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

Today's lesson

Targets:

- Obtain reco and gen mass spectra (your homework)
- write a C++ method to compute R
- Write a C++ method to compute c1, c2
- Write a C++ method to compute A_{FB}

ssh login to your account

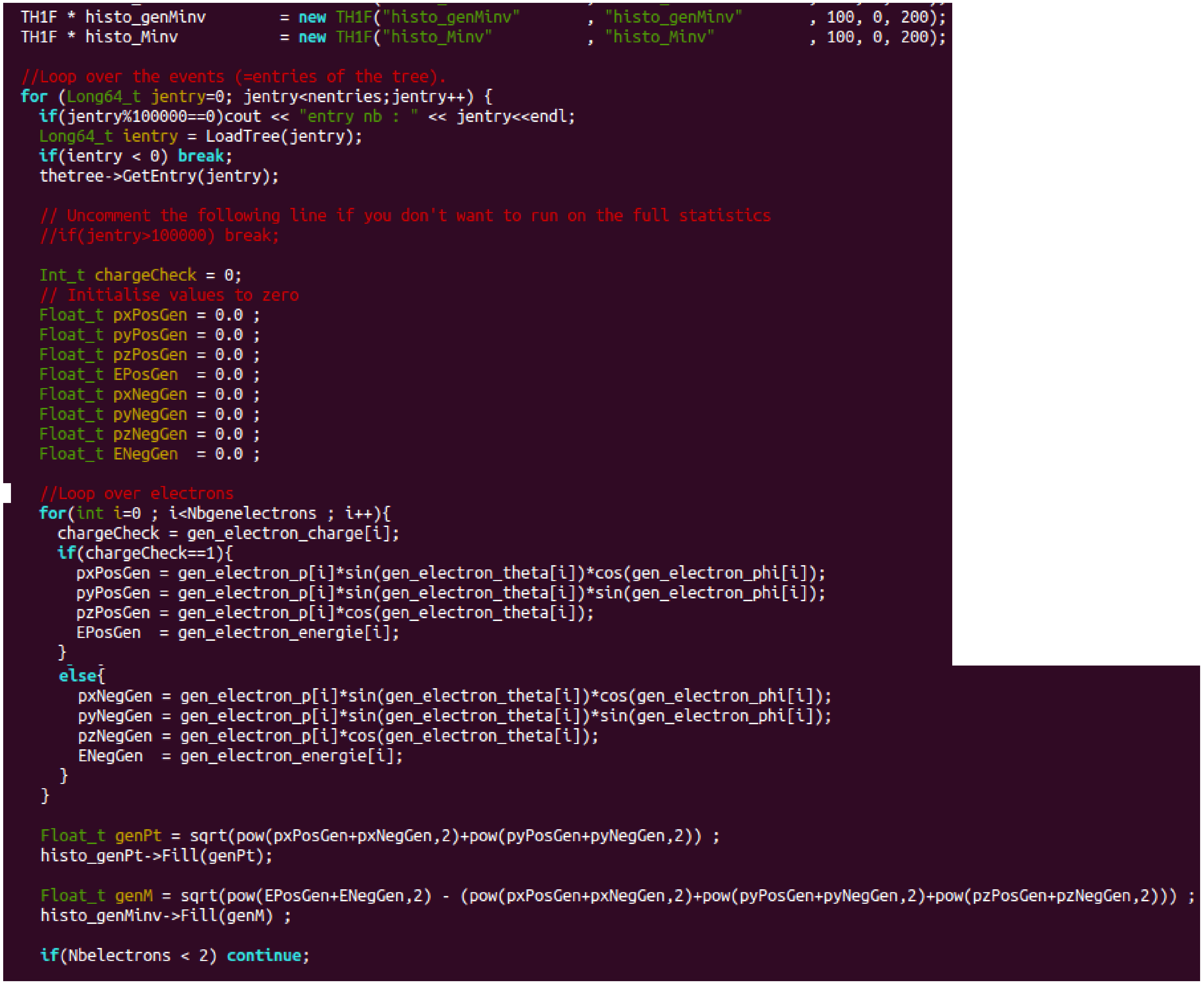
Login:
ssh -Y yourname@lxpub.iihe.ac.be
Password is: xxxxx
Now you pc is just a monitor: you are in fact using another remote pc!

To **set ROOT**:
source setter.sh

(the last source command is equivalent to:
cd CMSSW_8_0_8
source /cvmfs/cms.cern.ch/cmsset_default.sh
cmsenv
cd -
)

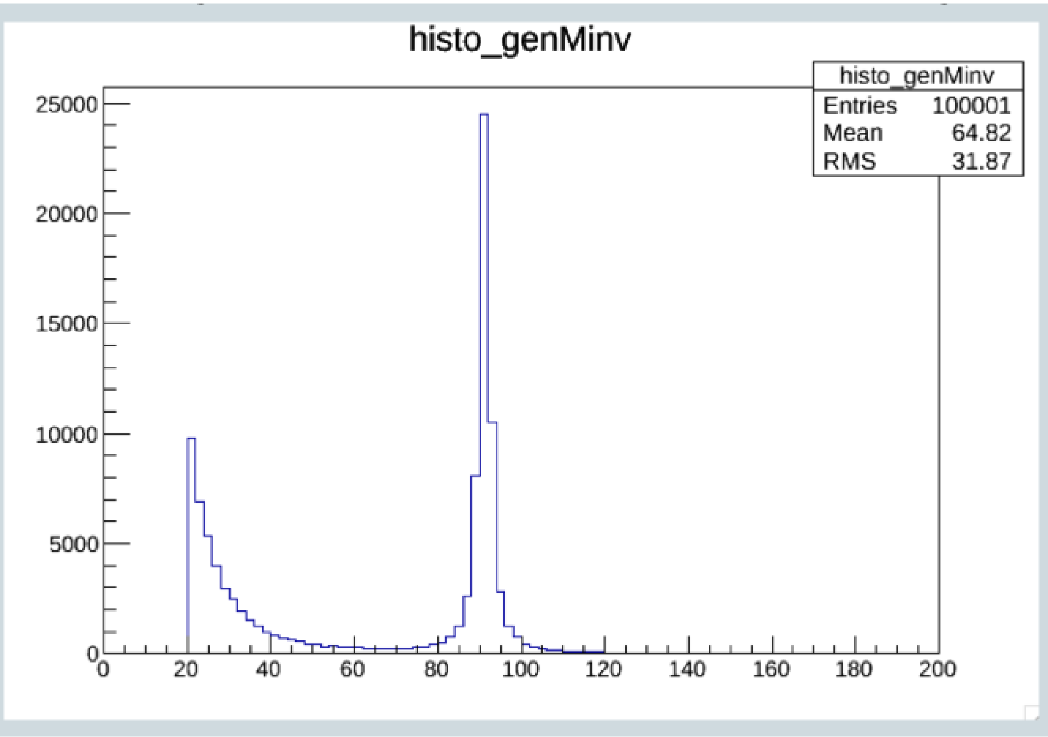
To do: InvMass histograms (Gen and Reco)

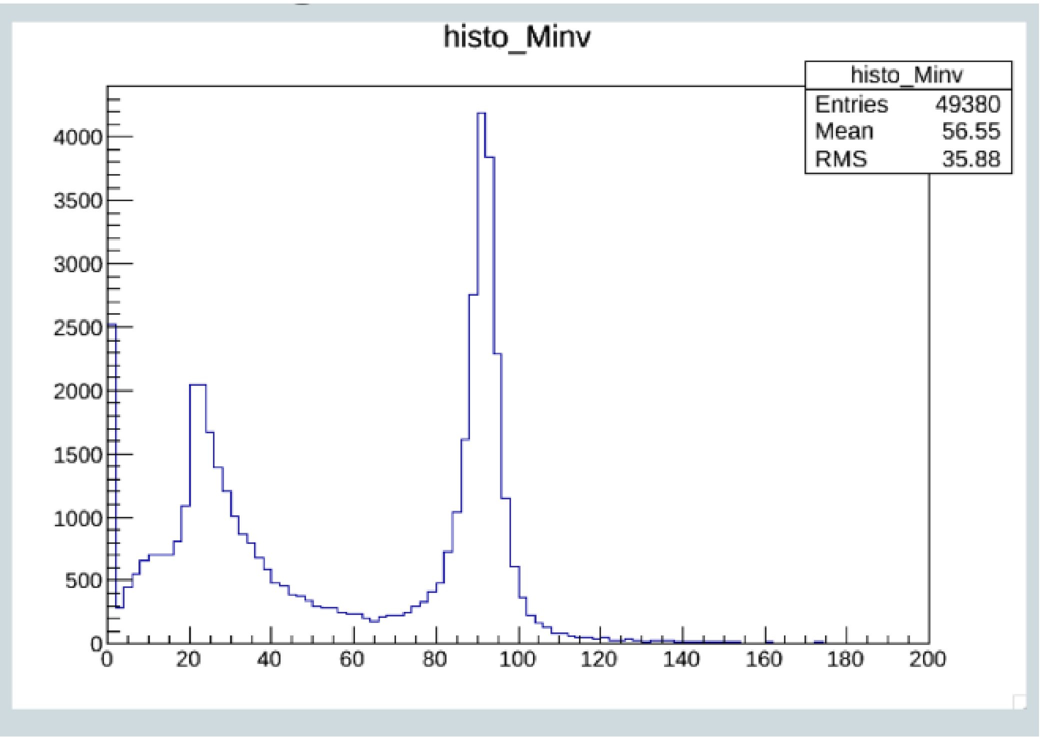
You have
15 minutes
to complete
it with Reco



```
TH1F * histo_genMinv        = new TH1F("histo_genMinv"       , "histo_genMinv"       , 100, 0, 200);
TH1F * histo_Minv           = new TH1F("histo_Minv"          , "histo_Minv"          , 100, 0, 200);

//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);

  // Uncomment the following line if you don't want to run on the full statistics
  //if(jentry>100000) break;

  Int_t chargeCheck = 0;
  // Initialise values to zero
  Float_t pxPosGen = 0.0 ;
  Float_t pyPosGen = 0.0 ;
  Float_t pzPosGen = 0.0 ;
  Float_t EPosGen  = 0.0 ;
  Float_t pxNegGen = 0.0 ;
  Float_t pyNegGen = 0.0 ;
  Float_t pzNegGen = 0.0 ;
  Float_t ENegGen  = 0.0 ;

  //Loop over electrons
  for(int i=0 ; i<Nbgenelectrons ; i++){
    chargeCheck = gen_electron_charge[i];
    if(chargeCheck==1){
      pxPosGen = gen_electron_p[i]*sin(gen_electron_theta[i])*cos(gen_electron_phi[i]);
      pyPosGen = gen_electron_p[i]*sin(gen_electron_theta[i])*sin(gen_electron_phi[i]);
      pzPosGen = gen_electron_p[i]*cos(gen_electron_theta[i]);
      EPosGen  = gen_electron_energie[i];
    }
    else{
      pxNegGen = gen_electron_p[i]*sin(gen_electron_theta[i])*cos(gen_electron_phi[i]);
      pyNegGen = gen_electron_p[i]*sin(gen_electron_theta[i])*sin(gen_electron_phi[i]);
      pzNegGen = gen_electron_p[i]*cos(gen_electron_theta[i]);
      ENegGen  = gen_electron_energie[i];
    }
  }

  Float_t genPt = sqrt(pow(pxPosGen+pxNegGen,2)+pow(pyPosGen+pyNegGen,2)) ;
  histo_genPt->Fill(genPt);

  Float_t genM = sqrt(pow(EPosGen+ENegGen,2) - (pow(pxPosGen+pxNegGen,2)+pow(pyPosGen+pyNegGen,2)+pow(pzPosGen+pzNegGen,2))) ;
  histo_genMinv->Fill(genM) ;

  if(Nbelectrons < 2) continue;
```

Invartiant mass histogram





If you wrote your code correctly you should see something like these two plots

These plots will be part of your final report:
Why gen and reco are different?
Why the gen histogram does not start at zero?

Check with my implementation: InvMass histograms Generated and reconstructed

This is how I did it:

```
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer_Mass.h  .
cp /ice3/phy_f_416_2016/Lesson_1/Analyzer_Mass.C .
```

Compile Analyzer_Mass and run the Loop method, so:

```
root -l
.L Analyzer_Mass.C++
Analyzer_Mass John
John.Loop()
```

A_{FB} depends on s' (the center-of-mass energy of the initial state quarks)

- How is s' related to the invariant mass of the electrons in the final state?

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!

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

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

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

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

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

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

What do we need?

At the end of this lesson we want to have the plot of A_{FB} vs s'

We need to compute c1 and c2
To compute c1 and c2 we need also R
R is a complex number: did you write down the real and imaginary part of R?
To compute c1 and c2 we also need |R| and Re(R)

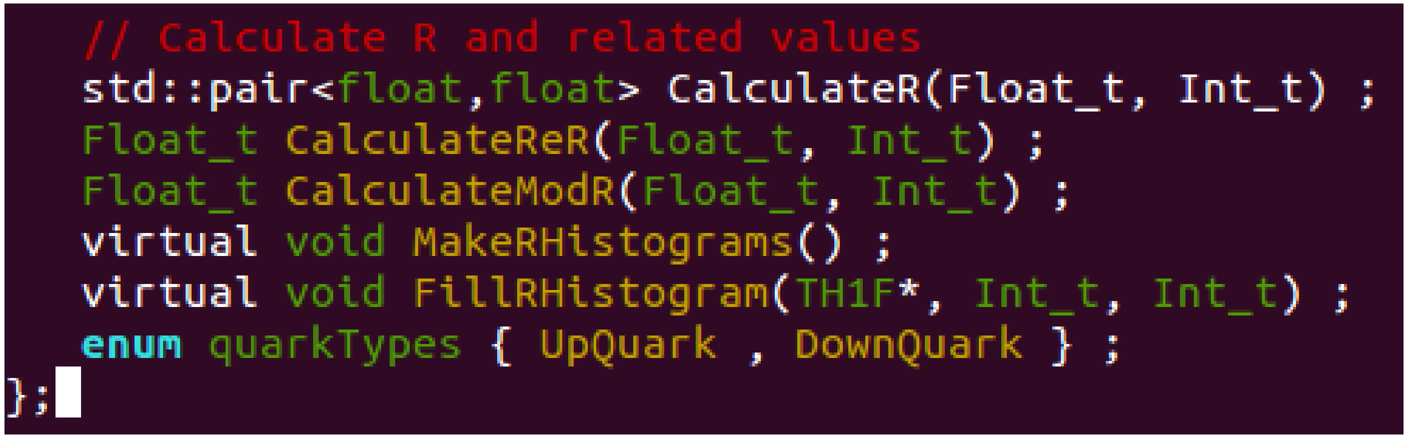
For sure we need at least 3 methods, let's call them:
CalculateR → inputs: s' and quark flavor outputs: a pair (real, imaginary)
CalculateReR → same inputs; output: a float
CalculateModR → same inputs; output: a float

R computation

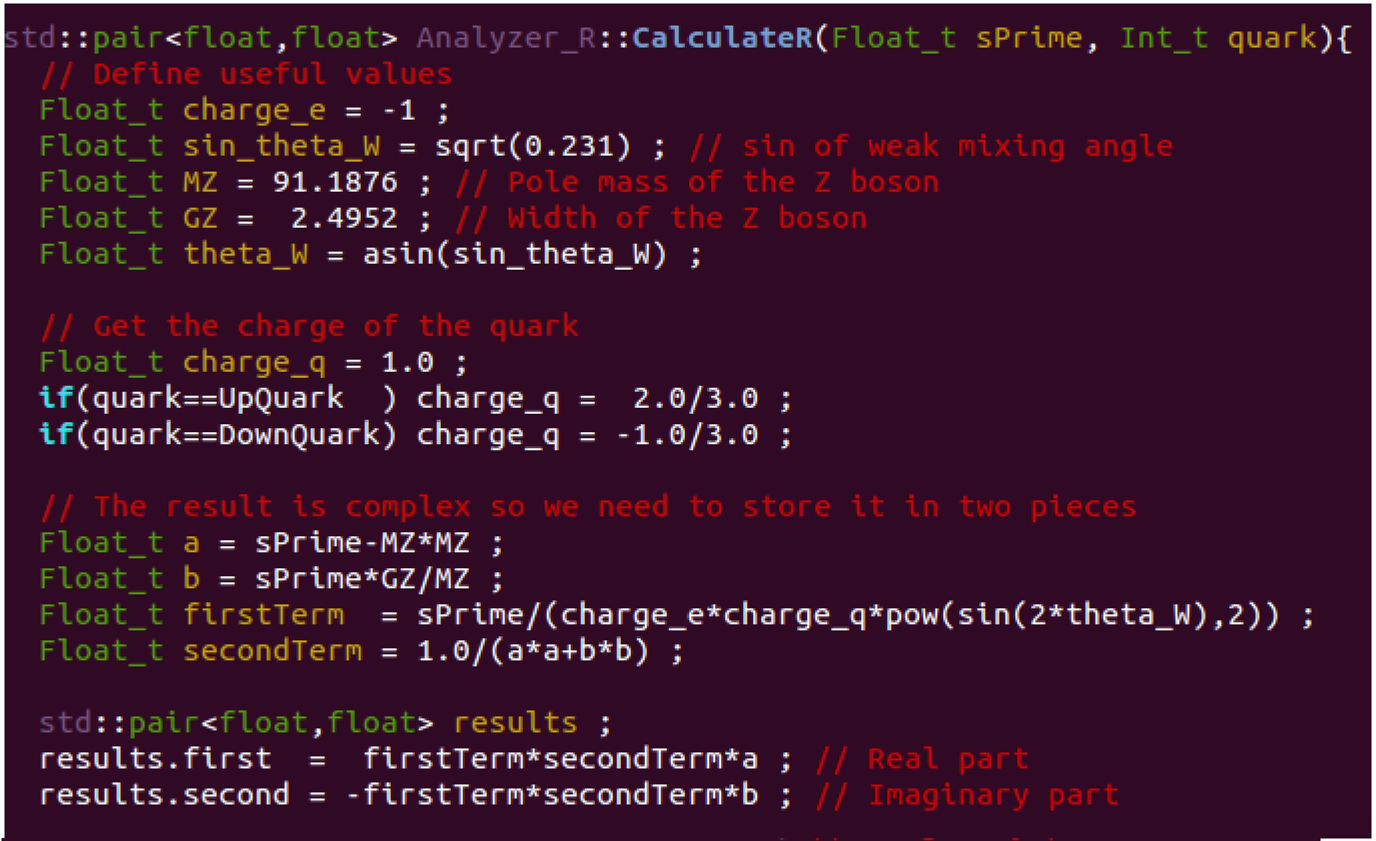
cp /ice3/phy_f_416_2016/Lesson_2/Analyzer_R.h .

Copy the declaration of the methods I have written

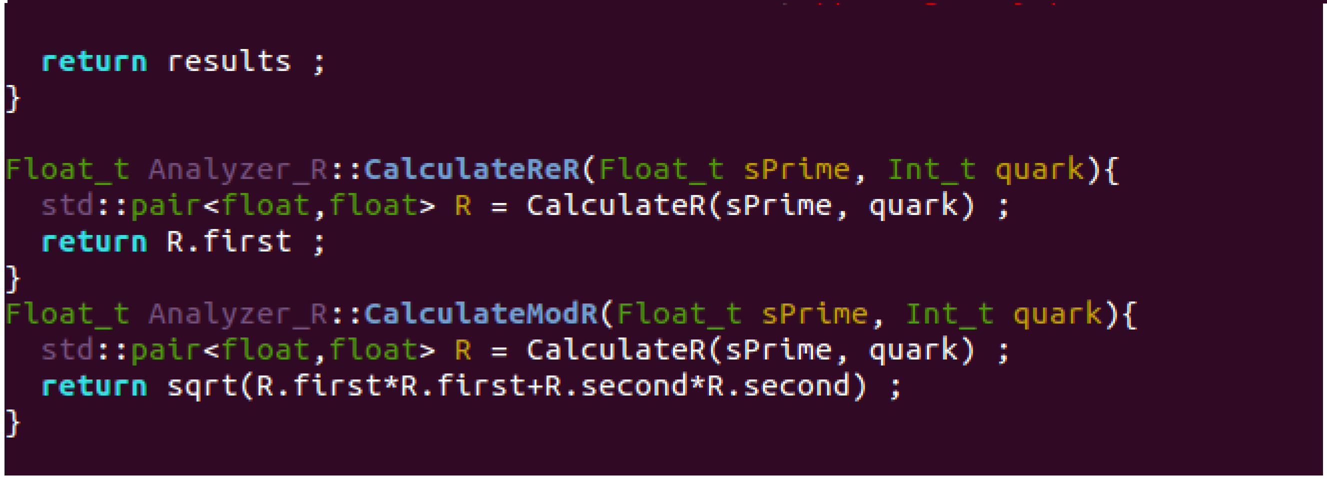
Now it's up to you to define the methods:
CalculateR, CalculateModR, CalculateReR



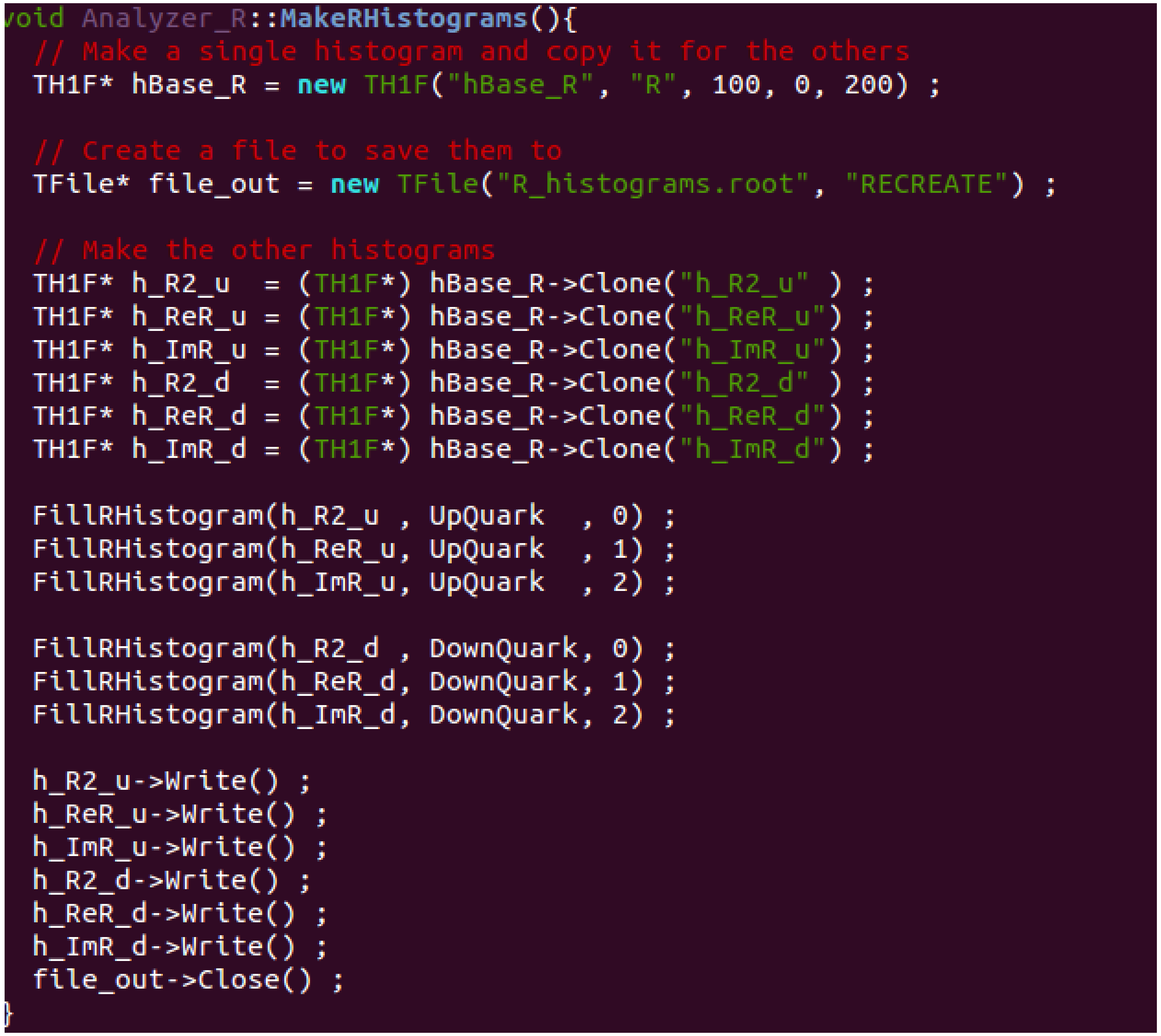
```
   // Calculate R and related values
   std::pair<float,float> CalculateR(Float_t, Int_t) ;
   Float_t CalculateReR(Float_t, Int_t) ;
   Float_t CalculateModR(Float_t, Int_t) ;
   virtual void MakeRHistograms() ;
   virtual void FillRHistogram(TH1F*, Int_t, Int_t) ;
   enum quarkTypes { UpQuark , DownQuark } ;
};
```



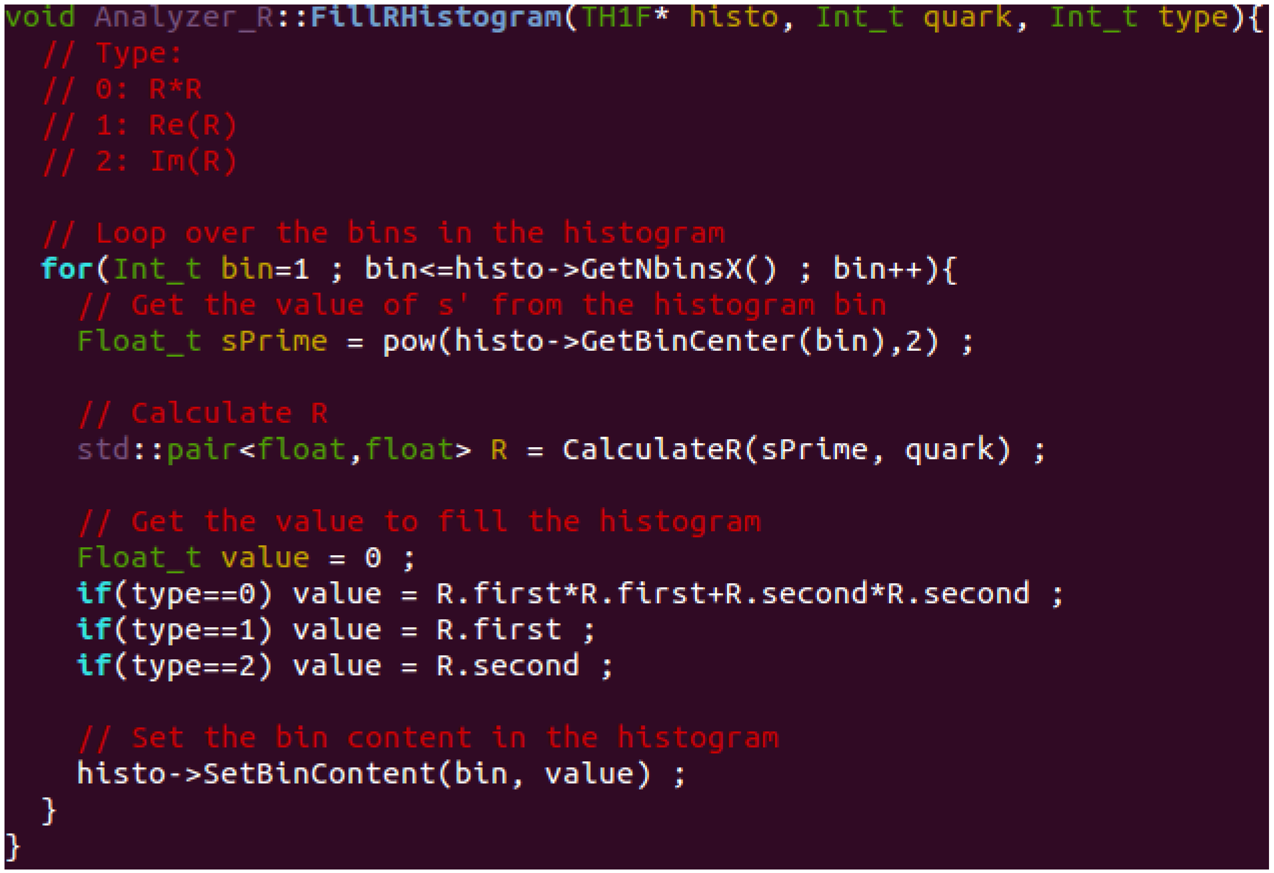
```
std::pair<float,float> Analyzer_R::CalculateR(Float_t sPrime, Int_t quark){
  // Define useful values
  Float_t charge_e = -1 ;
  Float_t sin_theta_W = sqrt(0.231) ; // sin of weak mixing angle
  Float_t MZ = 91.1876 ; // Pole mass of the Z boson
  Float_t GZ =  2.4952 ; // Width of the Z boson
  Float_t theta_W = asin(sin_theta_W) ;

  // Get the charge of the quark
  Float_t charge_q = 1.0 ;
  if(quark==UpQuark  ) charge_q =  2.0/3.0 ;
  if(quark==DownQuark) charge_q = -1.0/3.0 ;

  // The result is complex so we need to store it in two pieces
  Float_t a = sPrime-MZ*MZ ;
  Float_t b = sPrime*GZ/MZ ;
  Float_t firstTerm  = sPrime/(charge_e*charge_q*pow(sin(2*theta_W),2)) ;
  Float_t secondTerm = 1.0/(a*a+b*b) ;

  std::pair<float,float> results ;
  results.first  =  firstTerm*secondTerm*a ; // Real part
  results.second = -firstTerm*secondTerm*b ; // Imaginary part
```



```

  return results ;
}

Float_t Analyzer_R::CalculateReR(Float_t sPrime, Int_t quark){
  std::pair<float,float> R = CalculateR(sPrime, quark) ;
  return R.first ;
}
Float_t Analyzer_R::CalculateModR(Float_t sPrime, Int_t quark){
  std::pair<float,float> R = CalculateR(sPrime, quark) ;
  return sqrt(R.first*R.first+R.second*R.second) ;
}
```



```
void Analyzer_R::MakeRHistograms(){
  // Make a single histogram and copy it for the others
  TH1F* hBase_R = new TH1F("hBase_R", "R", 100, 0, 200) ;

  // Create a file to save them to
  TFile* file_out = new TFile("R_histograms.root", "RECREATE") ;

  // Make the other histograms
  TH1F* h_R2_u  = (TH1F*) hBase_R->Clone("h_R2_u" ) ;
  TH1F* h_ReR_u = (TH1F*) hBase_R->Clone("h_ReR_u") ;
  TH1F* h_ImR_u = (TH1F*) hBase_R->Clone("h_ImR_u") ;
  TH1F* h_R2_d  = (TH1F*) hBase_R->Clone("h_R2_d" ) ;
  TH1F* h_ReR_d = (TH1F*) hBase_R->Clone("h_ReR_d") ;
  TH1F* h_ImR_d = (TH1F*) hBase_R->Clone("h_ImR_d") ;

  FillRHistogram(h_R2_u , UpQuark  , 0) ;
  FillRHistogram(h_ReR_u, UpQuark  , 1) ;
  FillRHistogram(h_ImR_u, UpQuark  , 2) ;

  FillRHistogram(h_R2_d , DownQuark, 0) ;
  FillRHistogram(h_ReR_d, DownQuark, 1) ;
  FillRHistogram(h_ImR_d, DownQuark, 2) ;

  h_R2_u->Write() ;
  h_ReR_u->Write() ;
  h_ImR_u->Write() ;
  h_R2_d->Write() ;
  h_ReR_d->Write() ;
  h_ImR_d->Write() ;
  file_out->Close() ;
}
```



```
void Analyzer_R::FillRHistogram(TH1F* histo, Int_t quark, Int_t type){
  // Type:
  // 0: R*R
  // 1: Re(R)
  // 2: Im(R)

  // Loop over the bins in the histogram
  for(Int_t bin=1 ; bin<=histo->GetNbinsX() ; bin++){
    // Get the value of s' from the histogram bin
    Float_t sPrime = pow(histo->GetBinCenter(bin),2) ;

    // Calculate R
    std::pair<float,float> R = CalculateR(sPrime, quark) ;

    // Get the value to fill the histogram
    Float_t value = 0 ;
    if(type==0) value = R.first*R.first+R.second*R.second ;
    if(type==1) value = R.first ;
    if(type==2) value = R.second ;

    // Set the bin content in the histogram
    histo->SetBinContent(bin, value) ;
  }
}
```

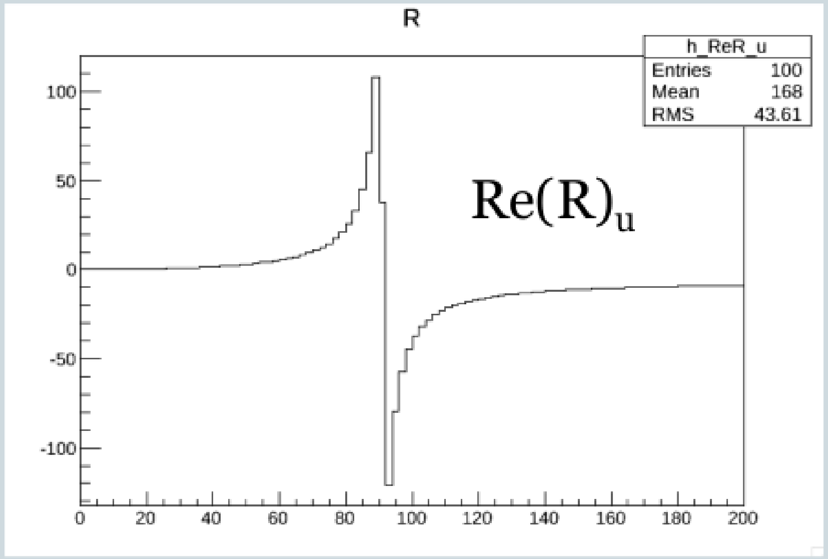
R computation

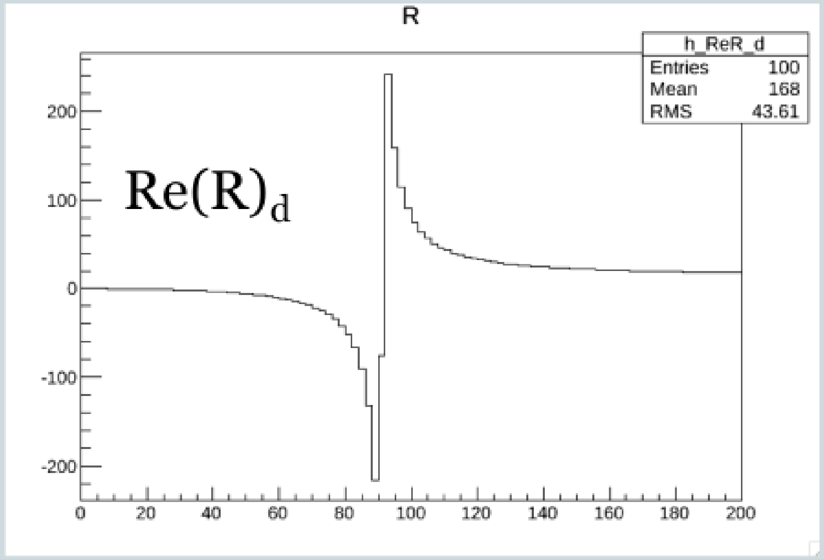
NOTE: If you don't want to overwrite what you wrote: change the names of your macros!

```
cp /ice3/phy_f_416_2016/Lesson_2/Analyzer_R.h  .
cp /ice3/phy_f_416_2016/Lesson_2/Analyzer_R.C  .
```

```
root -l
.L Analyzer_R.C++
Analyzer_R k
k.MakeRHistograms()
```

This will create a root file R_histograms.root
Open it with TBrowser and have a look





- Why are the shapes different?

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

Put these plots in your **final report** and explain them a bit

What else do we need?

We just computed R
To compute A_{FB} we still need c1 and c2 (both depending on R)

It's up to you to write the methods for c1 and c2:
Let me highlight the structure of the methods:

```
 Float_t CalculateC1(Float_t, Int_t) → What are the inputs?
 Float_t CalculateC2(Float_t, Int_t) ;
```

If you want, as I did, you can write another method:

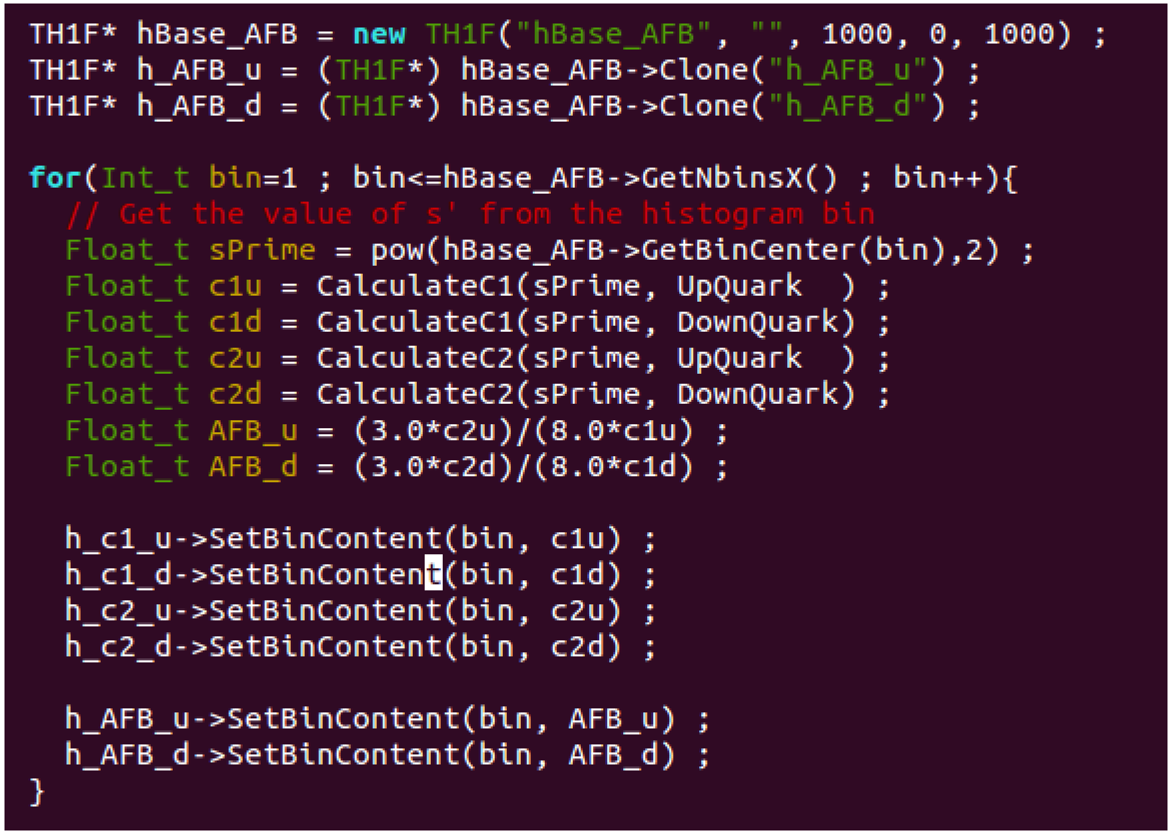
```
 Float_t CalculateC (Float_t, Int_t, Int_t) ;
```

If the last int is ==1 CalculateC1 is called, otherwise CalculateC2 is called.

A_{FB} prediction

Now that you know how to compute c1 and c2, you just have to write the final method that computes A_{FB}

→ Call it MakeAFBHistograms()
→ Create a histograms up to 1000 GeV of 1000 bins
→ bin per bin in s', fill the histogram with the A_{FB} value



```
TH1F* hBase_AFB = new TH1F("hBase_AFB", "", 1000, 0, 1000) ;
TH1F* h_AFB_u = (TH1F*) hBase_AFB->Clone("h_AFB_u") ;
TH1F* h_AFB_d = (TH1F*) hBase_AFB->Clone("h_AFB_d") ;

for(Int_t bin=1 ; bin<=hBase_AFB->GetNbinsX() ; bin++){
  // Get the value of s' from the histogram bin
  Float_t sPrime = pow(hBase_AFB->GetBinCenter(bin),2) ;
  Float_t c1u = CalculateC1(sPrime, UpQuark   ) ;
  Float_t c1d = CalculateC1(sPrime, DownQuark) ;
  Float_t c2u = CalculateC2(sPrime, UpQuark   ) ;
  Float_t c2d = CalculateC2(sPrime, DownQuark) ;
  Float_t AFB_u = (3.0*c2u)/(8.0*c1u) ;
  Float_t AFB_d = (3.0*c2d)/(8.0*c1d) ;

  h_c1_u->SetBinContent(bin, c1u) ;
  h_c1_d->SetBinContent(bin, c1d) ;
  h_c2_u->SetBinContent(bin, c2u) ;
  h_c2_d->SetBinContent(bin, c2d) ;

  h_AFB_u->SetBinContent(bin, AFB_u) ;
  h_AFB_d->SetBinContent(bin, AFB_d) ;
}
```

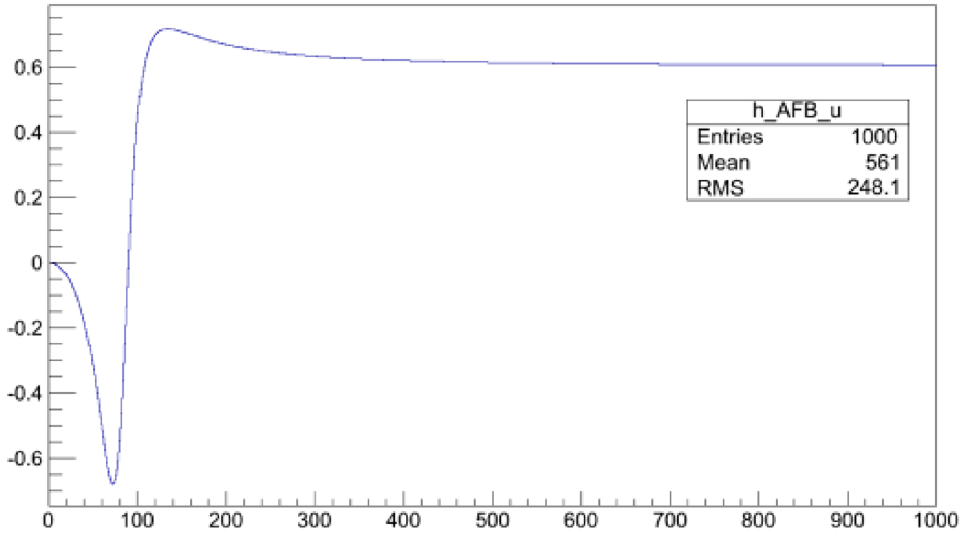
A_{FB} computation

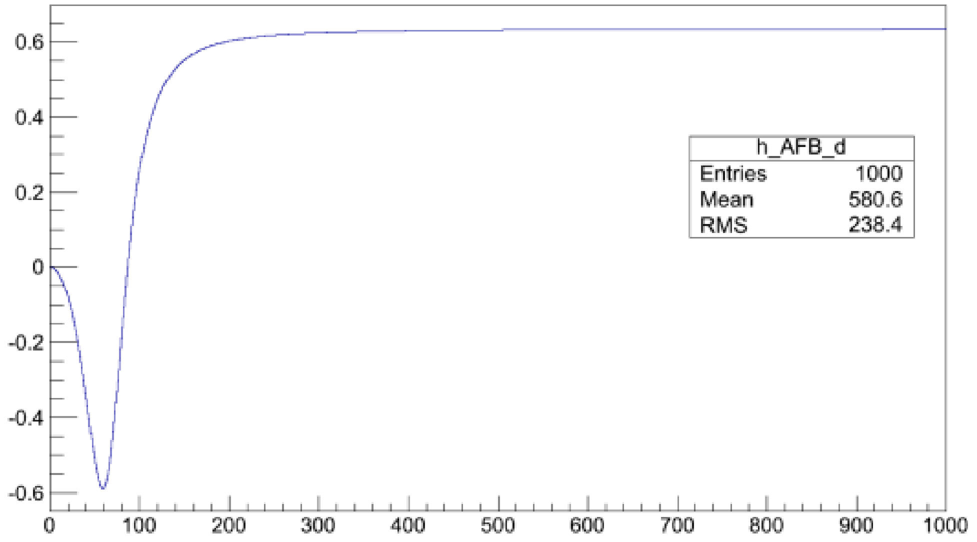
Recipe:

```
cp /ice3/phy_f_416_2016/Lesson_2/Analyzer_AFB.h  .
cp /ice3/phy_f_416_2016/Lesson_2/Analyzer_AFB.C  .

root -l
.L Analyzer_AFB.C++
Analyzer_AFB james
james.MakeAFBHistograms()
```

This will create a root file AFB_histograms.root
Open it with TBrowser and have a look

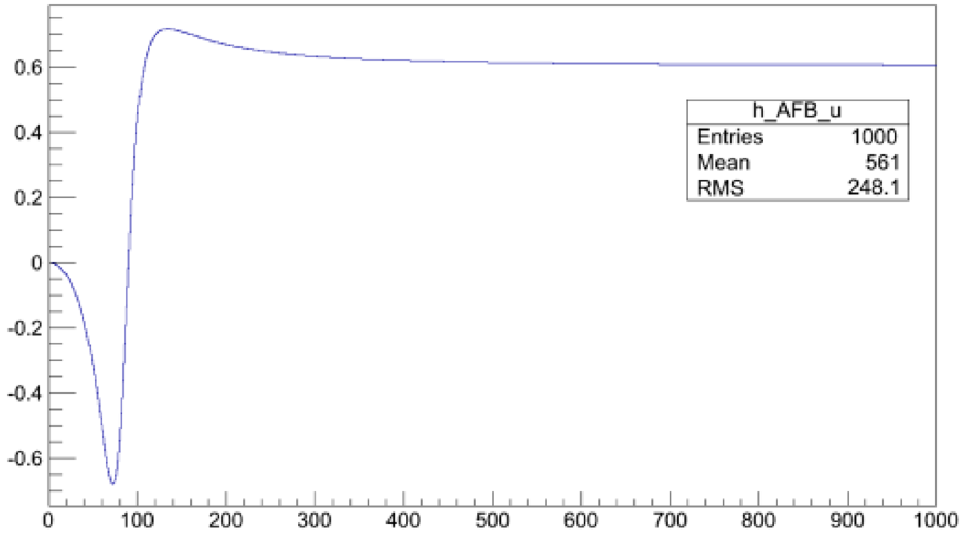
A_{FB} predictions

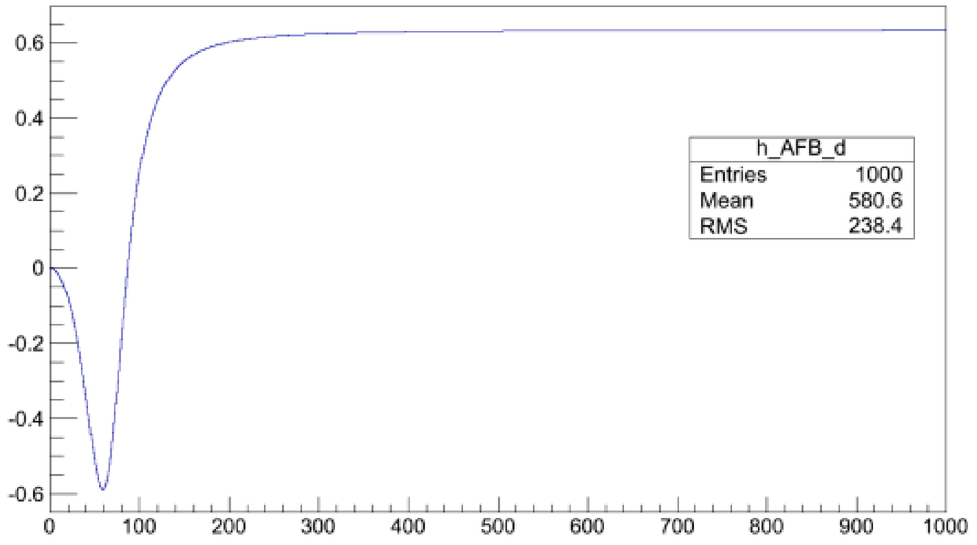




This is what you get applying the formula of A_{FB} as predicted by QFT
Put these plots in your **final report** and comment them

A_{FB} predictions





In the **predictions**: you fill the histograms depending on s' applying the formula
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?
We'll continue from here