Week 1. Computational methods for ODEs

Oriol Colomés

TU Delft

“Baseball players or cricketers do not need to be able to solve explicitly the non-linear differential equations which govern the flight of the ball. They just catch it”

Paul Ormerod

TU Delft

Following this unit you will avoid the need of years of training

TU Delft

Learning objectives

At the end of this week you will be able to:

Define and analyse numerical methods to solve Ordinary Differential Equations (ODEs). This entails:

1. Define a simple solver to approximate solutions of ODEs based on Taylor Series
2. Quantify the numerical error of an approximated solution
3. Define adaptive time stepping approaches to control the numerical error
4. Distinguish between different ODE solvers

1.1

Introduction

TU Delft

Introduction

Let's say we solved the following problem analytically and obtained its displacement $u(t)$:

$$m\ddot{u} + c\dot{u} + ku = F(t)$$

with Initial Conditions (IC):

$$u(0) = \dot{u}(0) = 0$$

From the analytical solution we can evaluate the displacement at any given point in time t^*: $u^* = u(t^*)$

Question: What do we do if we cannot (don't know how to) obtain an analytical solution to the problem?

Answer: Numerical solution

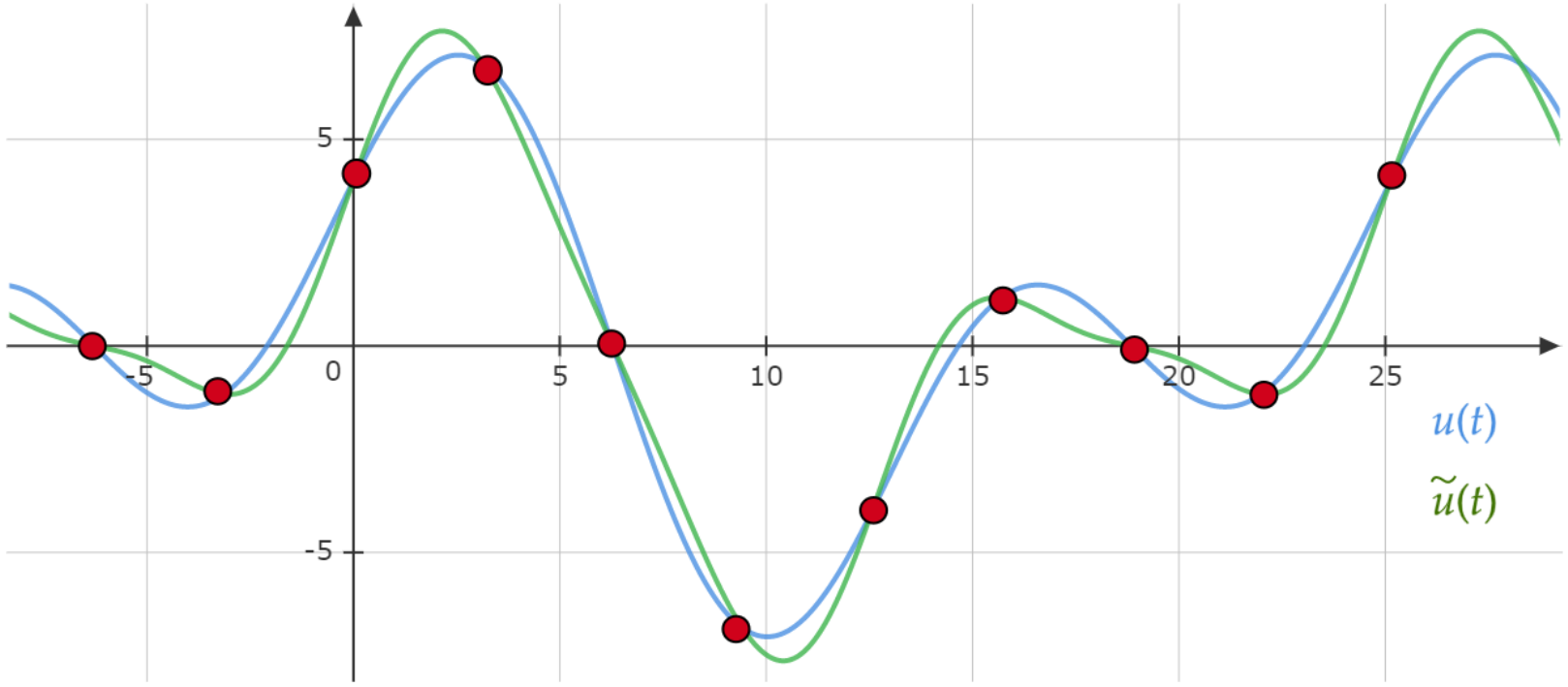
Question: Can we evaluate the numerical solution at any point t^* and still get the same result as the analytical solution?

Answer: It depends on the properties of the numerical and analytical solutions

Introduction

Question: Do we need the solution $u(t)$ at all points?

Answer: Often, we only need a “close-enough” solution: $\tilde{u}(t)$



TU Delft

Notation

Let's define some notation that we will use in this first week:

- For simplicity we assume we discretize in time using a **constant time step** Δt
- We aim to find the solution in the time interval $[0,T]$ using a total of N time steps. Then

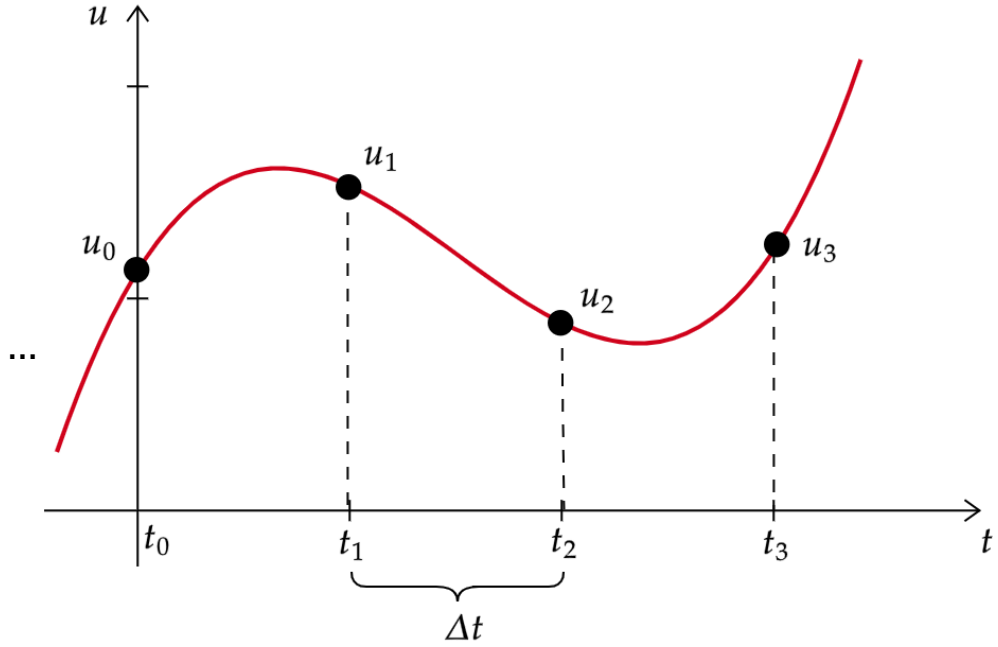
$$\Delta t = \frac{T}{N}$$

- This results in a set of $N+1$ points in time. We will index these points in time with index n: $n = 0,1, \dots N$
- Therefore, these points are located at $t_n = n\Delta t$.
- The total set of points in time is:

$$\boldsymbol{t} = [t_0, t_1, \dots, t_n, \dots, t_N\ ]^T$$

- We will use the following notation for brevity:

$$u(t_n) = u_n, \qquad \dot{u}(t_n\ ) = \dot{u}_n,$$



TU Delft

1.2

Taylor Series

TU Delft

Taylor Series

Let's start refreshing some theory…

A Taylor Series (TS) is *"a representation of a function based on an infinite sum of terms that are calculated from the function's derivatives at a single point"*

For an arbitrary function $f(x)$ and a given point a:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i$$

Or equivalently…

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f'''(a)}{6}(x-a)^3 + \ \ldots$$

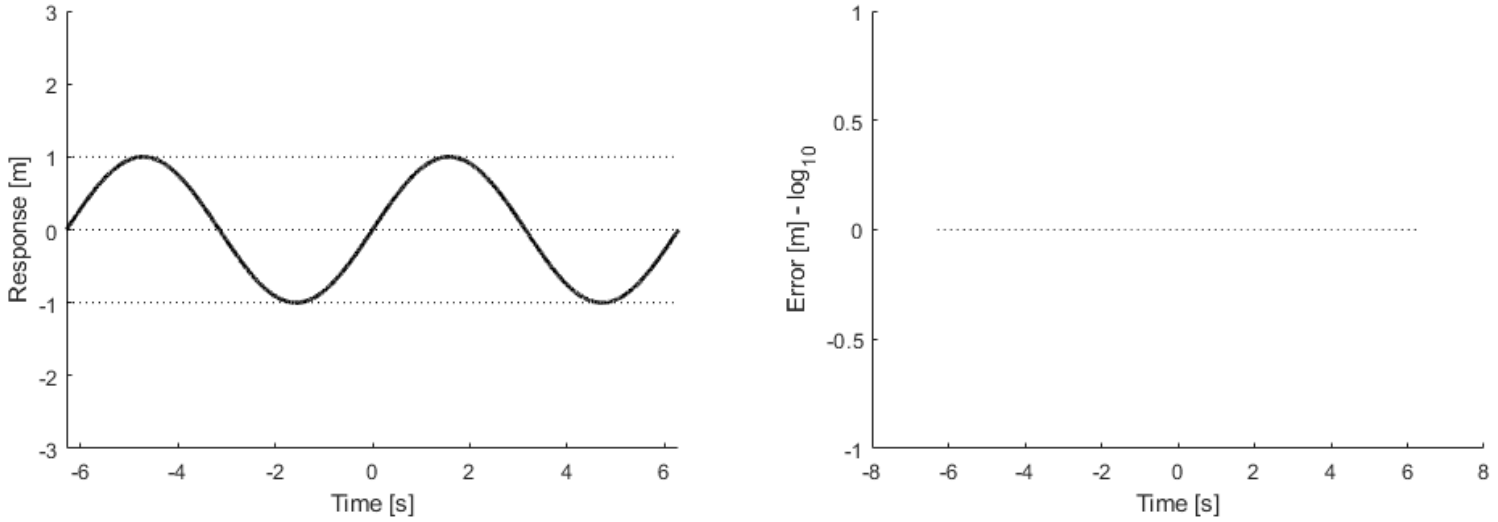
The TS is **exact** as long as we include **infinitely many terms**

TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$



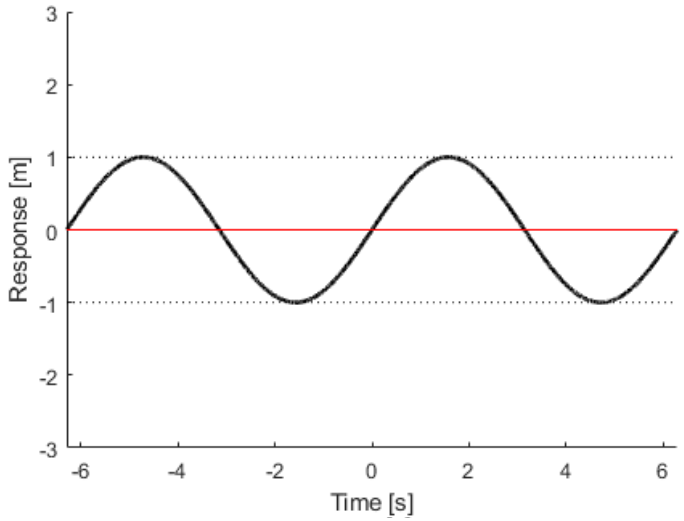
$$\sin(x) = \ ?$$

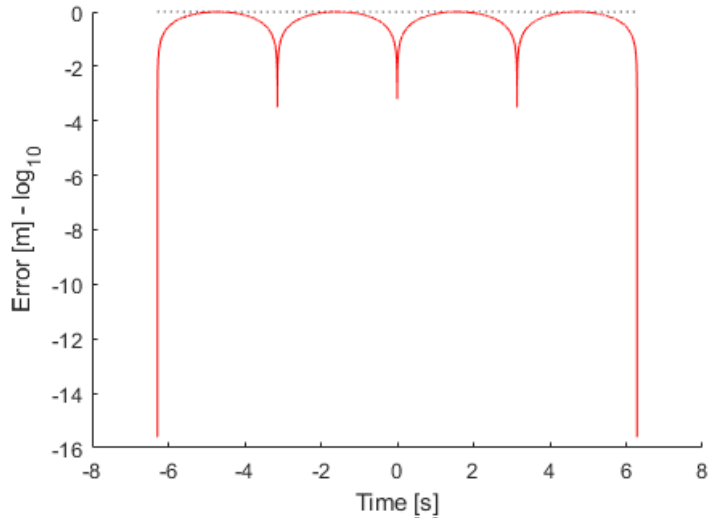
TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$





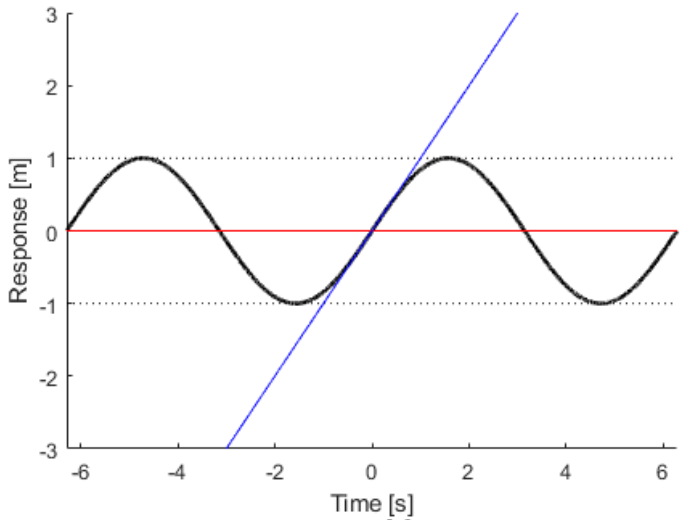
$$\sin(x) = 0 + \cdots$$

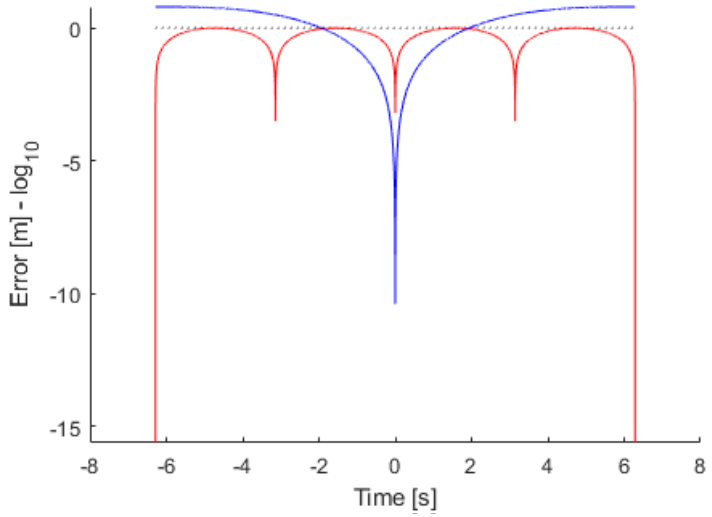
TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$





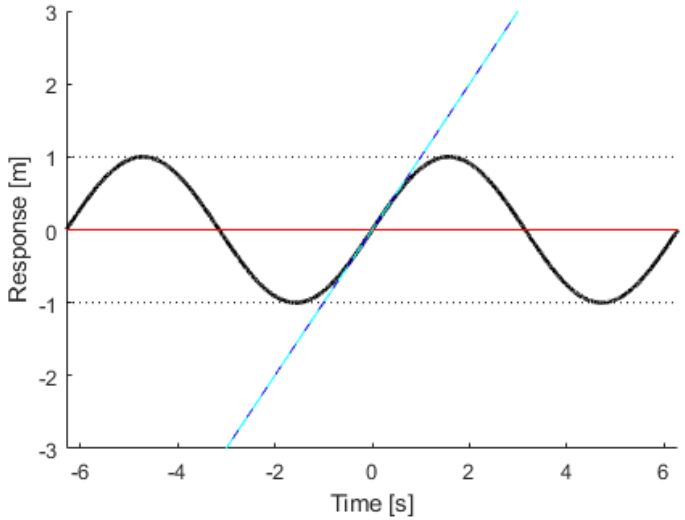
$$\sin(x) = 0 + x \ldots$$

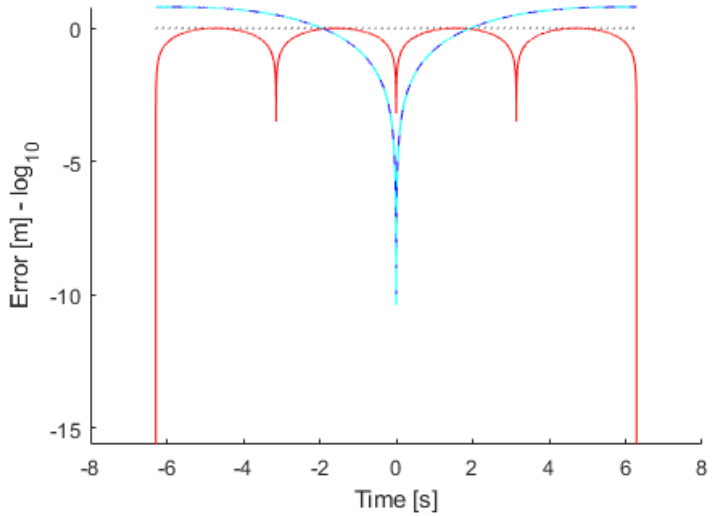
TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$





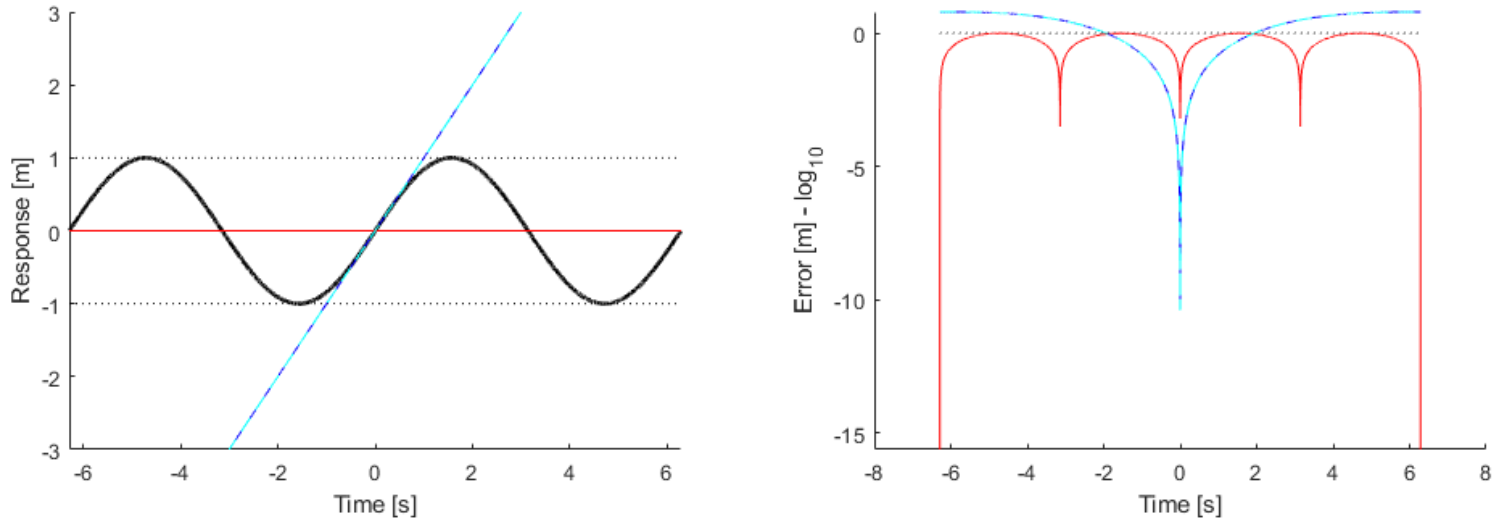
$$\sin(x) = 0 + x \ldots$$

TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$



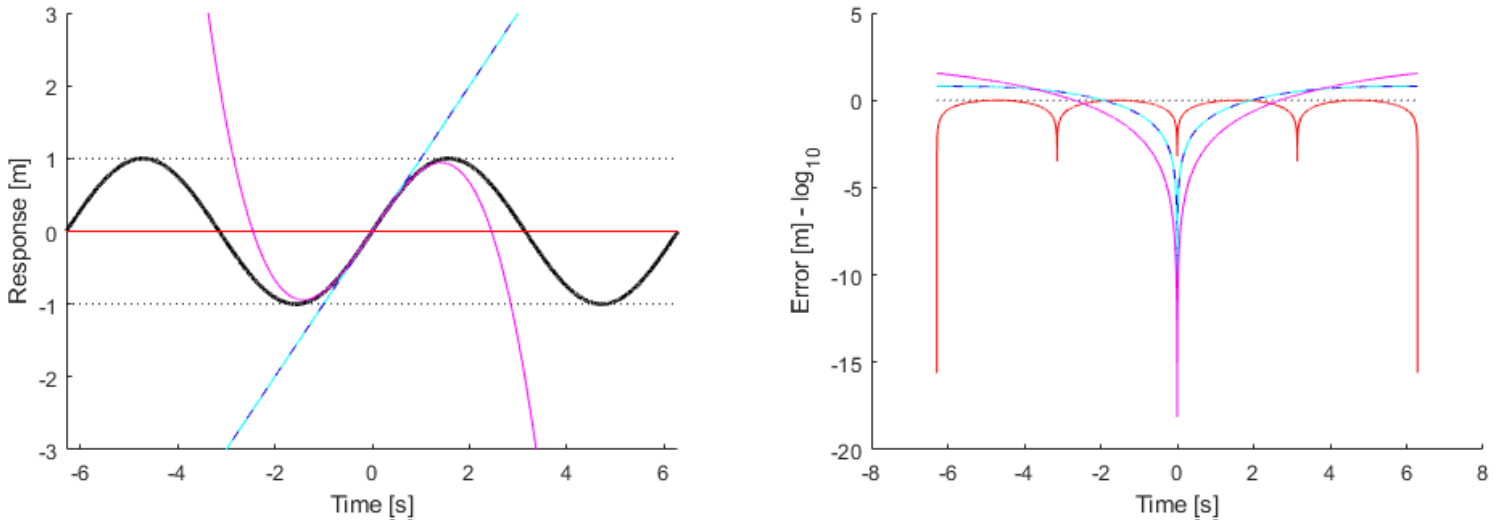
$$\sin(x) = 0 + x + 0 + \cdots$$

TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$



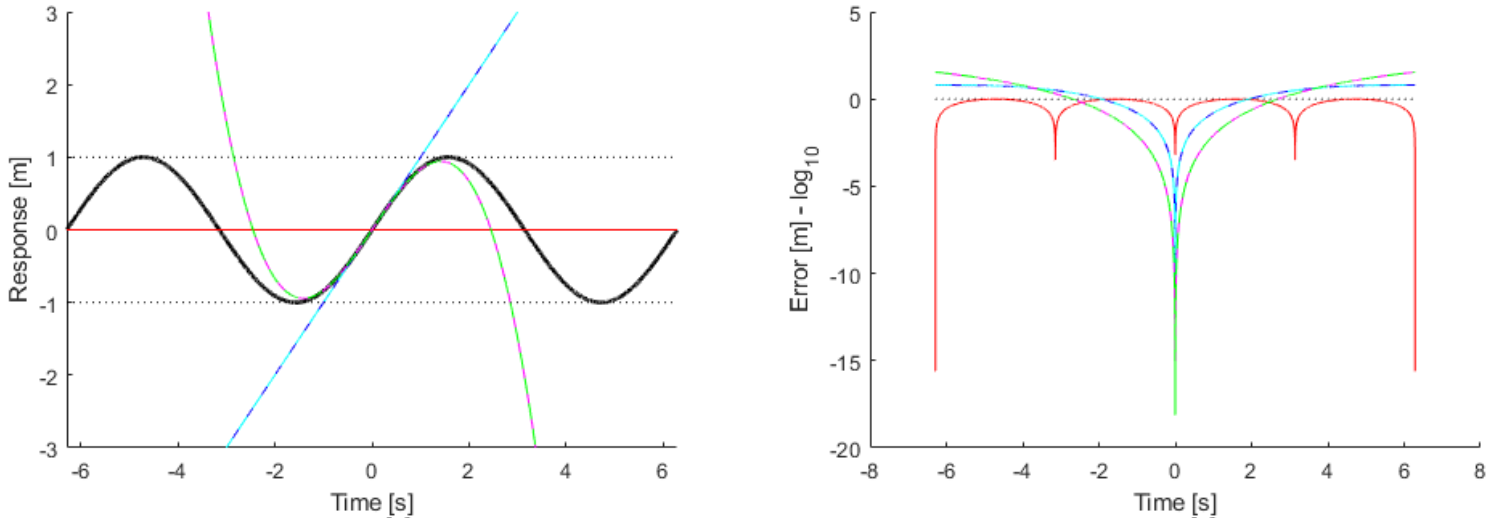
$$\sin(x) = 0 + x + 0 - \frac{x^3}{6} + \cdots$$

TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$



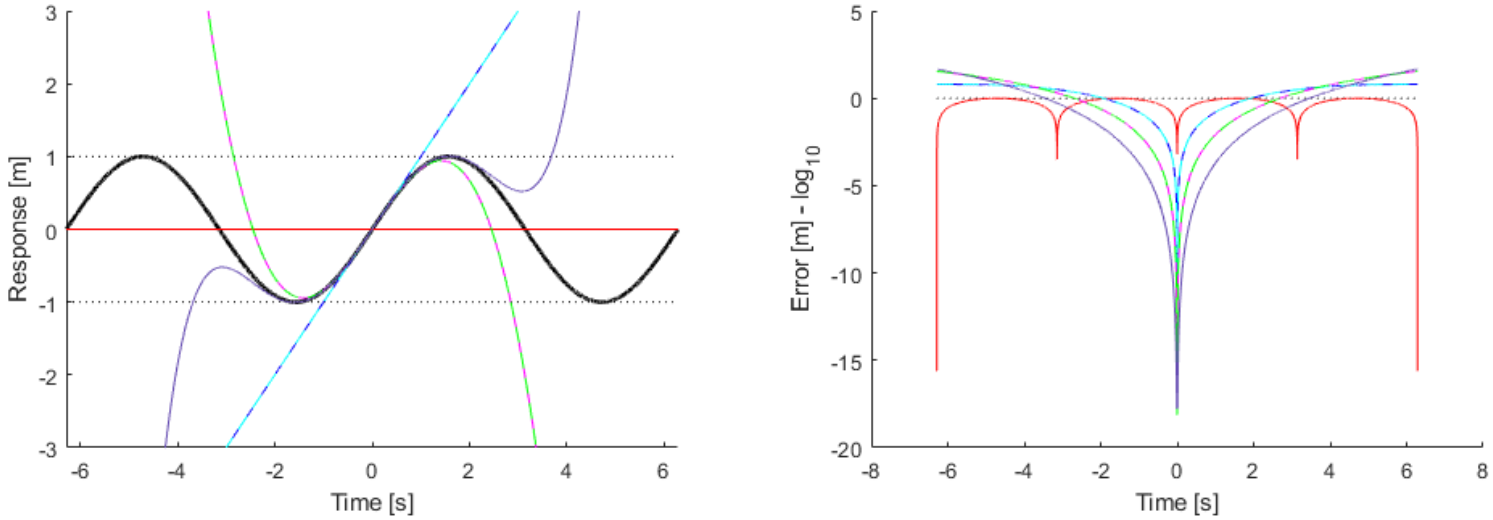
$$\sin(x) = 0 + x + 0 - \frac{x^3}{6} + 0 + \cdots$$

TU Delft

Taylor Series

Let's use the TS for a simple function:

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$



$$\sin(x) = 0 + x + 0 - \frac{x^3}{6} + 0 + \frac{x^5}{120} + \cdots$$

TU Delft

Taylor Series

Let's use the TS for a simple function:

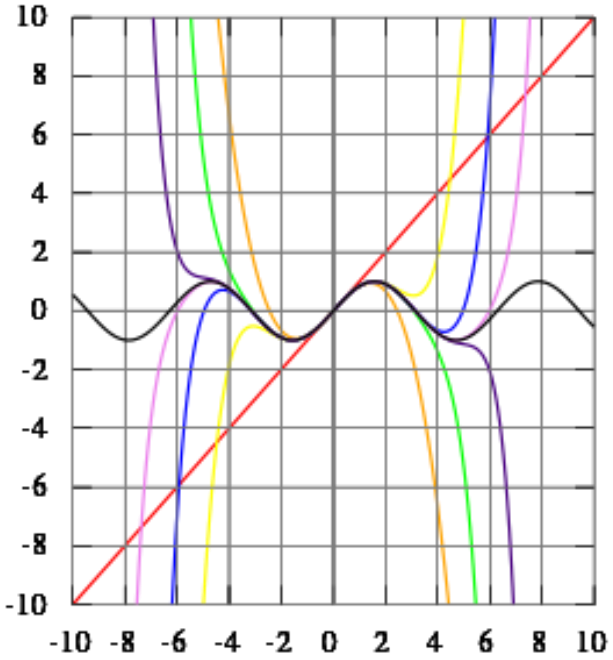
$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$

We can approximate a function $f(x)$ in the vicinity of some point a, if we know the derivatives of $f(x)$ at a.

$$\sin(x) = \mathrm{x} - \frac{\mathrm{x}^3}{6} + \frac{\mathrm{x}^5}{120} + \cdots$$

$$\sin(x) \approx \mathrm{x} - \frac{\mathrm{x}^3}{6} + \frac{\mathrm{x}^5}{120}$$

Since we truncated the expansion, we introduced a **truncation error** which grows with distance from a



TU Delft

Exercises

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i = f(a) + (x-a)\frac{\dot{f}(a)}{1!} + (x-a)^2\frac{\ddot{f}(a)}{2!} + (x-a)^3\frac{\dddot{f}(a)}{3!} + \cdots$$

Exercise: Apply the same process for the function $\cos(x)$

Exercise: Follow the same process as explained in the Jupyter Book and approximate the function $\cos(x)$. Plot the approximation and the error

1.3

Approximating ODEs using Taylor Series

TU Delft

Using Taylor series to solve ODEs

Let's see how we can use the Taylor series to solve an ODE. We start with our model problem:

$$m\ddot{u} + c\dot{u} + ku = F(t)$$

with Initial Conditions (IC):

$$u(0) = u_0, \qquad \dot{u}(0) = \dot{u}_0$$

From the equation of motion, we can also get the second derivative (acceleration):

$$\ddot{u}(0) = \ddot{u}_0 = \frac{(F(0) - ku_0 - c\dot{u}_0)}{m}$$

We want to find an approximation to the solution u ($\tilde{u}$) using the Taylor series

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i$$

Using Taylor series to solve ODEs

Let's take the expansion of $u(t)$ at a given time t^*:

$$u(t) = u(t^*) + (t - t^*)\frac{\dot{u}(t^*)}{1!} + (t - t^*)^2\frac{\ddot{u}(t^*)}{2!} + (t - t^*)^3\frac{\dddot{u}(t^*)}{3!} + \cdots$$

Question: How do we select t^*?

Using Taylor series to solve ODEs

Let's take the expansion of $u(t)$ at a given time t^*:

$$u(t) = u(t^*) + (t - t^*)\frac{\dot{u}(t^*)}{1!} + (t - t^*)^2\frac{\ddot{u}(t^*)}{2!} + (t - t^*)^3\frac{\dddot{u}(t^*)}{3!} + \cdots$$

Question: How do we select t^*?

Answer: At a point where we can evaluate the function and its derivatives, $t^* = t_0$

$$u(t) = u_0 + (t - t_0)\dot{u}_0 + (t - t_0)^2\frac{\ddot{u}_0}{2!} + (t - t_0)^3\frac{\dddot{u}_0}{3!} + \cdots$$

Using Taylor series to solve ODEs

Let's take the expansion of $u(t)$ at a given time t^*:

$$u(t) = u(t^*) + (t - t^*)\frac{\dot{u}(t^*)}{1!} + (t - t^*)^2\frac{\ddot{u}(t^*)}{2!} + (t - t^*)^3\frac{\dddot{u}(t^*)}{3!} + \cdots$$

Question: How do we select t^*?

Answer: At a point where we can evaluate the function and its derivatives, $t^* = t_0$

$$u(t) = u_0 + (t - t_0)\dot{u}_0 + (t - t_0)^2\frac{\ddot{u}_0}{2!} + (t - t_0)^3\frac{\dddot{u}_0}{3!} + \cdots$$

If we evaluate the function at $t_1 = t_0 + \Delta t$, we have

$$u(t_1) = u_1 = u_0 + \Delta t\dot{u}_0 + \Delta t^2\frac{\ddot{u}_0}{2!} + \Delta t^3\frac{\dddot{u}_0}{3!} + \cdots$$

Using Taylor series to solve ODEs

$$u(t_1) = u_1 = u_0 + \Delta t \dot{u}_0 + \Delta t^2 \frac{\ddot{u}_0}{2!} + \Delta t^3 \frac{\dddot{u}_0}{3!} + \cdots$$

Question: Can we evaluate this expression?

Using Taylor series to solve ODEs

$$u(t_1) = u_1 = u_0 + \Delta t \dot{u}_0 + \Delta t^2 \frac{\ddot{u}_0}{2!} + \Delta t^3 \frac{\dddot{u}_0}{3!} + \cdots$$

Question: Can we evaluate this expression?

Answer: We can only evaluate up to the third term, we don't know $\ddot{u}_0$. We can approximate the solution by truncating the expansion

$$u_1 \approx \tilde{u}_1 = u_0 + \Delta t \dot{u}_0 + \Delta t^2 \frac{\ddot{u}_0}{2!}$$

Using Taylor series to solve ODEs

$$\tilde{u}_1 = u_0 + \Delta t \dot{u}_0 + \Delta t^2 \frac{\ddot{u}_0}{2!}$$

We have the approximated solution at t_1, let's see if we can get $u(t_2)$

$$\tilde{u}_2 = \tilde{u}_1 + \Delta t \dot{\tilde{u}}_1 + \Delta t^2 \frac{\ddot{\tilde{u}}_1}{2!}$$

We know that $\ddot{\tilde{u}}_1 = \frac{1}{m}\left(F_1 - k\tilde{u} - c\dot{\tilde{u}}\right)$, so we just need to find $\dot{\tilde{u}}_1$. Using another Taylor expansion we have that

$$\dot{u}_1 = \dot{u}_0 + \Delta t \ddot{u}_0 + \frac{\Delta t^2}{2}\dddot{u}_0 + \cdots$$

$$\dot{u}_1 \approx \dot{\tilde{u}}_1 = \dot{u}_0 + \Delta t \ddot{u}_0$$

TU Delft

Using Taylor series to solve ODEs

This process can be generalized to all discrete points in our time interval t_i for $i = 1, \dots, N$. That is, knowing the solution $\tilde{u}_i$ and its time derivative $\dot{\tilde{u}}_i$, we can get the solution at t_{i+1} as:

1. Compute acceleration at t_i

$$\ddot{\tilde{u}}_i = \frac{1}{m}\left(F_i - k\tilde{u}_i - c\dot{\tilde{u}}_i\right)$$

2. Approximate solution at t_{i+1}

$$\tilde{u}_{i+1} = u_i + \Delta t_i \dot{u}_i + \Delta t_i^2 \frac{\ddot{u}_i}{2!}$$

3. Approximate time derivative at t_{i+1}

$$\dot{\tilde{u}}_{i+1} = \dot{u}_i + \Delta t_i \ddot{u}_i$$

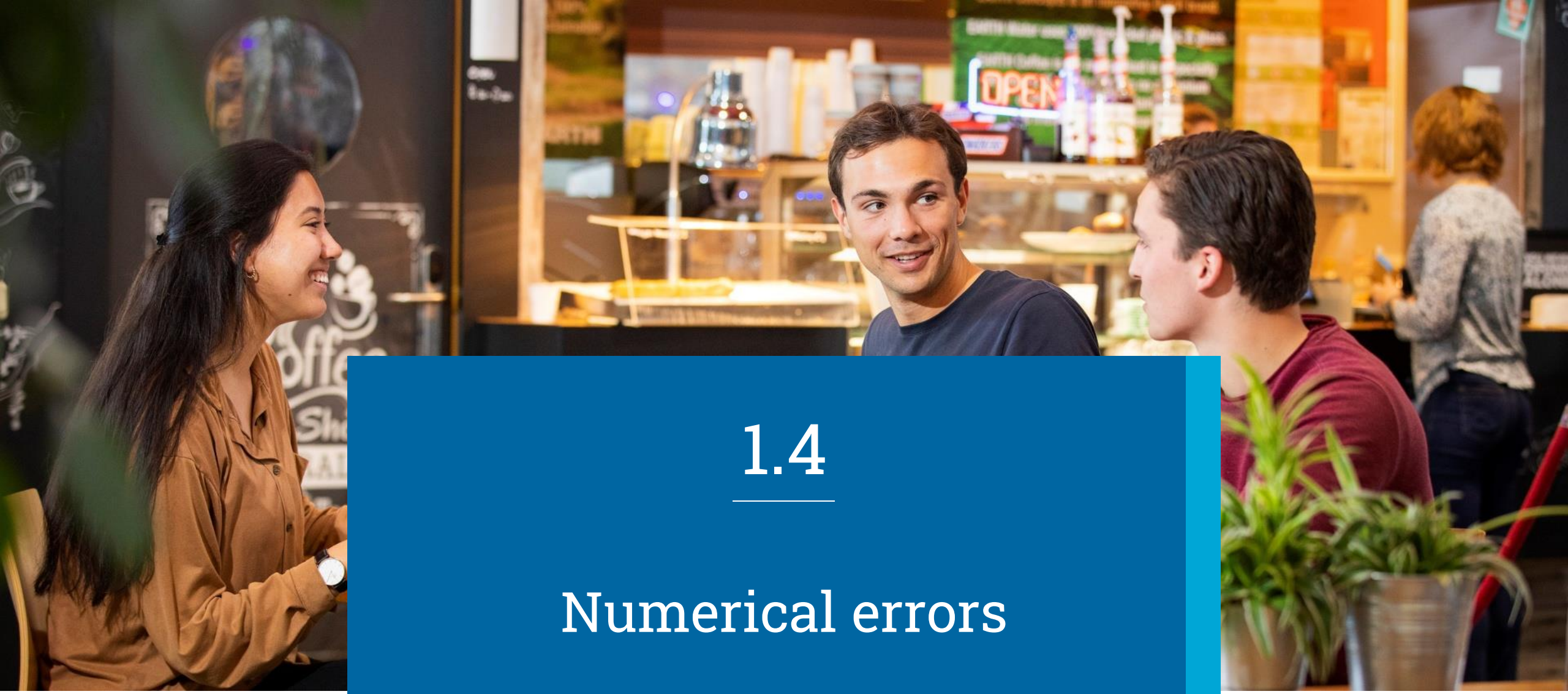
Exercise

Implement the algorithm described before to solve the ODE from $t = 0$ to $t = 1.0$ with $\Delta t = 0.01$:

$$3\ddot{u} + 0.5\dot{u} + 0.1u = 10$$

with Initial Conditions:

$$u(0) = \dot{u}(0) = 0$$



1.4

Numerical errors

TU Delft

Local truncation error

The exact expansion of a function and its derivative is:

$$u_{n+1} = u_n + \Delta t \dot{u}_n + \frac{\Delta t^2 \ddot{u}_n}{2} + \frac{\Delta t^3 \dddot{u}_n}{3!} + \cdots$$

$$\dot{u}_{n+1} = \dot{u}_n + \Delta t \ddot{u}_n + \frac{\Delta t^2 \dddot{u}_n}{2} + \cdots$$

The truncated Taylor series (approximation):

$$u_{n+1} \approx \tilde{u}_{n+1} = u_n + \Delta t \dot{u}_n + \Delta t^2 \frac{\ddot{u}_n}{2}$$

$$\dot{u}_{n+1} \approx \tilde{\dot{u}}_{n+1} = \dot{u}_n + \Delta t \ddot{u}_n$$

Local truncation error

The exact expansion of a function and its derivative is:

$$u_{n+1} = u_n + \Delta t \dot{u}_n + \frac{\Delta t^2 \ddot{u}_n}{2} + \frac{\Delta t^3 \dddot{u}_n}{3!} + \cdots$$

$$\dot{u}_{n+1} = \dot{u}_n + \Delta t \ddot{u}_n + \frac{\Delta t^2 \dddot{u}_n}{2} + \cdots$$

The truncated Taylor series (approximation):

$$u_{n+1} \approx \tilde{u}_{n+1} = u_n + \Delta t \dot{u}_n + \Delta t^2 \frac{\ddot{u}_n}{2}$$

$$\dot{u}_{n+1} \approx \tilde{\dot{u}}_{n+1} = \dot{u}_n + \Delta t \ddot{u}_n$$

Question: What is the error between u_{n+1} and $\tilde{u}_{n+1}$?

Local truncation error

The truncations introduced an error $\epsilon(t)$:

$$\epsilon_u(t_1) = \Delta t^3 \frac{\dddot{u}(t_0)}{3!} + \cdots = O(\Delta t^3)$$

$$\epsilon_{\dot{u}}(t_1) = \Delta t^2 \frac{\ddot{u}(t_0)}{2!} + \cdots = O(\Delta t^2)$$

Since the system converges as Δt decreases (the error become smaller), we are mainly interested in the rate of converge (how fast does the error reduces when reducing Δt).

The symbol $O(\Delta t^r)$ indicates the order of the truncation.

Example: Let's say we computed response using Δt and it was not accurate enough. We reduce Δt by a factor 10 in order to improve the converge. This reduces the error by a factor:

$$\frac{\left(\frac{\Delta t}{10}\right)^3 \frac{\dddot{u}_0}{3!}}{\Delta t^3 \frac{\dddot{u}_0}{3!}} = \frac{1}{10^3} \quad \ll \quad \frac{\left(\frac{\Delta t}{10}\right)^2 \frac{\ddot{u}_0}{2!}}{\Delta t^2 \frac{\ddot{u}_0}{2!}} = \frac{1}{10^2}$$

TU Delft

Local truncation error

Question: What can we conclude about the order of the truncation error of our ODE solver?

Local truncation error

Question: What can we conclude about the order of the truncation error of our ODE solver?

Answer: The higher the order of the solver, the quicker it will converge to the correct solution. Therefore, a higher order improves to accuracy of the solver.

Local truncation error

Question: What can we conclude about the order of the truncation error of our ODE solver?

Answer: The higher the order of the solver, the quicker it will converge to the correct solution. Therefore, a higher order improves to accuracy of the solver.

Based on this logic we would want to have to order as high as possible.

Question: Why is this not always the case in reality?

Local truncation error

Question: What can we conclude about the order of the truncation error of our ODE solver?

Answer: The higher the order of the solver, the quicker it will converge to the correct solution. Therefore, a higher order improves to accuracy of the solver.

Based on this logic we would want to have to order as high as possible.

Question: Why is this not always the case in reality?

Answer:

- Increased calculation time.
- Higher orders make use of higher derivatives, which require the function to be very smooth.

Local truncation error

Both errors, ϵ_u and $\epsilon_{\dot{u}}$, are added to together while evaluating the acceleration:

$$\ddot{u}_n = \frac{1}{m}(F_n - c\dot{u}_n - ku_n)$$

$$\ddot{u}_n = \frac{1}{m}\left(F_n - c\left(\tilde{\dot{u}}_n + \epsilon_{\dot{u}}\right) - k(\tilde{u}_n + \epsilon_u)\right) \sim O(\Delta t^2 + \Delta t^3)$$

Local truncation error

Both errors, ϵ_u and $\epsilon_{\dot{u}}$, are added to together while evaluating the acceleration:

$$\ddot{u}_n = \frac{1}{m}(F_n - c\dot{u}_n - ku_n)$$

$$\ddot{u}_n = \frac{1}{m}\left(F_n - c\left(\tilde{\dot{u}}_n + \epsilon_{\dot{u}}\right) - k(\tilde{u}_n + \epsilon_u)\right) \sim O(\Delta t^2 + \Delta t^3)$$

Question: Which error will become governing if we keep decreasing Δt, $\epsilon_u(t_1) = O(\Delta t^3)$ or $\epsilon_{\dot{u}}(t_1) = O(\Delta t^2)$?

Local truncation error

Both errors, ϵ_u and $\epsilon_{\dot{u}}$, are added to together while evaluating the acceleration:

$$\ddot{u}_n = \frac{1}{m}(F_n - c\dot{u}_n - ku_n)$$

$$\ddot{u}_n = \frac{1}{m}\left(F_n - c\left(\tilde{\dot{u}}_n + \epsilon_{\dot{u}}\right) - k(\tilde{u}_n + \epsilon_u)\right) \sim O(\Delta t^2 + \Delta t^3)$$

Question: Which error will become governing if we keep decreasing Δt, $\epsilon_u(t_1) = O(\Delta t^3)$ or $\epsilon_{\dot{u}}(t_1) = O(\Delta t^2)$?

Answer: $\epsilon_{\dot{u}}(t_1)$ because it is of a lower order! Mathematically: $O(\Delta t^3) + O(\Delta t^2) = O(\Delta t^2)$

Local truncation error

Both errors, ϵ_u and $\epsilon_{\dot{u}}$, are added to together while evaluating the acceleration:

$$\ddot{u}_n = \frac{1}{m}(F_n - c\dot{u}_n - ku_n)$$

$$\ddot{u}_n = \frac{1}{m}\left(F_n - c\left(\dot{\tilde{u}}_n + \epsilon_{\dot{u}}\right) - k(\tilde{u}_n + \epsilon_u)\right) \sim O(\Delta t^2 + \Delta t^3)$$

Question: Which error will become governing if we keep decreasing Δt, $\epsilon_u(t_1) = O(\Delta t^3)$ or $\epsilon_{\dot{u}}(t_1) = O(\Delta t^2)$?

Answer: $\epsilon_{\dot{u}}(t_1)$ because it is of a lower order! Mathematically: $O(\Delta t^3) + O(\Delta t^2) = O(\Delta t^2)$

Since there is a cost associated with this third order error, namely evaluating $\Delta t^2 \frac{\ddot{u}_n}{2}$, and we have just shown that there is no benefit, let's improve the efficient of our algorithm:

$$u_{n+1} \approx \tilde{u}_{n+1} = u_n + \Delta t\dot{u}_n$$

$$\dot{u}_{n+1} \approx \dot{\tilde{u}}_{n+1=}\dot{u}_n + \Delta t\ddot{u}_n$$

TU Delft

Local truncation error

Both errors, ϵ_u and $\epsilon_{\dot{u}}$, are added to together while evaluating the acceleration:

$$\ddot{u}_n = \frac{1}{m}(F_n - c\dot{u}_n - ku_n)$$

$$\ddot{u}_n = \frac{1}{m}\left(F_n - c\left(\dot{\tilde{u}}_n + \epsilon_{\dot{u}}\right) - k(\tilde{u}_n + \epsilon_u)\right) \sim O(\Delta t^2 + \Delta t^3)$$

Question: Which error will become governing if we keep decreasing Δt, $\epsilon_u(t_1) = O(\Delta t^3)$ or $\epsilon_{\dot{u}}(t_1) = O(\Delta t^2)$?

Answer: $\epsilon_{\dot{u}}(t_1)$ because it is of a lower order! Mathematically: $O(\Delta t^3) + O(\Delta t^2) = O(\Delta t^2)$

Since there is a cost associated with this third order error, namely evaluating $\Delta t^2 \frac{\ddot{u}_n}{2}$, and we have just shown that there is no benefit, let's improve the efficient of our algorithm:

$$\left.\begin{aligned} u_{n+1} \approx \tilde{u}_{n+1} &= u_n + \Delta t\dot{u}_n \\ \dot{u}_{n+1} \approx \dot{\tilde{u}}_{n+1} &= \dot{u}_n + \Delta t\ddot{u}_n \end{aligned}\right\}$$ **Forward Euler method!**

TU Delft

Local truncation error

Question: What can we conclude about the order of the truncation error of our ODE solver?

Answer: The higher the order of the solver, the quicker it will converge to the correct solution. Therefore, a higher order improves to accuracy of the solver.

Based on this logic we would want to have to order as high as possible.

Question: Why is this not always the case in reality?

Answer:

- Increased calculation time.
- Higher orders make use of higher derivatives, which require the function to be very smooth.

Global truncation error

The Forward Euler method has a local truncation error with a quadratic order of convergence ($O(\Delta t^2)$).

The global error of convergence is the **error at the final step**, which will have all the local step error accumulation. For a solution from $t = [0, T]$ with N time steps, i.e. $\Delta t = \frac{T}{N}$, we have that the total error will be:

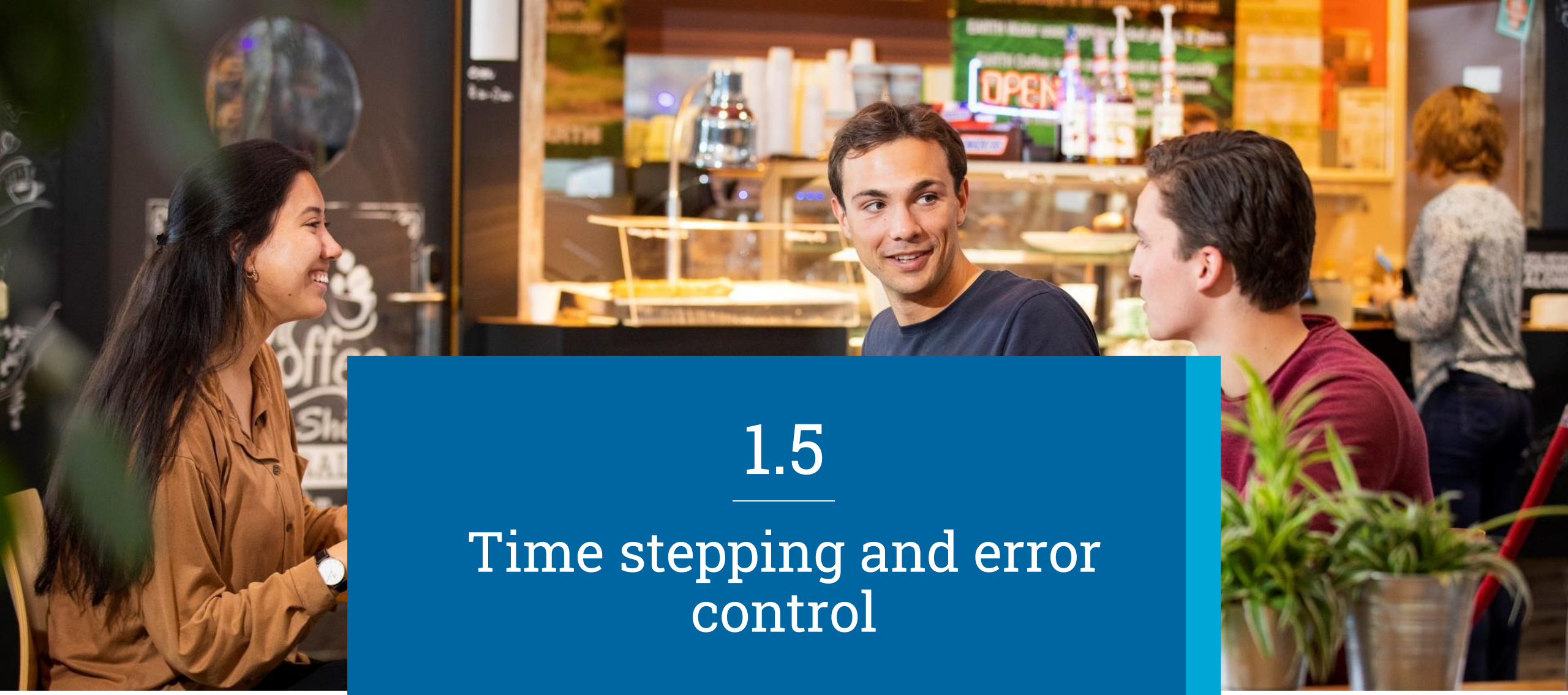
$$\epsilon(T) = \sum_{i=1}^{N} \epsilon_L \sim N\epsilon_L = \frac{T}{\Delta t}\epsilon_L \sim \frac{1}{\Delta t} O(\Delta t^2) = O(\Delta t)$$

Then, the global error will be of order 1, $O(\Delta t)$

Exercise

Workshop / Tutorial in jupyter book

TU Delft



1.5

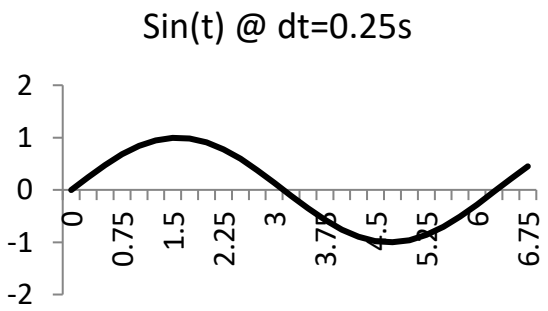
Time stepping and error control

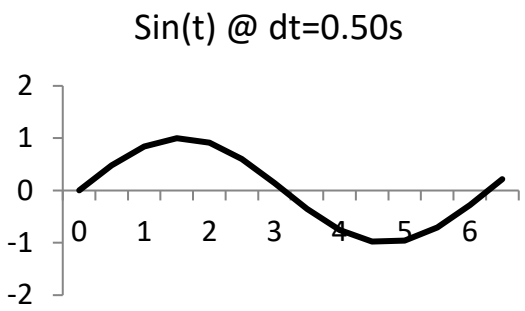
TU Delft

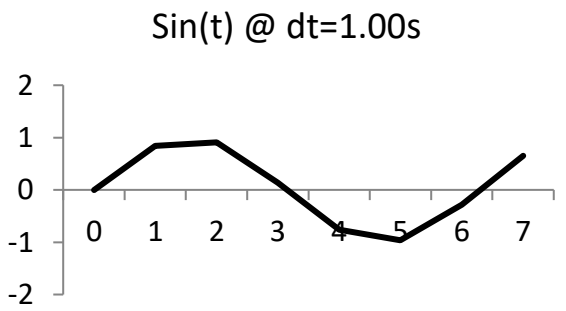
Time stepping

We have seen that the time step influences the error of the solution. How do we choose Δt?

Question: We want to solve the displacement of a system with a natural frequency of $\omega_n = 2\pi$. Which Δt should we choose?







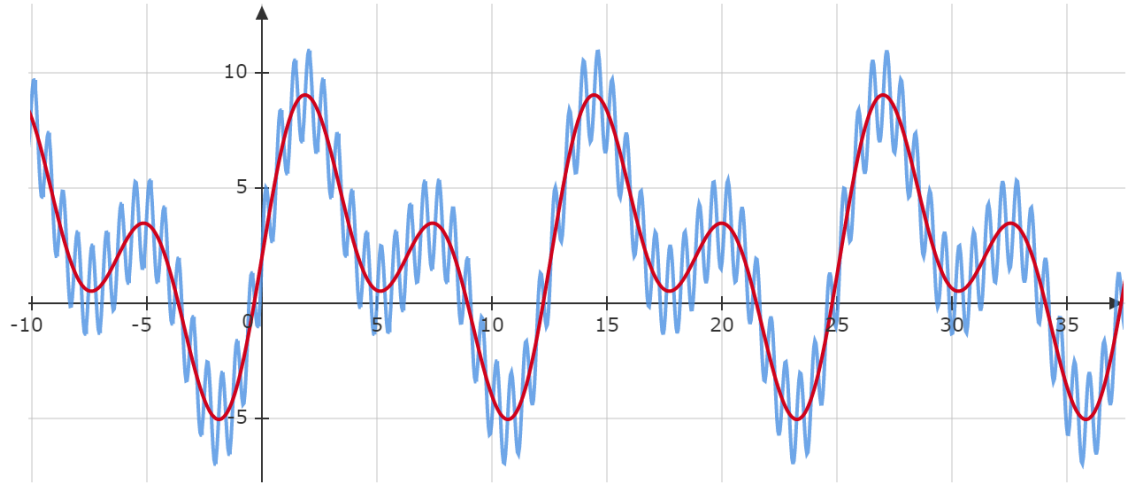
Answer: It depends on the balance accuracy-computational time.

TU Delft

Time stepping

We have seen that the time step influences the error of the solution. How do we choose Δt?

Question: If we have a system with many frequencies. Which frequency will limit the Δt?



Answer: The highest frequency will govern the time step size. To capture the high frequency signal we need small time steps.

TU Delft

Error control

Previously we found that the Forward Euler solver has the following truncation error:

$$\epsilon_u(t_{n+1}) = \frac{\Delta_t^2}{2}\ddot{u}_n + O(\Delta_t^3) \qquad \epsilon_{\dot{u}}(t_{n+1}) = \frac{\Delta_t^2}{2}\dddot{u}_n + O(\Delta_t^3)$$

We established that we can control this error by reducing the time step. Let's say we want out error to be below a certain tolerance.

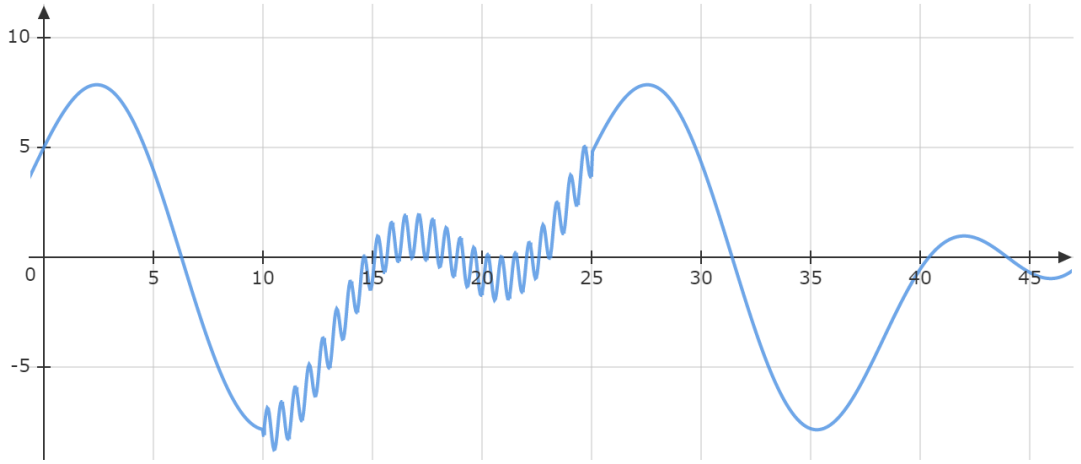
$$\epsilon_u(t_{n+1}) \leq \epsilon_{tol} \ , \qquad \epsilon_{\dot{u}}(t_{n+1}) \leq \epsilon_{tol}$$

Question: In a function like the one shown below, which part of the simulation will be governing for deciding Δt? Why?

Answer: The part with the highest $\ddot{u}_n$ and $\dddot{u}_n$.

Question: How can we avoid this issue?

Answer: We have to use a variable time step.



TU Delft

Error control

Let's say we want the error to be smaller than some tolerance τ (for both ϵ_u and $\epsilon_{\dot{u}}$). Here we'll do the analysis for ϵ_u:

$$\left|\frac{\Delta t^2}{2}\ddot{u}_n + O(\Delta t^3)\right| \leq \tau$$

Ignoring the higher order terms and solving for Δt gives:

$$\Delta t_n \leq \sqrt{2\frac{\tau}{|\ddot{u}_n|}}$$

Question: Does this step size guarantee the error is smaller than the tolerance?

Answer: No, we cannot guarantee that because we didn't account for the $O(\Delta t^3)$ error.

Error control

To also account for $\epsilon_{\dot{u}}$, the time step should be:

$$\Delta t_n \leq \min\left(\sqrt{2\frac{\tau}{|\ddot{u}_n|}}, \sqrt{2\frac{\tau}{|\ddot{u}_n|}}\right)$$

But we don't know $\ddot{u}_n$. We can approximate it by:

$$\dddot{u}_n \approx \frac{\ddot{u}_n - \ddot{u}_{n-1}}{\Delta t_{n-1}}$$

Error control

How do we choose the tolerance τ?

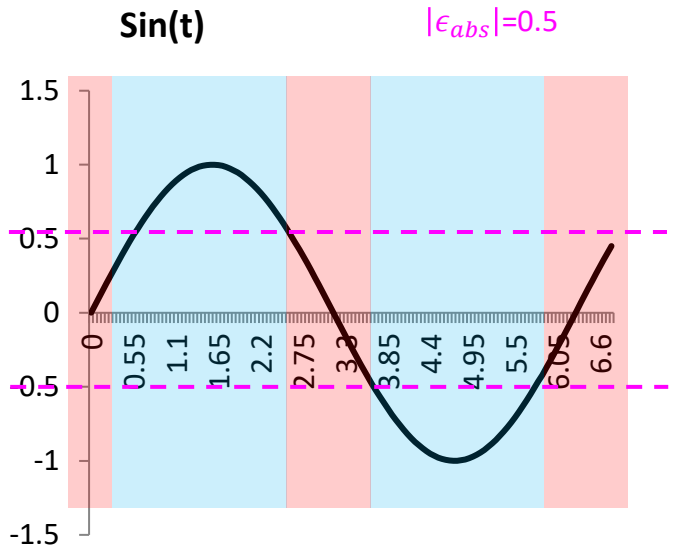
We can establish a relative tolerance with respect to the solution: $\tau = \epsilon_{rel}|u_n|$

$$\Delta t_n \leq \min\left(\sqrt{2\frac{\epsilon_{rel}|u_n|}{|\ddot{u}_n|}}, \sqrt{2\frac{\epsilon_{rel}|\dot{u}_n|}{|\ddot{u}_n|}}\right)$$

Question: What if the solution and the time derivative go to zero?

Answer: Add an absolute tolerance to prevent the time step to go to zero

$$\tau = \max(\epsilon_{rel}|u_n|, \epsilon_{abs})$$

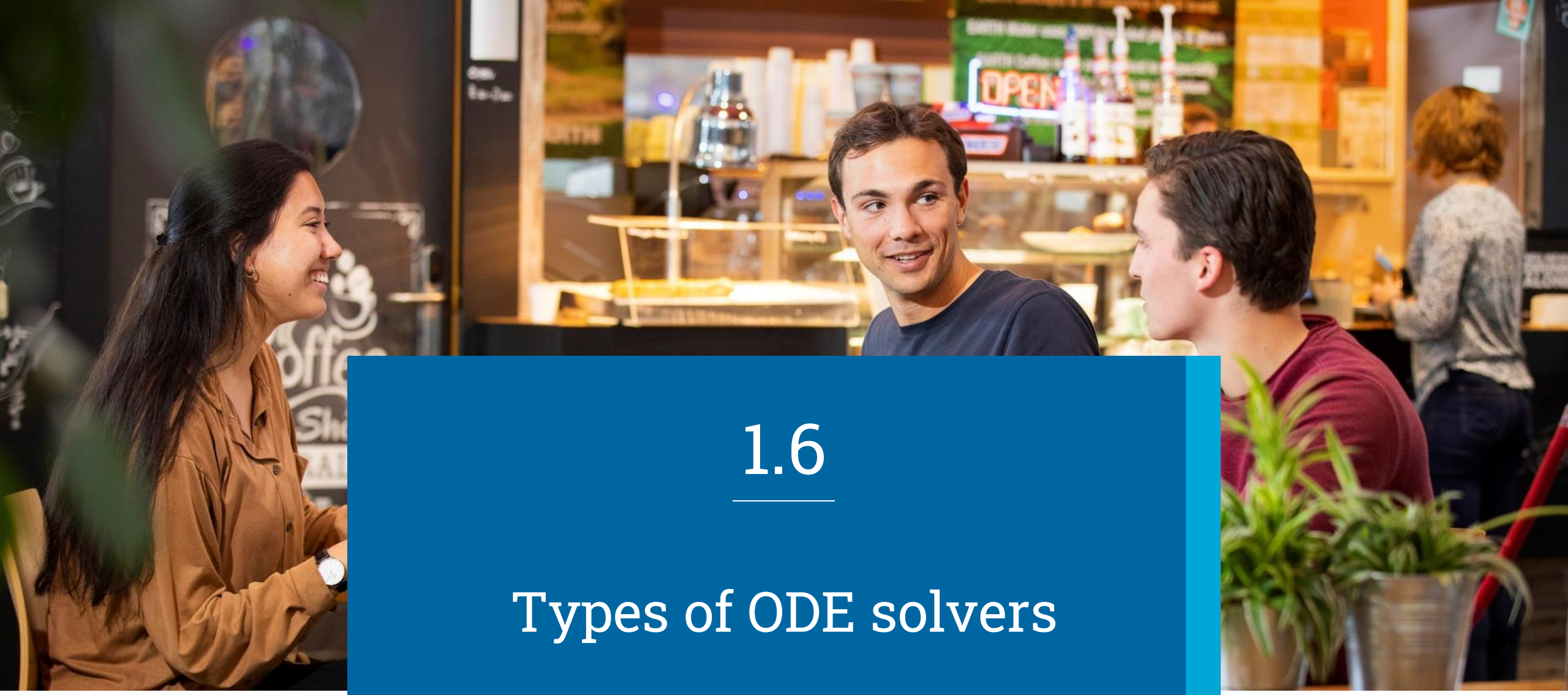


TU Delft

Error control

Implementation details:

- At the start of a calculation it is unclear how “fast” the system moves. Therefore, the solver must first find a suitable time step.
- To achieve this, we can iterate until it find a time step that meets the specified tolerances.
- Unless the system has a very abrupt change in its “speed”, the solver can keep using a similar / the same step size.



1.6

Types of ODE solvers

TU Delft

Types of ODE solvers

Going back to the compact form of our Forward Euler scheme:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + \Delta t\, \dot{\boldsymbol{q}}_n + O(\Delta t^2)$$

We can make it more abstract:

$$\boldsymbol{q}_{n+1} = f_{FE}(\boldsymbol{q}_n, \dot{\boldsymbol{q}}_n) + O(\Delta t^2)$$

Where f_{FE} is a function (in this case for our Forward Euler scheme) that takes the current state $\boldsymbol{q}_n$ and its derivative $\dot{\boldsymbol{q}}_n$ as input and computes the state at the next time-step with some error. And since $\dot{\boldsymbol{q}}_n$ can be computed from $\boldsymbol{q}_n$ (using the EOM):

$$\boldsymbol{q}_{n+1} = f_{FE}(\boldsymbol{q}_n) + O(\Delta t^2)$$

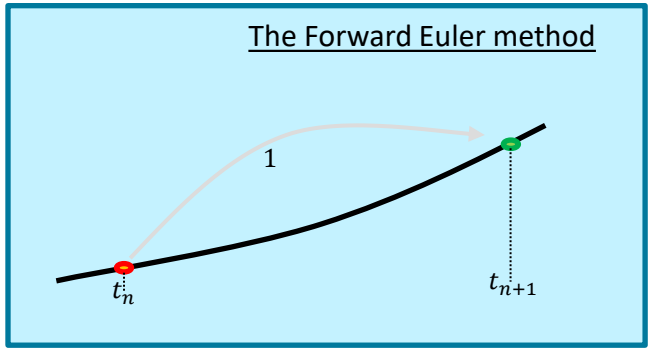
An example of other solver:

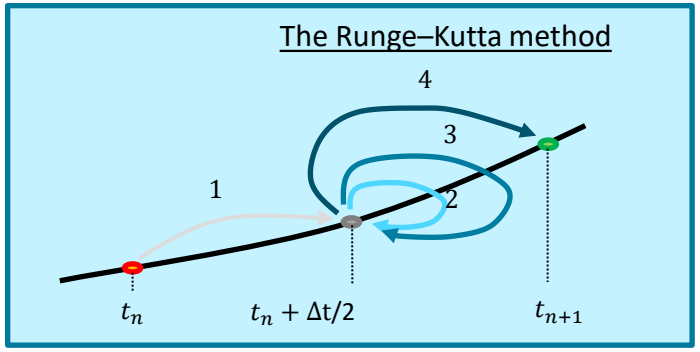
Using this form we can look at other ODE solvers. For instance, the 5th order Runga-Kutta scheme that you will be mostly using during the course:

$$\boldsymbol{q}_{n+1} = f_{RK45}(\boldsymbol{q}_n) + O(\Delta t^5)$$

Types of ODE solvers

One stage vs multi-stages:





Forward Euler uses 1 stage

$$\boldsymbol{q}_{n+1} = f_{FE}(\boldsymbol{q}_n) + 0(\Delta_t^2)$$

Runge-Kutta 45 uses 4 stages

$$\boldsymbol{q}_{n+1} = f_{RK45}(\boldsymbol{q}_n) + 0\left(\Delta_t^5\right)$$

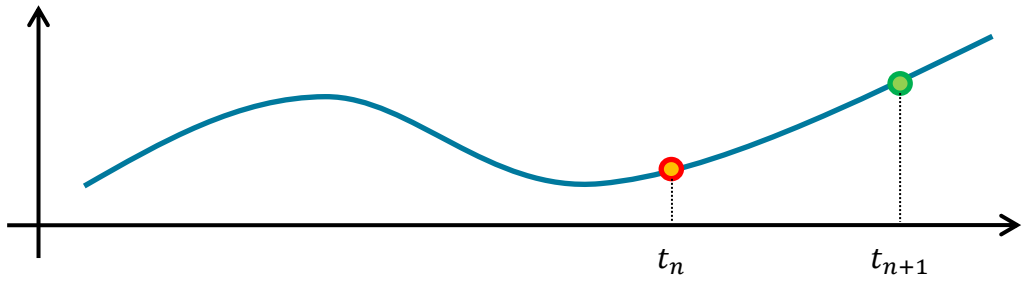
Question: Why not always use RK45?

Answer: An improved accuracy always requires using more points and therefore reduce the speed of the system. However, they also allow for larger time steps.

TU Delft

Types of ODE solvers

One step vs multi-step:



Forward Euler uses 1 step

$$\boldsymbol{q}_{n+1} = f_{FE}(\boldsymbol{q}_n) + O(\Delta_t^2)$$

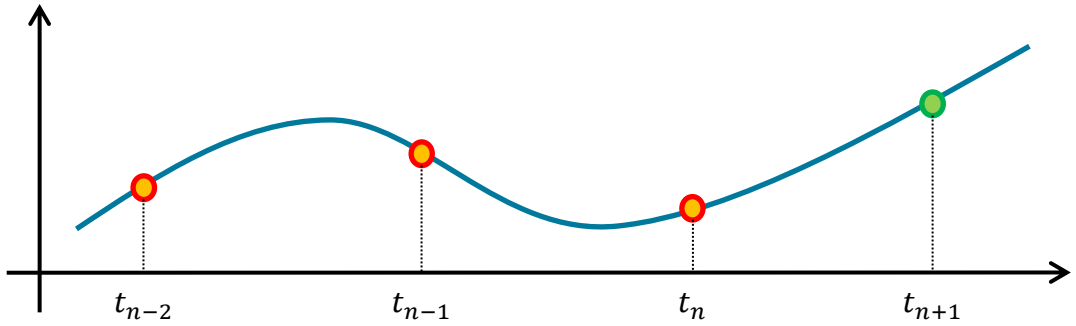
Runge-Kutta 45 uses 1 step

$$\boldsymbol{q}_{n+1} = f_{RK45}(\boldsymbol{q}_n) + O(\Delta_t^5)$$

Explicit multi-step methods:

2 step: $\boldsymbol{q}_{n+1} = f_{??}(\boldsymbol{q}_n, \boldsymbol{q}_{n-1}) + O(\Delta_t^?)$

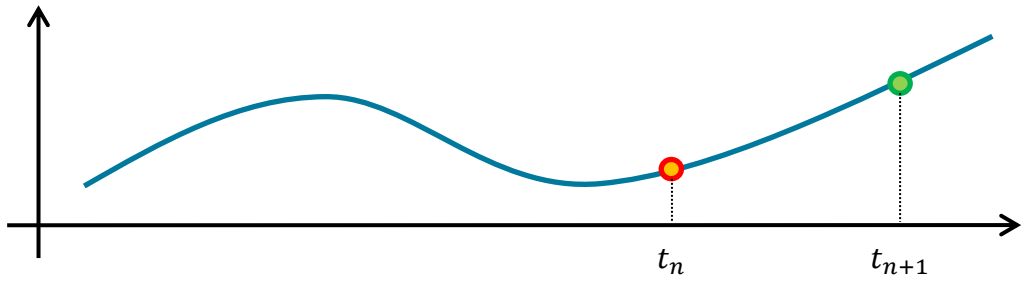
3 step: $\boldsymbol{q}_{n+1} = f_{??}(\boldsymbol{q}_n, \boldsymbol{q}_{n-1}, \boldsymbol{q}_{n-2}) + O(\Delta_t^?)$



TU Delft

Types of ODE solvers

Explicit vs implicit:



Forward Euler uses 1 step

$$\boldsymbol{q}_{n+1} = f_{FE}(\boldsymbol{q}_n) + O(\Delta_t^2)$$

Runge-Kutta 45 uses 1 step

$$\boldsymbol{q}_{n+1} = f_{RK45}(\boldsymbol{q}_n) + O(\Delta_t^5)$$

Implicit methods: (example Backward Euler)

$$\boldsymbol{q}_{n+1} = f_{??}(\boldsymbol{q}_n, \boldsymbol{q}_{n+1}) + O(\Delta_t^?)$$

Implicit methods have better stability properties, but they are often more expensive

TU Delft

Thank you for your attention

Oriol Colomés

TU Delft