Hands on: Analysis with UD tables

O2 Analysis Tutorial 4.0

Anisa Khatun

16-10-2024

PWG-UD Session

U.S. DEPARTMENT OF ENERGY
Office of Science

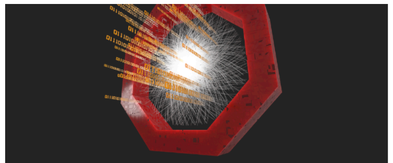
THE UNIVERSITY OF KANSAS

Outline

- Ingredients
- Two pion invariant mass analysis at mid rapidity
    - Using DG derive data UD tables
    - Using SG derive data UD tables
- Two muon invariant mass analysis at forward rapidity
    - Using UPC derive data UD tables
- Event by event analysis: derived tree

Ingredients

- Pre-requisites : Latest O2phsysics installed locally
- UD Tables: https://aliceo2group.github.io/analysis-framework/docs/datamodel/pwgTables.html#pwg-ud
- Resources: O2 Analysis tutorials https://indico.cern.ch/event/1425820/timetable/
- Webpage: https://alice-run3analysis.web.cern.ch/
- Subscribe to alice-dpg-general@cern.ch for data and MC processing news (optional)
- Asynchronous Quality Control weekly meeting https://indico.cern.ch/category/8532/ (optional)



o2-analysis-ud-sgcand-producer

Code file: SGCandProducer.cxx

- o2::aod::UDCollisions
- o2::aod::SGCollisions
- o2::aod::UDCollisionsSels
- o2::aod::UDCollsLabels
- o2::aod::UDTracks
- o2::aod::UDTracksCov
- o2::aod::UDTracksPID
- o2::aod::UDTracksExtra
- o2::aod::UDTracksDCA
- o2::aod::UDTracksFlags
- o2::aod::UDTracksLabels
- o2::aod::UDFwdTracks
- o2::aod::UDZdcs
- o2::aod::UDZdcsReduced

o2-analysis-ud-upccand-producer

Code file: UPCCandidateProducer.cxx

- o2::aod::UDMcCollisions
- o2::aod::UDMcParticles
- o2::aod::UDCollisions
- o2::aod::UDCollisionsSels
- o2::aod::UDCollisionsSelsCent
- o2::aod::UDCollisionsSelsFwd

Ingredients

Data sample for this tutorial

- SG derive data for midrapidity analysis :
  - Download a latest data file from Hyperloop train 274917.
    alien_cp alien:///alice/cern.ch/user/a/alihyperloop/jobs/0077/hy_777928/AOD/001/AO2D.root file://.
- UPC derive data for forward rapidity analysis:
  - Download a latest data file from Hyperloop train 274418
    alien_cp alien:///alice/cern.ch/user/a/alihyperloop/outputs/0027/274418/42684/AO2D.root file://.
- Include the following tutorial tasks in the CMakeList.txt (alice/O2Physics/Tutorials/PWGUD) and compile*

1. https://github.com/AliceO2Group/O2Physics/blob/master/Tutorials/PWGUD/UDTutorial_05.cxx
2. https://github.com/AliceO2Group/O2Physics/blob/master/Tutorials/PWGUD/UDTutorial_06.cxx
3. https://github.com/AliceO2Group/O2Physics/blob/master/Tutorials/PWGUD/UDTutorial_07.cxx

- Also, all the materials are there in this cernbox folder, including a README !

https://cernbox.cern.ch/s/mnm0BKx9nKN3gFS

***Useful instruction of compiling only specific parts use ninja**
e.g. ninja Tutorials/PWGUD/install

Two pion invariant mass analysis: DG tables

- We learnt it last time during O2 analysis tutorial 3.0
- To refresh, checkout the task

/Users/anisa/Alice/O2Physics/Tutorials/PWGUD/UDTutorial_02b.cxx

```
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"
#include "Common/DataModel/PIDResponse.h"
#include "PWGUD/DataModel/UDTables.h"
#include "PWGUD/Core/UDHelpers.h"
#include "PWGUD/Core/DGPIDSelector.h"
using namespace o2;
using namespace o2::framework;
using namespace o2::framework::expressions;
struct UDTutorial02b {…..
using CCs = soa::Join<aod::UDCollisions, aod::SGCollisions,
aod::UDCollisionsSels, aod::UDMcCollsLabels>;
using TCs = soa::Join<aod::UDTracks, aod::UDTracksExtra,
aod::UDTracksFlags, aod::UDTracksPID, aod::UDMcTrackLabels>;
using CC = CCs::iterator;
using TC = TCs::iterator;
}
```

To include UD information

To join collisions and tracks tables

Two pion invariant mass analysis: SG tables

As we learnt from the introductory slides about the important features of SGCandidateProducer table maker

Here we will use those features to see exclusive two pions candidates !!

```
#include "PWGUD/DataModel/UDTables.h"
#include "PWGUD/Core/SGSelector.h"
#include "PWGUD/Core/SGTrackSelector.h"
struct UDTutorial05{
SGSelector sgSelector;
Configurable<float> FV0_cut{"FV0", 100., "FV0A threshold"};
Configurable<float> FT0A_cut{"FT0A", 100., "FT0A threshold"};
Configurable<float> FT0C_cut{"FT0C", 50., "FT0C threshold"};
Configurable<float> FDDA_cut{"FDDA", 10000., "FDDA threshold"};
Configurable<float> FDDC_cut{"FDDC", 10000., "FDDC threshold"};
Configurable<float> ZDC_cut{"ZDC", 10., "ZDC threshold"};
Configurable<float> gap_Side{"gap", 2, "gap selection”};
using UDCollisionsFull = soa::Join<aod::UDCollisions, aod::SGCollisions,
aod::UDCollisionsSels, aod::UDZdcsReduced>;
}
```

To include SG producer headers

To handle FIT information in configurable

Two pion invariant mass analysis: SG tables

```
using UDCollisionsFull = soa::Join<aod::UDCollisions,
aod::SGCollisions, aod::UDCollisionsSels, aod::UDZdcsReduced>;
using udtracksfull = soa::Join<aod::UDTracks, aod::UDTracksPID,
aod::UDTracksExtra, aod::UDTracksFlags, aod::UDTracksDCA>;
void process(UDCollisionsFull::iterator const& collision,
udtracksfull const& tracks){
int gapSide = collision.gapSide();
if(gapSide < 0 || gapSide > 2)return;
float FIT_cut[5] = {FV0_cut, FT0A_cut, FT0C_cut, FDDA_cut,
FDDC_cut};
int truegapSide = sgSelector.trueGap(collision, FIT_cut[0],
FIT_cut[1], FIT_cut[2], ZDC_cut);
gapSide = truegapSide;
if (gapSide == gap_Side) { rest of the analysis  }
}
```

Accessing GAP information

Accessing FIT information

Selecting Gap side

Two pion invariant mass analysis: SG tables

```
std::vector<TLorentzVector> allTracks;
std::vector<TLorentzVector> onlyPionTracks;
std::vector<float> onlyPionSigma;
std::vector<decltype(tracks.begin())> rawPionTracks;
TLorentzVector p;

for (auto t : tracks) {
if (!trackselector(t, parameters)) continue;
TLorentzVector a;
a.SetXYZM(t.px(), t.py(), t.pz(), o2::constants::physics::MassPionCharged);
allTracks.push_back(a);
auto nSigmaPi = t.tpcNSigmaPi();
if (fabs(nSigmaPi) < PID_cut) {
onlyPionTracks.push_back(a);
onlyPionSigma.push_back(nSigmaPi);
rawPionTracks.push_back(t);}
      }

for (auto pion : onlyPionTracks) {
        p += pion;
      }

if (collision.numContrib() == 2) {
if ((rawPionTracks.size() == 2) && (onlyPionTracks.size() == 2)) {
          }
```

Create Lorentz vector to store important information

Track loop

Inside track loop create pion tracks and apply TPC PID

Outside in a separate loop add all the only pion tracks

Now select collisions with 2 PV tracks

Selecting good 2 PV tracks

Two pion invariant mass analysis: SG tables

Viola!

Now we can just fill invariant mass and pt

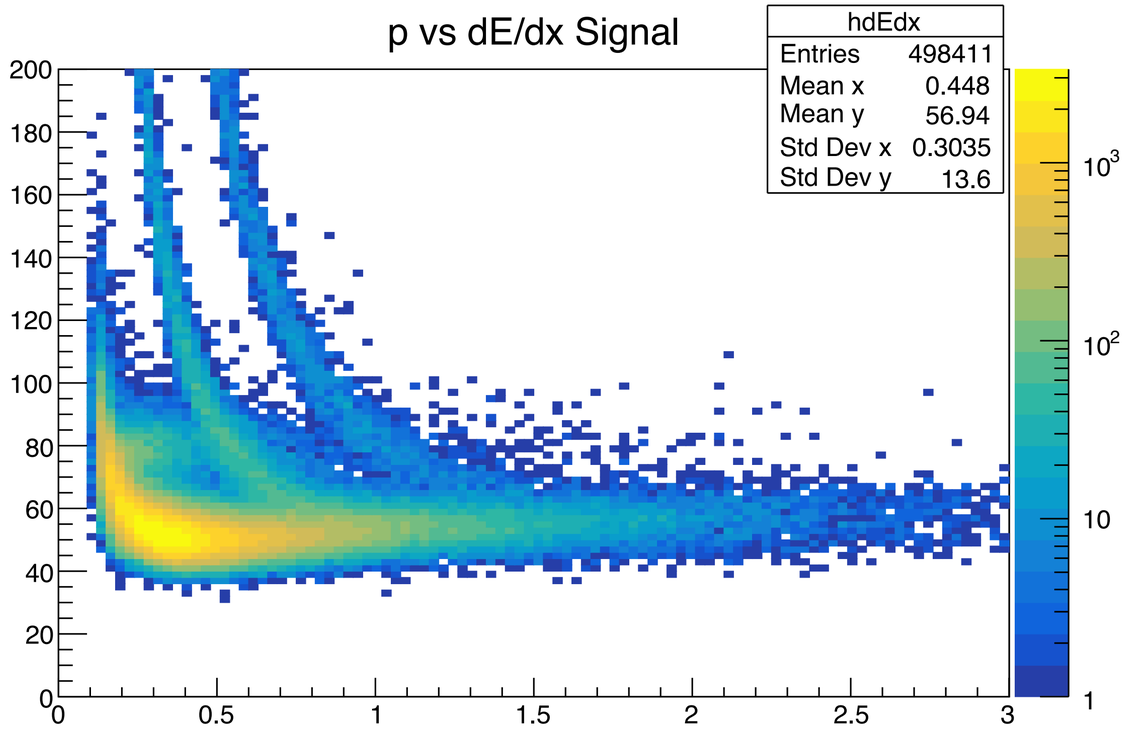
```
registry.fill(HIST("TwoPion/hMassUnlike"), p.M());
registry.fill(HIST("TwoPion/hPt"), p.Pt());
```

But we need to first check three important criteria

Two good tracks are with TPC pion sigma limit

```
if ((onlyPionSigma[0] * onlyPionSigma[0] +
onlyPionSigma[1] * onlyPionSigma[1]) > (PID_cut *
PID_cut))return;
```

Mid rapidity acceptance




```
if ((p.Rapidity() < -Rap_cut) || (p.Rapidity() > Rap_cut)) return;
```

Two opposite sign tracks

```
if (rawPionTracks[0].sign() != rawPionTracks[1].sign()) {}
```

Two pion invariant mass analysis: SG tables

Let's try some control plots

Now we can fill TPC signal information for each track

```
registry.fill(HIST("TwoPion/hTPCsig"),
rawPionTracks[0].tpcSignal(), rawPionTracks[1].tpcSignal());
```

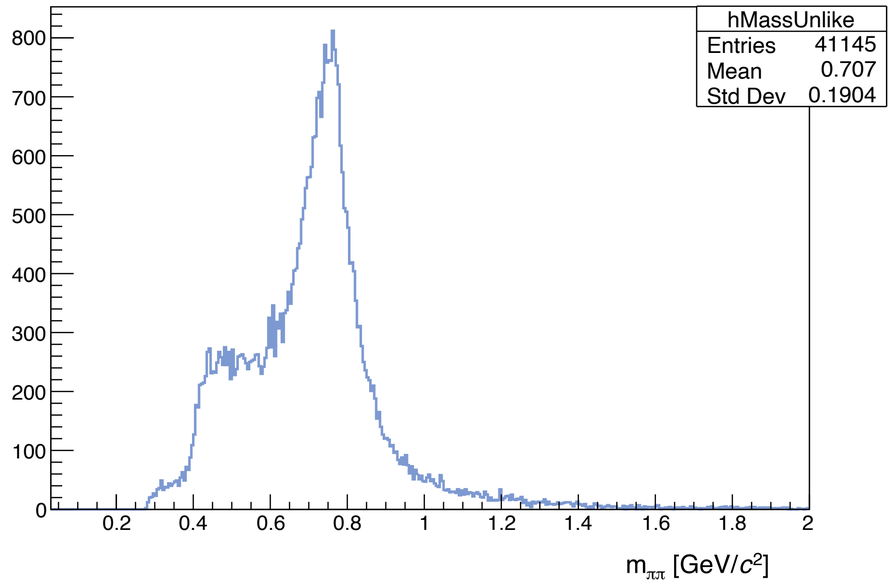
Track pt and track eta and…

```
registry.fill(HIST("TwoPion/hEta_t1"), onlyPionTracks[0].Eta());
```

```
registry.fill(HIST("TwoPion/hPtsingle_track2"),
onlyPionTracks[1].Pt());
```

Finally coherent ρ^0 candidates

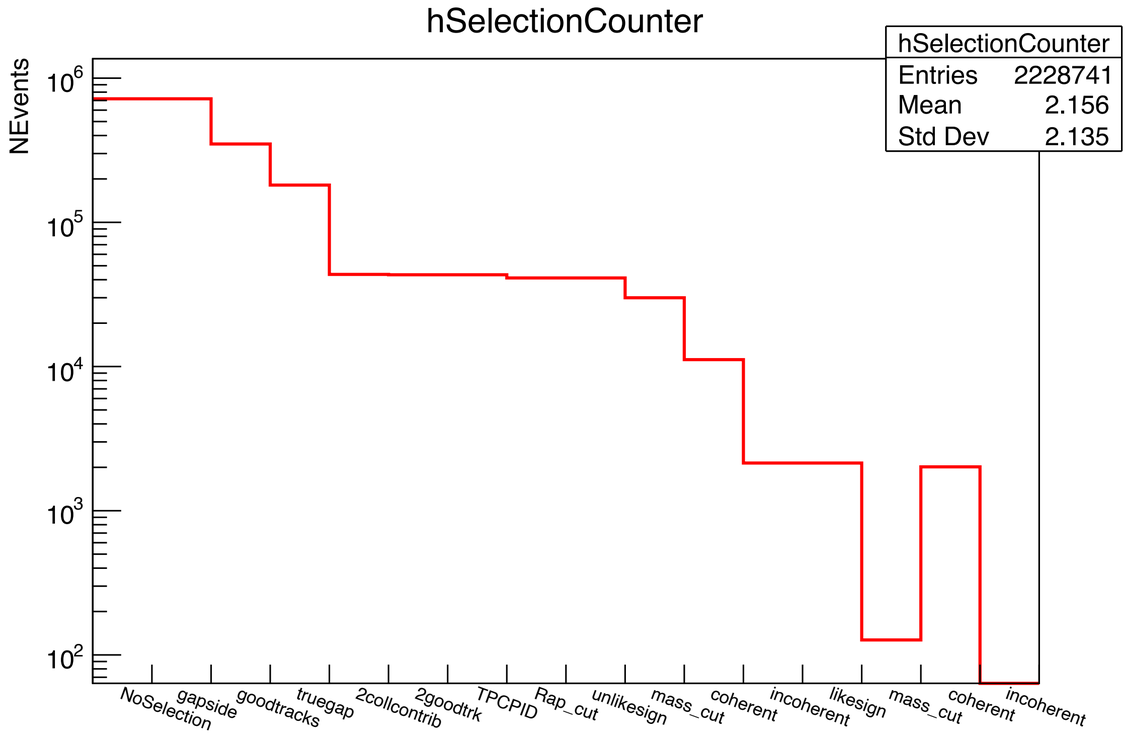
```
if (p.Pt() < Pt_coherent){
registry.fill(HIST("TwoPion/Coherent/hMassUnlikeCoherent"), p.M());
}
```



In the end the complete task should look like UDTutorial_05.cxx

```
o2-analysis-ud-udtutorial-05 --configuration json://udtutorial05.json --aod-file AO2D_SG_latest.root
```

Two pion invariant mass analysis: SG tables



Input file

```
O2-analysis-ud-udtutorial-05 -b --configuration://udtutorial05.json -aod-file AO2D_SG_latest.root
```

Make sure to provide proper path, O2 does not take path to the input file directly, e.g. Input/AO2D_SG_latest.root ✘

In that case we can create a txt file with path like input.txt !

Two muon invariant mass analysis: Forward rapidity

```
// O2Physics headers
#include "PWGUD/DataModel/UDTables.h"
#include "CCDB/BasicCCDBManager.h"
#include "DataFormatsParameters/GRPLHCIFData.h"
#include "DataFormatsParameters/GRPECSObject.h"

// ROOT headers
#include "TSystem.h"
#include "TDatabasePDG.h"
#include "TLorentzVector.h"
#include "TLorentzVector.h"
#include “TMath.h”

struct UDTutorial06 {

void init(InitContext const&)
    {Define histograms}

using UDCollisionsFwd = soa::Join<o2::aod::UDCollisions,o2::aod::UDCollisionsSels,
o2::aod::UDCollisionsSelsFwd>;
using fwdtraks = soa::Join<aod::UDFwdTracks,aod::UDFwdTracksExtra>;
```

Two muon invariant mass analysis: Forward rapidity

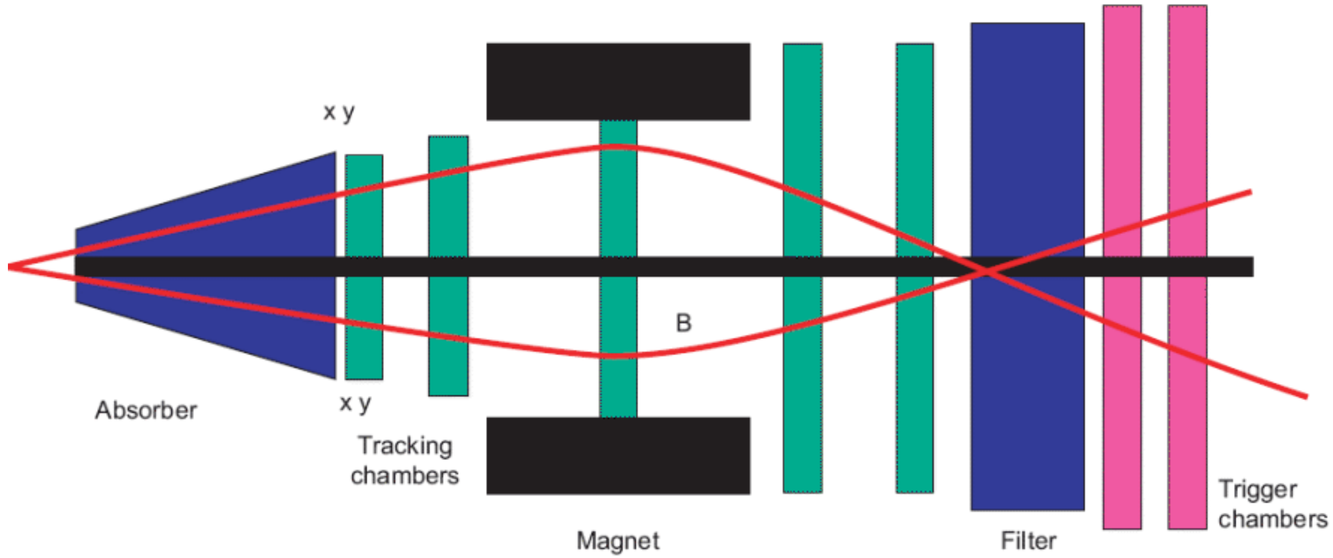
```
Inside struct,
template <typename TTrack1, typename TTrack2>
void processCandidate(UDCollisionsFwd::iterator const& collision, TTrack1& tr1, TTrack2& tr2)
 {
   // Vetoing V0A selection
            const auto& ampsV0A = collision.amplitudesV0A();
            const auto& ampsRelBCsV0A = collision.ampRelBCsV0A();
            for (unsigned int i = 0; i < ampsV0A.size(); ++i) {
              if (std::abs(ampsRelBCsV0A[i]) <= 1) {
                if (ampsV0A[i] > 100.)
                  return;
              }
            }

Similarly veto T0A selection
```

Muon arm is situated on the C side
Veto is applied on A side

The veto can be applied to either FV0A or FT0A or both
Veto is applied on current BC relative to FV0A or FT0A

Two muon invariant mass analysis: Forward rapidity

- $-4.0 < \eta < -2.5$ for each muon
- pT > 0 GeV/c
- 17.6 < Rabs< 89.5 (cm) for each muon
- Both μ^+ & μ^- matching the trigger, now MCH-MID matching
- $-4.0 < y < -2.5$ on dimuon pair.
- Unlike-sign dimuon pair.




```
if (tr1.rAtAbsorberEnd() < 17.6 || tr1.rAtAbsorberEnd() > 89.5) return;
if ((tr1.chi2MatchMCHMID() < 0) || (tr2.chi2MatchMCHMID() < 0)) return;
bool isUnlikeSign = (tr1.sign()+tr2.sign())==0;
if (p1.Eta() < -4.0 || p1.Eta() > -2.5) return;
if ((p1.Pt() < 0.0) || (p2.Pt() < 0.0)) return;
if ((p.Rapidity() < -4.0) || (p.Rapidity() > -2.5)) return;
```

After applying all the selections one can fill the invariant mass histogram!

Two muon invariant mass analysis: Forward rapidity

//Template that collects all collision IDs and track per collision

```
template <typename TTracks>
void collectCandIDs(std::unordered_map<int32_t, std::vector<int32_t>>& tracksPerCand, TTracks& tracks){
for (const auto& tr : tracks) {
int32_t candId = tr.udCollisionId();
if (candId < 0) {continue;}
tracksPerCand[candId].push_back(tr.globalIndex());
      }
 }
```

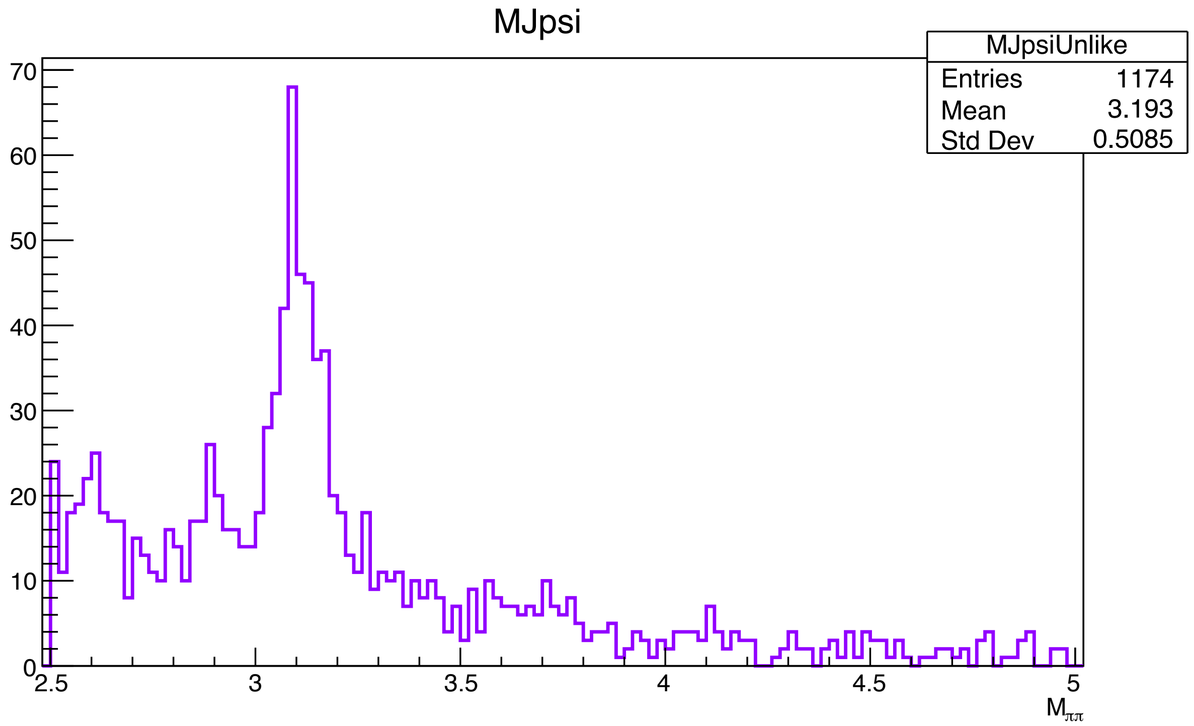
//Finally run the main process: candidates with two forward tracks associated with collision

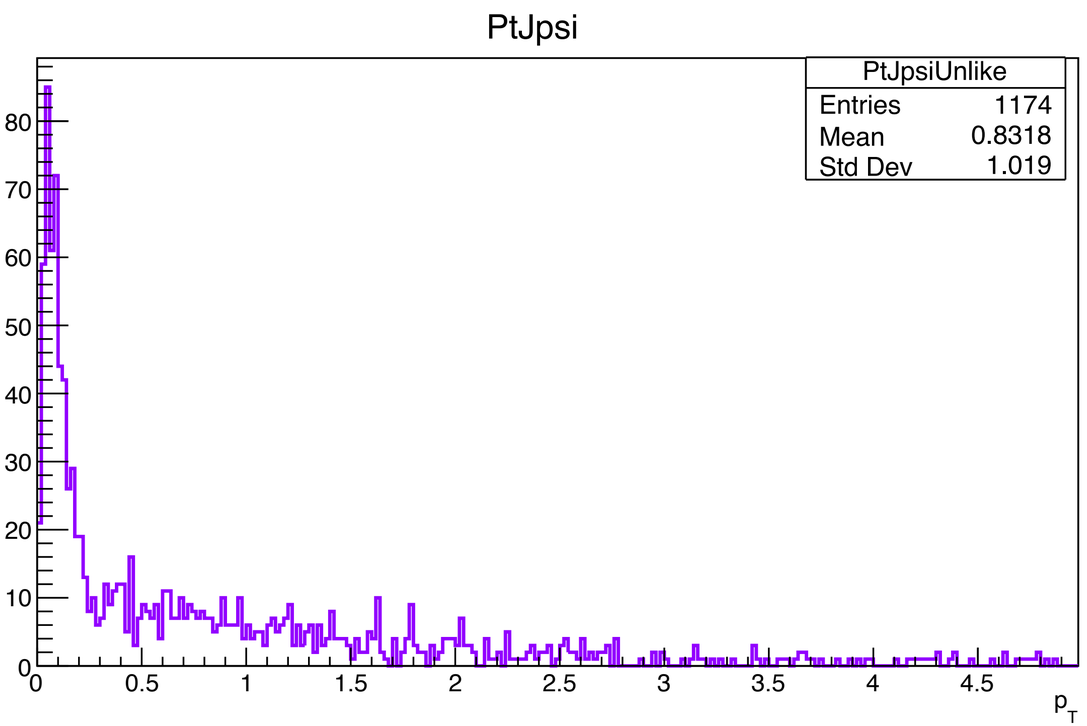
```
void process(UDCollisionsFwd const& eventCandidates,fwdtraks const& fwdTracks){
std::unordered_map<int32_t, std::vector<int32_t>> tracksPerCand;
collectCandIDs(tracksPerCand, fwdTracks);
// assuming that candidates have exatly 2 forward tracks
for (const auto& item : tracksPerCand) {
int32_t trId1 = item.second[0];
int32_t trId2 = item.second[1];
int32_t candID = item.first;
const auto& collision = eventCandidates.iteratorAt(candID);
const auto& tr1 = fwdTracks.iteratorAt(trId1);
const auto& tr2 = fwdTracks.iteratorAt(trId2);
processCandidate(collision, tr1, tr2);
      }
    }
```

To run the task

```
o2-analysis-ud-udtutorial-06 --aod-file @input.txt
```

Two muon invariant mass analysis: Forward rapidity





To run the task

```
o2-analysis-ud-udtutorial-06 --aod-file @input.txt
```

Event by event analysis: derived tree

Now we will try different approach of saving outputs
Instead of histograms, let's use AOD tree!
For this exercise, we use the same task of analysing two pion invariant mass (UDTutorial_05.cxx) and modify it

```
#include <TTree.h>
```

Don't forget to include required headers!

```
namespace o2::aod{
namespace tree{
// track tables
DECLARE_SOA_COLUMN(PT, Pt, float);
DECLARE_SOA_COLUMN(RAP, rap, float);
DECLARE_SOA_COLUMN(PHI, Phi, float);
DECLARE_SOA_COLUMN(MASS, mass, float);
} // namespace tree
DECLARE_SOA_TABLE(TREE, "AOD", "Tree",
tree::PT,
tree::RAP,
tree::PHI,
tree::MASS);
} // namespace o2::aod
```

Now, let's declare some columns and trees

This part should go before the struct

Create trees only for essential parameters, we have to keep output size in check!
Output size < 1GB for Hyperloop rule!

Event by event analysis: derived tree

Beginning of struct

```
struct UDTutorial07 {
  SGSelector sgSelector;
  Produces<aod::TREE> tree;
```

Rest of the configurable should be same as UDTutorial_05.cxx
Remove all the histograms, one can keep one or two histograms to cross-check the values between two tasks

```
void process(UDCollisionsFull::iterator const& collision, udtracksfull const& tracks)
  {
    std::vector<float> trackpt;
    std::vector<float> tracketa;
    std::vector<float> trackphi;
    std::vector<float> tracksign;
    std::vector<float> pitpcpid;
```

In addition to already defined Lorentz vectors, we add some more to store raw track information

```
if (gapSide == gap_Side) {
```

```
for (auto t : tracks) {
    same as UDTutorial_05.cxx
}
```

Track loop, same as before

Event by event analysis: derived tree

```
// Selecting collisions with Two PV contributors
if (collision.numContrib() == 2) {
/ Selecting only Two good tracks
if ((rawPionTracks.size() == 2) && (allTracks.size() == 2)) {

// Creating rhos
for (auto pion : onlyPionTracks) {
  p += pion;
  }

for (auto rtrk : rawPionTracks) {
TLorentzVector itrk;
itrk.SetXYZM(rtrk.px(), rtrk.py(), rtrk.pz(), o2::constants::physics::MassPionCharged);
trackpt.push_back(itrk.Pt());
tracketa.push_back(itrk.Eta());
trackphi.push_back(itrk.Phi());
tracksign.push_back(rtrk.sign());
pitpcpid.push_back(rtrk.tpcNSigmaPi());
}
```

Same as before

If one need to store raw track information to create Rho(0) mesons later!

Event by event analysis: derived tree

```
int sign = 0;
for (auto rawPion : rawPionTracks) {
sign += rawPion.sign();
 }
```

Storing charge information

Finally, filling tree

```
tree(p.Pt(), p.Y(), p.Phi(), p.M(), sign, collision.numContrib(), pitpcpid, trackpt,
tracketa, trackphi, tracksign);
```

Close struct and make sure all the braces are closed in order!

To run the task UDTutorial_07.cxx, use similar concept as table maker

```
o2-analysis-ud-udtutorial-07 -b  --configuration json://udtutorial07.json --aod-writer-json
OutputDirector_mini.json --aod-file AO2D_SG_latest.root
```

Output container

Input file

Event by event analysis: derived tree

```
{ "OutputDirector": {
  "debug_mode": true,
  "resfile": "AO2D_res",
  "OutputDescriptors": [
   {
      "table": "AOD/Tree/0"
    }
      ],
  "ntfmerge": 100000000
   }
}
```

Define output container

We want only one tree!

```
o2-analysis-ud-udtutorial-07 -b  --configuration json://
udtutorial07.json --aod-writer-json OutputDirector_mini.json
--aod-file AO2D_SG_latest.root
```

This will produce AO2D_res.root, you can choose your own output name !

The output tree will contain multiple data frames (DF_*******)

Run AO2D merger to add them.

```
o2-aod-merger --input AOD_res.root --output AO2D_res_final.root --max-size 10000000000000000000
```

Now our output tree is ready for analysis!

Event by event analysis: derived tree

Now our output tree is ready for analysis! It looks like a simple ROOT tree!

An example task to analyse this output tree is given `read_ao2dtree.C`

After running it we will get invariant mass of two pion histograms

The output from this task should be comparable to the results from UDTutorial_05.cxx

Exercise :

1. Create invariant mass histogram for two pions using direct mass information from tree and using track eta, pt, phi and sign information
2. Compare invariant mass histograms between UDTutorial_05.cxx and UDTutorial_07.cxx

Take away

- We have learnt how to analysis data using UD tables within UPC framework!
- Both at mid and forward rapidity
- Restoring event by event information in a reduced output tree

For more help and later

O2AT PWG-UD: a channel for the UD hands-on - https://mattermost.web.cern.ch/alice/channels/o2at-pwg-ud

Happy physicsing!!