O2 Analysis Tutorial 4.0

2024.10.17

Flow with Generic Framework

Zhiyong Lu
China Institute of Atomic Energy

CONTENTS

CONTENTS

Generic Framework (GFW)

Ante Bilandzic, etc., Phys. Rev. C 89, 064904 (2014)
Zuzana Moravcova, etc., Phys. Rev. C 103, 024913 (2021)

- A framework to calculate multi-particle correlation in any harmonics

For m-particle correlation in harmonics n_1, n_2, ..., n_m, we have:

$$\langle m \rangle_{n_1,n_2,\ldots,n_m} \equiv \left\langle e^{i(n_1\varphi_{k_1}+n_2\varphi_{k_2}+\cdots+n_m\varphi_{k_m})} \right\rangle \equiv \frac{\sum\limits_{\substack{k_1,k_2,\ldots,k_m=1 \\ k_1\neq k_2\neq\ldots\neq k_m}}^{M} w_{k_1}w_{k_2}\cdots w_{k_m}\, e^{i(n_1\varphi_{k_1}+n_2\varphi_{k_2}+\cdots+n_m\varphi_{k_m})}}{\sum\limits_{\substack{k_1,k_2,\ldots,k_m=1 \\ k_1\neq k_2\neq\ldots\neq k_m}}^{M} w_{k_1}w_{k_2}\cdots w_{k_m}} .$$

$$\mathrm{N}\langle m \rangle_{n_1,n_2,\ldots,n_m} \equiv \sum_{\substack{k_1,k_2,\ldots,k_m=1 \\ k_1\neq k_2\neq\ldots\neq k_m}}^{M} w_{k_1}w_{k_2}\cdots w_{k_m}\, e^{i(n_1\varphi_{k_1}+n_2\varphi_{k_2}+\cdots+n_m\varphi_{k_m})},$$

$$\mathrm{D}\langle m \rangle_{n_1,n_2,\ldots,n_m} \equiv \sum_{\substack{k_1,k_2,\ldots,k_m=1 \\ k_1\neq k_2\neq\ldots\neq k_m}}^{M} w_{k_1}w_{k_2}\cdots w_{k_m} = \mathrm{N}\langle m \rangle_{0,0,\ldots,0} .$$

Denominator is the real part of numerator

The recursive algorithm to get all $\mathrm{N}\langle \mathrm{m} \rangle$:

Q-vector

$$Q_{n,p} \equiv \sum_{k=1}^{M} w_k^p\, e^{in\varphi_k} .$$

$\mathrm{N}\langle 1 \rangle'_{n_1}$: return $Q_{n_1,1}$
$\mathrm{N}\langle m \rangle'_{n_1,\ldots,n_m}$:
    $C \leftarrow 0$
    for $k \leftarrow (m-1), 1$ do
        for each combination $c = \{c_1,\ldots,c_k\}$ of $\{n_1,\ldots,n_{m-1}\}$ do
            $q \leftarrow \sum_{j \text{ not in } c} n_j$
            $C \leftarrow C + (-1)^{m-k}\,(m-k-1)! \times \mathrm{N}\langle k \rangle'_{c_1,\ldots,c_k} \times Q_{q,m-k}$
        end for each c
    end for k
    return C .

Generic Framework (GFW)

Ante Bilandzic, etc., Phys. Rev. C 89, 064904 (2014)
Zuzana Moravcova, etc., Phys. Rev. C 103, 024913 (2021)

- A framework to calculate multi-particle correlation in any harmonics

$$
\begin{aligned}
\mathrm{N}\langle 1\rangle_{n_1} &= Q_{n_1,1}\,, \\
\mathrm{N}\langle 2\rangle_{n_1,n_2} &= \mathrm{N}\langle 1\rangle_{n_1} Q_{n_2,1} - Q_{n_1+n_2,2}\,, \\
\mathrm{N}\langle 3\rangle_{n_1,n_2,n_3} &= \mathrm{N}\langle 2\rangle_{n_1,n_2} Q_{n_3,1} - \mathrm{N}\langle 1\rangle_{n_1} Q_{n_2+n_3,2} - \mathrm{N}\langle 1\rangle_{n_2} Q_{n_1+n_3,2} + 2Q_{n_1+n_2+n_3,3}\,, \\
\mathrm{N}\langle 4\rangle_{n_1,n_2,n_3,n_4} &= \mathrm{N}\langle 3\rangle_{n_1,n_2,n_3} Q_{n_4,1} - \mathrm{N}\langle 2\rangle_{n_1,n_2} Q_{n_3+n_4,2} - \mathrm{N}\langle 2\rangle_{n_1,n_3} Q_{n_2+n_4,2} - \mathrm{N}\langle 2\rangle_{n_2,n_3} Q_{n_1+n_4,2} \\
&\quad + 2\mathrm{N}\langle 1\rangle_{n_1} Q_{n_2+n_3+n_4,3} + 2\mathrm{N}\langle 1\rangle_{n_2} Q_{n_1+n_3+n_4,3} + 2\mathrm{N}\langle 1\rangle_{n_3} Q_{n_1+n_2+n_4,3} - 6Q_{n_1+n_2+n_3+n_4,4}\,,
\end{aligned}
$$

For example:

two particle correlation (n = 2): $\langle 2\rangle_{2,-2} = \dfrac{\mathrm{N}\langle 2\rangle_{2,-2}}{\mathrm{N}\langle 2\rangle_{0,0}}$

four particle correlation (n = 2): $\langle 4\rangle_{2,2,-2,-2} = \dfrac{\mathrm{N}\langle 4\rangle_{2,2,-2,-2}}{\mathrm{N}\langle 4\rangle_{0,0,0,0}}$

Initialisation of Generic Framework (GFW)

The class definitions are in O2Physics/PWGCF/GenericFramework/Core/

- First, you should include the following header files to use GFW:

```
#include "GFWPowerArray.h"
#include "GFW.h"
#include "GFWCumulant.h"
```

- Then, define your GFW object:

```
GFW* fGFW = new GFW();
std::vector<GFW::CorrConfig> corrconfigs;
```

- Specify your desired η regions and correlations in init(InitContext const&):

```
//                         name,   η_min,  η_max, p_T bins, bitmask
fGFW->AddRegion("full", -0.8,  0.8,  1,        1);
//                                                  η regions,  harmonics,   name,        p_T diff flag
corrconfigs.push_back(fGFW->GetCorrelatorConfig("full    {2 -2}",   "ChFull22",   kFALSE));
fGFW->CreateRegions(); // Finalize the initialisation
```

$\langle 2 \rangle_{2,-2}$ in full η region

documentation: https://github.com/vvislavi/GFW_Core

Add η regions and correlation configs

- correlation configurations: help you retrive m-particle in a simple string

general syntax in GetCorrelatorConfig:

η regions, {harmonics}

“[POI] Ref [| Overlap] {harm1[,harm2,…,harmN]}”

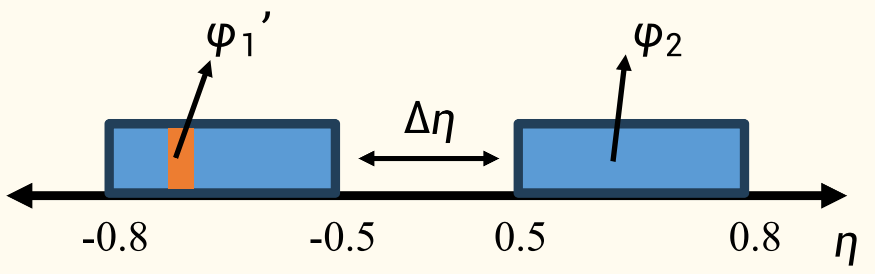
For example in more cases
Firstly define η regions:

```
fGFW->AddRegion("full", -0.8,  0.8,  1,  1);
fGFW->AddRegion("refN", -0.8, -0.5, 1, 1);
fGFW->AddRegion("refP", 0.5, 0.8, 1, 1);
fGFW->AddRegion("poiN", -0.8, -0.4, 1 + fPtAxis->GetNbins(), 2);
fGFW->AddRegion("olN", -0.8, -0.4, 1, 4);
```

poi and overlap are filled in differet bitmask

More correlations:

"full {2 -2}"	: $\langle 2\rangle_{2,-2}$ in full η region
"full {2 2 -2 -2}"	: $\langle 4\rangle_{2,2,-2,-2}$ in full η region
"refN {2} refP {-2}"	: $\langle 2\rangle_{2,-2}$ with |Δη|
"refN {2 2} refP {-2 -2}"	: $\langle 4\rangle_{2,2,-2-2}$ with |Δη|
"poiN refN | olN {2} refP {-2}"	: $\langle 2\rangle_{2,-2}$ in p_T differential



documentation: https://github.com/vvislavi/GFW_Core

Fill GFW in track loop

- After initialisation, we should fill GFW during track loop:

```
for (auto& track : tracks) {
//                                              pT bin                         NUA*NUE,   bitmask
      fGFW->Fill(track.eta(), 1, track.phi(), wacc * weff, 1); // this fills all the correlations with bitmask 1
    }
```

- if you have p_T differential correlations, then:

```
bool WithinPtPOI = (cfgCutPtPOIMin < track.pt()) && (track.pt() < cfgCutPtPOIMax); // within POI pT range
bool WithinPtRef = (cfgCutPtRefMin < track.pt()) && (track.pt() < cfgCutPtRefMax); // within RF pT range
if (WithinPtRef)
    fGFW->Fill(track.eta(), fPtAxis->FindBin(track.pt()) - 1, track.phi(), wacc * weff, 1); // bitmask 1 for ref
if (WithinPtPOI)
    fGFW->Fill(track.eta(), fPtAxis->FindBin(track.pt()) - 1, track.phi(), wacc * weff, 2); // bitmask 2 for poi
if (WithinPtPOI && WithinPtRef)
    fGFW->Fill(track.eta(), fPtAxis->FindBin(track.pt()) - 1, track.phi(), wacc * weff, 4); // bitmask 4 for overlap
```

documentation: https://github.com/vvislavi/GFW_Core

Fill histogram with correlation

- First, add a TProfile in initialisation:

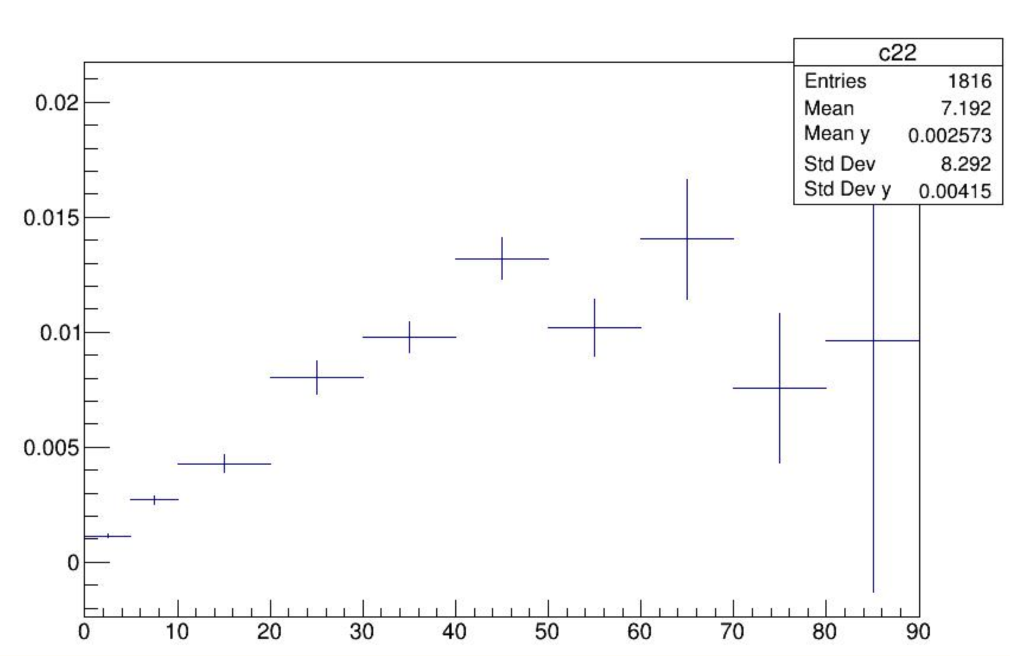
```
registry.add("c22", "", {HistType::kTProfile, {axisMultiplicity}});
```

- for every event, fill the histogram after track loop:

```
FillProfile(corrconfigs.at(0), HIST("c22"), cent);
```

```
template <char... chars>
void FillProfile(const GFW::CorrConfig& corrconf, const ConstStr<chars...>& tarName, const double& cent)
{
  double dnx, val;
  dnx = fGFW->Calculate(corrconf, 0, kTRUE).real();
  if (dnx == 0)
    return;
  if (!corrconf.pTDif) {
    val = fGFW->Calculate(corrconf, 0, kFALSE).real() / dnx;
    if (TMath::Abs(val) < 1)
      registry.fill(tarName, cent, val, dnx);
    return;
  }
  return;
}
```

$$\frac{\mathrm{N}\langle m\rangle_{n1,n2,\ldots,nm}}{\mathrm{N}\langle m\rangle_{0,0,\ldots,0}}$$



histogram "c22" is $\langle\langle 2\rangle\rangle_{2,-2}$

Then we can get $v_2\{2\}$ in local analysis

documentation: https://github.com/vvislavi/GFW_Core

Example code

O2Physics/Tutorials/PWGCF/FlowGenericFramework/src/flowGFWTutorial.cxx

```
// define global variables
GFW* fGFW = new GFW();
std::vector<GFW::CorrConfig> corrconfigs;

using aodCollisions = soa::Filtered<soa::Join<aod::Collisions, aod::EvSels, aod::CentRun2V0Ms>>;
using aodTracks = soa::Filtered<soa::Join<aod::Tracks, aod::TrackSelection, aod::TracksExtra>>;

void init(InitContext const&)
{
  ccdb->setURL(url.value);
  ccdb->setCaching(true);
  ccdb->setCreatedNotAfter(nolaterthan.value);

  // Add some output objects to the histogram registry
  registry.add("hPhi", "", {HistType::kTH1D, {axisPhi}});
  registry.add("hEta", "", {HistType::kTH1D, {axisEta}});
  registry.add("hVtxZ", "", {HistType::kTH1D, {axisVertex}});
  registry.add("hMult", "", {HistType::kTH1D, {{3000, 0.5, 3000.5}}});
  registry.add("hCent", "", {HistType::kTH1D, {{90, 0, 90}}});
  registry.add("c22", "", {HistType::kTProfile, {axisMultiplicity}});

  fGFW->AddRegion("full", -0.8, 0.8, 1, 1);
  corrconfigs.push_back(fGFW->GetCorrelatorConfig("full {2 -2}", "ChFull22", kFALSE));
  fGFW->CreateRegions();
}
```

Initialisation

Example code

O2Physics/Tutorials/PWGCF/FlowGenericFramework/src/flowGFWTutorial.cxx

```
void process(aodCollisions::iterator const& collision, aod::BCsWithTimestamps const&, aodTracks const& tracks)
{
  int Ntot = tracks.size();
  if (Ntot < 1)
    return;
  if (!collision.sel7())
    return;
  float vtxz = collision.posZ();
  registry.fill(HIST("hVtxZ"), vtxz);
  registry.fill(HIST("hMult"), Ntot);
  registry.fill(HIST("hCent"), collision.centRun2V0M());
  fGFW->Clear();
  const auto cent = collision.centRun2V0M();
  float weff = 1, wacc = 1;
  for (auto& track : tracks) {
    registry.fill(HIST("hPhi"), track.phi());
    registry.fill(HIST("hEta"), track.eta());

    fGFW->Fill(track.eta(), 1, track.phi(), wacc * weff, 1);
  }

  // Filling c22 with ROOT TProfile
  FillProfile(corrconfigs.at(0), HIST("c22"), cent);
}
```

event loop and track loop

Running your task locally

The provided scripts in indico will take care of downloading and running the tasks with the correct configurations (Thank Emil for providing the script!)

Download AO2D data files with the downloadAOD.sh script:

```
$> sh downloadAOD.sh
```

run the task:

```
$> sh run.sh
```

The tutorial code was written for Run2 dataset, to run on Run3 dataset, you need to make some changes and rebuild.
Change collision table, change centrality estimator, remove Run2 trigger

```
using aodCollisions = soa::Filtered<soa::Join<aod::Collisions, aod::EvSels, aod::CentRun2V0Ms>>; // Run2
using aodCollisions = soa::Filtered<soa::Join<aod::Collisions, aod::EvSels, aod::CentFT0Cs>>; //Run3
```

Download AO2D data files with the downloadAOD.sh script:

```
$> sh downloadAOD.sh run3
```

run the task:

```
$> sh run.sh run3
```

More useful classes

In O2Physics/PWGCF/GenericFramework/Core, more useful classes are also provided to help you get results easily.

FlowContainer: container for regular correlation, cumulant calculation
FlowPtContainer: container for m-particle pT correlation
GFWWeights: production and extraction of NUA weight

For realistic usage, please refer to the working code in:
O2Physics/PWGCF/Flow/Tasks/FlowTask.cxx Developed by me
O2Physics/PWGCF/GenericFramework/Tasks/flowGenericFramework.cxx Developed by Emil

FlowContainer

container for regular correlation, cumulant calculation

- Header:

```
#include "FlowContainer.h"
```

- Declare:

```
OutputObj<FlowContainer> fFC{FlowContainer("FlowContainer")};
```

- Initialisation:

```
TObjArray* oba = new TObjArray();
oba->Add(new TNamed("ChGap22", "ChGap22"));
for(Int_t i=0;i<fPtAxis->GetNbins();i++)
  oba->Add(new TNamed(Form("ChGap22_pt_%i",i+1),"ChGap22_pTDiff"));
fFC->SetName("FlowContainer");
fFC->SetXAxis(fPtAxis);  // pT differential X-axis
fFC->Initialize(oba, axisMultiplicity, cfgNbootstrap);
delete oba;        // Centrality/multiplicity X-axis, Number of subsamples
```

ObjArray holds each correlator,
which will be joined together in a TProfile2D

- Fill FlowContainer in event loop

```
for (uint l_ind = 0; l_ind < corrconfigs.size(); l_ind++) {
//                     correlation,    centrality, random number
    FillFC(corrconfigs.at(l_ind), cent, l_Random);
}
```

```
void FillFC(const GFW::CorrConfig& corrconf, const double& cent, const double& rndm)
{
  double dnx, val;
  dnx = fGFW->Calculate(corrconf, 0, kTRUE).real();
  if (dnx == 0)
    return;
  if (!corrconf.pTDif) {
    val = fGFW->Calculate(corrconf, 0, kFALSE).real() / dnx;
    if (TMath::Abs(val) < 1)
      fFC->FillProfile(corrconf.Head.c_str(), cent, val, dnx, rndm);
    return;
  }
  for (Int_t i = 1; i <= fPtAxis->GetNbins(); i++) {
    dnx = fGFW->Calculate(corrconf, i - 1, kTRUE).real();
    if (dnx == 0)
      continue;
    val = fGFW->Calculate(corrconf, i - 1, kFALSE).real() / dnx;
    if (TMath::Abs(val) < 1)
      fFC->FillProfile(Form("%s_pt_%i", corrconf.Head.c_str(), i), cent, val, dnx, rndm);
  }
  return;
}
```

FlowContainer

After receiving the analysis output file

- Fetch FlowContainer:

```
FlowContainer* fc = (FlowContainer*)file->Get("your-task/FlowContainer");
```

- Set ID:

```
fc->SetIDName("ChGap");
```

- If not using bootstrap errors, set propagation of errors:

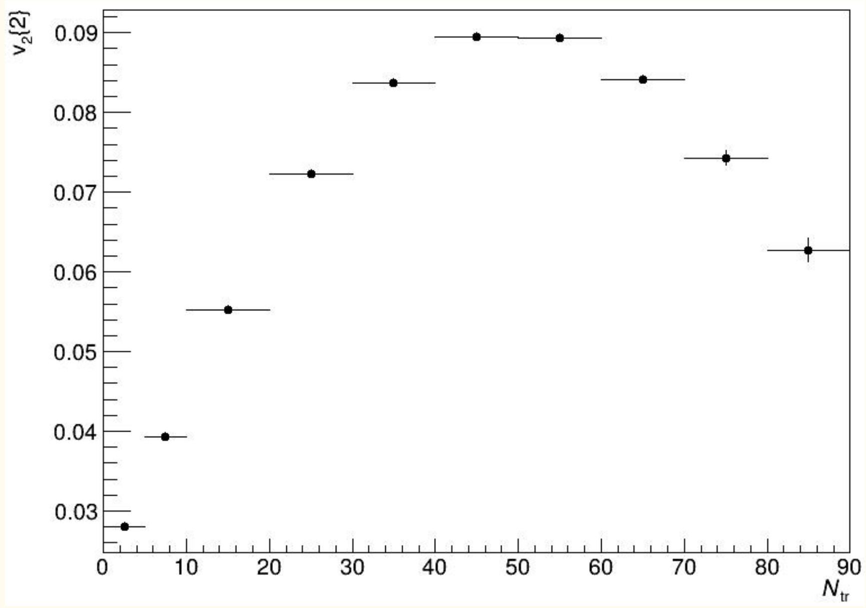
```
fc->SetPropagateErrors(kTRUE);
```

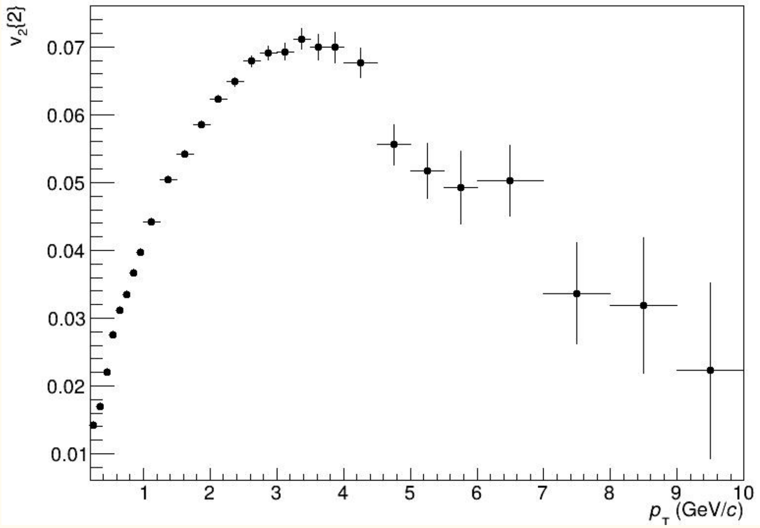
- Get $v_2\{2\}$:

```
TH1D* hv22 = fc->GetVN2VsMulti(2); // this function fetch object "IDName+22" and calculate v22
// so actually it require you to give the right object name in your code
```

- Get $v_2(p_T)$:

```
TH1D* hv22 = fc->GetVN2VsPt(2, 0, 4.9); // v2(pT) in 0~5% centrality
```





Exercises

```
cd ~/alice/sw/BUILD/O2Physics-latest/O2Physics
ninja install Tutorials/PWGCF/all
```

or

```
cd ~/alice/
aliBuild build O2Physics
```

modify the flowGFWTutorial.cxx locally
remember to rebuild o2physics after your changes:

- 1. Add more correlations: $\langle 2\rangle_{3,-3}$, $\langle 2\rangle_{4,-4}$, $\langle 4\rangle_{2,2,-2,-2}$
- 2. Add more η regions: smaller or larger $|\Delta\eta|$
- 3. Add p_T-differential v_2.
- 4. Modify the code to run on Run3 dataset (take a look at slide 11)
- 5. Add PID flow
- 6. Use FlowContainer to store correlations
- 7. Use GFWWeight to produce and apply NUA corrections

solution 1-6 can be found in: https://github.com/AliceO2Group/analysis-tutorials/tree/master/o2at-2/PWGCF/GenericFramework/src


THANKS

solution 7

Produce NUA

Declare GFWWeight object:

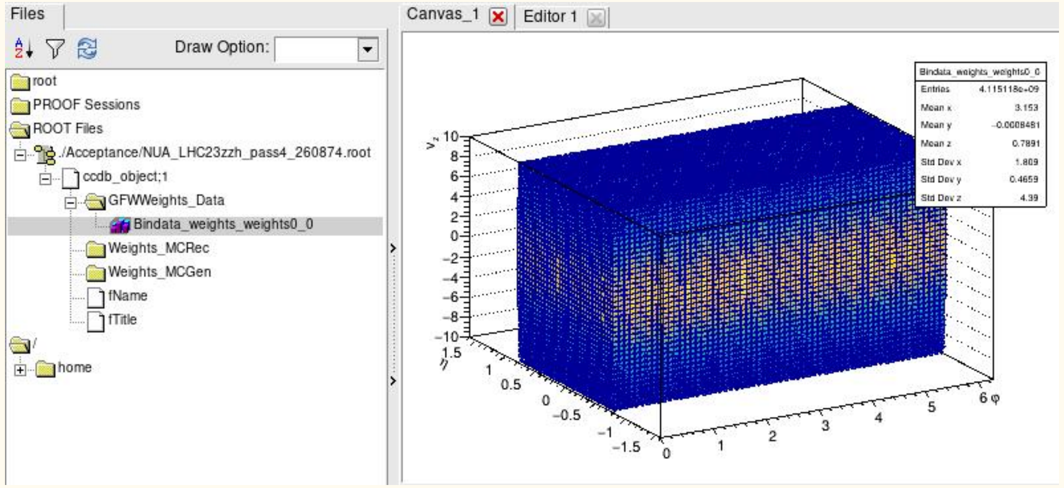
```
OutputObj<GFWWeights> fWeights{GFWWeights("weights")};
```

Init pT axis for GFWWeight:

```
o2::framework::AxisSpec axis = axisPt;
int nPtBins = axis.binEdges.size() - 1;
double* PtBins = &(axis.binEdges)[0];
fPtAxis = new TAxis(nPtBins, PtBins);
fWeights->SetPtBins(nPtBins, PtBins);
fWeights->Init(true, false);
```

Fill GFWWeight during track loop:

```
fWeights->Fill(track.phi(), track.eta(), vtxz, track.pt(), cent, 0);
```



The file you upload should look like this

After you get the output file, you can fetch your GFWWeight object and upload it to ccdb

solution 7

Apply NUA

set configurable and GFWWeight object to receive NUA file:

```
O2_DEFINE_CONFIGURABLE(cfgAcceptance, std::string, "", "CCDB path to acceptance object")
GFWWeights* mAcceptance = nullptr;
```

Load NUA file:

```
auto bc = collision.bc_as<aod::BCsWithTimestamps>();
mAcceptance = ccdb->getForTimeStamp<GFWWeights>(cfgAcceptance, bc.timestamp());
if (mAcceptance)
   LOGF(info, "Loaded acceptance weights from %s (%p)", cfgAcceptance.value.c_str(), (void*)mAcceptance);
else
   LOGF(warning, "Could not load acceptance weights from %s (%p)", cfgAcceptance.value.c_str(), (void*)mAcceptance);
```

Get NUA weight

```
wacc = mAcceptance->GetNUA(phi, eta, vtxz);
```

Then you can use the weight when filling the GFW object