.67e-12$ m s^{-1} $= 0.0052596$ cm yr^{-1}. The mentioned velocities are then 0.22cm/yr, 0.021cm/yr and 0.015cm/yr.

We now turn to the time evolution of the root mean square velocity:

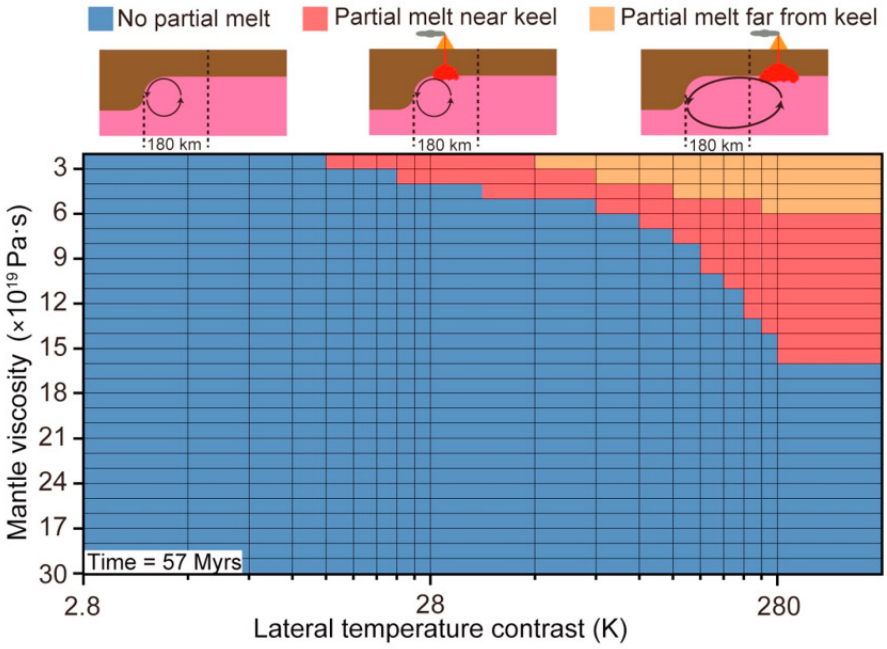


ν_{rms} of the mantle flow over an entire domain for $T_H = 0.001 - 0.1$

I have a fundamental problem with this plot as the velocity seems to start at zero. Since we are solving the incompressible Stokes equations with no elasticity, the velocity cannot be zero after the first solve. Indeed when I record the ν_{rms} as a function of time, I obtain the following left figure. If I then proceed to subtract the initial root mean square velocity value to all all measured value, I obtain the following right figure:



Left: ν_{rms} depending on Ra when $t = 8.8 \cdot 10^{-4}$. Different symbols represent the different values of Ra while different colors represent different small-scale convection styles along LAB: localized (red), weak (orange), and no (blue) EDC cells.



A map showing the cases of partial melting near the lithospheric keel (red), partial melting far from the keel (orange), and no partial melting (blue) with axes of mantle viscosity ($3 \cdot 10^{19}$ Pa s $- 30 \cdot 10^{19}$ Pa s) and lateral temperature contrast (2.8 K $-$ 280 K). Illustrations above the map show small-scale convection styles and partial melts pattern.

Current state

As of now there are still multiple problems.

- my vrms increases but then decreases
- i need to run a bunch of TH values and Ra number combinations. i need to script it all. but it takes a long time to run
- i am not sure that the initial T in the lith is correct
- i need to benchmark the code before running all models (Stokes, adv, diff)

Stone 169

equilibration of partially melted metamorphic rocks, replicating Stüwe (1995)



python_codes/fieldstone_169/text.tex

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_169`

This stone was suggested to me by Alissa Kotowski.

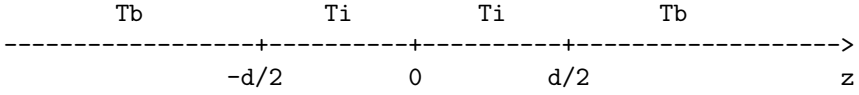
This is an attempt at replicating the publication by Kurt Stüwe. "Thermal buffering effects at the solidus. Implications for the equilibration of partially melted metamorphic rocks". In: *Tectonophysics* 248.1-2 (1995), pp. 39–51. DOI: 10.1016/0040-1951(94)00282-E. When replicating a study I always first reproduce the abstract to give context:

> A simple thermal model is used to investigate to what extent buffering effects of latent heat of fusion may prolong the thermal evolution of a one-dimensional, statically cooling, partially or completely melted heat source. For a melting model appropriate for crustal rocks, the time of cooling of rocks through the solidus may be about tripled by this process. Depending on the melting model, cooling may halt completely near the solidus for time spans comparable to the thermal time constant of the heat source.
> In order to test the influence of this buffering effect on the equilibration of partially melted metamorphic rocks, these thermal model results are coupled with a simple diffusion model that relates the degree of equilibration of a simple assemblage to temperature and cooling rate. It is shown that, for a broad range of initial heat source temperatures, freezing of selected mineral equilibria may occur in a comparably narrow temperature range at and around the solidus. The calculations may have some relevance to low-P high-T metamorphic terranes in as much as they may provide an explanation for the widely observed equilibration and partial re-equilibration of rocks near the crystallisation temperatures of partial melts.

Theory

The cooling history of an intrusion is described by the thermal relaxation of an initial one-dimensional, step-shaped temperature distribution

This initial condition may be described by $T = T_b$ in the region $|z| > d/2$ and $T = T_i$ in the region $|z| < d/2$ where T_b is the background temperature, T_i is the intrusion temperature, d is the intrusion thickness and z is the distance from the intrusion centre.

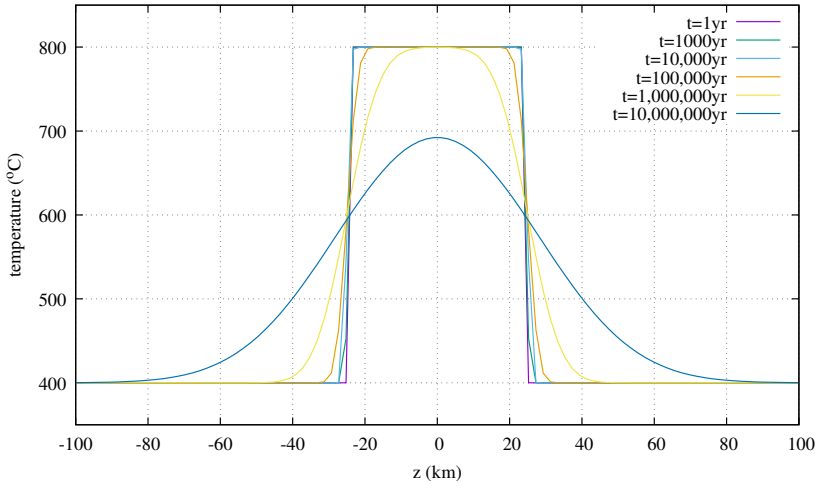


For this initial condition, and if no latent heat is considered, the thermal energy balance can be solved analytically to give temperature as a function of time, t, and distance by [169]:

$$T(z,t) = T_b + \frac{T_i - T_b}{2}\left(\mathrm{erf}\left(\frac{d/2+z}{\sqrt{4\kappa t}}\right) + \mathrm{erf}\left(\frac{d/2-z}{\sqrt{4\kappa t}}\right)\right) \tag{169.1}$$

where κ is the thermal diffusivity.

For the thermal parameters we use average values appropriate to the Earth's crust that are commonly used in the literature: $\rho = 2800\ \mathrm{kg\,m^{-3}}$, $k = 2.5\ \mathrm{W\,m^{-1}\,K^{-1}}$, $C_p = 1100\ \mathrm{J\,kg^{-1}\,K^{-1}}$. Both C_p and k are assumed to be independent of temperature[1] so that they can be combined into the thermal diffusivity $\kappa = 8.11688 \cdot 10^{-7}\ \mathrm{m^2\,s^{-1}}$. For the latent heat of fusion a value of $L = 320\ \mathrm{kJ\,kg^{-1}}$ is assumed. Parameter space is explored for intrusions of thicknesses $d = 20 - 100$ km, intrusion temperatures $T_i = 600 - 1000\ ^\circ$C and ambient background temperatures of $T_b = 400\ ^\circ$C.



Plot obtained with Eq. (169.1) for $T_b = 400\ ^\circ$C, $T_i = 800\ ^\circ$C and $d = 50$ km.

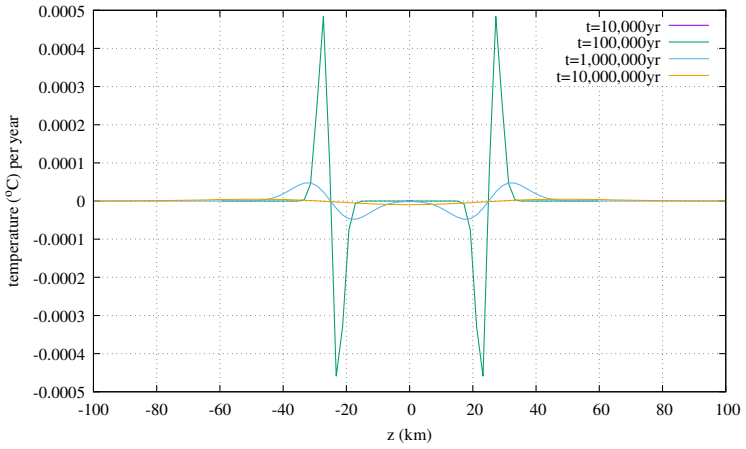
For many geological purposes it is useful to look at the temperature-cooling-rate evolution rather than at the temperature-time evolution. For the latent heat absent case, this may be found by differentiating the equation above with respect to time so that the cooling rate, s, is given by[2]:

$$s(z,t) = \frac{\partial T}{\partial t} = -\frac{T_i - T_b}{4\sqrt{\pi\kappa t^3}}\left(\frac{d/2+z}{\exp((d/2+z)^2/4\kappa t)} + \frac{d/2-z}{\exp((d/2-z)^2/4\kappa t)}\right)$$

We can plot this quantity in space at different times:

[1] Although not made explicit in the paper so is density

[2] I have not checked the maths



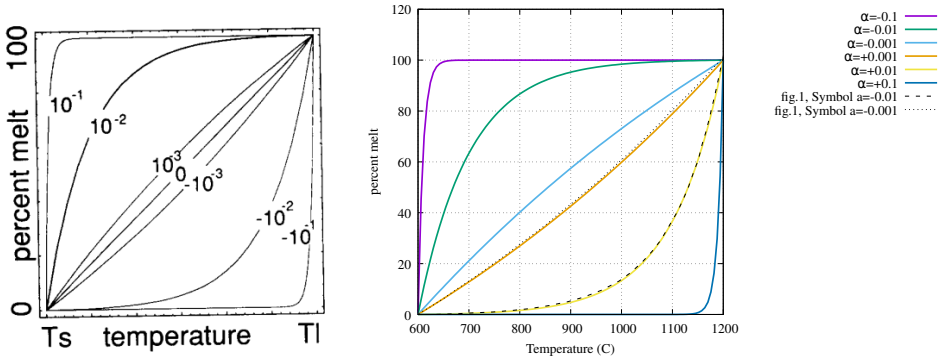
Plot obtained for $T_b = 400$ °C, $T_i = 800$ °C and $d = 50$ km.

Melting of multi-component metamorphic rocks is likely to occur between a solidus temperature, T_s, and a liquidus temperature, T_l. Between these two temperatures melting is likely to occur along a series of melt producing reactions but details of the melt volume-temperature relationships are poorly understood, in part because they depend on a range of ill-constrained variables, for example, the presence of a fluid (e.g., Vielzeuf and Holloway, 1988). Here a simple model function of the form:

$$v(T) = \frac{\exp(\alpha T) - \exp(\alpha T_s)}{\exp(\alpha T_l) - \exp(\alpha T_s)} \tag{169.2}$$

is used in which the rate of melt production between T_s and T_l can be controlled by varying α. Obviously, when $T \to T_s$ then $v \to 0$ (no melt) while when when $T \to T_l$ then $v \to 1$ (100% melt).

For the melting model, a solidus temperature of $T_s = 600$ °C and a liquidus temperature of $T_l = 1200$ °C are assumed.



Left: fig. 1 (taken from the paper) of melt volume against temperature taken as given by the melting model of Eq. (169.2). Different melting models may be explored by varying the parameter α, for which the diagram is contoured. In most of the following calculations $\alpha = -0.01$ is used (thick line). Right: my own version of the plot. Note that there is an obvious sign issue with the labels; also, although not so clear, the thick line seems to be for $\alpha = +0.01$ and not $\alpha = -0.01$ in fig. 1..

Side note: Based on my experience in STONE 167, the question remains whether the author used temperatures in Celsius or Kelvin in his Eq. (3). Fortunately, this does not matter since we have:

$$\begin{aligned} v(T) &= \frac{\exp(\alpha(T+273)) - \exp(\alpha(T_s+273))}{\exp(\alpha(T_l+273)) - \exp(\alpha(T_s+273))} \\ &= \frac{\exp(\alpha T)\exp(\alpha\cdot 273) - \exp(\alpha T_s)\exp(\alpha\cdot 273)}{\exp(\alpha T_l)\exp(\alpha\cdot 273) - \exp(\alpha T_s)\exp(\alpha\cdot 273)} \\ &= \frac{\exp(\alpha T) - \exp(\alpha T_s)}{\exp(\alpha T_l) - \exp(\alpha T_s)} \end{aligned}$$

This means that from Fig. 1 alone I cannot determine whether calculations were carried out in °C or K.

For positive α most of the melt production occurs near T_l, for negative a most of the melt production occurs near T_s (This seems to agree with my plot rather than the one in the paper?). For this melting model the thermal energy balance may not be solved analytically and a finite difference scheme was employed to obtain numerical solutions in the article.

These solutions were obtained by incorporating the effects of latent heat into a modified heat capacity (see here after) and the results were back checked by comparing them with the analytical solution of Stefan (1891) for $T_i = T_l$ and large negative α.

We start from the 1D heat transport equation with no advection term:

$$\rho C_p \frac{\partial T}{\partial t} + \rho L \frac{\partial v}{\partial t} = k \frac{\partial^2 T}{\partial z^2}$$

where $\partial v/\partial t$ describes the change in the proportion of melt with time. It is assumed that the intrusion is not completely molten, but that the melt percentage at is given by Eq. (169.2). In order to solve this equation it is useful to replace:

$$\frac{\partial v}{\partial t} = \frac{\partial v}{\partial T} \frac{\partial T}{\partial t}$$

so that now the heat transport PDE is given by

$$\rho C_p \frac{\partial T}{\partial t} + \rho L \frac{\partial v}{\partial T} \frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial z^2}$$

or,

$$\rho \left(C_p + L \frac{\partial v}{\partial T} \right) \frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial z^2}$$

and finally we obtain (what we call approach 1):

$$\boxed{\rho \underbrace{\left(C_p + L \frac{\partial v}{\partial T} \right)}_{C_p(T)} \frac{\partial T}{\partial t} = k \frac{\partial^2 T}{\partial z^2}}$$

or, (what we will call approach 2):

$$\boxed{\frac{\partial T}{\partial t} = \frac{k}{\underbrace{\rho \left(C_p + L \frac{\partial v}{\partial T} \right)}_{\kappa(T)}} \frac{\partial^2 T}{\partial z^2}}$$

Both are nonlinear 1D diffusion equations. The second approach is taken in the paper (see Eq. 8, obtained with FDM). In passing, let us note that the author 1) does not mention boundary conditions at all, 2) uses an explicit time discretisation, 3) does not elaborate about how he deals with the nonlinearities.

Concerning the boundary conditions, all what we have to do is consider a domain large enough so as to avoid boundary effects and prescribe T_b on both ends.

Remark: when discretised using FEM, we will obtain the following system:

$$\mathbb{M} \cdot \dot{\vec{\mathcal{T}}} + \mathbb{K}_d \cdot \vec{\mathcal{T}} = 0$$

In approach 1 the mass matrix $\mathbb{M}$ is nonlinear while the diffusion matrix $\mathbb{K}_d$ is linear. In approach 2 the mass matrix $\mathbb{M}$ is linear while the diffusion matrix $\mathbb{K}_d$ is nonlinear. When discretised using FDM and resorting to an explicit scheme we will naturally adopt approach 2.

From Eq. (169.2) we find that

$$\frac{\partial v}{\partial T} = \frac{\alpha \exp(\alpha T)}{\exp(\alpha T_l) - \exp(\alpha T_s)} = A \exp(\alpha T)$$

so that the heat capacity coefficient is given by

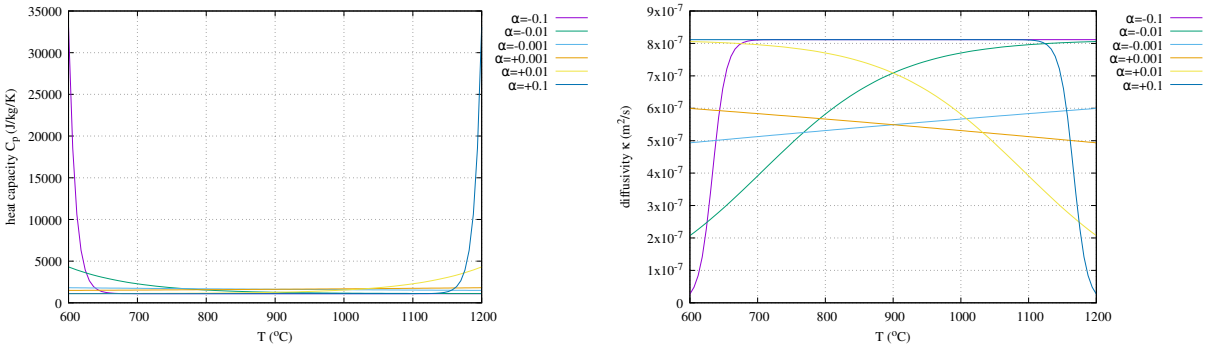
$$C_p(\alpha, T) = C_p + LA \exp(\alpha T)$$

and the diffusion coefficient by

$$\kappa(\alpha,T)=\frac{k}{\rho\left(C_p+LA\exp(\alpha T)\right)}$$

which is only valid for temperatures between T_s and T_l.

We can plot the heat capacity $C_p(\alpha,T)$ and heat diffusivity $\kappa(\alpha,T)$:



We find that the value of α can alter the value of κ by almost an order of magnitude.

At this stage we can make a few remarks about the context of the study:

1. melt is not moving (no advection),
2. the matrix (host rock) is not deforming,
3. 1D system implies an infinite extent in x,y directions.

Discretisation and implementation

Linear case (no melting)

We adopt here a very simple FDM approach, a forward in time, centered in space approach (FTCS) as explained in Section 6.4. Except for the nodes on the boundary (where the temperature is prescribed to be T_b) we have the following explicit stencil:

$$T_i^{n+1}=T_i^n+\frac{\kappa_0\delta t}{h^2}(T_{i+1}^n-2T_i^n+T_{i-1}^n)$$

with $\kappa_0=k/\rho/C_p$. This translates as follows in the code:

```
Time=0.
for istep in range(0,nstep):
    Time+=dt
    for i in range (0,nnx):
        if linear:
           kappa[i]=kappa0
           if i==0:
              Tnew[i]=Tleft
           elif i==nnx-1:
              Tnew[i]=Tright
           else:
              Tnew[i]=Told[i]+dt*kappa[i]/h**2*(Told[i+1]-2*Told[i]+Told[i-1])
           Ta[i]=analytical_solution(x[i],Time,kappa[i])
        else:
           ...
        #end if
    #end for
    Told[:]=Tnew[:]
#end for
```

We define the analytical solution function as follows:

```
def analytical_solution(x,t,kappa):
    return Tb+(Ti-Tb)/2*\
           (\
           erf((d/2+x)/np.sqrt(4*kappa*t)) +\
           erf((d/2-x)/np.sqrt(4*kappa*t)) \
           )
```

The timestep value is computed by means of a CFL conditions:

```
dt=CFL_nb*h**2/2/kappa
```

The initial temperature is prescribed as follows:

```
Told=np.zeros(nnx,dtype=np.float64)
Told[:]=Tb
for i in range(0,nnx):
    if abs(x[i])<d/2: Told[i]=Ti
```

Note that it would be trivial to produce an implicit code with Crank-Nicolson time stepping that would prove to be more stable and accurate but we will see in what follows that it makes little sense to do so (for now?).

Remark: the code also solves the nonlinear equation with melting but we will see that results are not satisfying so I have not bothered to describe this part of the code yet.

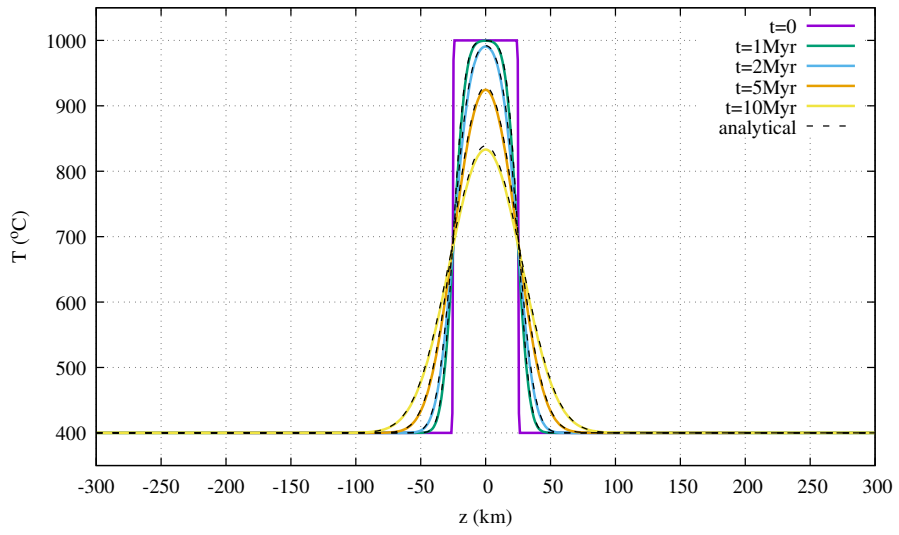
Results

Linear case (no melting)

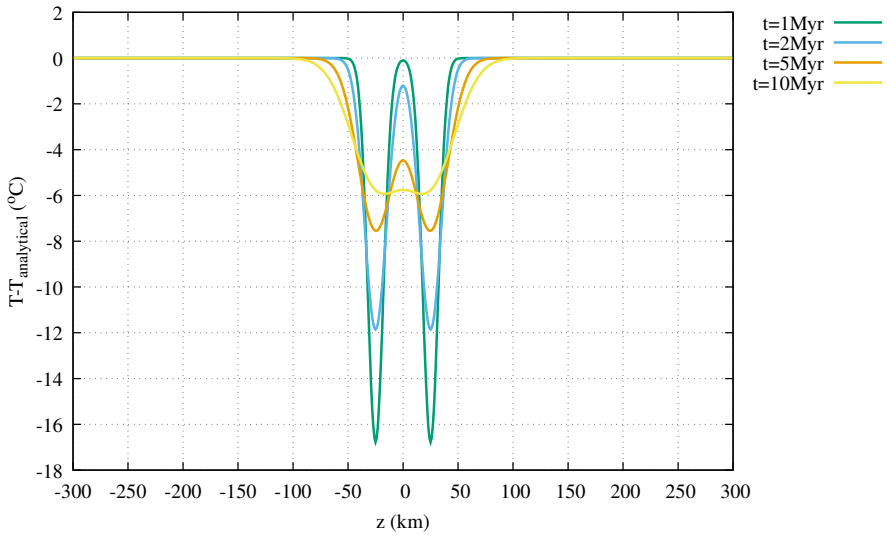
I have run the code with the following parameters:

```
nnx=601
nstep=1000000
Lx=600e3
rho=2800
Cp=1100
k=2.5
kappa0=k/rho/Cp
L=320e3
Tb=400+TKelvin
Ti=1000+TKelvin
Ts=600+TKelvin
Tl=1200+TKelvin
d=50e3
CFL_nb=0.1
tfinal=150e6*year
```

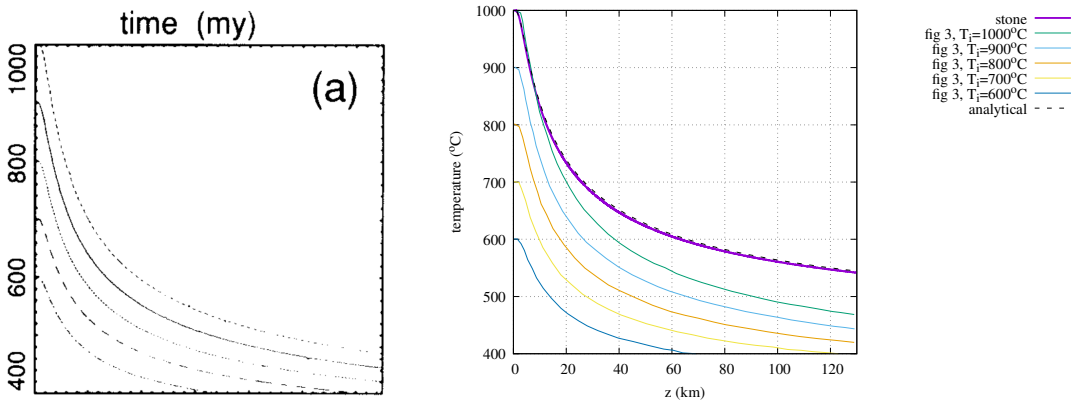
This means that the spatial resolution is 1 km, and the time step is approximately 1952 year. We find that the temperature field indeed follows the analytical solution:



The absolute error is only a few degrees at most:



We now turn to Fig. 3a of the paper which shows the cooling history of a sill-shaped intrusion of thickness $d = 50$ km at $z = 0$ for $T_i = 600 - 1000$ °C in the absence of melting. I have digitized[3] the lines of Fig. 3 and results are show here under:



Left: fig. 3a of the paper; Right: time evolution of the temperature at $z = 0$ for $T_i = 1000$ °C in purple, next to the digitized lines from the paper.

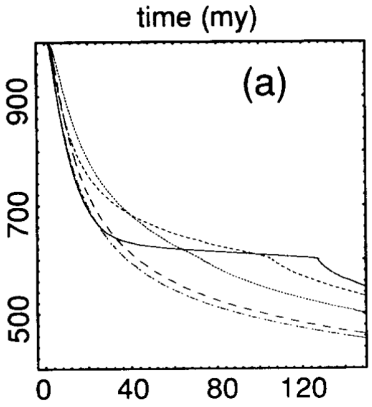
Since this is the linear case we have an analytical solution at our disposal in Eq. (169.1). I find that here again my results do agree well with the analytical solution but those of Stüwe do not (see green curve)! Note that the author specifies in the caption of Fig. 3 that it is indeed in the absence of latent heat.

If linear case results in the paper are off with respect to the analytical solution we cannot expect the nonlinear results (with melt) to match...

Nonlinear case (melting)

We now turn to Fig. 2 of the paper. In this case we set $d = 50$ km and $T_i = 1000$ °C.

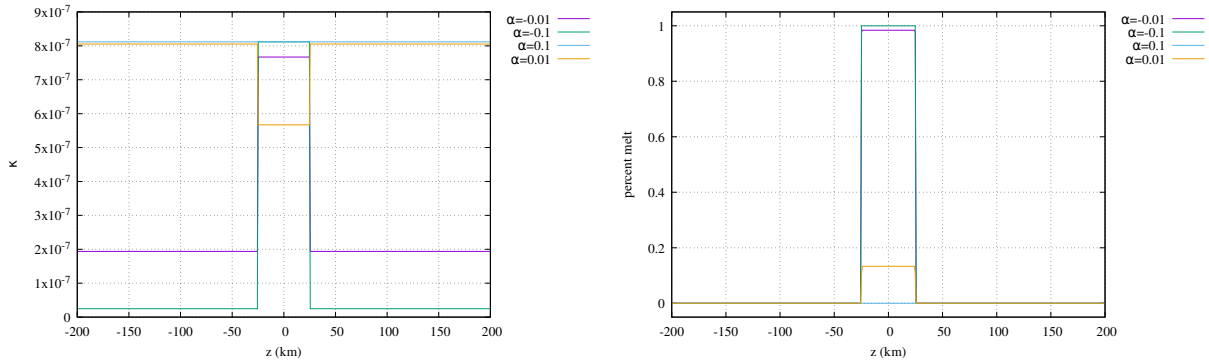
[3] `https://plotdigitizer.com/app`



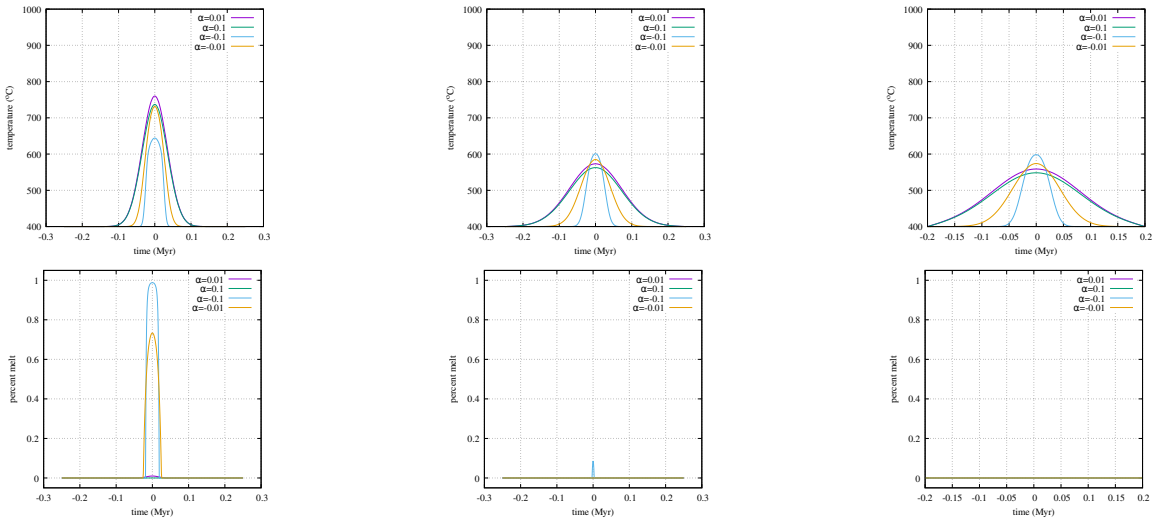
We have

$$A = \frac{\alpha}{\exp(\alpha T_l) - \exp(\alpha T_s)}$$

We can plot κ and the melt percentage in space at the end of the first timestep:

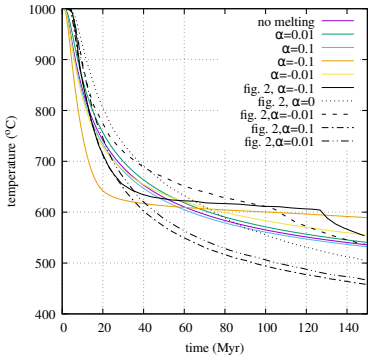


I now plot the temperature profile and the melt profile at three different times:



I have digitized[4] the lines of Fig. 2. finally I can look again at the temperature in the middle of the layer as a function of time:

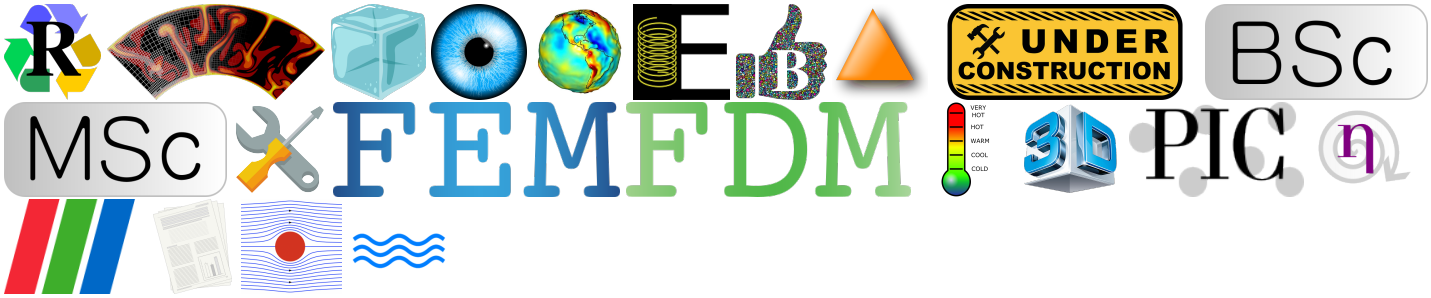
[4] `https://plotdigitizer.com/app`



Unsurprisingly my results do not match those of the paper ...

Stone 170

Title of the stone



python_codes/fieldstone_XXX/text.tex

keyword1
keyword2
keyword3
keyword4

P Code: `https://github.com/cedrict/fieldstone/tree/master/python_codes/fieldstone_XXX`

This stone was developed in collaboration with Donald Duck.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

To Do, open questions, future work?

- do smthg

 Relevant Literature:

xxxxYY

Appendix A

Classification

A.1 Stones I wrote/adapted/used for reviewing a paper

A.2 Stones which are replicating a paper



3, 28, 38, 39, 59, 63, 142, 143

A.3 Stones which served as a basis for a publication

A.4 Stones (partially) based on BSc/MSc thesis

BSc MSc 96

Bibliography

[1] Keiiti Aki and Paul G Richards. *Quantitative seismology*. 2002.

[2] M. Albers. "A local mesh refinement multigrid method for 3D convection problems with strongly variable viscosity". In: *J. Comp. Phys.* 160 (2000), pp. 126–150.

[3] J. Allen and C. Beaumont. "Continental Margin Syn-Rift Salt Tectonics at Intermediate Width Margins". In: *Basin Research* (2014). DOI: 10.1111/bre.12123.

[4] Philip A Allen and John R Allen. *Basin analysis: Principles and application to petroleum play assessment*. John Wiley & Sons, 2013.

[5] P.-A. Arrial and M.I. Billen. "Influence of geometry and eclogitization on oceanic plateau subduction". In: *Earth Planet. Sci. Lett.* 363 (2013), pp. 34–43. DOI: 10.1016/j.epsl.2012.12.011.

[6] M. Arroyo, C. Couder-Castaneda, A. Trujillo-Alcantara, I.E. Herrera-Diaz, and N. Vera-Chavez. "A Performance Study of a Dual Xeon-Phi Cluster for the Forward Modelling of Gravitational Fields". In: *Scientific Programming* (2015). DOI: 10.1155/2015/316012.

[7] W. Bai. "The quadrilateral Mini finite element for the Stokes problem". In: *Computer Methods in Applied Mechanics and Engineering* 143 (1997), pp. 41–47. DOI: 10.1016/S0045-7825(96)01146-2.

[8] Wolfgang Bangerth and Oliver Kayser-Herold. "Data structures and requirements for hp finite element software". In: *ACM Transactions on Mathematical Software (TOMS)* 36.1 (2009), pp. 1–31. DOI: 10.1145/1486525.1486529.

[9] Terence D Barr and Gregory A Houseman. "Deformation fields around a fault embedded in a non-linear ductile medium". In: *Geophy. J. Int.* 125.2 (1996), pp. 473–490. DOI: 10.1111/j.1365-246X.1996.tb00012.x.

[10] Terence D Barr and Gregory A Houseman. "Distribution of deformation around a fault in a non-linear ductile medium". In: *Geophys. Res. Lett.* 19.11 (1992), pp. 1145–1148. DOI: 10.1029/92GL00863.

[11] T.S. Baumann, B.J.P. Kaus, and A.A. Popov. "Constraining effective rheology through parallel joint geodynamic inversion". In: *Tectonophysics* 631 (2014), pp. 197–211. DOI: 10.1016/j.tecto.2014.04.037.

[12] C. Beaumont, R.A. Jamieson, J.P. Butler, and C.J. Warren. "Crustal structure: A key constraint on the mechanism of ultra-high-pressure rock exhumation". In: *Earth Planet. Sci. Lett.* 287 (2009), pp. 116–129. DOI: 10.1016/j.epsl.2009.08.001.

[13] Christopher Beaumont. "The evolution of sedimentary basins on a viscoelastic lithosphere: theory and examples". In: *Geophy. J. Int.* 55.2 (1978), pp. 471–497. DOI: 10.1111/j.1365-246X.1978.tb04283.x.

[14] J.M. Becker and M. Bevis. "Love's problem". In: *Geophy. J. Int.* 156 (2004), pp. 171–178. DOI: 10.1111/j.1365-246X.2003.02150.x.

[15] M. Behr. "On the Application of Slip Boundary Condition on Curved Boundaries". In: *Int. J. Num. Meth. Fluids* 45 (2004), pp. 43–51. DOI: 10.1002/fld.663.

[16] A. Bellas, Sh. Zhong, D. Bercovici, and E. Mulyukova. "Dynamic weakening with grain-damage and implications for slab detachment". In: *Phys. Earth. Planet. Inter.* 285 (2018), pp. 76–90. DOI: 10.1016/j.pepi.2018.09.001.

[17] D. Bercovici, G. Schubert, and G.A. Glatzmaier. "Three-dimensional convection of an infinite Prandtl-number compressible fluid in a basally heated spherical shell". In: *J. Fluid Mech.* 239 (1992), pp. 683–719. DOI: 10.1017/S0022112092004580.

[18] B. Deglo de Besses, A. Magnin, and P. Jay. "Sphere drag in a viscoplastic fluid". In: *AIChE Journal* 50.10 (2004), pp. 2627–2629. DOI: 10.1002/aic.10252.

[19] M.J. Beuchert and Y.Y. Podladchikov. "Viscoelastic mantle convection and lithospheric stresses". In: *Geophy. J. Int.* 183 (2010), pp. 35–63. DOI: 10.1111/j.1365-246X.2010.04708.x.

[20] Jilles van den Beukel and Rinus Wortel. "Temperatures and shear stresses in the upper part of a subduction zone". In: *Geophys. Res. Lett.* 14.10 (1987), pp. 1057–1060. DOI: 10.1029/GL014i010p01057.

[21] Peter Bird. "An updated digital model of plate boundaries". In: *Geochem. Geophys. Geosyst.* 4.3 (2003). DOI: 10.1029/2001GC000252.

[22] Richard J Blakely. *Potential theory in gravity and magnetic applications*. Cambridge university press, 1996.

[23] B. Blankenbach et al. "A benchmark comparison for mantle convection codes". In: *Geophy. J. Int.* 98 (1989), pp. 23–38. DOI: 10.1111/j.1365-246X.1989.tb05511.x.

[24] P. B. Bochev, C. R. Dohrmann, and M. D. Gunzburger. "Stabilization of Low-order Mixed Finite Elements for the Stokes Equations". In: *SIAM Journal on Numerical Analysis* 44.1 (2006), pp. 82–101. DOI: 10.1137/s0036142905444482.

[25] Pavel B Bochev, Max D Gunzburger, and Richard B Lehoucq. "On stabilized finite element methods for the Stokes problem in the small time step limit". In: *Int. J. Num. Meth. Fluids* 53.4 (2007), pp. 573–597.

[26] D. Boffi and L. Gastaldi. "On the quadrilateral $Q_2 - P_1$ element for the Stokes problem". In: *Int. J. Num. Meth. Fluids* 39 (2002), pp. 1001–1011. DOI: 10.1002/fld.358.

[27] Daniele Boffi, Franco Brezzi, and Michel Fortin. "Finite elements for the Stokes problem". In: *Mixed finite elements, compatibility conditions, and applications: Lectures given at the CIME Summer School held in Cetraro, Italy, June 26-July 1, 2006*. Springer, 2008, pp. 45–100. DOI: 10.1007/978-3-540-78319-0_2.

[28] Daniele Boffi, Nicola Cavallini, Francesca Gardini, and Lucia Gastaldi. "Local mass conservation of Stokes finite elements". In: *Journal of scientific computing* 52.2 (2012), pp. 383–400. DOI: 10.1007/s10915-011-9549-4.

[29] Fabian Böhm. "Matrix-free Implementation and Evaluation of the Enriched Galerkin Finite Element Method for the Stokes Problem with Varying Viscosity". MA thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg, 2023.

[30] O. Botella and R. Peyret. "Benchmark spectral results on the lid-driven cavity flow". In: *Computers and Fluids* 27.4 (1998), pp. 421–433.

[31] Bernard P Boudreau. *Diagenetic models and their implementation*. Vol. 505. 1997. ISBN: 978-3-642-64399-6. DOI: 10.1007/978-3-642-60421-8.

[32] W.F. Brace and D.L. Kohlstedt. "Limits on lithospheric stress imposed by laboratory experiments". In: *J. Geophys. Res.: Solid Earth* 85 (1980), pp. 6248–6252. DOI: 10.1029/JB085iB11p06248.

[33] D. Braess. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*. Cambridge, 2007. ISBN: 978-0-521705189.

[34] J.P. Brandenburg, E.H. Hauri, P.E. van Keken, and C.J. Ballentine. "A multiple-system study of the geochemical evolution of the mantle with force-balanced plates and thermochemical effects". In: *Earth Planet. Sci. Lett.* 276 (2008), pp. 1–13. DOI: 10.1016/j.epsl.2008.08.027.

[35] J. Braun and M. Sambridge. "Modelling landscape evolution on geological time scales: a new method based on irregular spatial discretisation". In: *Basin Research* 9 (1997), pp. 27–52. DOI: 10.1046/j.1365-2117.1997.00030.x.

[36] J. Braun, C. Thieulot, P. Fullsack, M. DeKool, and R.S. Huismans. "DOUAR: a new three-dimensional creeping flow model for the solution of geological problems". In: *Phys. Earth. Planet. Inter.* 171 (2008), pp. 76–91. DOI: 10.1016/j.pepi.2008.05.003.

[37] J.-C. de Bremaecker. "Is the oceanic lithosphere elastic or viscous?" In: *J. Geophys. Res.: Solid Earth* 82.14 (1977), pp. 2001–2004.

[38] Howard Brenner. "The slow motion of a sphere through a viscous fluid towards a plane surface". In: *Chemical engineering science* 16.3-4 (1961), pp. 242–251. DOI: 10.1016/0009-2509(61)80035-3.

[39] M Breuer and U Hansen. "Turbulent convection in the zero Reynolds number limit". In: *EPL (Europhysics Letters)* 86.2 (2009), p. 24004. DOI: 10.1209/0295-5075/86/24004.

[40] C.-H. Bruneau and M. Saad. "The 2D lid-driven cavity problem revisited". In: *Computers & Fluids* 35 (2006), pp. 326–348. DOI: 10.1016/j.compfluid.2004.12.004.

[41] RH Brzesowsky, SJT Hangx, N Brantut, and CJ Spiers. "Compaction creep of sands due to time-dependent grain failure: Effects of chemical environment, applied stress, and grain size". In: *J. Geophys. Res.: Solid Earth* 119.10 (2014), pp. 7521–7541. DOI: 10.1002/2014JB011277.

[42] RH Brzesowsky, CJ Spiers, CJ Peach, and SJT Hangx. "Failure behavior of single sand grains: theory versus experiment". In: *J. Geophys. Res.: Solid Earth* 116.B6 (2011). DOI: 10.1029/2010JB008120.

[43] RH Brzesowsky, CJ Spiers, CJ Peach, and SJT Hangx. "Time-independent compaction behavior of quartz sands". In: *J. Geophys. Res.: Solid Earth* 119.2 (2014), pp. 936–956. DOI: 10.1002/2013JB010444.

[44] Rudolf Heinrich Brzesowsky. "Micromechanics of sand grain failure and sand compaction". PhD thesis. Faculteit Aardwetenschappen, Utrecht University, 1995.

[45] WF Budd and TH Jacka. "A review of ice rheology for ice sheet modelling". In: *Cold Regions Science and Technology* 16.2 (1989), pp. 107–144. DOI: 10.1016/0165-232X(89)90014-1.

[46] Richard Gordon Budynas. *Advanced strength and applied stress analysis*. McGraw-Hill, 1999.

[47] H.H. Bui, R. Fukugawa, K. Sako, and S. Ohno. "Lagrangian meshfree particles method (SPH) for large deformation and failure flows of geomaterial using elastic-plastic soil constitutive model". In: *Int. J. Numer. Anal. Geomech.* 32.12 (2008), pp. 1537–1570.

[48] Susanne J.H. Buiter. "Surface deformation resulting from subduction and slab detachment". PhD thesis. Utrecht Universiteit, 2000.

[49] Hans-Peter Bunge and Mark A Richards. "The origin of large scale structure in mantle convection: effects of plate motions and viscosity stratification". In: *Geophys. Res. Lett.* 23.21 (1996), pp. 2987–2990.

[50] Steffi Burchardt, Hemin Koyi, and Harro Schmeling. "Strain pattern within and around denser blocks sinking within Newtonian salt structures". In: *Journal of Structural Geology* 33.2 (2011), pp. 145–153. DOI: 10.1016/j.jsg.2010.11.007.

[51] Steffi Burchardt, Hemin Koyi, and Harro Schmeling. "The influence of viscosity contrast on the strain pattern and magnitude within and around dense blocks sinking through Newtonian rock salt". In: *Journal of Structural Geology* 35 (2012), pp. 102–116. DOI: 10.1016/j.jsg.2011.07.007.

[52] Steffi Burchardt, Hemin Koyi, Harro Schmeling, and Lukas Fuchs. "Sinking of anhydrite blocks within a Newtonian salt diapir: modelling the influence of block aspect ratio and salt stratification". In: *Geophy. J. Int.* 188.3 (2012), pp. 763–778. DOI: 10.1111/j.1365-246X.2011.05290.x.

[53] Robert Wilbur Burgess. "The uniform motion of a sphere through a viscous liquid". In: *American Journal of Mathematics* 38.1 (1916), pp. 81–96. DOI: 10.2307/2370545.

[54] E. Burman and P. Hansbo. "Edge stabilization for the generalized Stokes problem: A continuous interior penalty method". In: *Computer Methods in Applied Mechanics and Engineering* 195 (2006), pp. 2393–2410. DOI: 10.1016/j.cma.2005.05.009.

[55] Evgene B Burov and Michel Diament. "The effective elastic thickness (T_e) of continental lithosphere: what does it really mean?" In: *J. Geophys. Res.: Solid Earth* 100.B3 (1995), pp. 3905–3927.

[56] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L.C. Wilcox. "Parallel scalable adjoint-based adaptive solution of variable-viscosity Stokes flow problems". In: *Computer Methods in Applied Mechanics and Engineering* 198 (2009), pp. 1691–1700. DOI: 10.1016/j.cma.2008.12.015.

[57] C. Burstedde et al. "Large-scale adaptive mantle convection simulation". In: *Geophy. J. Int.* 192 (2013), pp. 889–906. DOI: 10.1093/gji/ggs070.

[58] C. Burstedde et al. "Scalable Adaptive Mantle Convection Simulation on Petascale Supercomputers". In: *ACM/IEEE SC Conference Series, 2008* (2008).

[59] F.H. Busse et al. "3D convection at infinite Prandtl number in Cartesian geometry - a benchmark comparison". In: *Geophysical & Astrophysical Fluid Dynamics* 75.1 (1994), pp. 39–59. DOI: 10.1080/03091929408203646.

[60] J.P. Butler and C. Beaumont. "Subduction zone decoupling/retreat modeling explains south Tibet (Xigaze) and other supra-subduction zone ophiolites and their UHP mineral phases". In: *Earth Planet. Sci. Lett.* 463 (2017), pp. 101–117. DOI: 10.1016/j.epsl.2017.01.025.

[61] J.P. Butler, C. Beaumont, and R.A. Jamieson. "The Alps 2: Controls on crustal subduction and (ultra)high-pressure rock exhumation in Alpine-type orogens". In: *J. Geophys. Res.: Solid Earth* 119.7 (2014), pp. 5987–6022. DOI: 10.1002/2013JB010799.

[62] Marco Camesasca, Miron Kaufman, and Ica Manas-Zloczower. "Quantifying fluid mixing with the Shannon entropy". In: *Macromolecular theory and simulations* 15.8 (2006), pp. 595–607. DOI: 10.1002/mats.200600037.

[63] P Carré, AJF Metherell, and TJ Quinn. "Apropos of "The Gravitational Field of a 111 Tetrahedron"". In: *Metrologia* 23.2 (1986), p. 119. DOI: 10.1088/0026-1394/23/2/007.

[64] Carsten Carstensen, Karonline Köhler, Daniel Peterseim, and Mira Schedensack. "Comparison results for the Stokes equations". In: *Applied Numerical Mathematics* 95 (2015), pp. 118–129. DOI: 10.1016/j.apnum.2013.12.005.

[65] J.S. Chen, C. Pan, and T.Y.P. Chang. "On the control of pressure oscillation in bilinear-displacement constant-pressure element". In: *Computer Methods in Applied Mechanics and Engineering* 128 (1995), pp. 137–152.

[66] L Chen, H Nguyen-Xuan, T Nguyen-Thoi, KY Zeng, and SC Wu. "Assessment of smoothed point interpolation methods for elastic mechanics". In: *International Journal for Numerical Methods in Biomedical Engineering* 26.12 (2010), pp. 1635–1655. DOI: 10.1002/cnm.1251.

[67] M.V. Chertova, T. Geenen, A. van den Berg, and W. Spakman. "Using open sidewalls for modelling self-consistent lithosphere subduction dynamics". In: *Solid Earth* 3 (2012), pp. 313–326. DOI: 10.5194/se-3-313-2012.

[68] M.V. Chertova, W. Spakman, T. Geenen, A.P. van den Berg, and D.J.J. van Hinsbergen. "Underpinning tectonic reconstructions of the western Mediterranean region with dynamic slab evolution from 3-D numerical modeling". In: *J. Geophys. Res.: Solid Earth* 119 (2014), 10.1002/ 2014JB011150.

[69] E. Choi and K.D. Petersen. "Making Coulomb angle-oriented shear bands in numerical tectonic models". In: *Tectonophysics* 657 (2015), pp. 94–101. DOI: 10.1016/j.tecto.2015.06.026.

[70] E. Choi, E. Tan, L.L. Lavier, and V.M. Calo. "DynEarthSol2D: An efficient unstructured finite element method to study long-term tectonic deformation". In: *J. Geophys. Res.: Solid Earth* 118 (2013), pp. 1–16. DOI: 10.1002/jgrb.50148.

[71] R.R. Christensen. "An Eulerian technique for thermomechanical modeling of lithospheric extension". In: *J. Geophys. Res.: Solid Earth* 97.B2 (1992), pp. 2015–2036. DOI: 10.1029/91JB02642.

[72] Edmund Christiansen and Knud D. Andersen. "Computation of collapse states with von Mises type yield condition". In: *Int. J. Num. Meth. Eng.* 46 (1999), pp. 1185–1202.

[73] Edmund Christiansen and Ole S. Pedersen. "Automatic mesh refinement in limit analysis". In: *Int. J. Num. Meth. Eng.* 50 (2001), pp. 1331–1346.

[74] H. Ciskova, A. van den Berg, and M. Jacobs. "Impact of compressibility on heat transport characteristics of large terrestrial planets". In: *Phys. Earth. Planet. Inter.* 268 (2017), pp. 65–77.

[75] H. Ciskova, J. van Hunen, A.P. van den Berg, and N.J. Vlaar. "The influence of rheological weakening and yield stress on the interaction of slabs with the 670 km discontinuity". In: *Earth Planet. Sci. Lett.* 199 (2002), pp. 447–457. DOI: 10.1016/S0012-821X(02)00586-1.

[76] H. Čížková, A.P. van den Berg, W. Spakman, and Ctirad Matyska. "The viscosity of the earth's lower mantle inferred from sinking speed of subducted lithosphere". In: *Phys. Earth. Planet. Inter.* 200–201 (2012), pp. 56–62. DOI: 10.1016/j.pepi.2012.02.010.

[77] Marin K Clark, John WM Bush, and Leigh H Royden. "Dynamic topography produced by lower crustal flow against rheological strength heterogeneities bordering the Tibetan Plateau". In: *Geophy. J. Int.* 162.2 (2005), pp. 575–590. DOI: 10.1111/j.1365-246X.2005.02580.x.

[78] Nicolas Clausolles, Pauline Collon, and Guillaume Caumon. "Generating variable shapes of salt geobodies from seismic images and prior geological knowledge". In: *Interpretation* 7.4 (2019), T829–T841. DOI: 10.1190/INT-2019-0032.1.

[79] C. P. Conrad and L. Husson. "Influence of dynamic topography on sea level and its rate of change". In: *Lithosphere* 1.2 (2009), pp. 110–120. DOI: 10.1130/L32.1.

[80] F. Crameri et al. "A comparison of numerical surface topography calculations in geodynamic modelling: an evaluation of the 'sticky air' method". In: *Geophy. J. Int.* 189 (2012), pp. 38–54. DOI: 10.1111/j.1365-246X.2012.05388.x.

[81] C.A. Currie, C. Beaumont, and R.S. Huismans. "The fate of subducted sediments: a case for backarc intrusion and underplating". In: *Geology* 35.12 (2007), pp. 1111–1114.

[82] François Cuvelier, Caroline Japhet, and Gilles Scarella. "An efficient way to assemble finite element matrices in vector languages". In: *BIT Numerical Mathematics* 56 (2016), pp. 833–864. DOI: 10.1007/s10543-015-0587-4.

[83] FA Dahlen and Jeroen Tromp. *Theoretical global seismology*. Princeton university press, 1998.

[84] D.R. Davies, C.R. Wilson, and S.C. Kramer. "Fluidity: A fully unstructured anisotropic adaptive mesh computational modeling framework for geodynamics". In: *Geochem. Geophys. Geosyst.* 12.6 (2011). DOI: 10.1029/2011GC003551.

[85] R.O. Davis and A.P.S. Selvadurai. *Elasticity and Geomechanics*. Cambridge University Press, 1996.

[86] P. Davy and P. Cobbold. "Indentation tectonics in nature and experiment. 1. Experiments scaled for gravity". In: *Bulletin of the Geological Institutions of Uppsala* 14 (1988), pp. 129–141.

[87] Albert de Montserrat, Jason P Morgan, and Jörg Hasenclever. "LaCoDe: a Lagrangian two-dimensional thermo-mechanical code for large-strain compressible visco-elastic geodynamical modeling". In: *Tectonophysics* 767 (2019), p. 228173. DOI: 10.1016/j.tecto.2019.228173.

[88] Y. Deubelbeiss and B.J.P. Kaus. "Comparison of Eulerian and Lagrangian numerical techniques for the Stokes equations in the presence of strongly varying viscosity". In: *Phys. Earth. Planet. Inter.* 171 (2008), pp. 92–111. DOI: 10.1016/j.pepi.2008.06.023.

[89] G. Dhatt and G. Hubert. "A study of penalty elements for incompressible laminar flows". In: *Int. J. Num. Meth. Fluids* 6 (1986), pp. 1–19.

[90] I. Dione, C. Tibirna, and J. Urquiza. "Stokes equations with penalised slip boundary conditions". In: *International Journal of Computational Fluid Dynamics* 27.6-7 (2013), pp. 283–296. DOI: 10.1080/10618562.2013.821114.

[91] Ibrahima Dione and José M Urquiza. "Penalty: finite element approximation of Stokes equations with slip boundary conditions". In: *Numerische Mathematik* 129.3 (2015), pp. 587–610. DOI: 10.1007/s00211-014-0646-9.

[92] Seok-Hyeon Do and Byung-Dal So. "Finite element modeling for 2D viscoelastic buckling formation with mechanical strength of mantle and lithosphere". In: *J. Geol. Soc. Korea* 55.3 (2019), pp. 315–332. DOI: 10.14770/jgsk.2019.55.3.315.

[93] C.R. Dohrmann and P.B. Bochev. "A stabilized finite element method for the Stokes problem based on polynomial pressure projections". In: *Int. J. Num. Meth. Fluids* 46 (2004), pp. 183–201. DOI: 10.1002/fld.752.

[94] Jean Donea and Antonio Huerta. *Finite Element Methods for Flow Problems*. John Wiley & Sons, 2003. ISBN: 978-0-471-49666-3.

[95] J.R. Dormand and P.J. Prince. "A family of embedded Runge-Kutta formulae". In: *Journal of Computational and Applied Mathematics* 6.1 (1980), pp. 19–26.

[96] Jim Douglas Jr, Juan E Santos, Dongwoo Sheen, and Xiu Ye. "Nonconforming Galerkin methods based on quadrilateral elements for second order elliptic problems". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 33.4 (1999), pp. 747–770. DOI: 10.1051/m2an:1999161.

[97] T. Duretz, D.A. May, T.V. Gerya, and P.J. Tackley. "Discretization errors and free surface stabilisation in the finite difference and marker-in-cell method for applied geodynamics: A numerical study". In: *Geochem. Geophys. Geosyst.* 12.Q07004 (2011). DOI: 10.1029/2011GC003567.

[98] T. Duretz, A. Souche, R. de Borst, and L. Le Pourhiet. "The Benefits of Using a Consistent Tangent Operator for Viscoelastoplastic Computations in Geodynamics". In: *Geochem. Geophys. Geosyst.* 19 (2018). DOI: 10.1029/2018GC007877.

[99] Thibault Duretz, René de Borst, Philippe Yamato, and Laetitia Le Pourhiet. "Towards robust and predictive geodynamic modelling: the way forward in frictional plasticity". In: *Geophys. Res. Lett.* 47 (2020), e2019GL086027. DOI: 10.1029/2019GL086027.

[100] WB Durham and LA Stern. "Rheological properties of water ice—Applications to satellites of the outer planets". In: *Annual Review of Earth and Planetary Sciences* 29.1 (2001), pp. 295–330. DOI: 10.1146/annurev.earth.29.1.295.

[101] A.M. Dziewonski and D.L. Anderson. "Preliminary reference Earth model". In: *Phys. Earth. Planet. Inter.* 25 (1981), pp. 297–356. DOI: 10.1016/0031-9201(81)90046-7.

[102] R. Eid. "Higher order isoparametric finite element solution of Stokes flow ". In: *Applied Mathematics and Computation* 162 (2005), pp. 1083–1101. DOI: 10.1016/j.amc.2004.01.014.

[103] Göran Ekström. "Global Detection and Location of Seismic Sources by Using Surface Waves". In: *Bulletin of the Seismological Society of America* 96.4A (2006), pp. 1201–1212. ISSN: 0037-1106. DOI: 10.1785/0120050175.

[104] Göran Ekström, Meredith Nettles, and AM Dziewoński. "The global CMT project 2004–2010: Centroid-moment tensors for 13,017 earthquakes". In: *Phys. Earth. Planet. Inter.* 200 (2012), pp. 1–9. DOI: 10.1016/j.pepi.2012.04.002.

[105] H. Elman, D. Silvester, and A. Wathen. *Finite Elements and Fast Iterative Solvers*. Oxford Science Publications, 2014. ISBN: 978-0-19-967879-2.

[106] M.S. Engelman, R.L. Sani, and P.M. Gresho. "The implementation of normal and/or tangential boundary conditions in finite element codes for incompressible fluid flow". In: *Int. J. Num. Meth. Fluids* 2 (1982), pp. 225–238. DOI: 10.1002/fld.1650020302.

[107] Philip C England and Dave A May. "The Global Range of Temperatures on Convergent Plate Interfaces". In: *Geochem. Geophys. Geosyst.* 22 (2021), e2021GC009849. DOI: 10.1029/2021GC009849.

[108] E. Erturk. "Discussions on Driven Cavity Flow". In: *Int. J. Num. Meth. Fluids* 60 (2009), pp. 275–294.

[109] Alessandro M Forte and David B Rowley. "Earth's isostatic and dynamic topography—A critical perspective". In: *Geochem. Geophys. Geosyst.* 23.9 (2022), e2021GC009740. DOI: 10.1029/2021GC009740.

[110] M. Fortin. "Old and new finite elements for incompressible flows". In: *Int. J. Num. Meth. Fluids* 1 (1981), pp. 347–364. DOI: 10.1002/fld.1650010406.

[111] M. Fortin and A. Fortin. "Experiments with several elements for viscous incompressible flows". In: *Int. J. Num. Meth. Fluids* 5 (1985), pp. 911–928. DOI: 10.1002/fld.1650051005.

[112] Haakon Fossen and Basil Tikoff. "The deformation matrix for simultaneous simple shearing, pure shearing and volume change, and its application to transpression-transtension tectonics". In: *Journal of Structural Geology* 15.3-5 (1993), pp. 413–422. DOI: 10.1016/0191-8141(93)90137-Y.

[113] L. Fourel, S. Goes, and G. Morra. "The role of elasticity in slab bending". In: *Geochem. Geophys. Geosyst.* 15 (2014), pp. 4507–4525. DOI: 10.1002/2014GC005535.

[114] Alain Fournier, Don Fussell, and Loren Carpenter. "Computer rendering of stochastic models". In: *Communications of the ACM* 25.6 (1982), pp. 371–384. DOI: doi:10.1145/358523.358553.

[115] DA Frank-Kamenetskii. *Diffusion and heat transfer in chemical kinetics*. New York: Plenum Press, 1969.

[116] M.R.T. Fraters, W. Bangerth, C. Thieulot, A.C. Glerum, and W. Spakman. "Efficient and Practical Newton Solvers for Nonlinear Stokes Systems in Geodynamic Problems". In: *Geophy. J. Int.* 218 (2019), pp. 873–894. DOI: 10.1093/gji/ggz183.

[117] Scott W French and Barbara Romanowicz. "Broad plumes rooted at the base of the Earth's mantle beneath major hotspots". In: *Nature* 525.7567 (2015), p. 95. DOI: 10.1038/nature14876.

[118] J. de Frutos, V. John, and J. Novo. "Projection methods for incompressible flow problems with WENO finite difference schemes". In: *J. Comp. Phys.* 309 (2016), pp. 368–386.

[119] Hiromi Fujimoto and Yoshibumi Tomoda. "Lithospheric thickness anomaly near the trench and possible driving force of subduction". In: *Tectonophysics* 112.1-4 (1985), pp. 103–110. DOI: 10.1016/0040-1951(85)90174-X.

[120] Yoshio Fukao and Masayuki Obayashi. "Subducted slabs stagnant above, penetrating through, and trapped below the 660 km discontinuity". In: *J. Geophys. Res.: Solid Earth* 118.11 (2013), pp. 5920–5938. DOI: 10.1002/2013JB010466.

[121] Jean Furstoss. "Approche numérique de l'évolution microstructurale des péridotites". PhD thesis. 2020.

[122] Fanny Garel, Catherine Thoraval, Andréa Tommasi, Sylvie Demouchy, and D Rhodri Davies. "Using thermo-mechanical models of subduction to constrain effective mantle viscosity". In: *Earth Planet. Sci. Lett.* 539 (2020), p. 116243. DOI: 10.1016/j.epsl.2020.116243.

[123] Julien Gasc et al. "Deep-focus earthquakes: From high-temperature experiments to cold slabs". In: *Geology* 50 (2022), pp. 1018–1022. DOI: 10.1130/G50084.1.

[124] R. Gassmöller, H. Lokavarapu, E. M. Heien, E. G. Puckett, and W. Bangerth. "Flexible and scalable particle-in-cell methods with adaptive mesh refinement for geodynamic computations". In: *Geochem. Geophys. Geosyst.* 19.9 (2018), pp. 3596–3604. DOI: 10.1029/2018GC007508.

[125] M. Gerbault, A.N.B. Poliakov, and M. Daignieres. "Prediction of faulting from the theories of elasticity and plasticity: what are the limits?" In: *Journal of Structural Geology* 20 (1998), pp. 301–320.

[126] T.V. Gerya, D.A. May, and T. Duretz. "An adaptive staggered grid finite difference method for modeling geodynamic Stokes flows with strongly variable viscosity". In: *Geochem. Geophys. Geosyst.* 14.4 (2013). DOI: 10.1002/ggge.20078.

[127] Taras Gerya. "Geodynamics of the early Earth: Quest for the missing paradigm". In: *Geology* 47.10 (2019), pp. 1006–1007.

[128] Taras Gerya. *Numerical Geodynamic Modelling*. Cambridge University Press, 2010.

[129] Taras Gerya. *Numerical Geodynamic Modelling - 2nd edition*. Cambridge University Press, 2019. ISBN: 978-1-107-14314-2.

[130] U. Ghia, K.N. Ghia, and C.T. Shin. "High-Re Solutions for incompressible flow using the Navier-Stokes equations and a multigrid method". In: *J. Comp. Phys.* 48 (1982), pp. 387–411.

[131] Alfred Gierer and Hans Meinhardt. "A theory of biological pattern formation". In: *Kybernetik* 12.1 (1972), pp. 30–39. DOI: 10.1007/BF00289234.

[132] John W Glen. "The creep of polycrystalline ice". In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 228.1175 (1955), pp. 519–538. DOI: 10.1098/rspa.1955.0066.

[133] A. Glerum, C. Thieulot, M. Fraters, C. Blom, and W. Spakman. "Nonlinear viscoplasticity in ASPECT: benchmarking and applications to subduction". In: *Solid Earth* 9.2 (2018), pp. 267–294. DOI: 10.5194/se-9-267-2018.

[134] Heiko Goelzer, Alexander Robinson, Helene Seroussi, and Roderik Sw Van De Wal. "Recent progress in Greenland ice sheet modelling". In: *Current climate change reports* 3.4 (2017), pp. 291–302. DOI: 10.1007/s40641-017-0073-y.

[135] C Goetze. "The mechanisms of creep in olivine". In: *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 288.1350 (1978), pp. 99–119. DOI: 10.1098/rsta.1978.0008.

[136] DL Goldsby and DL Kohlstedt. "Superplastic deformation of ice: Experimental observations". In: *J. Geophys. Res.: Solid Earth* 106.B6 (2001), pp. 11017–11030. DOI: 10.1029/2000JB900336.

[137] Karine Gouriet et al. "Dislocation dynamics modelling of the power-law breakdown in olivine single crystals: Toward a unified creep law for the upper mantle". In: *Earth Planet. Sci. Lett.* 506 (2019), pp. 282–291. DOI: 10.1016/j.epsl.2018.10.049.

[138] J. Grandy. *Efficient Computation of Volume of Hexahedral Cells*. Tech. rep. UCRL-ID-128886. Lawrence Livermore National Laboratory, 1997.

[139] Jean-Pierre Gratier, Pascal Favreau, and François Renard. "Modeling fluid transfer along California faults when integrating pressure solution crack sealing and compaction processes". In: *J. Geophys. Res.: Solid Earth* 108.B2 (2003). DOI: 10.1029/2001JB000380.

[140] P.M. Gresho, S.T. Chan, M.A. Christon, and A.C. Hindmarsch. "A little more on stabilised Q_1Q_1 for transient viscous incompressible flow". In: *Int. J. Num. Meth. Fluids* 21 (1995), pp. 837–856. DOI: 10.1002/fld.1650211005.

[141] P.M. Gresho, R.L. Lee, R.L. Sani, M.K. Maslanik, and B.E. Eaton. "The consistent Galerkin FEM for computing derived boundary quantities in thermal and/or fluid problems". In: *Int. J. Num. Meth. Fluids* 7 (1987), pp. 371–394. DOI: 10.1002/fld.1650070406.

[142] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerical simulation in Fluid Dynamics*. SIAM, 1997.

[143] D.J. Griffiths. *Introduction to Electrodynamics - third edition*. Prentice-Hall, 1999.

[144] Ólafur Gudmundsson and Malcolm Sambridge. "A regionalized upper mantle (RUM) seismic model". In: *J. Geophys. Res.: Solid Earth* 103.B4 (1998), pp. 7121–7136.

[145] Frédéric Gueydan and Jacques Précigout. "Modes of continental rifting as a function of ductile strain localization in the lithospheric mantle". In: *Tectonophysics* 612 (2014), pp. 18–25. DOI: 10.1016/j.tecto.2013.11.029.

[146] R. Hall and W. Spakman. "Mantle structure and tectonic history of SE Asia". In: *Tectonophysics* 658 (2015), pp. 14–45. DOI: 10.1016/j.tecto.2015.07.003.

[147] William R Halter, Emilie Macherel, and Stefan M Schmalholz. "A simple computer program for calculating stress and strain rate in 2D viscous inclusion-matrix systems". In: *Journal of Structural Geology* 160 (2022), p. 104617. DOI: 10.1016/j.jsg.2022.104617.

[148] LN Hansen, ME Zimmerman, and David L Kohlstedt. "Grain boundary sliding in San Carlos olivine: Flow law parameters and crystallographic-preferred orientation". In: *J. Geophys. Res.: Solid Earth* 116.B8 (2011). DOI: 10.1029/2011JB008220.

[149] F.H. Harlow and J.E. Welch. "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface". In: *The physics of fluids* 8.12 (1965), p. 2182.

[150] Nathan J Harris, Steven E Benzley, and Steven J Owen. "Conformal Refinement of All-Hexahedral Element Meshes Based on Multiple Twist Plane Insertion". In: *IMR*. 2004, pp. 157–168.

[151] T. Heister, J. Dannberg, R. Gassmöller, and W. Bangerth. "High Accuracy Mantle Convection Simulation through Modern Numerical Methods. II: Realistic Models and Problems". In: *Geophy. J. Int.* 210.2 (2017), pp. 833–851. DOI: 10.1093/gji/ggx195.

[152] Timo Heister and Gerd Rapin. "Efficient augmented Lagrangian-type preconditioning for the Oseen problem using Grad-Div stabilization". In: *Int. J. Num. Meth. Fluids* 71.1 (2013), pp. 118–134. DOI: 10.1002/fld.3654.

[153] Christian Helanow and Josefin Ahlkrona. "Stabilized equal low-order finite elements in ice sheet modeling–accuracy and robustness". In: *Computational Geosciences* 22.4 (2018), pp. 951–974. DOI: 10.1007/s10596-017-9713-5.

[154] Yoshio Hiramatsu and Yukitoshi Oka. "Determination of the tensile strength of rock by a compression test of an irregular test piece". In: *International Journal of Rock Mechanics and Mining Sciences & Geomechanics Abstracts*. Vol. 3. 1966, pp. 89–90. DOI: 10.1016/0148-9062(66)90002-7.

[155] G. Hirth and D.L. Kohlstedt. "Rheology of the upper mantle and the mantle wedge: A view from the experimentalists". In: *in Inside the Subduction Factory, ed. J. Eiler, Geophysical Monograph American Geophysical Union, Washington, D.C.* 138 (2003), pp. 83–105.

[156] Greg Hirth, Christian Teyssier, and James W Dunlap. "An evaluation of quartzite flow laws based on comparisons between experimentally and naturally deformed rocks". In: *International Journal of Earth Sciences* 90.1 (2001), pp. 77–87. DOI: 10.1007/s005310000152.

[157] Anne Hoger and Donald E Carlson. "Determination of the stretch and rotation in the polar decomposition of the deformation gradient". In: *Quarterly of applied mathematics* 42.1 (1984), pp. 113–117. DOI: xxxx.

[158] K. Hosseini, K.J. Matthews, K. Sigloch, G.E. Shephard, M. Domeier, and Maria Tsekhmistrenko. "SubMachine: Web-Based Tools for Exploring Seismic Tomography and Other Models of Earth's Deep Interior". In: *Geochem. Geophys. Geosyst.* 19 (2018), pp. 1464–1483. DOI: 10.1029/2018GC007431.

[159] Yunqing Huang and Shangyou Zhang. "A lowest order divergence-free finite element on rectangular grids". In: *Frontiers of Mathematics in China* 6 (2011), pp. 253–270. DOI: 10.1007/s11464-011-0094-0.

[160] T.J.R. Hughes, L.P. Franca, and M. Balestra. "A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška-Brezzi condition: A stable Petrov-Galerkin formulation of the Stokes problem accommodating equal-order interpolations". In: *Computer Methods in Applied Mechanics and Engineering* 59.1 (1986), pp. 85–99.

[161] Hoon Huh, Choong Ho Lee, and Wei H. Yang. "A general algorithm for plastic flow simulation by finite element limit analysis". In: *International Journal of Solids and Structures* 36 (1999), pp. 1193–1207. DOI: 10.1016/S0020-7683(97)00347-8.

[162] R. Huismans and C. Beaumont. "Depth-dependent extension, two-stage breakup and cratonic underplating at rifted margins". In: *Nature* 473 (2011), pp. 74–79. DOI: 10.1038/nature09988.

[163] Ritske S Huismans and Christopher Beaumont. "Rifted continental margins: The case for depth-dependent extension". In: *Earth Planet. Sci. Lett.* 407 (2014), pp. 148–162. DOI: 10.1016/j.epsl.2014.09.032.

[164] Christian Hüttig and Kai Stemmer. "Finite volume discretization for dynamic viscosities on Voronoi grids". In: *Phys. Earth. Planet. Inter.* 171.1-4 (2008), pp. 137–146. DOI: 10.1016/j.pepi.2008.07.007.

[165] Tobin Isaac, Georg Stadler, and Omar Ghattas. "Solution of nonlinear Stokes equations discretized by high-order finite elements on nonconforming and anisotropic meshes, with application to ice sheet dynamics". In: *SIAM Journal on Scientific Computing* 37.6 (2015), B804–B833. DOI: 10.1137/140974407.

[166] Kazuhiko Ishii and Simon R Wallis. "High-and low-stress subduction zones recognized in the rock record". In: *Earth Planet. Sci. Lett.* 531 (2020), p. 115935. DOI: 10.1016/j.epsl.2019.115935.

[167] J. Ita and S.D. King. "Sensitivity of convection with an endothermic phase change to the form of governing equations, initial conditions, boundary conditions, and equation of state". In: *J. Geophys. Res.: Solid Earth* 99.B8 (1994), pp. 15, 919–15, 938. DOI: 10.1029/94JB00852.

[168] Antoine B Jacquey and Mauro Cacace. "Multiphysics Modeling of a Brittle-Ductile Lithosphere: 1. Explicit Visco-Elasto-Plastic Formulation and Its Numerical Implementation". In: *J. Geophys. Res.: Solid Earth* 125.1 (2020), e2019JB018474. DOI: 10.1029/2019JB018474.

[169] JC Jaeger. "Thermal effects of intrusions". In: *Reviews of geophysics* 2.3 (1964), pp. 443–466. DOI: 10.1029/RG002i003p00443.

[170] Eleanor W Jenkins, Volker John, Alexander Linke, and Leo G Rebholz. "On the parameter choice in grad-div stabilization for the Stokes equations". In: *Advances in Computational Mathematics* 40.2 (2014), pp. 491–516. DOI: 10.1007/s10444-013-9316-1.

[171] Volker John. *Finite Element Methods for Incompressible Flow Problems*. Springer, 2016. ISBN: 978-3-319-45749-9. DOI: 10.1007/978-3-319-45750-5.

[172] Volker John, Alexander Linke, Christian Merdon, Michael Neilan, and Leo G Rebholz. "On the divergence constraint in mixed finite element methods for incompressible flows". In: *SIAM review* 59.3 (2017), pp. 492–544. DOI: 10.1137/15M1047696.

[173] L. Jolivet, P. Davy, and P. Cobbold. "Right-lateral shear along the Northwest Pacific margin and the India-Eurasia collision". In: *Tectonics* 9.6 (1990), pp. 1409–1419. DOI: 10.1029/TC009i006p01409.

[174] A. Jourdon and D. A. May. "An efficient parallel method to compute lithostatic pressure in thermo-mechanical geodynamic models". In: *Solid Earth* 13 (2022), pp. 1107–1125. DOI: 10.5194/se-13-1107-2022.

[175] L.M. Kachanov. *Fundamentals of the Theory of Plasticity*. Dover Publications, Inc., 2004. ISBN: 0-486-43583-0.

[176] Elias Karabelas, Gundolf Haase, Gernot Plank, and Christoph M Augustin. "Versatile stabilized finite element formulations for nearly and fully incompressible solid mechanics". In: *Computational mechanics* 65.1 (2020), pp. 193–215. DOI: 10.1007/s00466-019-01760-w.

[177] Tomoo Katsura, Akira Yoneda, Daisuke Yamazaki, Takashi Yoshino, and Eiji Ito. "Adiabatic temperature profile in the mantle". In: *Phys. Earth. Planet. Inter.* 183.1-2 (2010), pp. 212–218. DOI: 10.1016/j.pepi.2010.07.001.

[178] B.J.P. Kaus. "Factors that control the angle of shear bands in geodynamic numerical models of brittle deformation". In: *Tectonophysics* 484 (2010), pp. 36–47. DOI: 10.1016/j.tecto.2009.08.042.

[179] B.J.P. Kaus and T.W. Becker. "Effects of elasticity on the Rayleigh-Taylor instability: implications for large-scale geodynamics". In: *Geophy. J. Int.* 168.843–862 (2007). DOI: 10.1111/j.1365-246X.2006.03201.x.

[180] B.J.P. Kaus, H. Mühlhaus, and D.A. May. "A stabilization algorithm for geodynamic numerical simulations with a free surface". In: *Phys. Earth. Planet. Inter.* 181 (2010), pp. 12–20. DOI: 10.1016/j.pepi.2010.04.007.

[181] M. Kawaguti. "Numerical solution of the Navier-Stokes equations for the flow in a two-dimensional cavity". In: *Journal of the Physical Society of Japan* 16.12 (1961), pp. 2307–2315.

[182] P.E. van Keken, I. Wada, N. Sime, and G.A. Abers. "Thermal structure of the forearc in subduction zones: a comparison of methodologies". In: *Geochem. Geophys. Geosyst.* 20 (2019), pp. 3268–3288. DOI: 10.1029/2019GC008334.

[183] T. Keller, D.A. May, and B.J.P. Kaus. "Numerical modelling of magma dynamics coupled to tectonic deformation of lithosphere and crust". In: *Geophy. J. Int.* 195.3 (2013), pp. 1406–1442. DOI: 10.1093/gji/ggt306.

[184] L. H. Kellogg and S. D. King. "The effect of temperature dependent viscosity on the structure of new plumes in the mantle: Results of a finite element model in a spherical, axisymmetric shell". In: *Earth Planet. Sci. Lett.* 148.1-2 (1997), pp. 13–26. DOI: 10.1016/S0012-821X(97)00025-3.

[185] LH Kellogg and DL Turcotte. "Mixing and the distribution of heterogeneities in a chaotically convecting mantle". In: *J. Geophys. Res.: Solid Earth* 95.B1 (1990), pp. 421–432. DOI: 10.1029/JB095iB01p00421.

[186] Louise H Kellogg. "Chaotic mixing in the Earth's mantle". In: *Advances in geophysics*. Vol. 34. 1993, pp. 1–33. DOI: 10.1016/S0065-2687(08)60433-1.

[187] Amir Khan et al. "Upper mantle structure of Mars from InSight seismic data". In: *Science* 373.6553 (2021), pp. 434–438. DOI: 10.1126/science.abf2966.

[188] W.S. Kiefer and B. Hager. "Geoid anomalies and dynamic topography from convection in cylindrical geometry: applications to mantle plumes on Earth and Venus". In: *Geophy. J. Int.* 108 (1992), pp. 198–214. DOI: 10.1111/j.1365-246X.1992.tb00850.x.

[189] Walter S Kiefer. "Melting in the Martian mantle: Shergottite formation and implications for present-day mantle convection on Mars". In: *Meteoritics & Planetary Science* 38.12 (2003), pp. 1815–1832. DOI: 10.1111/j.1945-5100.2003.tb00017.x.

[190] Dae-Hee Kim and Byung-Dal So. "Effects of rheology and mantle temperature structure on edge-driven convection: Implications for partial melting and dynamic topography". In: *Phys. Earth. Planet. Inter.* 303 (2020), p. 106487. DOI: 10.1016/j.pepi.2020.106487.

[191] Madlen Kimmritz and Malte Braack. "Equal-order finite elements for the hydrostatic Stokes problem". In: *Computational Methods in Applied Mathematics Comput. Methods Appl. Math.* 12.3 (2012), pp. 306–329.

[192] S. King et al. "A community benchmark for 2D Cartesian compressible convection in the Earth's mantle". In: *Geophy. J. Int.* 180 (2010), pp. 73–87.

[193] Scott D King. "Geoid and topographic swells over temperature-dependent thermal plumes in spherical-axisymmetric geometry". In: *Geophys. Res. Lett.* 24.23 (1997), pp. 3093–3096. DOI: 10.1029/97GL53154.

[194] Scott D King. "Mantle convection, the asthenosphere, and Earth's thermal history". In: *Geological Society of America Special Papers* 514 (2015), SPE514–07. DOI: 10.1130/2015.2514(07).

[195] S.H. Kirby and A.K. Kronenberg. "Rheology of the lithosphere: Selected topics". In: *Reviews of Geophysics* 25.6 (1987). DOI: 10.1029/RG025i006p01219.

[196] Brigitte Knapmeyer-Endrun et al. "Thickness and structure of the martian crust from InSight seismic data". In: *Science* 373.6553 (2021), pp. 438–443. DOI: 10.1126/science.abf8966.

[197] Nils Kohl, Marcus Mohr, Sebastian Eibl, and Ulrich Rüde. "A massively parallel Eulerian-Lagrangian method for advection-dominated transport in viscous fluids". In: *SIAM J. SCI. COMPUT.* 44.3 (2022), pp. C260–C285. DOI: 10.1137/21M1402510.

[198] Henk Kooi and Christopher Beaumont. "Escarpment evolution on high-elevation rifted margins: Insights derived from a surface processes model that combines diffusion, advection, and reaction". In: *J. Geophys. Res.: Solid Earth* 99.B6 (1994), pp. 12191–12209. DOI: 10.1029/94JB00047.

[199] J.R. Koseff and R.L. Street. "The Lid-Driven Cavity Flow: A Synthesis of Qualitative and Quantitative Observations". In: *J. Fluids Eng* 106 (1984), pp. 390–398.

[200] M. Kronbichler, T. Heister, and W. Bangerth. "High accuracy mantle convection simulation through modern numerical methods". In: *Geophy. J. Int.* 191 (2012), pp. 12–29. DOI: 10.1111/j.1365-246X.2012.05609.x.

[201] E.-J. N. Kuiper, J. H. P. de Bresser, M. R. Drury, J. Eichler, G. M. Pennock, and I. Weikusat. "Using a composite flow law to model deformation in the NEEM deep ice core, Greenland: Part 2 the role of grain size and premelting on ice deformation at high homologous temperature". In: *The Cryosphere Discussions* 2019 (2019), pp. 1–30. DOI: 10.5194/tc-2018-275.

[202] E.-J. N. Kuiper, I. Weikusat, J. H. P. de Bresser, D. Jansen, G. M. Pennock, and M. R. Drury. "Using a composite flow law to model deformation in the NEEM deep ice core, Greenland: Part 1 the role of grain size and grain size distribution on the deformation of Holocene and glacial ice". In: *The Cryosphere Discussions* 2019 (2019), pp. 1–32. DOI: 10.5194/tc-2018-274.

[203] E.N. Kuiper. "Flow in naturally deformed ice: a cryogenic electron microscopy and modelling study of the NEEM ice core". PhD thesis. Utrecht University, 2019.

[204] B Kustowski, G Ekström, and AM Dziewoński. "Anisotropic shear-wave velocity structure of the Earth's mantle: A global model". In: *J. Geophys. Res.: Solid Earth* 113.B6 (2008). DOI: 10.1029/2007JB005169.

[205] B.P. Lamichhane. "A quadrilateral 'mini' finite element for the Stokes problem using a single bubble function". In: *International Journal of Numerical Analysis and Modeling* 14.6 (2017), pp. 869–878.

[206] T.B. Larsen, D.A. Yuen, A.V. Malevsky, and J.L. Smedsmo. "Dynamics of strongly time-dependent convection with non-Newtonian temperature-dependent viscosity". In: *Phys. Earth. Planet. Inter.* 94.1-2 (1996), pp. 75–103. DOI: 10.1016/0031-9201(95)03082-4.

[207] W. Layton. "Weak imposition of 'no-slip' conditions in finite element methods". In: *Computers and Mathematics with Applications* 38 (1999), pp. 129–142. DOI: 10.1016/S0898-1221(99)00220-5.

[208] R. Lee, P. Gresho, and R. Sani. "Smoothing techniques for certain primitive variable solutions of the Navier-Stokes equations". In: *Int. J. Num. Meth. Eng.* 14 (1979), pp. 1785–1804.

[209] A. Lenardic and W. M. Kaula. "A numerical treatment of geodynamic viscous flow problems involving the advection of material interfaces". In: *J. Geophys. Res.: Solid Earth* 98.B5 (1993), pp. 8243–8260. DOI: 10.1029/92JB02858.

[210] W. Leng and S. Zhong. "Implementation and application of adaptive mesh refinement for thermo-chemical mantle convection studies". In: *Geochem. Geophys. Geosyst.* 12.4 (2011). DOI: 10.1029/2010GC003425.

[211] W. Leng and S. Zhong. "Viscous heating, adiabatic heating and energetic consistency in compressible mantle convection". In: *Geophy. J. Int.* 173 (2008), pp. 693–702. DOI: 10.1111/j.1365-246X.2008.03745.x.

[212] Wei Leng and Michael Gurnis. "Shape of thermal plumes in a compressible mantle with depth-dependent viscosity". In: *Geophys. Res. Lett.* 39.5 (2012). DOI: 10.1029/2012GL050959.

[213] Wei Leng, Lili Ju, Yan Xie, Tao Cui, and Max Gunzburger. "Finite element three-dimensional Stokes ice sheet dynamics model with enhanced local mass conservation". In: *J. Comp. Phys.* 274 (2014), pp. 299–311.

[214] Ben Q Li. *Discontinuous finite elements in fluid dynamics and heat transfer*. Springer Science & Business Media, 2006.

[215] J. Li, Y. He, and Z. Chen. "Performance of several stabilized finite element methods for the Stokes equations based on the lowest equal-order pairs". In: *Computing* 86 (2009), pp. 37–51. DOI: 10.1007/s00607-009-0064-5.

[216] Qingsong Li and Walter S Kiefer. "Mantle convection and magma production on present-day Mars: Effects of temperature-dependent rheology". In: *Geophys. Res. Lett.* 34.16 (2007). DOI: 10.1029/2007GL030544.

[217] E. Rune Lindgren. "The motion of a sphere in an incompressible viscous fluid at Reynolds numbers considerably less than one". In: *Physica Scripta* 60 (1999), pp. 97–110.

[218] Alexander Linke and Leo G Rebholz. "Pressure-induced locking in mixed methods for time-dependent (Navier–) Stokes equations". In: *J. Comp. Phys.* 388 (2019), pp. 350–356. DOI: 10.1016/j.jcp.2019.03.010.

[219] X. Liu and S. Zhong. "Analyses of marginal stability, heat transfer and boundary layer properties for thermal convection in a compressible fluid with infinite Prandtl number". In: *Geophy. J. Int.* 194 (2013), pp. 125–144. DOI: 10.1093/gji/ggt117.

[220] Aurelie Louis–Napoleon, Muriel Gerbault, Thomas Bonometti, Cedric Thieulot, Roland Martin, and Olivier Vanderhaeghe. "3D numerical modeling of crustal polydiapirs with Volume-Of-Fluid methods". In: *Geophy. J. Int.* 222 (2020), pp. 474–506. DOI: 10.1093/gji/ggaa141.

[221] Augustus Edward Hough Love. "IX. The stress produced in a semi-infinite solid by pressure on part of the boundary". In: *Philosophical Transactions of the Royal Society of London. Series A* 228.659-669 (1929), pp. 377–420. DOI: 10.1098/rsta.1929.0009.

[222] Philip K Maini, Thomas E Woolley, Ruth E Baker, Eamonn A Gaffney, and S Seirin Lee. "Turing's model for biological pattern formation and the robustness problem". In: *Interface focus* 2.4 (2012), pp. 487–496. DOI: 10.1098/rsfs.2011.0113.

[223] L.E. Malvern. *Introduction to the mechanics of a continuous medium*. Prentice-Hall, Inc., 1969.

[224] Takeshi Matsumoto and Yoshibumi Tomoda. "Numerical simulation of the initiation of subduction at the fracture zone". In: *Journal of Physics of the Earth* 31.3 (1983), pp. 183–194. DOI: 10.4294/jpe1952.31.183.

[225] Ctirad Matyska, David A Yuen, Renata M Wentzcovitch, and Hana Čížková. "The impact of variability in the rheological activation parameters on lower-mantle viscosity stratification and its dynamics". In: *Phys. Earth. Planet. Inter.* 188.1-2 (2011), pp. 1–8. DOI: 10.1016/j.pepi.2011.05.012.

[226] D.A. May. "Volume reconstruction of point cloud data sets derived from computational geodynamic simulations". In: *Geochem. Geophys. Geosyst.* 13.5 (2012), Q05019. DOI: 10.1029/2012GC004170.

[227] D.A. May, J. Brown, and L. Le Pourhiet. "A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous Stokes flow". In: *Computer Methods in Applied Mechanics and Engineering* 290 (2015), pp. 496–523. DOI: 10.1016/j.cma.2015.03.014.

[228] Dave A May, Jed Brown, and Laetitia Le Pourhiet. "pTatin3D: High-performance methods for long-term lithospheric dynamics". In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*. IEEE Press. 2014, pp. 274–284. DOI: 10.1109/SC.2014.28.

[229] D. McKenzie. "Some remarks on the development of sedimentary basins". In: *Earth Planet. Sci. Lett.* 40 (1978), pp. 25–32. DOI: 10.1016/0012-821X(78)90071-7.

[230] Dan McKenzie. "Finite deformation during fluid flow". In: *Geophy. J. Int.* 58.3 (1979), pp. 689–715. DOI: 10.1111/j.1365-246X.1979.tb04803.x.

[231] Dan McKenzie and James Jackson. "The relationship between strain rates, crustal thickening, palaeomagnetism, finite strain and fault movements within a deforming zone". In: *Earth Planet. Sci. Lett.* 65.1 (1983), pp. 182–202. DOI: 10.1016/0012-821X(83)90198-X.

[232] Dan McKenzie, James Jackson, and Keith Priestley. "Thermal structure of oceanic and continental lithosphere". In: *Earth Planet. Sci. Lett.* 233.3-4 (2005), pp. 337–349. DOI: 10.1016/j.epsl.2005.02.005.

[233] Dan P McKenzie, Jean M Roberts, and Nigel O Weiss. "Convection in the Earth's mantle: towards a numerical simulation". In: *Journal of Fluid Mechanics* 62.3 (1974), pp. 465–538. DOI: 10.1017/S0022112074000784.

[234] A.K. McNamara and S. Zhong. "Degree-one mantle convection: Dependence on internal heating and temperature-dependent rheology". In: *Geophys. Res. Lett.* 32.L01301 (2005).

[235] AJF Metherell and TJ Quinn. "The gravitational field of a 111 tetrahedron". In: *Metrologia* 22.2 (1986), p. 87. DOI: 10.1088/0026-1394/22/2/003.

[236] P. Molnar and P. Tapponnier. "Relation of the tectonics of eastern China to the India-Eurasia collision: Application of the slip-line field theory to large-scale continental tectonics". In: *Geology* 5 (1977), pp. 212–216.

[237] Jason P Morgan, Jorge M Taramón, and Jörg Hasenclever. "Shape-preserving finite elements in cylindrical and spherical geometries: The double Jacobian approach". In: *Int. J. Num. Meth. Fluids* 92.6 (2020), pp. 635–668. DOI: 10.1002/fld.4799.

[238] W.J. Morgan. "Gravity Anomalies and Convection Currents. 1: A Sphere and Cylinder Sinking beneath the Surface of a Viscous Fluid". In: *J. Geophys. Res.: Solid Earth* 70.24 (1965). DOI: 10.1029/JZ070i024p06175.

[239] P. Moulik and G. Ekström. "The relationships between large-scale variations in shear velocity, density, and compressional velocity in the Earth's mantle". In: *J. Geophys. Res.: Solid Earth* 121 (2016). DOI: 10.1002/2015JB012679.

[240] Thomas Nagler, Helmut Rott, Markus Hetzenecker, Jan Wuite, and Pierre Potin. "The Sentinel-1 mission: New opportunities for ice sheet observations". In: *Remote Sensing* 7.7 (2015), pp. 9371–9389. DOI: 10.3390/rs70709371.

[241] SM Nakiboglu and Kurt Lambeck. "A study of the Earth's response to surface loading with application to Lake Bonneville". In: *Geophy. J. Int.* 70.3 (1982), pp. 577–620. DOI: 10.1111/j.1365-246X.1982.tb05975.x.

[242] S. Norburn and D. Silvester. "Fourier analysis of stabilized Q1-Q1 mixed finite element approximation". In: *SIAM J. Numer. Anal.* 39 (2001), pp. 817–833. DOI: 10.1137/S0036142999362274.

[243] Clifford U Nwoji, Hyginus N Onah, Benjamin O Mama, and Charles C Ike. "Solution of elastic half space problem using Boussinesq displacement potential functions". In: *Asian Journal of Applied Sciences* 5.5 (2017). DOI: xxxx.

[244] C. O'Neill, L. Moresi, D. Müller, R. Albert, and F. Dufour. "Ellipsis 3D: a particle-in-cell finite element hybrid code for modelling mantle convection and lithospheric deformation". In: *Computers and Geosciences* 32 (2006), pp. 1769–1779. DOI: 10.1016/j.cageo.2006.04.006.

[245] Masayuki Obayashi et al. "Finite frequency whole mantle P wave tomography: Improvement of subducted slab images". In: *Geophys. Res. Lett.* 40.21 (2013), pp. 5652–5657.

[246] Maxim Olshanskii, Gert Lube, Timo Heister, and Johannes Löwe. "Grad–div stabilization and subgrid pressure models for the incompressible Navier–Stokes equations". In: *Computer Methods in Applied Mechanics and Engineering* 198.49-52 (2009), pp. 3975–3988. DOI: 10.1016/j.cma.2009.09.005.

[247] F. Pan and A. Acrivos. "Steady flows in rectangular cavities". In: *J. Fluid Mech.* 28.4 (1967), pp. 643–655. DOI: 10.1017/S002211206700237X.

[248] Barry Parsons and John G Sclater. "An analysis of the variation of ocean floor bathymetry and heat flow with age". In: *J. Geophys. Res.: Solid Earth* 82.5 (1977), pp. 803–827. DOI: 10.1029/JB082i005p00803.

[249] L. Pasol, M. Chaoui, S. Yahiaoui, and F. Feuillebois. "Analytical solutions for a spherical particle near a wall in axisymmetrical polynomial creeping flows". In: *Physics of Fluids* 17.7 (2005), p. 073602. DOI: 10.1063/1.1955272.

[250] G. Peltzer and P. Tapponnier. "Formation and evolution of strike-slip faults, rifts, and basins during the india-asia collision: an experimental approach". In: *J. Geophys. Res.: Solid Earth* 93.B12 (1988), pp. 15085–15177. DOI: 10.1029/JB093iB12p15085.

[251] A. Pinelli and A. Vacca. "Chebyshev collocation method and multidomain decomposition for the incompressible Navier-Stokes equations". In: *International Journal for numerical methods in fluids* 18 (1994), pp. 781–799. DOI: 10.1002/fld.1650180806.

[252] Adina E Pusok, Boris JP Kaus, and Anton A Popov. "On the Quality of Velocity Interpolation Schemes for Marker-in-Cell Method and Staggered Grids". In: *Pure Appl. Geophys.* 174 (2017), pp. 1071–1089. DOI: 10.1007/s00024-016-1431-8.

[253] P. Puster, B. H. Hager, and T. H. Jordan. "Mantle convection experiments with evolving plates". In: *Geophys. Res. Lett.* 22.16 (1995), pp. 2223–2226. DOI: 10.1029/95GL01998.

[254] Matthieu Quinquis. "A numerical study of subduction zone dynamics using linear viscous to thermo-mechanical model setups including (de)hydration processes". PhD thesis. Charles University in Prague, 2014.

[255] T. Rabczuk, P.M.A. Areias, and T. Belytschko. "A simplified mesh-free method for shear bands with cohesive surfaces ". In: *Int. J. Num. Meth. Eng.* 69 (2007), pp. 993–1021. DOI: 10.1002/nme.1797.

[256] A. Ramachandran. "Parallel adaptive finite element simulation using distribued quad-tree/octree forest". MA thesis. Institute for Structural Mechanics, Ruhr University Bochum, 2016.

[257] Hans Ramberg. "Instability of layered systems in the field of gravity". In: *Phys. Earth. Planet. Inter.* 1 (1968), pp. 427–447. DOI: 10.1016/0031-9201(68)90014-9.

[258] Ekkehard Ramm et al. *Error-controlled adaptive finite elements in solid mechanics*. 2003.

[259] Hannah L Redmond and Scott D King. "A numerical study of a mantle plume beneath the Tharsis Rise: Reconciling dynamic uplift and lithospheric support models". In: *J. Geophys. Res.: Planets* 109.E9 (2004). DOI: 10.1029/2003JE002228.

[260] J. Revenaugh and B. Parsons. "Dynamic topography and gravity anomalies for fluid layers whose viscosity varies exponentially with depth". In: *Geophysical Journal of the Royal Astronomical Society* 90.2 (1987), pp. 349–368. DOI: 10.1111/j.1365-246X.1987.tb00731.x.

[261] J.G. Rice and R.J. Schnipke. "An equal-order velocity-pressure formulation that does not exhibit spurious pressure modes". In: *Computer Methods in Applied Mechanics and Engineering* 58 (1986), pp. 135–149.

[262] FD Richards, MJ Hoggard, LR Cowton, and NJ White. "Reassessing the thermal structure of oceanic lithosphere with revised global inventories of basement depths and heat flow measurements". In: *J. Geophys. Res.: Solid Earth* 123.10 (2018), pp. 9136–9161. DOI: 10.1029/2018JB015998.

[263] M.A. Richards, H.-P. Bunge, Y. Ricard, and J.R. Baumgardner. "Polar wandering in mantle convection models". In: *Geophys. Res. Lett.* 26.12 (1999), pp. 1777–1780. DOI: 10.1029/1999GL900331.

[264] J. Ritsema, A. Deuss, H.J. van Heijst, and J.H. Woodhouse. "S40rts: a degree-40 shear- velocity model for the mantle from new rayleigh wave dispersion, teleseismic traveltime and normal-mode splitting function measurements". In: *Geophy. J. Int.* 184 (2011), pp. 1223–1236.

[265] Jeroen Ritsema, Hendrik Jan van Heijst, and John H Woodhouse. "Complex shear wave velocity structure imaged beneath Africa and Iceland". In: *Science* 286.5446 (1999), pp. 1925–1928. DOI: 10.1126/science.286.5446.1925.

[266] Jeroen Ritsema, Hendrik Jan Van Heijst, and John H Woodhouse. "Global transition zone tomography". In: *J. Geophys. Res.: Solid Earth* 109.B2 (2004). DOI: 10.1029/2003JB002610.

[267] J. H. Roberts and S. Zhong. "Degree-1 convection in the Martian mantle and the origin of the hemispheric dichotomy". In: *J. Geophys. Res.: Solid Earth* 111.E6 (2006). DOI: 10.1029/2005JE002668.

[268] B. Root, J. Sebera, W. Szwillus, C. Thieulot, Z. Martinec, and J. Fullea. "Benchmark forward gravity schemes: the gravity field of a realistic lithosphere model WINTERC-G". In: *Solid Earth* 13 (2022), pp. 849–873. DOI: 10.5194/se-13-849-2022.

[269] I. Rose, B. Buffet, and T. Heister. "Stability and accuracy of free surface time integration in viscous flows". In: *Phys. Earth. Planet. Inter.* 262 (2017), pp. 90–100.

[270] L Royden and CE Keen. "Rifting process and thermal evolution of the continental margin of eastern Canada determined from subsidence curves". In: *Earth Planet. Sci. Lett.* 51.2 (1980), pp. 343–361. DOI: 10.1016/0012-821X(80)90216-2.

[271] Johann Rudi, Yu-hsuan Shih, and Georg Stadler. "Advanced Newton methods for geodynamical models of Stokes flow with viscoplastic rheologies". In: *Geochem. Geophys. Geosyst.* 21.9 (2020), e2020GC009059. DOI: 10.1029/2020GC009059.

[272] M.H. Sadd. *Elasticity. Theory, applications and numerics (Third edition)*. Elsevier, 2014.

[273] Malcolm Sambridge and Ólafur Gudmundsson. "Tomographic systems of equations with irregular cells". In: *J. Geophys. Res.: Solid Earth* 103.B1 (1998), pp. 773–781. DOI: 10.1029/97JB02602.

[274] H. Samuel. "Time domain parallelization for computational geodynamics". In: *Geochem. Geophys. Geosyst.* 13.1 (2012), Q01003. DOI: 10.1029/2011GC003905.

[275] H. Samuel and M. Evonuk. "Modeling advection in geophysical flows with particle level sets". In: *Geochem. Geophys. Geosyst.* 11.8 (2010). DOI: 10.1029/2010GC003081.

[276] Henri Samuel. "A deformable particle-in-cell method for advective transport in geodynamic modelling". In: *Geophy. J. Int.* 214.3 (2018), pp. 1744–1773.

[277] R.L. Sani, P.M. Gresho, R.L. Lee, and D.F. Griffiths. "The cause and cure (?) of the spurious pressures generated by certain FEM solutions of the incompressible Navier-Stokes equations: part 1". In: *Int. J. Num. Meth. Fluids* 1 (1981), pp. 17–43. DOI: 10.1002/fld.1650010104.

[278] R.L. Sani, P.M. Gresho, R.L. Lee, D.F. Griffiths, and M. Engelman. "The cause and cure (?) of the spurious pressures generated by certain FEM solutions of the incompressible Navier-Stokes equations: part 2". In: *Int. J. Num. Meth. Fluids* 1 (1981), pp. 171–204. DOI: 10.1002/fld.1650010206.

[279] Atuo Sato and Erik G Thompson. "Finite element models for creeping convection". In: *J. Comp. Phys.* 22.2 (1976), pp. 229–244. DOI: 10.1016/0021-9991(76)90077-2.

[280] J.C. Savage and R.O. Burford. "Geodetic Determination of Relative Plate Motion in Central California". In: *J. Geophys. Res.: Solid Earth* 78.5 (1973), pp. 832–845.

[281] S.M. Schmalholz. "A simple analytical solution for slab detachment". In: *Earth Planet. Sci. Lett.* 304 (2011), pp. 45–54. DOI: 10.1016/j.epsl.2011.01.011.

[282] H. Schmeling et al. "A benchmark comparison of spontaneous subduction models - Towards a free surface". In: *Phys. Earth. Planet. Inter.* 171 (2008), pp. 198–223. DOI: 10.1016/j.pepi.2008.06.028.

[283] D.W. Schmid and Y.Y. Podlachikov. "Analytical solutions for deformable elliptical inclusions in general shear". In: *Geophy. J. Int.* 155 (2003), pp. 269–288. DOI: 10.1046/j.1365-246X.2003.02042.x.

[284] J Schnakenberg. "Simple chemical reaction systems with limit cycle behaviour". In: *Journal of theoretical biology* 81.3 (1979), pp. 389–400. DOI: 10.1016/0022-5193(79)90042-0.

[285] G.E. Schneider, G.D. Raithby, and M.M. Yovanovich. "Finite-element solution procedures for solving the incompressible Navier-Stokes equations using equal order variable interpolation". In: *Numerical Heat Transfer* 1 (1978), pp. 433–451.

[286] Teseo Schneider, Yixin Hu, Xifeng Gao, Jérémie Dumas, Denis Zorin, and Daniele Panozzo. "A Large-Scale Comparison of Tetrahedral and Hexahedral Elements for Solving Elliptic PDEs with the Finite Element Method". In: *ACM Transactions on Graphics (TOG)* 41.3 (2022), pp. 1–14. DOI: 10.1145/3508372.

[287] G. Schubert, D.L. Turcotte, and P. Olson. *Mantle Convection in the Earth and Planets*. Cambridge University Press, 2001. ISBN: 0-521-70000-0. DOI: 10.1017/CBO9780511612879.

[288] Gerald Schubert, D. Bercovici, and G.A. Glatzmaier. "Mantle dynamics in Mars and Venus: Influence of an immobile lithosphere on three-dimensional mantle convection". In: *J. Geophys. Res.: Solid Earth* 95.B9 (1990), pp. 14105–14129.

[289] Melchior Schuh-Senlis, Cedric Thieulot, Paul Cupillard, and Guillaume Caumon. "Towards the application of Stokes flow equations to structural restoration simulations". In: *Solid Earth* 11 (2020), pp. 1909–1930. DOI: 10.5194/se-11-1909-2020.

[290] APS Selvadurai. "On Boussinesq's problem". In: *International Journal of Engineering Science* 39.3 (2001), pp. 317–322. DOI: 10.1016/S0020-7225(00)00043-4.

[291] P.N. Shankar and M.D. Deshpande. "Fluid mechanics in the driven cavity". In: *Annu. Rev. Fluid Mech.* 32 (2000), pp. 93–136.

[292] Simon G Sheldon, Jørgen P Steffensen, Steffen B Hansen, Trevor J Popp, and Sigfús J Johnsen. "The investigation and experience of using ESTISOL 240 and COASOL for ice-core drilling". In: *Annals of Glaciology* 55.68 (2014), pp. 219–232.

[293] Jonathan Richard Shewchuk. "Reprint of: Delaunay refinement algorithms for triangular mesh generation". In: *Computational Geometry* 47.7 (2014), pp. 741–778. DOI: 10.1016/j.comgeo.2014.02.005.

[294] Nathan Sime and Cian R Wilson. "Automatic weak imposition of free slip boundary conditions via Nitsche's method: application to nonlinear problems in geodynamics". In: *arXiv preprint arXiv:2001.10639* (2020).

[295] N.A. Simmons, A.M. Forte, L. Boschi, and S.P. Grand. "GyPSuM: A joint tomographic model of mantle density and seismic wave speeds". In: *J. Geophys. Res.: Solid Earth* 115.B12310 (2010). DOI: 10.1029/2010JB007631.

[296] Guy Simpson. "A dynamic model to investigate coupling between erosion, deposition, and three-dimensional (thin-plate) deformation". In: *J. Geophys. Res.: Earth Surface* 109.F2 (2004). DOI: 10.1029/2003JF000078.

[297] Guy Simpson. "Dynamic interactions between erosion, deposition, and three-dimensional deformation in compressional fold belt settings". In: *J. Geophys. Res.: Earth Surface* 109.F3 (2004). DOI: 10.1029/2003JF000111.

[298] Guy Simpson. *Practical Finite Element Modeling in Earth Science Using Matlab*. Wiley-Blackwell, 2017. ISBN: 978-1-119-24862-0.

[299] Guy Simpson and Sébastien Castelltort. "Coupled model of surface water flow, sediment transport and morphological evolution". In: *Computers and Geosciences* 32.10 (2006), pp. 1600–1614. DOI: 10.1016/j.cageo.2006.02.020.

[300] Guy Simpson and Fritz Schlunegger. "Topographic evolution and morphology of surfaces evolving in response to coupled fluvial and hillslope sediment transport". In: *J. Geophys. Res.: Solid Earth* 108.B6 (2003). DOI: 10.1029/2002JB002162.

[301] Gaia Siravo et al. "Slab flattening and the rise of the Eastern Cordillera, Colombia". In: *Earth Planet. Sci. Lett.* 512 (2019), pp. 100–110. DOI: 10.1016/j.epsl.2019.02.002.

[302] M. Spiegelman, D.A. May, and C. Wilson. "On the solvability of incompressible Stokes with viscoplastic rheologies in geodynamics". In: *Geochem. Geophys. Geosyst.* 17 (2016), pp. 2213–2238. DOI: 10.1002/2015GC006228.

[303] G. Stadler, M. Gurnis, C. Burstedde, L.C. Wilcox, L. Alisic, and O. Ghattas. "The dynamics of plate tectonics and mantle flow: from local to global scales". In: *Science* 329 (2010), pp. 1033–1038. DOI: 10.1126/science.1191223.

[304] Simon C. Stähler et al. "Seismic detection of the martian core". In: *Science* 373.6553 (2021), pp. 443–448. DOI: 10.1126/science.abi7730.

[305] C. Stein, J. Lowman, and U. Hansen. "A comparison of mantle convection models featuring plates". In: *Geochem. Geophys. Geosyst.* 15 (2014), pp. 2689–2698. DOI: 10.1002/2013GC005211.

[306] B Steinberger and R Holme. "Mantle flow models with core-mantle boundary constraints and chemical heterogeneities in the lowermost mantle". In: *J. Geophys. Res.: Solid Earth* 113.B5 (2008).

[307] B. Steinberger and A.R. Calderwood. "Models of large-scale viscous flow in the Earth's mantle with constraints from mineral physics and surface observations". In: *Geophy. J. Int.* 167 (2006), pp. 1461–1481. DOI: 10.1111/j.1365-246X.2006.03131.x.

[308] Bernhard Steinberger and Thorsten W Becker. "A comparison of lithospheric thickness models". In: *Tectonophysics* 746 (2018), pp. 325–338. DOI: 10.1016/j.tecto.2016.08.001.

[309] Bernhard Steinberger, Stephanie C Werner, and Trond H Torsvik. "Deep versus shallow origin of gravity anomalies, topography and volcanism on Earth, Venus and Mars". In: *Icarus* 207.2 (2010), pp. 564–577. DOI: 10.1016/j.icarus.2009.12.025.

[310] Cheryl A Stewart and Donald L Turcotte. "The route to chaos in thermal convection at infinite Prandtl number: 1. Some trajectories and bifurcations". In: *J. Geophys. Res.: Solid Earth* 94.B10 (1989), pp. 13707–13717. DOI: 10.1029/JB094iB10p13707.

[311] Margaret Stimson and George Barker Jeffery. "The motion of two spheres in a viscous fluid". In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 111.757 (1926), pp. 110–116. DOI: 10.1098/rspa.1926.0053.

[312] K Stüwe and M Sandiford. "Contribution of deviatoric stresses to metamorphic P-T paths: an example appropriate to low-P, high-T metamorphism". In: *Journal of Metamorphic Geology* 12.4 (1994), pp. 445–454. DOI: 10.1111/j.1525-1314.1994.tb00034.x.

[313] Kurt Stüwe. "Heat sources of Cretaceous metamorphism in the Eastern Alps—a discussion". In: *Tectonophysics* 287.1-4 (1998), pp. 251–269. DOI: 10.1016/S0040-1951(98)80072-3.

[314] Kurt Stüwe. "Thermal buffering effects at the solidus. Implications for the equilibration of partially melted metamorphic rocks". In: *Tectonophysics* 248.1-2 (1995), pp. 39–51. DOI: 10.1016/0040-1951(94)00282-E.

[315] J. Suckale, J.-C. Nave, and B.H. Hager. "It takes three to tango: 1. Simulating buoyancy-driven flow in the presence of large viscosity contrasts". In: *J. Geophys. Res.: Solid Earth* 115.B07409 (2010). DOI: 10.1029/2009JB006916.

[316] E.M. Syracuse, P.E. van Keken, and G.A. Abers. "The global range of subduction zone thermal models". In: *Phys. Earth. Planet. Inter.* 183 (2010), pp. 73–90.

[317] P. Tackley. "Three-dimensional models of mantle convection: Influence of phase transitions and temperature-dependent viscosity". PhD thesis. California Institute of Technology, 1994, p290.

[318] E. Tan and M. Gurnis. "Compressible thermochemical convection and application to lower mantle structures". In: *J. Geophys. Res.: Solid Earth* 112.B06304 (2007).

[319] Paul Tapponnier and Peter Molnar. "Slip-line field theory and large-scale continental tectonics". In: *Nature* 264 (Nov. 1976), pp. 319–324. DOI: 10.1038/264319a0.

[320] V Tenczer, K Stüwe, and TD Barr. "Pressure anomalies around cylindrical objects in simple shear". In: *Journal of Structural Geology* 23.5 (2001), pp. 777–788.

[321] Andy R Terrel, L Ridgway Scott, Matthew Gregg Knepley, Robert C Kirby, and Garth N Wells. "Finite elements for incompressible fluids". In: *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Ed. by Anders Logg, Kent-Andre Mardal, and Garth Wells. Vol. 84. 2012, pp. 385–397. DOI: 10.1007/978-3-642-23099-8_20.

[322] Magdala Tesauro, Pascal Audet, Mikhail K Kaban, Roland Bürgmann, and Sierd Cloetingh. "The effective elastic thickness of the continental lithosphere: Comparison between rheological and inverse approaches". In: *Geochem. Geophys. Geosyst.* 13.9 (2012). DOI: 10.1029/2012GC004162.

[323] Tayfun E Tezduyar, Sanjay Mittal, SE Ray, and R Shih. "Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements". In: *Computer Methods in Applied Mechanics and Engineering* 95.2 (1992), pp. 221–242.

[324] M. Thielmann, D.A. May, and B.J.P. Kaus. "Discretization errors in the Hybrid Finite Element Particle-In-Cell Method". In: *Pure Appl. Geophys.* 171.9 (2014), pp. 2164–2184. DOI: 10.1007/s00024-014-0808-9.

[325] C. Thieulot. "ELEFANT: a user-friendly multipurpose geodynamics code". In: *Solid Earth Discussions* 6 (2014), pp. 1949–2096. DOI: 10.5194/sed-6-1949-2014.

[326] C. Thieulot. "FANTOM: two- and three-dimensional numerical modelling of creeping flows for the solution of geological problems". In: *Phys. Earth. Planet. Inter.* 188.1 (2011), pp. 47–68. DOI: 10.1016/j.pepi.2011.06.011.

[327] C. Thieulot. "GHOST: Geoscientific Hollow Sphere Tesselation". In: *Solid Earth* 9.1–9 (2018). DOI: 10.5194/se-9-1169-2018.

[328] C. Thieulot and W. Bangerth. "On the choice of finite element for applications in geodynamics". In: *Solid Earth* 13 (2022), pp. 229–249. DOI: 10.5194/se-13-229-2022.

[329] C. Thieulot and W. Bangerth. "On the choice of finite element for applications in geodynamics. Part II: A comparison of simplex and hypercube elements". In: *Solid Earth* (2024). DOI: Subm..

[330] C. Thieulot, P. Fullsack, and J. Braun. "Adaptive octree-based finite element analysis of two- and three-dimensional indentation problems". In: *J. Geophys. Res.: Solid Earth* 113 (2008), B12207. DOI: 10.1029/2008JB005591.

[331] Basil Tikoff and Haakon Fossen. "Simultaneous pure and simple shear: the unifying deformation matrix". In: *Tectonophysics* 217.3-4 (1993), pp. 267–283. DOI: 10.1016/0040-1951(93)90010-H.

[332] N. Tosi et al. “A community benchmark for viscoplastic thermal convection in a 2-D square box”. In: *Geochem. Geophys. Geosyst.* 16.7 (2015), pp. 2175–2196. DOI: 10.1002/2015GC005807.

[333] R.A. Trompert and U. Hansen. “Mantle convection simulations with rheologies that generate plate-like behaviour ”. In: *Nature* 395 (1998), pp. 686–689. DOI: 10.1038/27185.

[334] R.A. Trompert and U. Hansen. “On the Rayleigh number dependence of convection with a strongly temperature-dependent viscosity”. In: *Physics of Fluids* 10.2 (1998), pp. 351–360.

[335] D.L. Turcotte and G. Schubert. *Geodynamics, 2nd edition.* Cambridge, 2012. ISBN: 9780511807442. DOI: 10.1017/CBO9780511807442.

[336] J.M. Urquiza, A. Garon, and M.-I. Farinas. “Weak imposition of the slip boundary condition on curved boundaries for Stokes flow”. In: *J. Comp. Phys.* 256 (2014), pp. 748–767. DOI: 10.1016/j.jcp.2013.08.045.

[337] Jean d’Amour Uwiduhaye, Hideki Mizunaga, and Hakim Saibi. “A case history: 3-D gravity modeling using hexahedral element in Kinigi geothermal field, Rwanda”. In: *Arabian Journal of Geosciences* 12 (2019), pp. 1–13. DOI: 10.1007/s12517-019-4249-8.

[338] P.E. van Keken, S.D. King, H. Schmeling, U.R. Christensen, D. Neumeister, and M.-P. Doin. “A comparison of methods for the modeling of thermochemical convection”. In: *J. Geophys. Res.: Solid Earth* 102.B10 (1997), pp. 22, 477–22, 495.

[339] P.E. van Keken et al. “A community benchmark for subduction zone modelling”. In: *Phys. Earth. Planet. Inter.* 171 (2008), pp. 187–197. DOI: 10.1016/j.pepi.2008.04.015.

[340] PE Van Keken, S Kita, and J Nakajima. “Thermal structure and intermediate-depth seismicity in the Tohoku-Hokkaido subduction zones”. In: *Solid Earth* 3.2 (2012), pp. 355–364. DOI: 10.5194/se-3-355-2012.

[341] Yannis Voet. “On the fast assemblage of finite element matrices with application to nonlinear heat transfer problems”. In: *Applied Mathematics and Computation* 436 (2023), p. 127516. DOI: 10.1016/j.amc.2022.127516.

[342] R.R.B. von Frese, W.J. Hinze, L.W. Braile, and A.J. Luca. “Spherical-Earth Gravity and Magnetic Anomaly Modeling by Gauss-Legendre Quadrature Integration”. In: *J. Geophys.* 49 (1981), pp. 234–242.

[343] L. Vynnytska, M.E. Rognes, and S.R. Clark. “Benchmarking FEniCS for mantle convection simulations”. In: *Computers and Geosciences* 50 (2013), pp. 95–105. DOI: 10.1016/j.cageo.2012.05.012.

[344] H. Wang, R. Agrusta, and J. van Hunen. “Advantages of a conservative velocity interpolation (CVI) scheme for particle-in-cell methods with application in geodynamic modeling”. In: *Geochem. Geophys. Geosyst.* 16 (2015). DOI: 10.1002/2015GC005824.

[345] Zhijie Wang and Radoslaw L Michalowski. “Contact fatigue in silica sand - Observations and modeling”. In: *Geomechanics for Energy and the Environment* 4 (2015), pp. 88–99. DOI: 10.1016/j.gete.2015.07.003.

[346] C.J. Warren, C. Beaumont, and R.A. Jamieson. “Formation and exhumation of ultra-high-pressure rocks during continental collision: Role of detachment in the subduction channel”. In: *Geochem. Geophys. Geosyst.* 9 (2008). DOI: 10.1029/2007GC001839.

[347] C.J. Warren, C. Beaumont, and R.A. Jamieson. “Modelling tectonic styles and ultra-high pressure (UHP) rock exhumation during the transition from oceanic subduction to continental collision”. In: *Earth Planet. Sci. Lett.* 267 (2008), pp. 129–145.

[348] Clare J Warren, Christopher Beaumont, and Rebecca A Jamieson. “Deep subduction and rapid exhumation: Role of crustal strength and strain weakening in continental subduction and ultrahigh-pressure rock exhumation”. In: *Tectonics* 27.6 (2008).

[349] Jessica M Warren and Greg Hirth. “Grain size sensitive deformation mechanisms in naturally deformed peridotites”. In: *Earth Planet. Sci. Lett.* 248.1-2 (2006), pp. 438–450. DOI: 10.1016/j.epsl.2006.06.006.

[350] D. Weeraratne and M. Manga. "Transitions in the style of mantle convection at high Rayleigh numbers". In: *Earth Planet. Sci. Lett.* 160 (1998), pp. 563–568.

[351] R.A. Werner and D.J. Scheeres. "Exterior gravitation of a polyhedron derived and compared with harmonic and mascon gravitation representations of asteroid 4769 Castalia". In: *Celestial Mechanics and Dynamical Astronomy* 65 (1997), pp. 313–344. DOI: 10.1007/BF00053511.

[352] David Wichmann, Philippe Delandmeter, Henk A Dijkstra, and Erik van Sebille. "Mixing of passive tracers at the ocean surface and its implications for plastic transport modelling". In: *Environmental Research Communications* 1.11 (2019), p. 115001. DOI: 10.1088/2515-7620/ab4e77.

[353] Teng-Fong Wong and Li-Chun Wu. "Tensile stress concentration and compressive failure in cemented granular material". In: *Geophys. Res. Lett.* 22.13 (1995), pp. 1649–1652. DOI: 10.1029/95GL01596.

[354] P. Yamato, L. Husson, T.W. Becker, and K. Pedoja. "Passive margins getting squeezed in the mantle convection vice". In: *Tectonics* 32 (2013), pp. 1559–1570. DOI: 10.1002/2013TC003375.

[355] Haibin Yang, Zurab Chemia, Irina M Artemieva, and Hans Thybo. "Control on off-rift magmatism: A case study of the Baikal Rift Zone". In: *Earth Planet. Sci. Lett.* 482 (2018), pp. 501–509. DOI: 10.1016/j.epsl.2017.11.040.

[356] S. Yang and Y. Shi. "Three-dimensional numerical simulation of glacial trough forming process". In: *Science China: Earth Sciences* (2015). DOI: 10.1007/s11430-015-5120-8.

[357] Masaki Yoshida, Satoru Honda, Motoyuki Kido, and Yasuyuki Iwase. "Numerical simulation for the prediction of the plate motions". In: *Earth, planets and space* 53.7 (2001), pp. 709–721. DOI: 10.1186/BF03352399.

[358] Warren Clarence Young and Richard Gordon Budynas. *Roark's formulas for stress and strain*. Vol. 7. McGraw-Hill New York, 2002.

[359] X. Yu and F. Tin-Loi. "A simple mixed finite element for static limit analysis". In: *Computers and Structures* 84 (2006), pp. 1906–1917.

[360] S.T. Zalesak. "Fully Multidimensional Flux-Corrected Transport Algorithms for Fluids". In: *J. Comp. Phys.* 31 (1979), pp. 335–362. DOI: 10.1016/0021-9991(79)90051-2.

[361] Iris van Zelst, Fabio Crameri, Adina E. Pusok, Anne Glerum, Julianne Dannberg, and Cedric Thieulot. "101 Geodynamic modelling: How to design, interpret, and communicate numerical studies of the solid Earth". In: *Solid Earth* 13 (2022), pp. 583–637. DOI: 10.5194/se-13-583-2022.

[362] Iris van Zelst, Cedric Thieulot, and Timothy J. Craig. "The effect of temperature-dependent material properties on simple thermal models of subduction zones". In: *Solid Earth* 14 (2023), pp. 683–707. DOI: 10.5194/se-14-683-2023.

[363] Guoxiang Zhang and Junyu Xiang. "Eight-node conforming straight-side quadrilateral element with high-order completeness (QH8-C1)". In: *Int. J. Num. Meth. Eng.* (2020). DOI: 10.1002/nme.6360.

[364] Huai Zhang, Lili Ju, Max Gunzburger, Todd Ringler, and Stephen Price. "Coupled models and parallel simulations for three-dimensional full-Stokes ice sheet modeling". In: *Numerical Mathematics: Theory, Methods and Applications* 4.3 (2011), pp. 396–418. DOI: 10.1017/S1004897900000416.

[365] Jiaxiang Zhang, Teng-Fong Wong, and Daniel M Davis. "Micromechanics of pressure-induced grain crushing in porous rocks". In: *J. Geophys. Res.: Solid Earth* 95.B1 (1990), pp. 341–352. DOI: 10.1029/JB095iB01p00341.

[366] N. Zhang, S. Zhong, W. Leng, and Z.-X. Li. "A model for the evolution of the Earth's mantle structure since the Early Paleozoic". In: *J. Geophys. Res.: Solid Earth* 115.B06401 (2010). DOI: 10.1029/2009JB006896.

[367] Shangyou Zhang. "A Family of $Q_{k+1,k} \times Q_{k,k+1}$ Divergence-Free Finite Elements on Rectangular Grids". In: *SIAM journal on numerical analysis* 47.3 (2009), pp. 2090–2107. DOI: 10.1137/080728949.

[368] Shangyou Zhang. "On the P1 Powell-Sabin divergence-free finite element for the Stokes equations". In: *Journal of Computational Mathematics* (2008), pp. 456–470. DOI: xxx.

[369] Shangyou Zhang and Mo Mu. "Stable $Q_k - Q_{k-1}$ mixed finite elements with discontinuous pressure". In: *Journal of Computational and Applied Mathematics* 301 (2016), pp. 188–200. DOI: 10.1016/j.cam.2016.01.030.

[370] S. Zhong, M. Gurnis, and G. Hulbert. "Accurate determination of surface normal stress in viscous flow from a consistent boundary flux method". In: *Phys. Earth. Planet. Inter.* 78.1-2 (1993), pp. 1–8. DOI: 10.1016/0031-9201(93)90078-N.

[371] Shijie Zhong. "Analytic solutions for Stokes' flow with lateral variations in viscosity". In: *Geophy. J. Int.* 124.1 (1996), pp. 18–28. DOI: 10.1111/j.1365-246X.1996.tb06349.x.

[372] O.C. Zienkiewicz, M. Huang, and M. Pastor. "Localization problems in plasticity using Finite Elements with adaptive remeshing". In: *International Journal for Numerical and Analytical Methods in Geomechanics* 19 (1995), pp. 127–148. DOI: 10.1002/nag.1610190205.

[373] O.C. Zienkiewicz, C. Humpheson, and R.W. Lewis. "Associated and non-associated visco-plasticity and plasticity in soil mechanics". In: *Géotechnique* 25.4 (1975), pp. 671–689. DOI: 10.1680/geot.1975.25.4.671.

[374] Zahari Zlatev, Jerzy Wasniewski, and Kjeld Schaumburg. "Y12M: Solution of Large and Sparse Systems of Linear Algebraic Equations - Documentation of Subroutines". In: *Lecture Notes in Computer Science* 121 (1981), pp. 1–31. DOI: xxxx.

[375] T. Zwinger, R. Greve, O. Gagliardini, T. Shiraiwa, and M. Lyly. "A full Stokes-flow thermo-mechanical model for firn and ice applied to the Gorshkov crater glacier, Kamchatka". In: *Annals of Glaciology* 45 (2007), pp. 29–37.

Contributors

A. Hendrickx, 308, 895
A. Kotowski, 1106
A. van den Berg, 1089

B. Myhill, 428
B. Root, 513, 615, 634

C. Conrad, 1052
C. Leblanc, 998

D. Bonté, 523
D. Cayuela-Barcham, 1106
D. Duck, 1129

E. Hoogen, 65
E. van der Wiel, 301, 892, 953
E.G.P. Puckett, 47, 105

F. Garel, 846
F. Gueydan, 820, 941, 998

I. van Zelst, 384, 500

J. Austermann, 514
J. Jansen, 19, 26, 310, 472, 733
J. Mos, 19
J. Naliboff, 276
J. Veenhof, 943
J. Wolbers, 875
J.C. Afonso, 1052

L. van de Wiel, 195, 202, 315, 743, 829
L. Verbrugh, 607

M. Blasweiler, 615, 875

N. Ribe, 301, 885, 888

P. Pitard, 669

R. Elbertsen, 154, 190, 404, 513
R. Hassani, 110, 527
R. Maguire, 513
R. Meghezi, 901, 903

S. Hassing, 505

T. Broerse, 566, 854
T. Shinohara, 321, 343
T. Weir, 117, 120
Th. Sanders, 143

W. Klessens, 733

Z. Erdős, 656, 668

Elements

$P_1^+ \times P_1$, 691
$P_2^+ \times P_{-1}$, 691
$P_2^+ \times P_{-1}$, 336
$QH8 - C1 \times Q_1$, 283
$Q_1 \times P_0$, 19, 24, 26, 31, 70, 269
$Q_1 \times Q_1$-stab, 110
$Q_1^+ \times Q_1$, 408, 434, 437
$Q_1^{++} \times Q_1$, 493
$Q_2 \times P_{-1}$, 691
$Q_2 \times Q_1$, 691
$Q_2 \times Q_1$, 269, 283, 287, 333
$Q_2 \times Q_1$, 370
$Q_2^{(8)} \times Q_1$, 283, 287
$Q_3 \times Q_2$, 97, 269
$Q_4 \times Q_3$, 269

Donea & Huerta mms, 19, 70, 97, 110, 269, 283, 691, 760

Sinking block, 287
SolCx mms, 110
SolVi mms, 110