How to analyze data with ALICE software

D. Caffarri
Postdoc researcher at NIKHEF (Amsterdam)
davide.caffarri@nikhef.nl

Structure of the software

- ROOT is the main analysis tool we use to analyze data at CERN.
  - fully developed at CERN
  - tool for statistical analysis, plotting, define data format, …
  - here you can find the slides Panos presented about ROOT
- AliRoot is the 1st ALICE specific software developed by scientists of the ALICE Collaboration
  - C++ structured code
  - Build “on top” of ROOT (that’s why we are somehow dependent on the ROOT version when we install it)
  - Mainly used for Reconstruction, Simulation, Calibration, General Analysis framework.
  - AliRoot version are tagged and deployed on the filesystem (CVMFS) or GRID when big modifications are needed —> experts decide.
  - Very likely you might not need to investigate or modify this code.

Structure of the software

- AliPhysics is the 2nd ALICE specific software developed by scientists of the ALICE Collaboration
  - C++ structured code
  - Build “on top” of AliRoot (that’s why we are somehow dependent units version when we install it)
  - Mainly used for single user analysis YOU!
  - AllPhysics version are tagged and deployed on the filesystem (CVFNS) or GRID once per day—> done for analyzers
  - Very likely you will deal with this code.
  - The code is divided in directories, one per Physics Working Group

About ALICE Structure

- Physics Working Groups are the groups that organize the physics measurements and their interpretations within the ALICE Collaboration
  - Each PWG incorporates a group of analysis that cover similar topics
  - Yours will be Correlations and Fluctuations one (PWGCF)

- Each PWG is divided in Physics Analysis Group (PAG)
  - This is the first step of the analysis approval “procedure” when you want to show your results.
  - Here you can present also technical problems and results that are still work in progress.
  - Usually the different PAGs share the similar structure of the code and same analysis tool
  - ex Balance Functions code is in

Your class

`~/alice/AliPhysics/PWGCF/EBE/BalanceFunctions/Ali…`

PWG PAG Specific analysis

Where are the data?

- Depending on which data you want/need to use you might structure your code in different way.
- ALICE provide a “general analysis framework” that is what we suggest to use to develop your analysis but some preliminary studies can be done also in different ways.

- Data you use can be:
  - On your laptop → very limited amount of data (few hundred of events). Just use for testing that your code works, basic debug (i.e. your histograms are filled properly, no basic segmentation violations, …)
  - On Stoomboot → you can analyze in principle a full data sample, we stored a couple of them there. Useful for code development, comparisons and fast look at Quality Assurance results.
  - On GRID → “our” large data distributed system. All data samples we collected since 2009 and MC productions are available there. Useful if you have to use the full data statistics.

ALICE data structure

- You will run your analysis on Analysis Objects Data (AOD) data
  - they contain information already processed (you should not worry about detector calibration) on the **tracks** (pT, eta, phi, …) that you have on each **event**

  - Each event contains all information about the collisions: centrality, primary vertex position, Nb of tracks produced in the events, kinematic variables of the tracks.

- A certain amount of events are stored in one **run.** A run is defined from the data taking procedure (Online system). Very likely from a change in the detector configuration. Not all runs are good, there might be problems in the detector configuration. Run list to be used are defined by the Performance WG after extensive QA and can be found here.

  But also foresee your QA plots in your analysis. It is always good to cross check!

- A series of runs with similar characteristics are grouped in period. For example all PbPb 2015 (Run2 data) where collected during the LHC15o period.

Few things about C++

- Your analysis task will be a C++ class.
- A class contains variables and functions (called also methods) as members
- Member variables are usually accessed via methods

- Example:

```
class Rectangle {
private:
 int width, height;
public:
 Rectangle (int, int);
 int GetArea() {return height * width};
}
```

Few things about C++

- Class can inherit from one other
- This avoid to repeat common code for all classes, but can make the code a bit tricky
- Example:

```
class Polygon {
  private:
   int width, height;
  }

class Rectangle : public Polygon{
  public:
    Rectangle (int, int);
    int GetArea() {return height * width};
  }

class Triangle : public Polygon{
  public:
    Triangle (int, int);
    int GetArea() {return (height * width)/2};
  }
```

- Rectangle is derived from the base class Polygon and inherits its members
- while studying already produced code if you cannot find members or methods check the base class

How to create your class

- We usually call the analysis code *Analysis Task*
- Following official code your main method will “automatically” loop over the events
  - so you have to think of a code that does the same thing over all events
- All analysis are derived from `AliAnalysisTaskSE`
  - This provide the common framework where we do the analysis
  - you have to stick to some common rules on how build your analysis

`AliAnalysisTaskSE::AliAnalysisTaskSE();`	Constructor
`AliAnalysisTaskSE::AliAnalysisTaskSE(const char*);`	Copy Constructor
`AliAnalysisTaskSE::UserCreateOutputObjects();`	Histos definition
`AliAnalysisTaskSE::UserExec(Option_t*);`	Analysis
`AliAnalysisTaskSE::Terminate(Option_t*);`	End of the analysis

- Fixed format to build your class:
  - header file .h → contains function prototypes
  - implementation file .cxx → where the code is implemented
  - AddTask.C → creates an instance of your class and configures it

Header File AliAnalysisTaskMyTask.h

```
#ifndef ALIANALYSISTASkMYTASK_H
#define ALIANALYSISTASKMYTASK_H
class AliAnalysisTaskMyTask : public AliAnalysisTaskSE {
public:
  //constructors
  AliAnalysisTaskMyTask();
  AliAnalysisTaskMyTask(const char *name);
  // destructor
  virtual ~ AliAnalysisTaskMyTask();
  //called once at the beginning of the runtime
  virtual void UserCreateOutputObjects();
  //called for each event
  virtual void UserExec(Option_t* option);
  //called at the end of the analysis
  virtual void Terminate(Option_t* option);
  ClassDef (AliAnalysisTaskMyTask, 1);
  };
  #endif
```

Header File AliAnalysisTaskMyTask.h

```
#ifndef ALIANALYSISTASKBFPSI_H
#define ALIANALYSISTASKBFPSI_H
class AliAnalysisTaskMyTask : public AliAnalysisTaskSE {
public:
  //constructors
  AliAnalysisTaskMyTask();
  AliAnalysisTaskMyTask(const char *name);
  // destructor
  virtual ~ AliAnalysisTaskMyTask();
  //called once at the beginning of the runtime
  virtual void UserCreateOutputObjects();
  //called for each event
  virtual void UserExec(Option_t* option);
  //called at the end of the analysis
  virtual void Terminate(Option_t* option);

  private:
  AliAODEvent *fAOD; //! input event
  TList* fOutputList; //! output list
  TH1F *fHistPt; //! histogram
  ClassDef (AliAnalysisTaskMyTask, 1);
  };
  #endif
```

- Class members must be defined in the header!
- Pointers to objects that are initialized in the UserCreateOutputObjects() should have //!
- The Output List is used to have one output objects with many different other objects inside

Implementation AliAnalysisTaskMyTask.cxx

```
AliAnalysisTaskMyTask::AliAnalysisTaskMyTask():
  AliAnalysisTaskSE(),
  fAOD(0x0), fOutputList(0x0),  fHistPt(0),
  {
  }

AliAnalysisTaskMyTask::AliAnalysisTaskMyTask(const char *name):
  AliAnalysisTaskSE(),
  fAOD(0x0), fOutputList(0x0),  fHistPt(0),
  {
  DefineInput(0, TChain::Class());
  DefineOutput(1, TList::Class());
}
```

- Two constructors where:
  - we initialize the data members to their default values,
  - we said to the task what to expect as input and output

Implementation AliAnalysisTaskMyTask.cxx

```
AliAnalysisTaskMyTask::AliAnalysisTaskMyTask():
  AliAnalysisTaskSE(),
  fAOD(0x0), fOutputList(0x0),  fHistPt(0),
  {
  }

AliAnalysisTaskMyTask::AliAnalysisTaskMyTask(const char *name):
  AliAnalysisTaskSE(),
  fAOD(0x0), fOutputList(0x0),  fHistPt(0),
  {
  DefineInput(0, TChain::Class());
  DefineOutput(1, TList::Class());
}

AliAnalysisTaskMyTask::UserCreateOutputObjects() {

fOutputList = new TList();
fOutputList->SetOwner(kTRUE);

fHistPt = new TH1F("fHistPt", "fHistPt", 100, 0, 100);
fOutputList ->Add(fHistPt);

PostData(1, fOutputList);
}
```

- Create a list
- Create a hits and add it to the list
- Add the list to the output file

- Remember to include the needed classes: TList and TH1F in this case

Implementation AliAnalysisTaskMyTask.cxx

```
AliAnalysisTaskMyTask::UserExec(Option_t *){

  AliAODEvent *fAOD = dynamic_cast<AliAODEvent *>(InputEvent());
  if (!fAOD) return;

  Int_t nTracks=fAOD->GetNumberOfTracks();
  for(Int_t i=0; i< nTracks; i++){
    AliAODTrack *track = static_cast<AliAODTrack*>(fAOD->GetTrack(i));
    if (!track) continue;
    fHistPt->Fill(track->Pt());
  }

  PostData(1, fOutputList);
}
```

- Take the event from the analysis manager
- From the event get the tracks and loop over them
- Fill your output histogram
- Save the updated output

Configuration macro: AddTaskMyTask.C

```
AliAnalysisTaskMyTask *AddMyTask(TString name =“name”){

  AliAnalysisManger *mgr = AliAnalysisManager::GetAnalysisManager();

  TString fileName = AliAnalysisManager::GetCommonFileName();
  fileName+=“_MyTask”;

  AliAnalysisTaskMyTask *task = new AliAnalysisTaskMyTask(name.Data());
  mgr->AddTask(task);

  mgr->ConnectInput(task, 0, GetCommmonInputContainer());

  mgr->ConnectOutput(task, 1, mgr, CreateContainer(“MyOutputContainer”,
  TList::Class(), AliAnalysisManager::kOutputContainer,
  filename.Data());

  return task;

}
```

- Get the analysis manager
- Initialize your task
- Connect input container (only one)
- Connect all output containers.

How to run the analysis locally: runAnalysis.C

```
void runAnalysis(){
  gROOT->ProcessLine(“.include $ROOTSYS/include”);
  gROOT->ProcessLine(“.include $ALICE_ROOT/include”);
  gROOT->ProcessLine(“.include $ALICE_PHYSICS/include”);

  AliAnalysisManager *mgr = new AliAnalysisManager(“AnalysisMyTask”);

  AliAODInputHandler *aodH = new AliAODInputHandler();
  mgr->SetInputEventHandler(aodH);

  gROOT->LoadMacro(“AliAnalysisTaskMyTask.cxx++g”);

  gROOT->LoadMacro(“AddMyTask.C”);

  AliAnalysisTaskMyTask *task = AddMyTask();

  TChain *chain = new TChain(“aodTree”);
  chain->Add(“…/../../AliAOD.root”);

  mgr->StartAnalysis(“local”, chain);

}
```

- Say where are the file to be included
- Initialize the analysis manager and the type of input you need
- Compile your task, load your configuration macro.
- Initialize your analysis task
- Define your input.