Phys_f_416:
A_{FB} asymmetry measurement with the CMS detector (III)

Today's lesson

Target:

- Measure A_{FB} in simultated data (simulation) and data

- **Simulation:**
  /ice3/phy_f_416_2016/samples/DYM20.root
  Drell-Yan sample with a mass cut at 20 GeV (the sharp cut I showed you last time)
- **Data:**
  /ice3/phy_f_416_2016/samples/data_13TeV.root
  Data correspond to **real data** taken by the **CMS** experiment during all **2015**, for a total integrated luminosity of **L=2.7/fb**

ssh login to your account

Login:
ssh -Y yourname@lxpub.iihe.ac.be
Password is: xxxxx
Now you pc is just a monitor: you are in fact using another remote pc!

SET ROOT:
source setter.sh

A_{FB} asymmetry

The total cross section is:

$$\sigma = \int_\Omega \frac{d\sigma_{\gamma+Z}}{d\Omega} d\Omega = \frac{4\pi}{3}\frac{\alpha^2}{s'} c_1$$

where $\alpha = e^2/(4\pi)$.

$\sigma_F = \sigma_{\theta<\pi/2}$ and $\sigma_B = \sigma_{\theta>\pi/2}$ $\longrightarrow$ $A_{FB} = \dfrac{\sigma_F - \sigma_B}{\sigma_F + \sigma_B} = \dfrac{3}{8}\dfrac{c_2}{c_1}$

$$\boxed{A_{FB} = \frac{3}{8}\frac{c_2}{c_1}}$$

- A_{FB} depends on s' via R!

$$c_1 = 1 + 2\operatorname{Re}(R) g_{Vl} g_{Vq} + |R|^2 \left(g_{Vl}^2 + g_{Al}^2\right)\left(g_{Vq}^2 + g_{Aq}^2\right)$$

$$c_2 = 4\operatorname{Re}(R) g_{Al} g_{Aq} + 8|R|^2 g_{Vl} g_{Al} g_{Vq} g_{Aq}$$

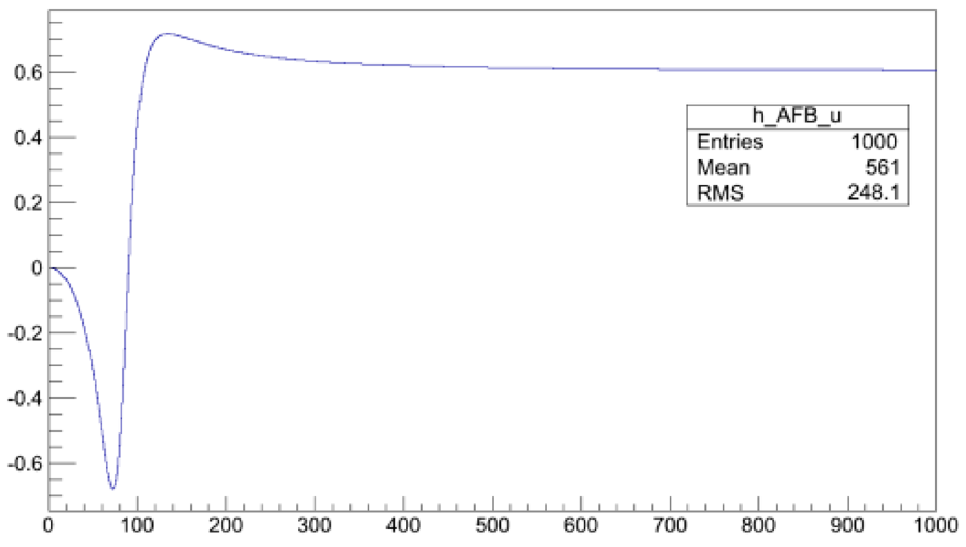
$$R = \frac{1}{Q_l Q_q \sin^2 2\theta_W} \frac{s'}{s' - M_Z^2 + is' \Gamma_Z / M_Z}$$

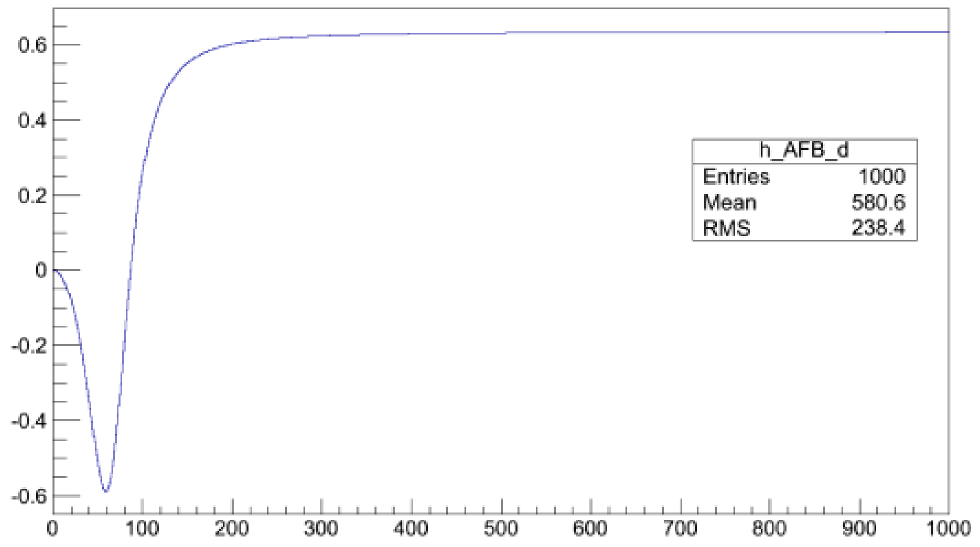
$$g_{Al,q} = -I^3_{Wl,q}$$

$$g_{Vl,q} = I^3_{Wl,q} - 2Q_{l,q} \sin^2\theta_W$$

	Q	I^3_W
e	-1	-1/2
u	2/3	1/2
d	-1/3	-1/2

A_{FB} predictions



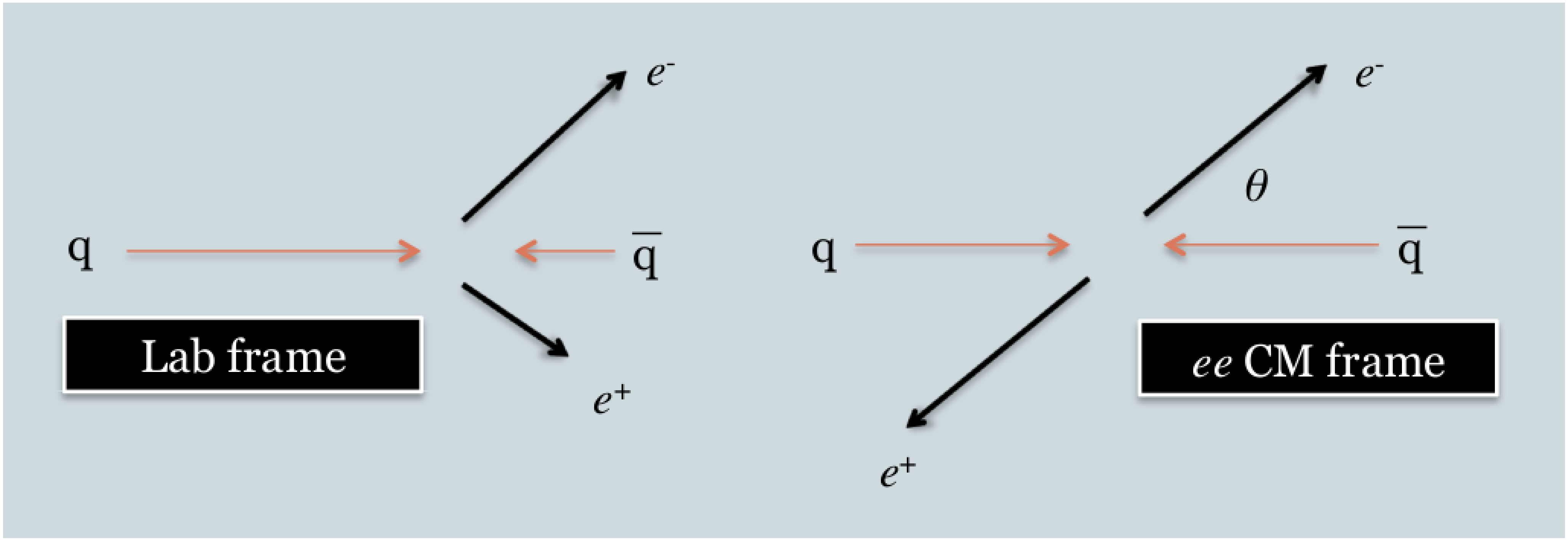


In the **predictions**: you fill the histograms depending on s' applying the formula.
NOTE: Actually the proton is uud not only u(d): the closest prediction is a mixture of 2/3 up and 1/3 down !

In **data** (and simulation of data): you have events → you have to compute the s' of the event → and decide if the electron goes foward or backward

In data: A_{FB} = (# forward electrons - # backward electrons)/(total # of electrons)
Problem: how do you decide if an electron goes forward?

A_{FB} is defined in the Center-of-mass frame



The electron is said to go in the forward region if theta < pi/2

Problems:

1. We measure objects in the lab frame → our 4-vectors must be boosted in the CM
2. Theta is the angle (in the CM) between the electron and the initial quark BUT our detector doesn't measure quarks (but jets) + in the final state there are no quarks

How do we solve problem #1?

1. We measure objects in the lab frame → our 4-vectors must be boosted in the CM

Easy! We use a special ROOT Class: TLorentzVector

TLorentzVector electron_plus_p4(ep_px, ep_py, ep_pz, ep_E); //electron plus
TLorentzVector electron_minus_p4(em_px, em_py, em_pz, em_E); //electron plus
TLorentzVector ee_pair=electron_plus_p4 + electron_minus_p4; //this is the 4-vector of the ee pair in the final state.

Every 4-vector can be boosted in the center-of-mass frame simply with:

Vector.Boost(-ee_pair.BoostVector());

How do we solve problem #2?

Theta is the angle (in the CM) between the electron and the initial quark BUT our detector doesn't measure quarks (but jets) + in the final state there are no quarks

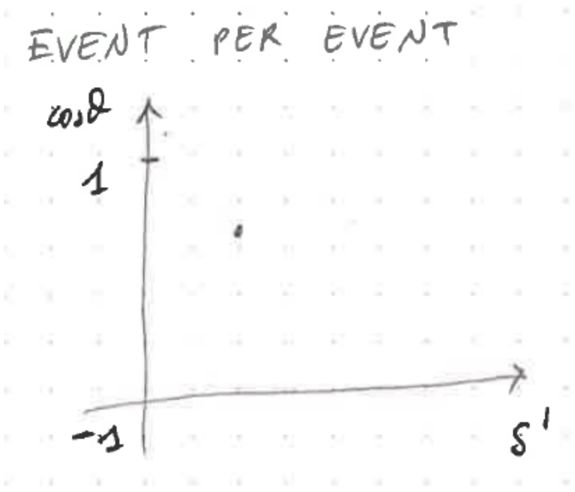
With the following reasoning:
1. The **initial quarks** were ~ along the beam pipe: along the **z-axis**
2. The D-Y process starts with the annihilation of a q anti-q pair
3. Either the quark or the anti-q carries bigger momentum along the z-axis → **the ee system in the final state will be boosted in the z direction**
3. The proton is ~uud → to anti-quark belongs to the sea → the anti-q probably carries less momentum than the quark
4. Conclusion: **the direction of the boost of ee is probably the direction of the quark in the initial state!**

So, to compute theta (cos(theta))

Scenario #0: simulation only (the easiest: let me call it **gen-gen**)

event per event (entry per entry of your tree):

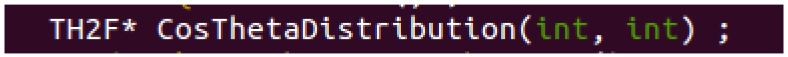
1. Take the generated electron
2. Take the generated positron
3. Build the ee pair
4. Take the generated quark
5. Take the generated anti-quark
6. Boost the electron, positron, quark, anti-quark in the CM
7. Compute the angle between the electron and the quark
8. Compute the invariant mass of the ee pair (s' of the event)
9. Fill a 2D histogram with s' and cos(theta)



I'll give you a skeleton to start with

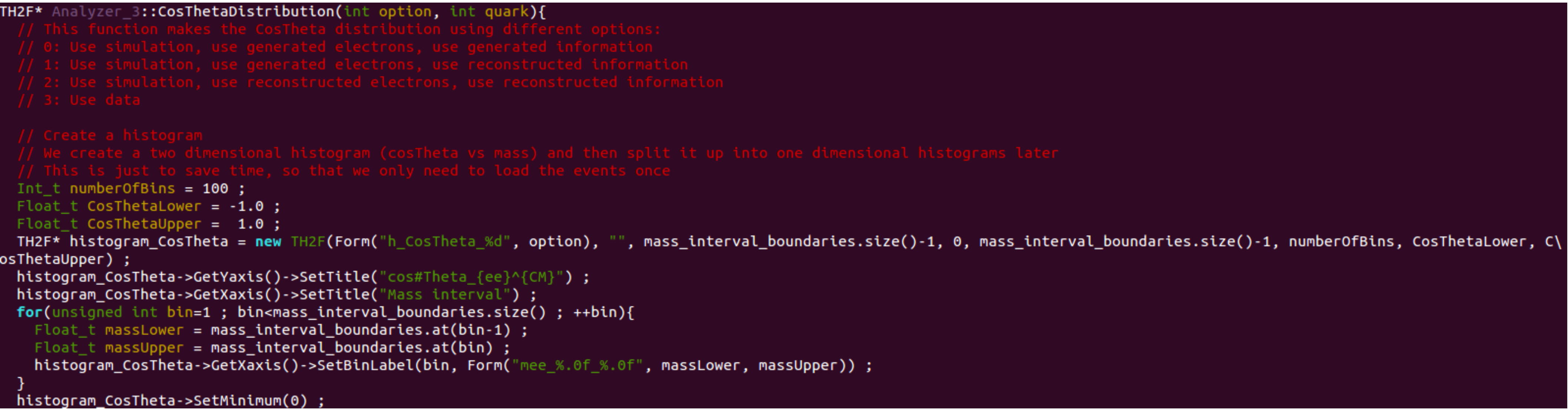
```
cp /ice3/phy_f_416_2016/Lesson_3/Analyzer_3.h .
cp /ice3/phy_f_416_2016/Lesson_3/Analyzer_3.C .
```

Have a look at the method CosThetaDistribution



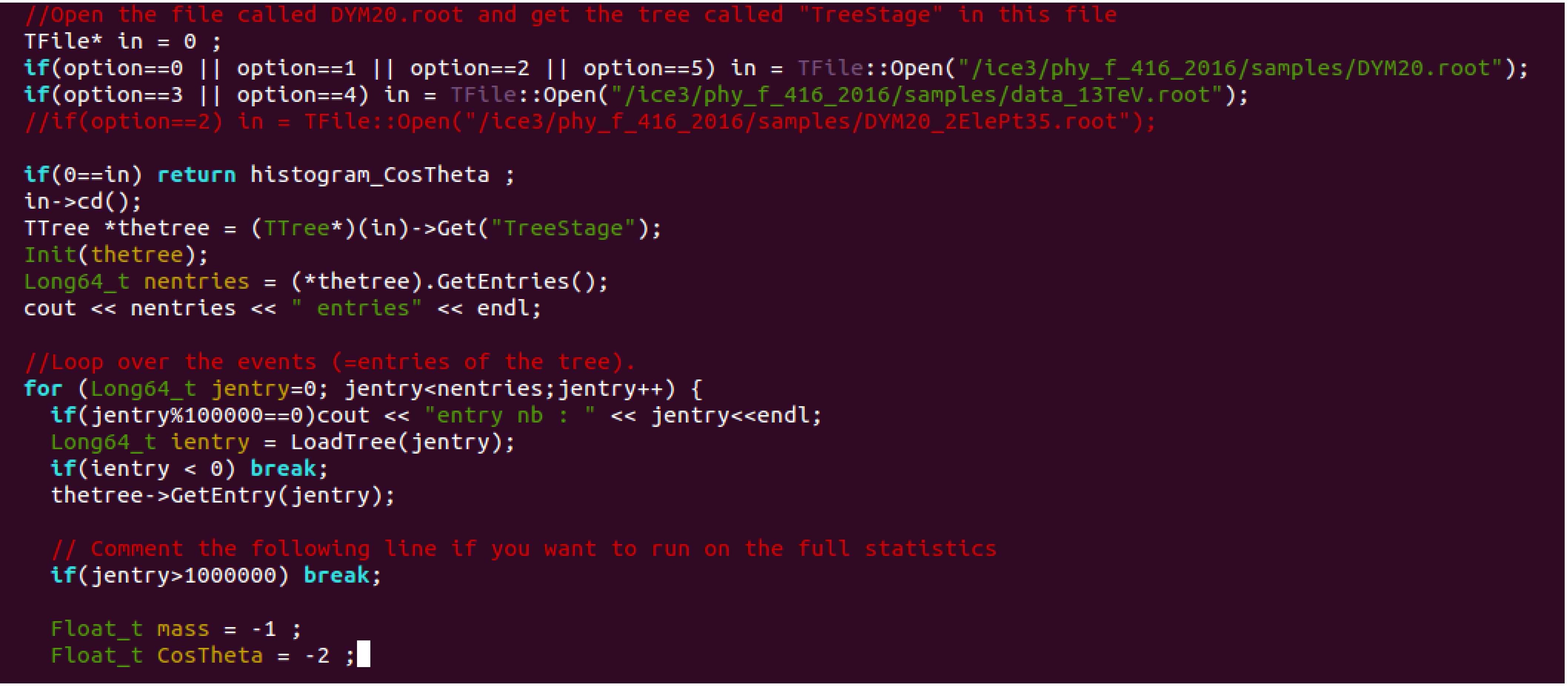
```
TH2F* CosThetaDistribution(int, int) ;
```

→ Definition of the method:
The first int is the scenario
The second int is the quark flavor
What is the output?



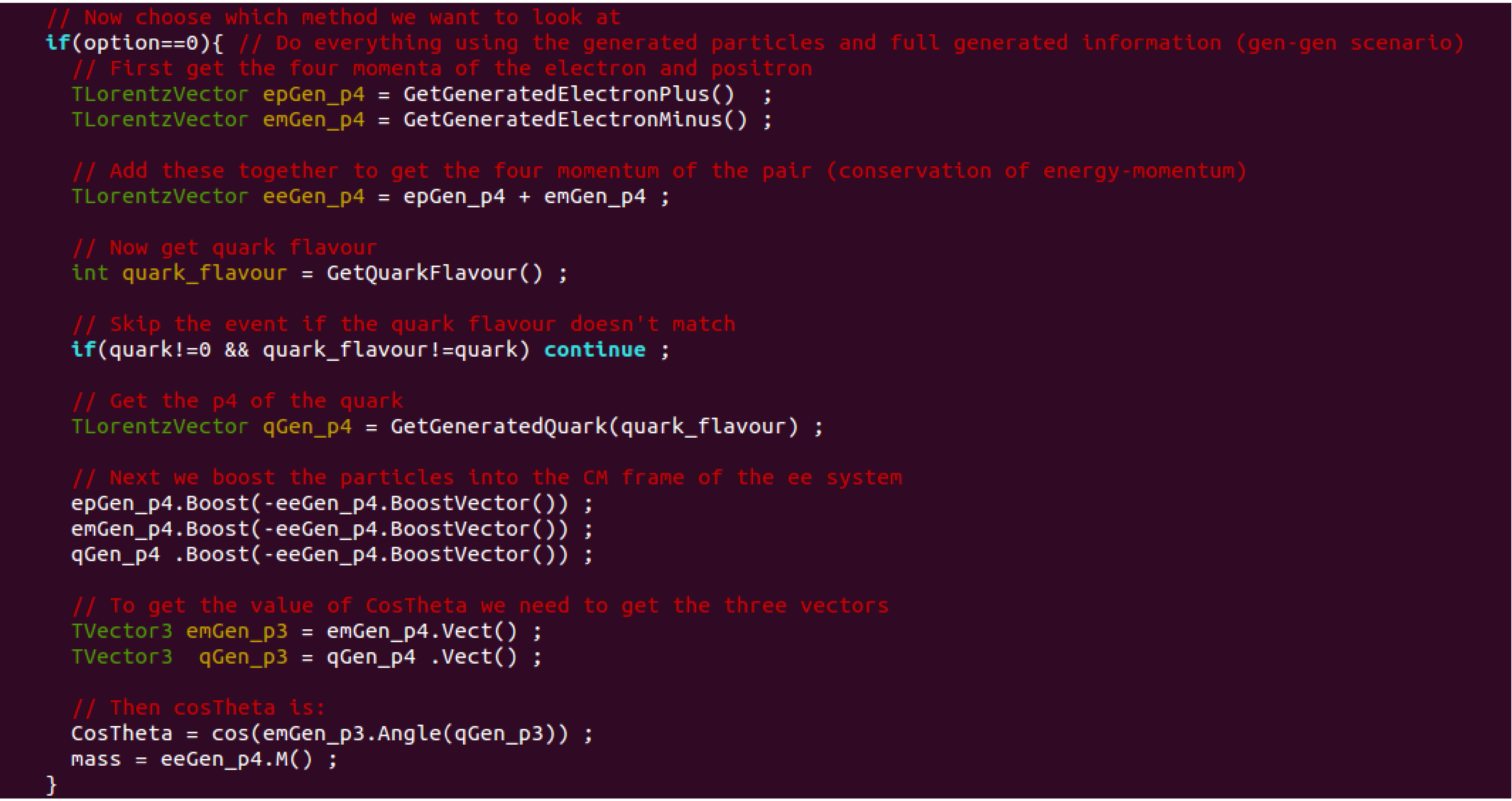
```
TH2F* Analyzer_3::CosThetaDistribution(int option, int quark){
  // This function makes the CosTheta distribution using different options:
  // 0: Use simulation, use generated electrons, use generated information
  // 1: Use simulation, use generated electrons, use reconstructed information
  // 2: Use simulation, use reconstructed electrons, use reconstructed information
  // 3: Use data

  // Create a histogram
  // We create a two dimensional histogram (cosTheta vs mass) and then split it up into one dimensional histograms later
  // This is just to save time, so that we only need to load the events once
  Int_t numberOfBins = 100 ;
  Float_t CosThetaLower = -1.0 ;
  Float_t CosThetaUpper =  1.0 ;
  TH2F* histogram_CosTheta = new TH2F(Form("h_CosTheta_%d", option), "", mass_interval_boundaries.size()-1, 0, mass_interval_boundaries.size()-1, numberOfBins, CosThetaLower, C\
osThetaUpper) ;
  histogram_CosTheta->GetYaxis()->SetTitle("cos#Theta_{ee}^{CM}") ;
  histogram_CosTheta->GetXaxis()->SetTitle("Mass interval") ;
  for(unsigned int bin=1 ; bin<mass_interval_boundaries.size() ; ++bin){
    Float_t massLower = mass_interval_boundaries.at(bin-1) ;
    Float_t massUpper = mass_interval_boundaries.at(bin) ;
    histogram_CosTheta->GetXaxis()->SetBinLabel(bin, Form("mee_%.0f_%.0f", massLower, massUpper)) ;
  }
  histogram_CosTheta->SetMinimum(0) ;
```

I'll give you a skeleton to start with



```
//Open the file called DYM20.root and get the tree called "TreeStage" in this file
TFile* in = 0 ;
if(option==0 || option==1 || option==2 || option==5) in = TFile::Open("/ice3/phy_f_416_2016/samples/DYM20.root");
if(option==3 || option==4) in = TFile::Open("/ice3/phy_f_416_2016/samples/data_13TeV.root");
//if(option==2) in = TFile::Open("/ice3/phy_f_416_2016/samples/DYM20_2ElePt35.root");

if(0==in) return histogram_CosTheta ;
in->cd();
TTree *thetree = (TTree*)(in)->Get("TreeStage");
Init(thetree);
Long64_t nentries = (*thetree).GetEntries();
cout << nentries << " entries" << endl;

//Loop over the events (=entries of the tree).
for (Long64_t jentry=0; jentry<nentries;jentry++) {
  if(jentry%100000==0)cout << "entry nb : " << jentry<<endl;
  Long64_t ientry = LoadTree(jentry);
  if(ientry < 0) break;
  thetree->GetEntry(jentry);

  // Comment the following line if you want to run on the full statistics
  if(jentry>1000000) break;

  Float_t mass = -1 ;
  Float_t CosTheta = -2 ;
```

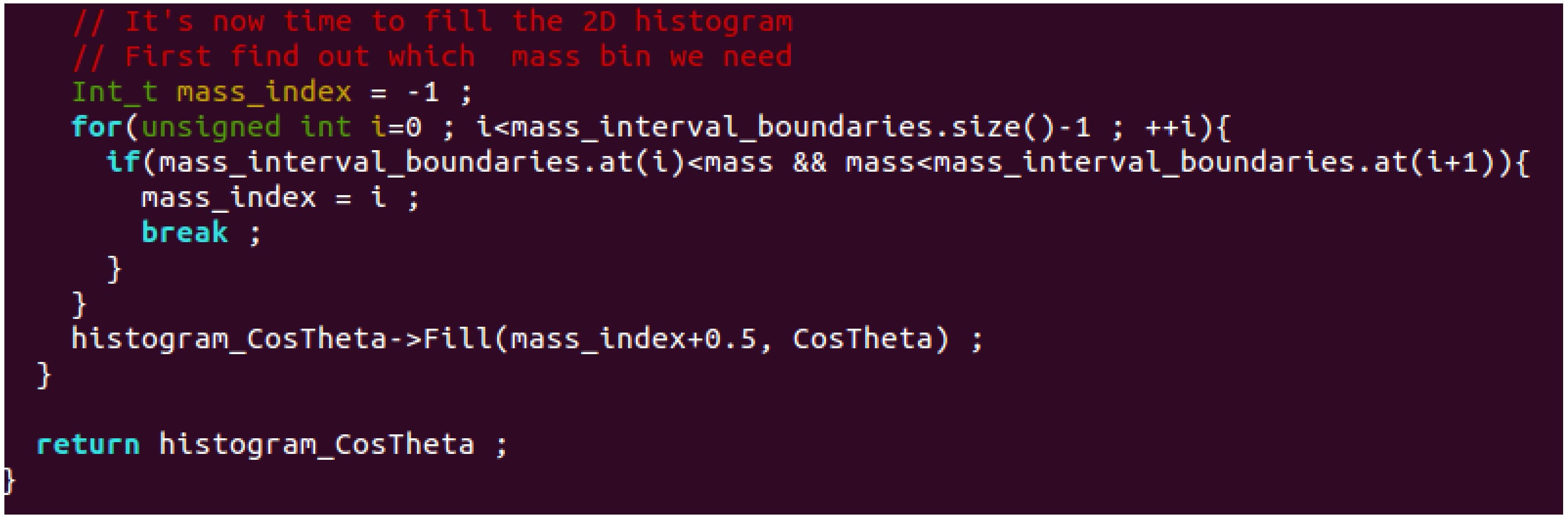
Scenario 0: gen-gen



```
// Now choose which method we want to look at
if(option==0){ // Do everything using the generated particles and full generated information (gen-gen scenario)
  // First get the four momenta of the electron and positron
  TLorentzVector epGen_p4 = GetGeneratedElectronPlus()  ;
  TLorentzVector emGen_p4 = GetGeneratedElectronMinus() ;

  // Add these together to get the four momentum of the pair (conservation of energy-momentum)
  TLorentzVector eeGen_p4 = epGen_p4 + emGen_p4 ;

  // Now get quark flavour
  int quark_flavour = GetQuarkFlavour() ;

  // Skip the event if the quark flavour doesn't match
  if(quark!=0 && quark_flavour!=quark) continue ;

  // Get the p4 of the quark
  TLorentzVector qGen_p4 = GetGeneratedQuark(quark_flavour) ;

  // Next we boost the particles into the CM frame of the ee system
  epGen_p4.Boost(-eeGen_p4.BoostVector()) ;
  emGen_p4.Boost(-eeGen_p4.BoostVector()) ;
  qGen_p4 .Boost(-eeGen_p4.BoostVector()) ;

  // To get the value of CosTheta we need to get the three vectors
  TVector3 emGen_p3 = emGen_p4.Vect() ;
  TVector3  qGen_p3 = qGen_p4 .Vect() ;

  // Then cosTheta is:
  CosTheta = cos(emGen_p3.Angle(qGen_p3)) ;
  mass = eeGen_p4.M() ;
}
```

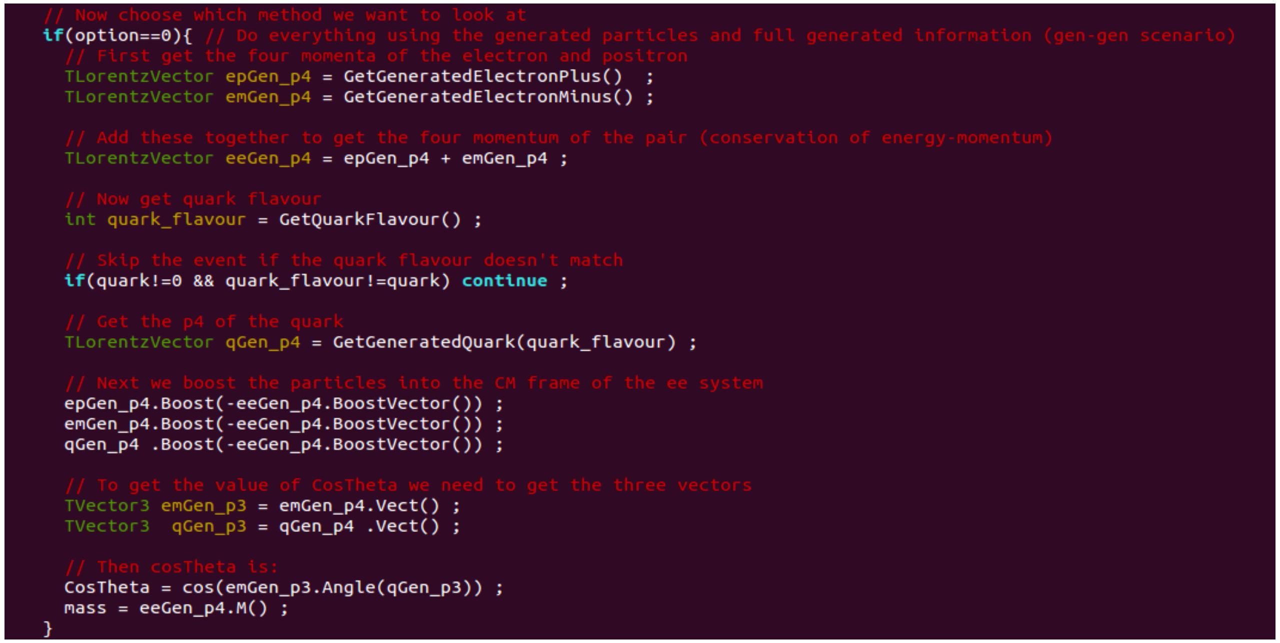
This is scenario 0: we only looked at generated information
What are the other possible scenarios?

And close the method



```
    // It's now time to fill the 2D histogram
    // First find out which  mass bin we need
    Int_t mass_index = -1 ;
    for(unsigned int i=0 ; i<mass_interval_boundaries.size()-1 ; ++i){
      if(mass_interval_boundaries.at(i)<mass && mass<mass_interval_boundaries.at(i+1)){
        mass_index = i ;
        break ;
      }
    }
    histogram_CosTheta->Fill(mass_index+0.5, CosTheta) ;
  }

  return histogram_CosTheta ;
}
```

Other scenarios



```
  // Now choose which method we want to look at
  if(option==0){ // Do everything using the generated particles and full generated information (gen-gen scenario)
    // First get the four momenta of the electron and positron
    TLorentzVector epGen_p4 = GetGeneratedElectronPlus()  ;
    TLorentzVector emGen_p4 = GetGeneratedElectronMinus() ;

    // Add these together to get the four momentum of the pair (conservation of energy-momentum)
    TLorentzVector eeGen_p4 = epGen_p4 + emGen_p4 ;

    // Now get quark flavour
    int quark_flavour = GetQuarkFlavour() ;

    // Skip the event if the quark flavour doesn't match
    if(quark!=0 && quark_flavour!=quark) continue ;

    // Get the p4 of the quark
    TLorentzVector qGen_p4 = GetGeneratedQuark(quark_flavour) ;

    // Next we boost the particles into the CM frame of the ee system
    epGen_p4.Boost(-eeGen_p4.BoostVector()) ;
    emGen_p4.Boost(-eeGen_p4.BoostVector()) ;
    qGen_p4 .Boost(-eeGen_p4.BoostVector()) ;

    // To get the value of CosTheta we need to get the three vectors
    TVector3 emGen_p3 = emGen_p4.Vect() ;
    TVector3  qGen_p3 = qGen_p4 .Vect() ;

    // Then cosTheta is:
    CosTheta = cos(emGen_p3.Angle(qGen_p3)) ;
    mass = eeGen_p4.M() ;
  }
```

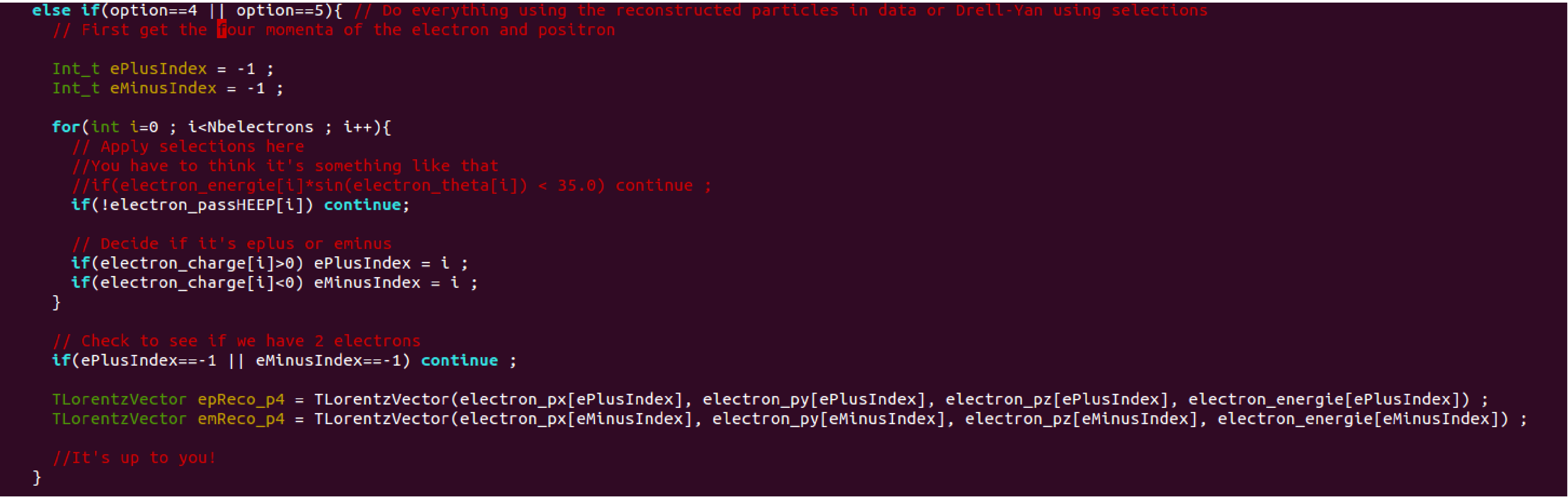
Now that you undestood the logic of the method:
→ Add other options for the other possible scenarios
Scenario #1: Use generated electrons but compute cos(theta) in the "reconstructed way" i.e. between the electron and the z direction of the ee pair (a.k.a gen-reco)
Scenario #2: Same as #1 but use reconstructed electrons (a.k.a reco-reco)
Scenario #3: Same as #2 but use data (if you look at slide 12 you opened different files depending on the option)

You have ~30 minutes to fill the other options in the method

Final scenarios

Predictions are "pure" in the sense that you simulate only DY process
Data are "dirt" in the sense that what you measure is not only DY: a photon or a jet could be misidentified as electron → that's why we apply a selection : a serie of cuts that rejects "bad" electrons

Scenario #4: Like Scenario #3, but applying the official high energy electron pair selection (heep) on data
Scenario #5: Like Scenario #2, but applying the official high energy electron pair selection (heep) on simulation



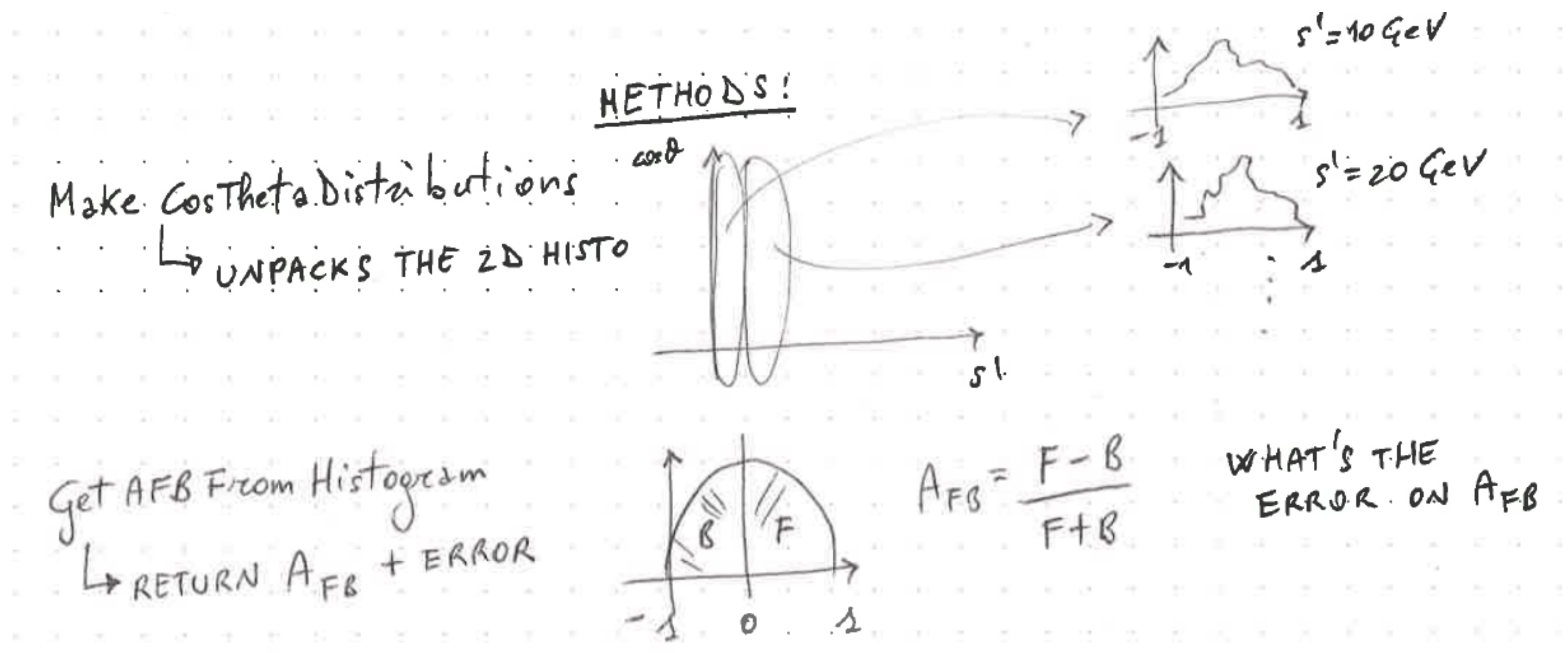
```
else if(option==4 || option==5){ // Do everything using the reconstructed particles in data or Drell-Yan using selections
  // First get the four momenta of the electron and positron

  Int_t ePlusIndex = -1 ;
  Int_t eMinusIndex = -1 ;

  for(int i=0 ; i<Nbelectrons ; i++){
    // Apply selections here
    //You have to think it's something like that
    //if(electron_energie[i]*sin(electron_theta[i]) < 35.0) continue ;
    if(!electron_passHEEP[i]) continue;

    // Decide if it's eplus or eminus
    if(electron_charge[i]>0) ePlusIndex = i ;
    if(electron_charge[i]<0) eMinusIndex = i ;
  }

  // Check to see if we have 2 electrons
  if(ePlusIndex==-1 || eMinusIndex==-1) continue ;

  TLorentzVector epReco_p4 = TLorentzVector(electron_px[ePlusIndex], electron_py[ePlusIndex], electron_pz[ePlusIndex], electron_energie[ePlusIndex]) ;
  TLorentzVector emReco_p4 = TLorentzVector(electron_px[eMinusIndex], electron_py[eMinusIndex], electron_pz[eMinusIndex], electron_energie[eMinusIndex]) ;

  //It's up to you!
}
```

Final step

Once you have filled the 2D histogram of s' and cos(theta) you have just to:

1. Fix a s' bin (x-bin)
2. Unpack the 2D histogram slice per slice in s'
2. Integrate how many events have cos(theta) > 0
3. Integrate how many events have cos(theta) < 0
4. Measure the A_{FB} as (2-3)(2+3)

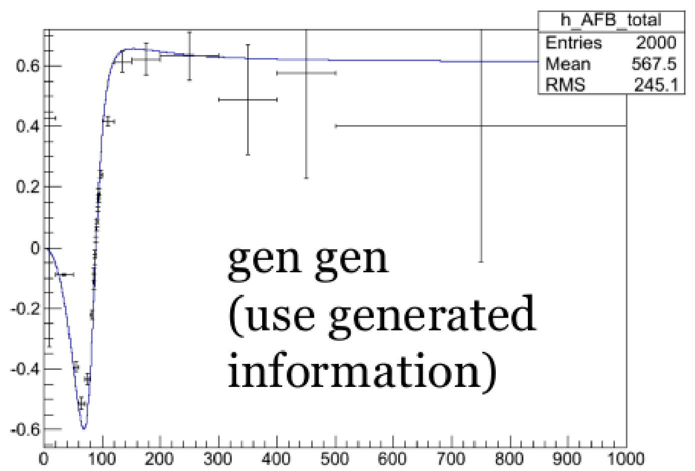
I provide you with 2 powerful methods: ~easy to write but time consuming

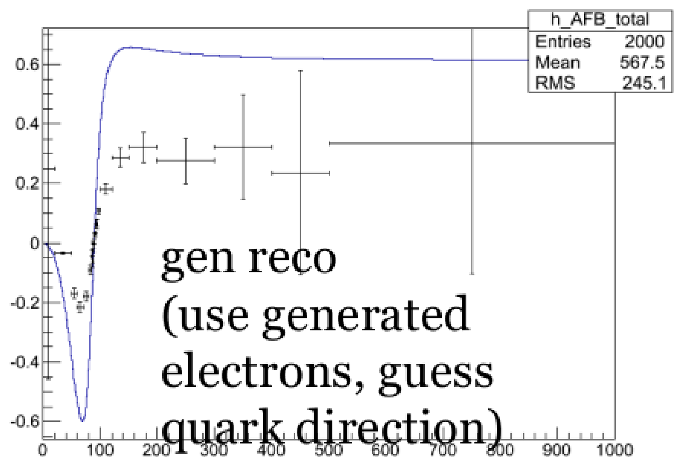

METHODS!
Make CosThetaDistributions
↳ UNPACKS THE 2D HISTO
cosθ
s'
s' = 10 GeV
s' = 20 GeV
-1
1
Get AFB From Histogram
↳ RETURN A_{FB} + ERROR
B
F
-1
0
1
$A_{FB} = \frac{F-B}{F+B}$
WHAT'S THE ERROR ON A_{FB}

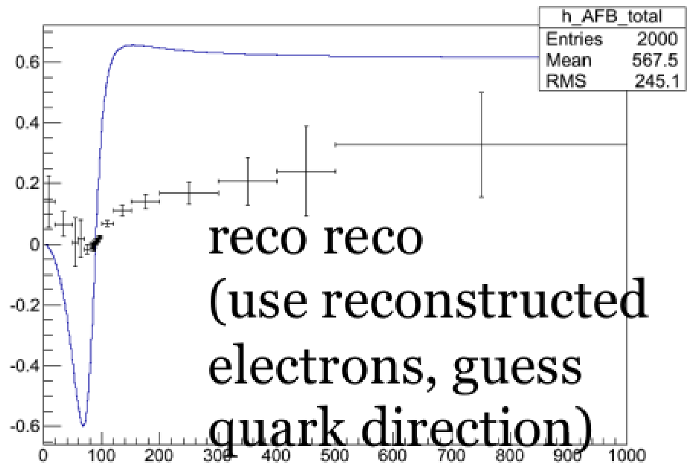
Putting everything together

```
root -l
.L Analyzer_3.C++
Analyzer_3 Test
Test. MakeCosThetaDistributions()
Test.MakeAFBPlots() (→ this will produce AFB_graphs.root with all the scenarios)
Test.CompareAFB() (→ to superimpose scenario #1 and predictions for example)
```

CompareAFB method is NOT complete: **it's up to you** to produce all the scenarios comparisons





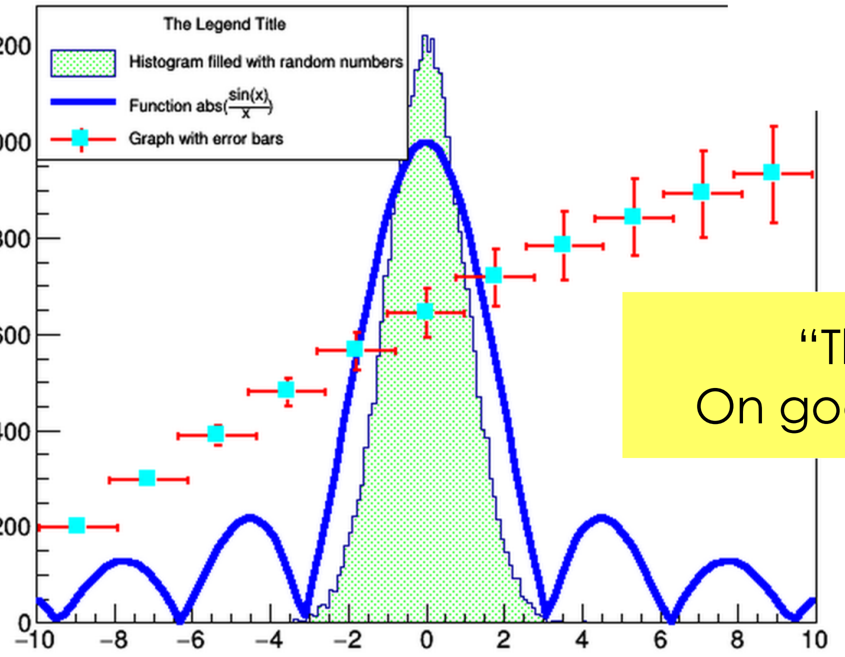


Here I am comparing:
Predictions (blue curve) and MC simulations (black points)

One of these 3 plots will be similar (hopefully) to real data: which one?
Go on with the comparisons **real data vs prediction** and real **data vs ?**

Next lesson

Homework:
Make the plots of the **AFB in real data**, without applying any selection and after **applying our heep selection**
Produce **all the comparisons**: MC vs predictions, data vs predictions, data vs MC
IMPORTANT: Put a **clear legend** on the plots (not like me in slide 17)
No solutions this time: it's your job :-)



"Tlegend example"
On google to find the code

→ Next tuesday (our last lesson):
We'll have a look at your codes and finalize all the plots to put in your report