RooFit_Basics

October 11, 2017

alt text

1 Introduction

If you want to follow this notebook you need to install ...

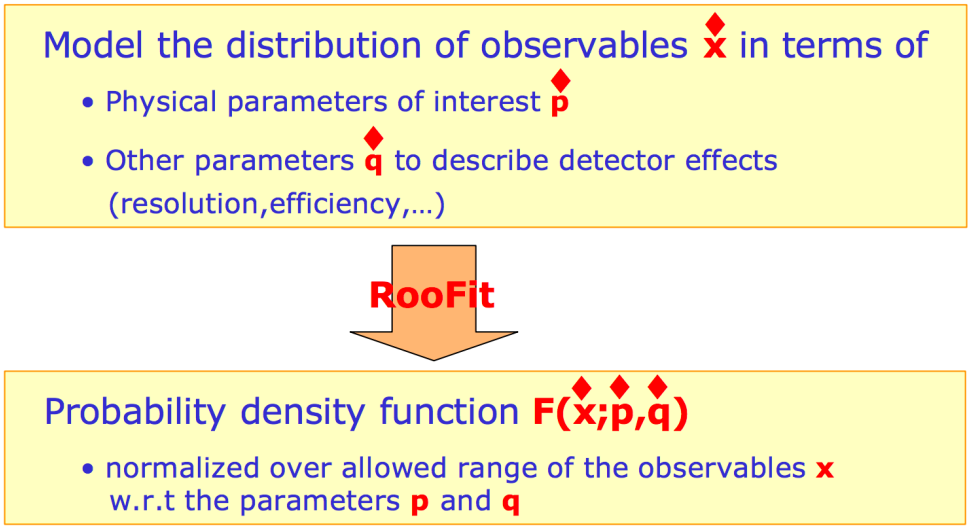
- ROOT (at least v6-06-06): https://root.cern.ch/building-root
- Jupyter: http://jupyter.readthedocs.io/en/latest/install.html

Alternatively, you can grab a VM Ubuntu with the above already installed from here (the admin pword is "stats"): https://cernbox.cern.ch/index.php/s/dIm4WLZU3jyjL4x

You can get this notebook from GitHub: https://github.com/nucleosynthesis/RooFitBasics

Open any notebook by typing in a terminal `root --notebook`

RooFit is a OO analysis environment built on ROOT. It is essentially a collection of classes designed to augment root for data modeling whose aim is summarised below (shamelessly stolen from Wouter Verkerke)...



alt text

We will cover a few of the basics in the session today but note there are many more tutorials available at this link: https://root.cern.ch/root/html600/tutorials/roofit/index.html

1.1 RooFit objects

In Roofit, any variable, data point, function, PDF ... is represented by a c++ object

The most basic of these is the `RooRealVar`. Let's create one which will represent the mass of some hypothetical particle, we name it and give it an initial starting value and range.

```
In [1]: RooRealVar MH("MH","mass of the Hypothetical Boson (H-boson) in GeV",125,120,130);
        MH.Print();
```

```
RooFit v3.60 -- Developed by Wouter Verkerke and David Kirkby
                Copyright (C) 2000-2013 NIKHEF, University of California & Stanford University
                All rights reserved, please read http://roofit.sourceforge.net/license.txt

RooRealVar::MH = 125  L(120 - 130)
```

ok, great. This variable is now an object we can play around with. We can access this object and modify it's properties, such as its value.

```
In [2]: MH.setVal(130);
        MH.getVal();
```

In particle detectors we typically don't observe this particle mass but usually define some observable which is *sensitive* to this mass. Lets assume we can detect and reconstruct the decay products of the H-boson and measure the invariant mass of those particles. We need to make another variable which represents that invariant mass

```
In [3]: RooRealVar mass("m","m (GeV)",100,80,200);
        mass.Print();

RooRealVar::m = 100  L(80 - 200)
```

In the perfect world we would perfectly measure the exact mass of the particle in every single event. However, our detectors are usually far from perfect so there will be some resolution effect. Lets assume the resolution of our measurement of the invariant mass is 10 GeV and call it "sigma"

```
In [4]: RooRealVar sigma("resolution","#sigma",10,0,20);
        sigma.Print();

RooRealVar::resolution = 10  L(0 - 20)
```

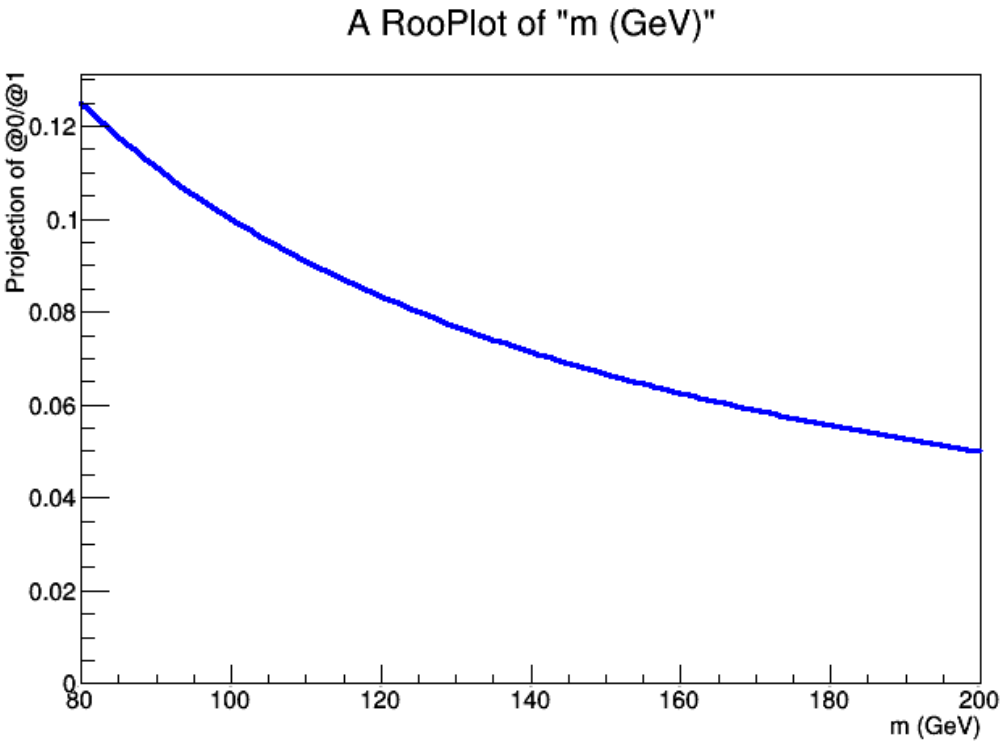
More exotic variables can be constructed out of these `RooRealVars` using `RooFormulaVars`. For example, suppose we wanted to make a function out of the variables which represented the relative resolution as a function of the hypothetical mass MH.

```
In [5]: RooFormulaVar func("R","@0/@1",RooArgList(sigma,mass));
        func.Print("v");

--- RooAbsArg ---
  Value State: DIRTY
  Shape State: DIRTY
  Attributes:
  Address: 0x7f03907ee000
  Clients:
  Servers:
    (0x7f03908e8848,V-) RooRealVar::resolution "#sigma"
    (0x7f03908e8430,V-) RooRealVar::m "m (GeV)"
  Proxies:
    actualVars ->
      1)  resolution
      2)           m
--- RooAbsReal ---

  Plot label is "R"
    --- RooFormula ---
    Formula: "@0/@1"
    (resolution,m)
```

Notice how there is a list of the variables we passed (the servers or "actual vars"). We can now plot the function. RooFit has a special plotting object `RooPlot` which keeps track of the objects (and their normalisations) which we want to draw. Since RooFit doesn't know the difference between which objects are/aren't dependant, we need to tell it.

Right now, we have the relative resolution as $R(m,\sigma)$ whereas we want to plot $R_\sigma(m)$.

```
In [6]: TCanvas *can = new TCanvas();

        //make the x-axis the "mass"
        RooPlot *plot = mass.frame();

        func.plotOn(plot);
        plot->Draw();

        can->Draw();
```



The main objects we are interested in using from RooFit are "probability denisty functions" or (PDFs). We can construct the PDF

$$f(m|M_H,\sigma)$$

as a simple Gaussian shape for example or a `RooGaussian` in RooFit language (think McDonald's logic, everything is a `RooSomethingOrOther`)

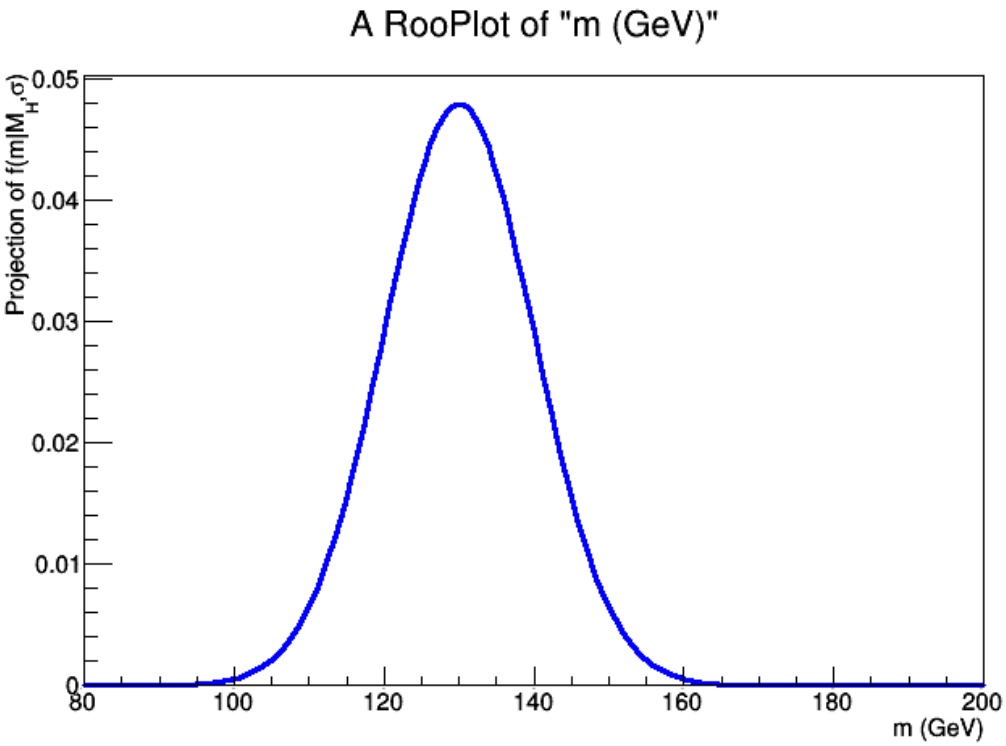
```
In [7]: RooGaussian gauss("gauss","f(m|M_{H},#sigma)",mass,MH,sigma);
        gauss.Print("V");
```

```
--- RooAbsArg ---
  Value State: DIRTY
  Shape State: DIRTY
  Attributes:
  Address: 0x7f03907ee480
  Clients:
  Servers:
    (0x7f03908e8430,V-) RooRealVar::m "m (GeV)"
    (0x7f03908e8018,V-) RooRealVar::MH "mass of the Hypothetical Boson (H-boson) in GeV"
    (0x7f03908e8848,V-) RooRealVar::resolution "#sigma"
  Proxies:
    x -> m
    mean -> MH
    sigma -> resolution
--- RooAbsReal ---

  Plot label is "gauss"
--- RooAbsPdf ---
Cached value = 0
```

Notice how the gaussian PDF, like the `RooFormulaVar` depends on our `RooRealVar` objects, these are its servers. Its evaluation will depend on their values.

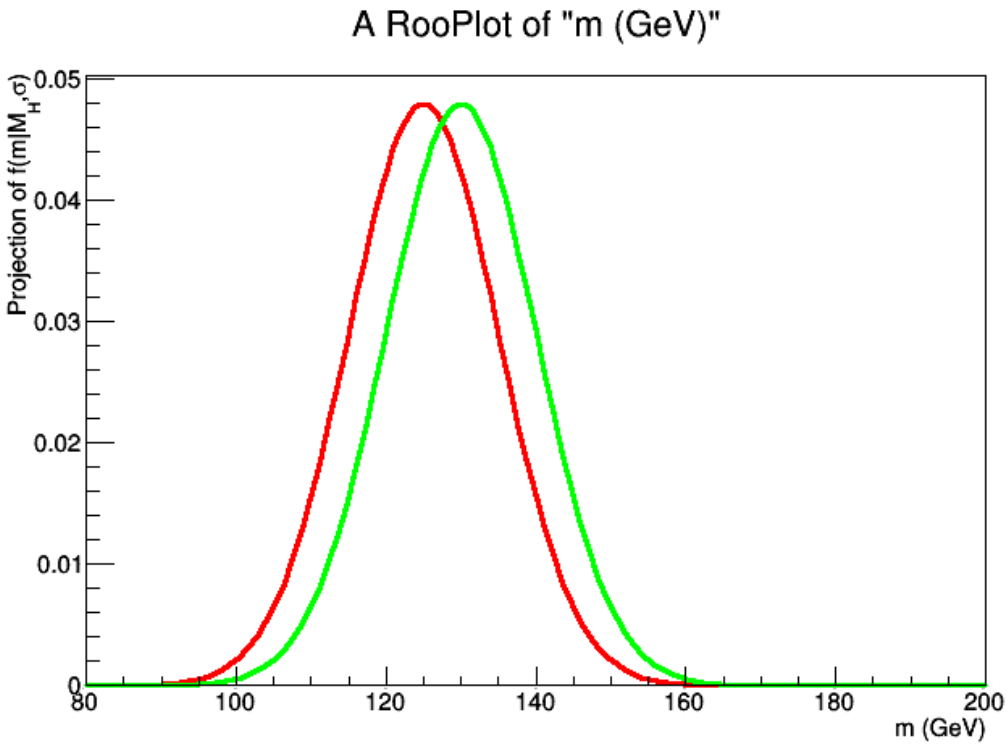
The main difference between PDFs and Functions in RooFit is that PDFs are automatically normalised to unitiy, hence they represent a probability density, you don't need to normalise yourself. Lets plot it for the current values of m and σ

```
In [8]: plot = mass.frame();

        gauss.plotOn(plot);
        plot->Draw();

        can->Draw();
```



If we change the value of MH, the PDF gets updated at the same time.

In [9]:
```
MH.setVal(125);
gauss.plotOn(plot,RooFit::LineColor(kRed));

MH.setVal(135);
gauss.plotOn(plot,RooFit::LineColor(kGreen));

plot->Draw();

can->Update();
can->Draw();
```



PDFs can be used to generate Monte Carlo data. One of the benefits of RooFit is that to do so only uses a single line of code!

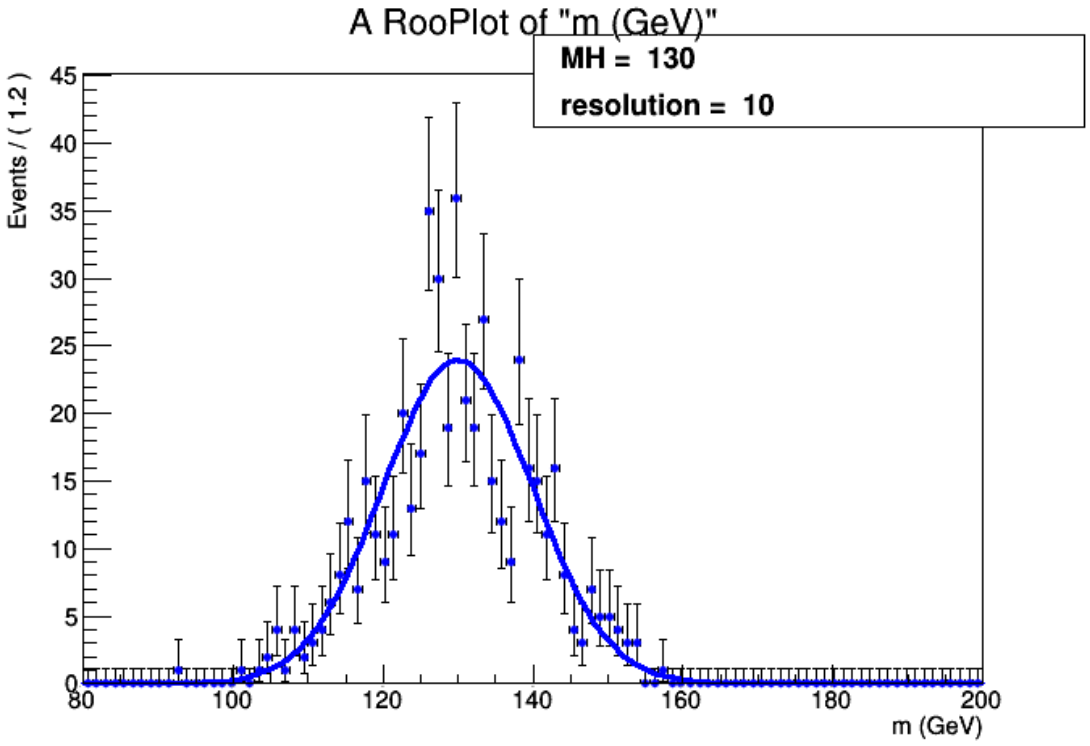
As before, we have to tell RooFit which variables to generate in (e.g which are the observables for an experiment). In this case, each of our events will be a single value of "mass" m.

```
In [10]: RooDataSet *data = (RooDataSet*) gauss.generate(RooArgSet(mass),500);
         //The arguments are the set of observables, follwed by the number of events

         data->Print();
```

```
RooDataSet::gaussData[m] = 500 entries
```

Now we can plot the data as with other RooFit objects.

```
In [11]: plot = mass.frame();

         data->plotOn(plot);
         gauss.plotOn(plot);
         gauss.paramOn(plot);

         plot->Draw();
         can->Update();
         can->Draw();
```



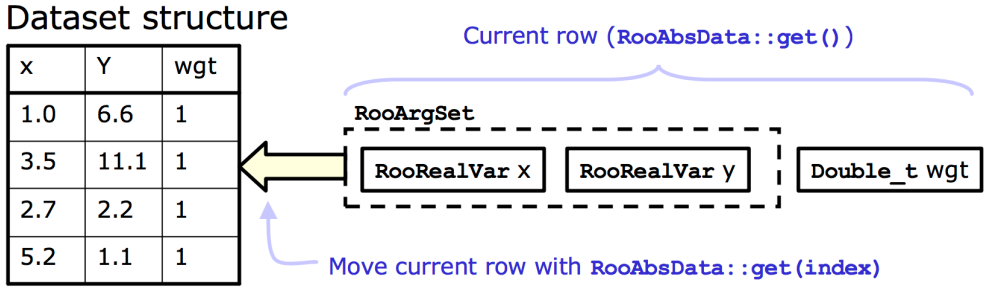
Of course we're not in the business of generating MC events, but collecting real data!. Next we will look at using real data in RooFit...

1.2 RooFit datasets

A dataset is essentially just a collection of points in N-dimensional (N-observables) space. There are two basic implementations in RooFit,

1) an "unbinned" dataset - `RooDataSet`
2) a "binned" dataset - `RooDataHist`

both of these use the same basic structure as below



alt text

Lets create an empty dataset where the only observable, the mass. Points can be added to the dataset one by one ...

```
In [12]: RooDataSet mydata("dummy","My dummy dataset",RooArgSet(mass));
         // We've made a dataset with one observable (mass)

         mass.setVal(123.4);
         mydata.add(RooArgSet(mass));
         mass.setVal(145.2);
         mydata.add(RooArgSet(mass));
         mass.setVal(170.8);
         mydata.add(RooArgSet(mass));


         mydata.Print();

RooDataSet::dummy[m] = 3 entries
```

There are also other ways to manipulate datasets in this way as shown in the diagram below

- Appending

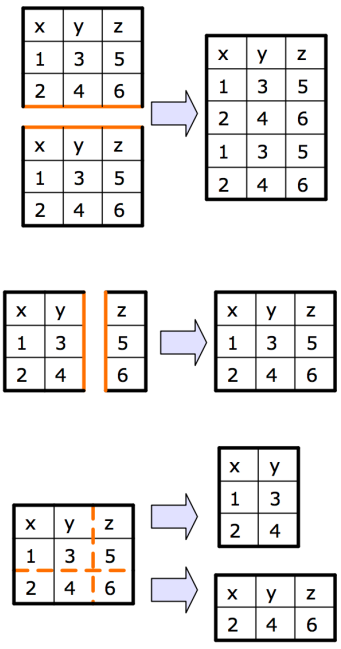
```
RooDataSet d1("d1","d1",RooArgSet(x,y,z));
RooDataSet d2("d2","d2",RooArgSet(x,y,z));

d1.append(d2) ;
```

- Merging

```
RooDataSet d1("d1","d1",RooArgSet(x,y) ;
RooDataSet d2("d2","d2",RooArgSet(z)) ;

d1.merge(d2) ;
```

- Reducing

```
RooDataSet d1("d1","d1",RooArgSet(x,y,z) ;

RooDataSet* d2 = d1.reduce(RooArgSet(x,y));

RooDataSet* d3 = d1.reduce("x>1");
```



alt text

Luckily there are also Constructors for a `RooDataSet` from a `TTree` and for a `RooDataHist` from a `TH1` so its simple to convert from your usual ROOT objects.

Let's take an example dataset put together already.

```
In [13]: TFile *file = TFile::Open("tutorial.root");
         file->ls();
```

```
TFile**         tutorial.root
 TFile*         tutorial.root
  KEY: RooWorkspace    workspace;1    Tutorial Workspace
  KEY: TProcessID      ProcessID0;1   48737500-e7e5-11e6-be6f-0d0011acbeef
```

Inside the file, there is something called a `RooWorkspace`. This is just the RooFit way of keeping a persistent link between the objects for a model. It is a very useful way to share data and PDFs/functions etc among CMS collaborators.

Let's take a look at it. It contains a `RooDataSet` and one variable. This time we called our variable (or observable) `CMS_hgg_mass`, let's assume now that this is the invariant mass of photon pairs where we assume our H-boson decays to photons.

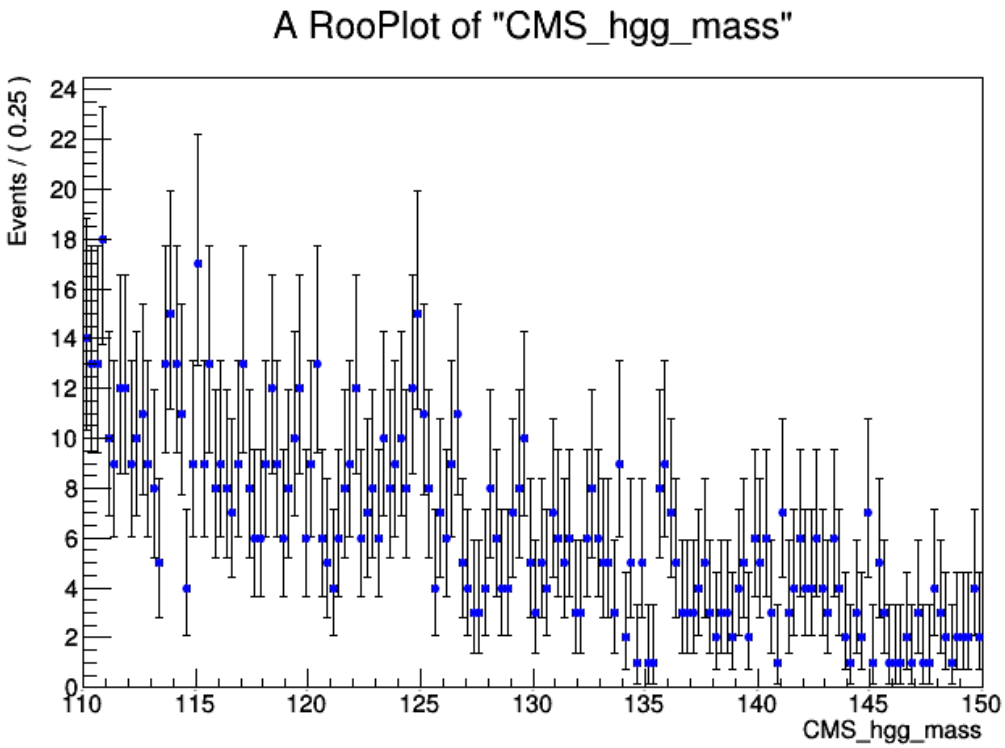
```
In [14]: RooWorkspace *wspace = (RooWorkspace*) file->Get("workspace");
         wspace->Print("v");
```

```
RooWorkspace(workspace) Tutorial Workspace contents

variables
---------
(CMS_hgg_mass)

datasets
--------
RooDataSet::dataset(CMS_hgg_mass)
```

Let's have a look at the data. The `RooWorkspace` has several accessor functions, we will use the `RooWorkspace::data` one.

There are also `RooWorkspace::var`, `RooWorkspace::function` and `RooWorkspace::pdf` with (hopefully) obvious purposes.

```
In [15]: RooDataSet *hgg_data = (RooDataSet*) wspace->data("dataset");
         RooRealVar *hgg_mass = (RooRealVar*) wspace->var("CMS_hgg_mass");

         plot = hgg_mass->frame();

         hgg_data->plotOn(plot,RooFit::Binning(160));
         // Here we've picked a certain number of bins just for plotting purposes

         plot->Draw();
         can->Update();
         can->Draw();
```



1.3 Likelihoods and Fitting to data

The data we have in our file doesn't look like a Gaussian distribution. Instead, we could probably use something like an exponential to describe it.

There is an exponential pdf already in RooFit (yep you guessed it) `RooExponential`. For a pdf, we only need one parameter which is the exponential slope α so our pdf is,

$$f(m|\alpha) = \frac{1}{N} e^{-\alpha m}$$

Where of course, $N = \int_{110}^{150} e^{-\alpha m} dm$ is the normalisation constant.

You can fund a bunch of available RooFit functions here: https://root.cern.ch/root/html/ROOFIT_ROOFIT_Index.html

There is also support for a generic pdf in the form of a `RooGenericPdf`, check this link: https://root.cern.ch/doc/v608/classRooGenericPdf.html

```
In [16]: RooRealVar alpha("alpha","#alpha",-0.05,-0.2,0.01);
         RooExponential expo("exp","exponential function",*hgg_mass,alpha);
```

We can use RooFit to tell us to estimate the value of α using this dataset. You will learn more about parameter estimation but for now we will just assume you know about maximising likelihoods. This maximum likelihood estimator is common in HEP and is known to give unbiased estimates for things like distribution means etc.

This also introduces the other main use of PDFs in RooFit. They can be used to construct likelihoods easily.

The likelihood $\mathcal{L}$ is defined for a particluar dataset (and model) as being proportional to the probability to observe the data assuming some pdf. For our data, the probability to observe an event with a value in an interval bounded by a and b is given by,

$$P\left(m \,\epsilon\, [a,b]\right) = \int_a^b f(m|\alpha)dm$$

As that interval shrinks we can say this probability just becomes equal to $f(m|\alpha)dm$

The probability to observe the dataset we have is given by the product of such probabilities for each of our data points, so that

$$\mathcal{L}(\alpha) \propto \prod_i f(m_i|\alpha)$$

Note that for a specific dataset, the dm factors which should be there are constnant. They can therefore be absorbed into the constant of proportionality!

The maximum likelihood esitmator for α, usually written as $\hat{\alpha}$, is found by maximising $L(\alpha)$.

Note that this won't depend on the value of the constant of proportionality so we can ignore it. This is true in most scenarios because usually only the ratio of likelihoods is needed, in which the constant factors out.

Obviously this multiplication of exponentials can lead to very large (or very small) numbers which can lead to numerical instabilities. To avoid this, we can take logs of the likelihood. Its also common to multiply this by -1 and minimize the resulting Negative Log Likelihood : $-Log\mathcal{L}(\alpha)$.

RooFit can construct the NLL for us.

```
In [17]: RooNLLVar *nll = (RooNLLVar*) expo.createNLL(*hgg_data);
         nll->Print("v");

--- RooAbsArg ---
  Value State: DIRTY
  Shape State: DIRTY
  Attributes:
  Address: 0x7f0364ead1b0
  Clients:
  Servers:
    (0x7f0374016000,V-) RooRealVar::alpha "#alpha"
  Proxies:
    paramSet ->
      1)  alpha
--- RooAbsReal ---

  Plot label is "nll_exp_dataset"
```

Notice that the NLL object knows which RooRealVar is the parameter because it doesn't find that one in the dataset. This is how RooFit distiguishes between observables and parameters.

RooFit has an interface to Minuit via the `RooMinimizer` class which takes the NLL as an argument.

```
In [18]: RooMinimizer minim(*nll);
         minim.minimize("Minuit","migrad");
         // Minuit is the program and "migrad" is
         // the minimization routine which uses gradient descent
```

```
**********
**    1 **SET PRINT           1
**********
**********
**    2 **SET NOGRAD
**********
PARAMETER DEFINITIONS:
   NO.   NAME         VALUE      STEP SIZE      LIMITS
    1 alpha       -5.00000e-02  2.10000e-02   -2.00000e-01  1.00000e-02
**********
**    3 **SET ERR         0.5
**********
**********
**    4 **SET PRINT           1
**********
**********
**    5 **SET STR           1
**********
NOW USING STRATEGY  1: TRY TO BALANCE SPEED AGAINST RELIABILITY
**********
**    6 **MIGRAD         500           1
**********
FIRST CALL TO USER FUNCTION AT NEW START POINT, WITH IFLAG=4.
START MIGRAD MINIMIZATION.  STRATEGY  1.  CONVERGENCE WHEN EDM .LT. 1.00e-03
FCN=3589.52 FROM MIGRAD    STATUS=INITIATE        4 CALLS           5 TOTAL
                    EDM= unknown      STRATEGY= 1      NO ERROR MATRIX
 EXT PARAMETER               CURRENT GUESS       STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  alpha       -5.00000e-02   2.10000e-02   2.24553e-01  -9.91191e+01
                                ERR DEF= 0.5
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=3584.68 FROM MIGRAD    STATUS=CONVERGED      18 CALLS          19 TOTAL
                    EDM=1.4449e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                       STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  alpha       -4.08262e-02   2.91959e-03   1.33905e-03  -3.70254e-03
                                ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  1    ERR DEF=0.5
 8.527e-06
```

RooFit has found the best fit value of alpha for this dataset. It also estimates an uncertainty on

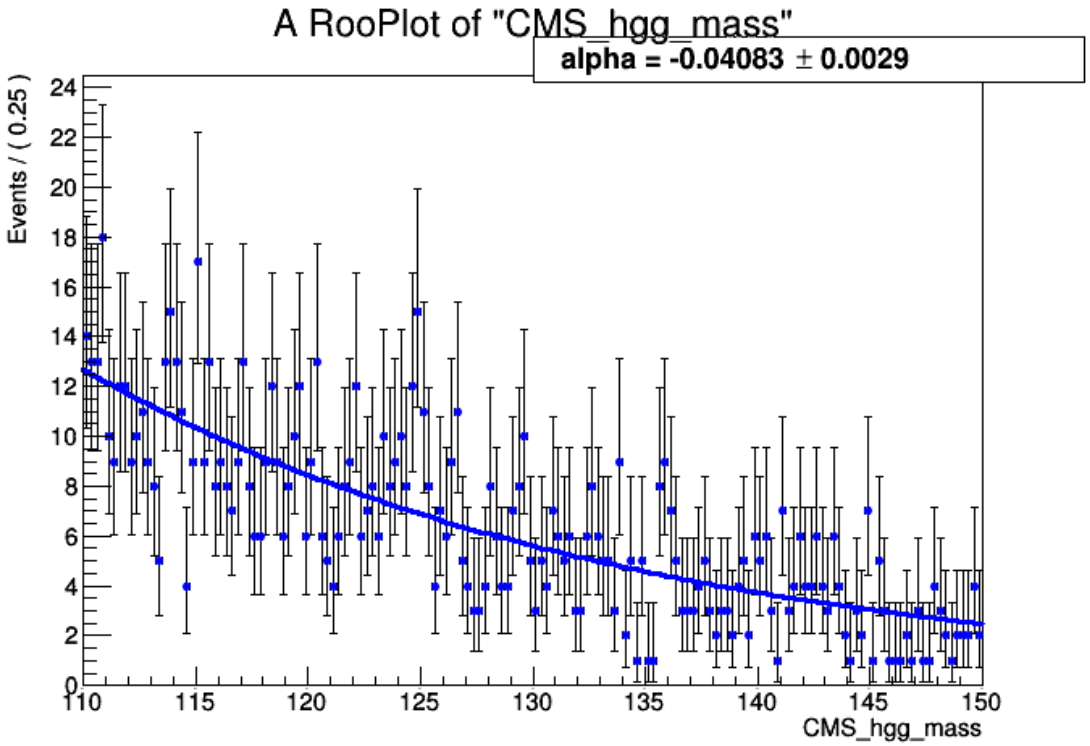
alpha using the Hessian matrix from the fit.

```
In [19]: alpha.Print("v");
```

```
--- RooAbsArg ---
  Value State: clean
  Shape State: clean
  Attributes:
  Address: 0x7f0374016000
  Clients:
    (0x7f0374016418,V-) RooExponential::exp "exponential function"
    (0x7f0364ead1b0,V-) RooNLLVar::nll_exp_dataset "-log(likelihood)"
    (0x7f0364f79a40,V-) RooExponential::exp "exponential function"
    (0x7f0364eb7070,V-) RooRealIntegral::exp_Int[CMS_hgg_mass] "Integral of exponential function
  Servers:
  Proxies:
--- RooAbsReal ---

  Plot label is "alpha"
--- RooAbsRealLValue ---
  Fit range is [ -0.2 , 0.01 ]
--- RooRealVar ---
  Error = 0.00291959
```

Lets plot the resulting exponential on the data. Notice that the value of $\hat{\alpha}$ is used for the exponential.

```
In [20]: expo.plotOn(plot);
         expo.paramOn(plot);
         plot->Draw();
         can->Update();
         can->Draw();
```



It looks like there could be a small region near 125 GeV for which our fit doesn't quite go through the points. Maybe our hypothetical H-boson isn't so hypothetical after all!

Let's see what happens if we include some resonant signal into the fit. We can take our Gaussian function again and use that as a signal model. A reasonable value for the resolution of a resonant signal with a mass around 125 GeV decaying to a pair of photons is around a GeV.

```
In [21]: sigma.setVal(1.);
         sigma.setConstant();

         MH.setVal(125);
         MH.setConstant();

         RooGaussian hgg_signal("signal","Gaussian PDF",*hgg_mass,MH,sigma);
```

By setting these parameters constant, RooFit knows (either when creating the NLL by hand or when using `fitTo`) that there is not need to fit for these parameters.

We need to add this to our exponential model and fit a "Sigmal+Background model" by creating a `RooAddPdf`. In RooFit there are two ways to add PDFs, recursively where the fraction of yields for the signal and background is a parameter or absolutely where each PDF has its own normalisation. We're going to use the second one.

```
In [22]: RooRealVar norm_s("norm_s","N_{s}",0,500);
         RooRealVar norm_b("norm_b","N_{b}",0,1000);
```

```
         const RooArgList components(hgg_signal,expo);
         const RooArgList coeffs(norm_s,norm_b);

         RooAddPdf model("model","f_{s+b}",components,coeffs);
         model.Print("v");
```

```
--- RooAbsArg ---
  Value State: DIRTY
  Shape State: DIRTY
  Attributes:
  Address: 0x7f035d237830
  Clients:
  Servers:
    (0x7f0374016928,V-) RooGaussian::signal "Gaussian PDF"
    (0x7f035d237000,V-) RooRealVar::norm_s "N_{s}"
    (0x7f0374016418,V-) RooExponential::exp "exponential function"
    (0x7f035d237418,V-) RooRealVar::norm_b "N_{b}"
  Proxies:
    !refCoefNorm ->
    !pdfs ->
      1)  signal
      2)     exp
    !coefficients ->
      1)  norm_s
      2)  norm_b
--- RooAbsReal ---

  Plot label is "model"
--- RooAbsPdf ---
Cached value = 0
```

A nice feature of RooFit is that once we've constructed a pdf like this, we can import this new model into our `RooWorkspace` and show off our new discovery to our LHC friends (if we weren't about 5 years too late!)

```
In [23]: wspace->import(model);
         wspace->Print("v");
```

```
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooAddPdf::model
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooGaussian::signal
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooRealVar::MH
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooRealVar::resolution
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooRealVar::norm_s
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooExponential::exp
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooRealVar::alpha
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooRealVar::norm_b

RooWorkspace(workspace) Tutorial Workspace contents
```

```
variables
---------
(CMS_hgg_mass,MH,alpha,norm_b,norm_s,resolution)

p.d.f.s
-------
RooExponential::exp[ x=CMS_hgg_mass c=alpha ] = 0.00246917
RooAddPdf::model[ norm_s * signal + norm_b * exp ] = 0.00164611
RooGaussian::signal[ x=CMS_hgg_mass mean=MH sigma=resolution ] = 2.14505e-106

datasets
--------
RooDataSet::dataset(CMS_hgg_mass)
```

Ok now lets fit the model. Note this time we add the option `Extended()` which tells RooFit that we care about the overall number of observed events in the data n too. It will add an additional Poisson term in the likelihood to account for this so our likelihood this time looks like,

$$L_{s+b}(N_s, N_b, \alpha) = \frac{N_s + N_b^n e^{N_s+N_b}}{n!} \cdot \prod_i^n [c f_s(m_i|M_H, \sigma) + (1-c) f_b(m_i|\alpha)]$$

where $c = \frac{N_s}{N_s + N_b}$, $f_s(m|M_H, \sigma)$ is the Gaussian signal pdf and $f_b(m|\alpha)$ is the exponential pdf. Remember that M_H and σ are fixed so that they are no longer parameters of the likelihood.

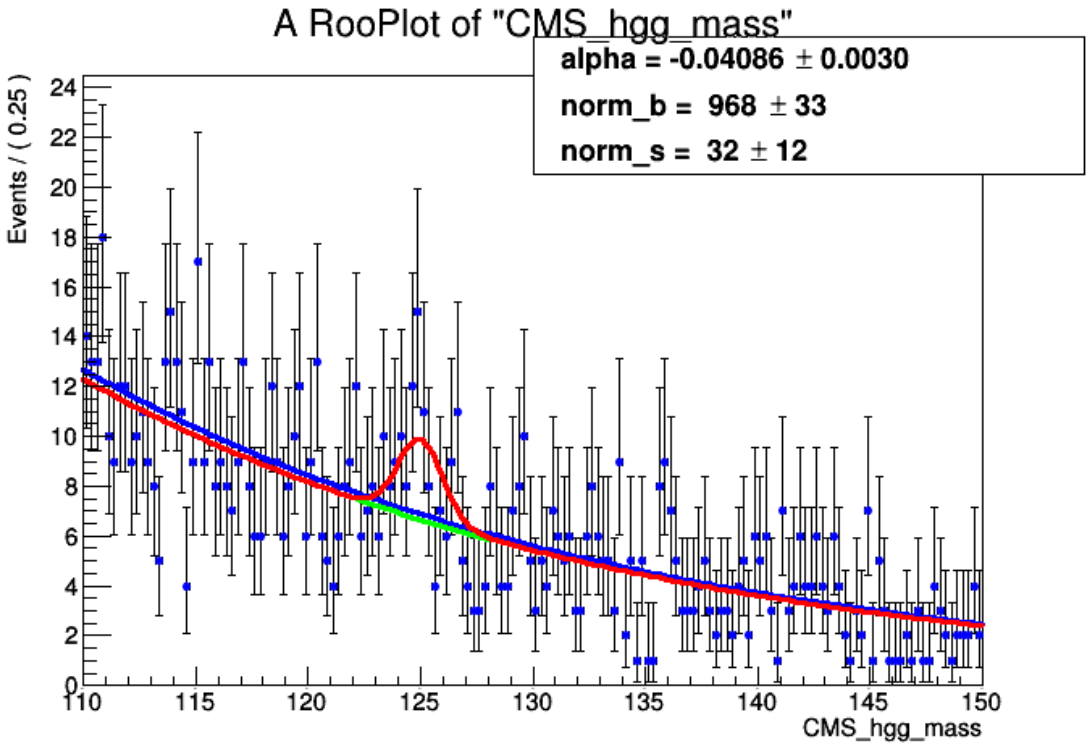
There is a simpler interface for maximum likelihood fits which is the `RooAbsPdf::fitTo` method. With this simple method, RooFit will construct the negative log-likelihood function, from the pdf, and minimize all of the free parameters in one step...

```
In [24]: model.fitTo(*hgg_data,RooFit::Extended());

[#1] INFO:Minization -- RooMinuit::optimizeConst: activating const optimization
[#1] INFO:Minization --  The following expressions have been identified as constant and will be
[#1] INFO:Minization --  The following expressions will be evaluated in cache-and-track mode: (e
 **********
 **   13 **MIGRAD        1500           1
 **********
 FIRST CALL TO USER FUNCTION AT NEW START POINT, WITH IFLAG=4.
 START MIGRAD MINIMIZATION.  STRATEGY  1.  CONVERGENCE WHEN EDM .LT. 1.00e-03
 FCN=-2120.78 FROM MIGRAD    STATUS=INITIATE       10 CALLS          11 TOTAL
                     EDM= unknown      STRATEGY= 1      NO ERROR MATRIX
  EXT PARAMETER               CURRENT GUESS       STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  alpha       -4.08262e-02   2.91959e-03   3.24715e-02   1.08581e+00
   2  norm_b       5.00000e+02   1.00000e+02   2.01358e-01  -3.78970e+02
   3  norm_s       2.50000e+02   5.00000e+01   2.01358e-01   1.28969e+02
                               ERR DEF= 0.5
```

```
 MIGRAD MINIMIZATION HAS CONVERGED.
 MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
 COVARIANCE MATRIX CALCULATED SUCCESSFULLY
 FCN=-2327.53 FROM MIGRAD    STATUS=CONVERGED      94 CALLS          95 TOTAL
                     EDM=1.69352e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  alpha       -4.08642e-02   2.96855e-03   1.10067e-03   3.14027e-02
   2  norm_b       9.67779e+02   3.25875e+01   5.71058e-03   3.86347e-03
   3  norm_s       3.22413e+01   1.17999e+01   3.08089e-03  -1.85003e-03
                               ERR DEF= 0.5
 EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  3    ERR DEF=0.5
  8.816e-06  1.296e-04 -1.306e-04
  1.296e-04  1.074e+03 -1.066e+02
 -1.306e-04 -1.066e+02  1.397e+02
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2      3
        1  0.00374   1.000  0.001 -0.004
        2  0.27520   0.001  1.000 -0.275
        3  0.27522  -0.004 -0.275  1.000
 **********
 **   18 **HESSE        1500
 **********
 COVARIANCE MATRIX CALCULATED SUCCESSFULLY
 FCN=-2327.53 FROM HESSE     STATUS=OK             16 CALLS         111 TOTAL
                     EDM=1.6964e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                INTERNAL      INTERNAL
  NO.   NAME      VALUE            ERROR       STEP SIZE       VALUE
   1  alpha       -4.08642e-02   2.96856e-03   2.20133e-04   5.41683e-01
   2  norm_b       9.67779e+02   3.26103e+01   2.28423e-04   1.20984e+00
   3  norm_s       3.22413e+01   1.18078e+01   1.23236e-04  -1.05730e+00
                               ERR DEF= 0.5
 EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  3    ERR DEF=0.5
  8.816e-06  1.353e-04 -1.353e-04
  1.353e-04  1.076e+03 -1.076e+02
 -1.353e-04 -1.076e+02  1.399e+02
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2      3
        1  0.00387   1.000  0.001 -0.004
        2  0.27745   0.001  1.000 -0.277
        3  0.27747  -0.004 -0.277  1.000
[#1] INFO:Minization -- RooMinuit::optimizeConst: deactivating const optimization
```

In [25]:
```
model.plotOn(plot,RooFit::Components("exp"),RooFit::LineColor(kGreen));
model.plotOn(plot,RooFit::LineColor(kRed));
model.paramOn(plot);
```

```
can->Clear();
plot->Draw();
can->Update();
can->Draw();
```




```
[#1] INFO:Plotting -- RooAbsPdf::plotOn(model) directly selected PDF components: (exp)
[#1] INFO:Plotting -- RooAbsPdf::plotOn(model) indirectly selected PDF components: ()
```

What about if we also fit for the mass (M_H)? we can easily do this by removing the constant setting on MH...

```
In [26]: MH.setConstant(false);
         model.fitTo(*hgg_data,RooFit::Extended());
```

```
[#1] INFO:Minization -- RooMinuit::optimizeConst: activating const optimization
[#1] INFO:Minization --  The following expressions will be evaluated in cache-and-track mode: (s
 **********
 **    13 **MIGRAD        2000           1
 **********
 FIRST CALL TO USER FUNCTION AT NEW START POINT, WITH IFLAG=4.
 START MIGRAD MINIMIZATION.  STRATEGY  1.  CONVERGENCE WHEN EDM .LT. 1.00e-03
 FCN=-2327.53 FROM MIGRAD    STATUS=INITIATE       12 CALLS          13 TOTAL
                     EDM= unknown      STRATEGY= 1      NO ERROR MATRIX
```

```
 EXT PARAMETER                CURRENT GUESS       STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  MH           1.25000e+02   1.00000e+00   2.01358e-01   1.12741e+01
  2  alpha       -4.08642e-02   2.96856e-03   3.30080e-02   3.11789e-02
  3  norm_b       9.67779e+02   3.26103e+01   2.56477e-01   2.32889e-03
  4  norm_s       3.22413e+01   1.18078e+01   9.77951e-02  -1.18908e-03
                               ERR DEF= 0.5
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=-2327.96 FROM MIGRAD    STATUS=CONVERGED      67 CALLS          68 TOTAL
                     EDM=1.18454e-05    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                       STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  MH           1.24628e+02   3.98149e-01   2.66543e-03   2.49097e-02
  2  alpha       -4.07708e-02   2.97196e-03   1.10093e-03   8.35083e-02
  3  norm_b       9.66098e+02   3.25754e+01   5.96710e-03   5.19746e-04
  4  norm_s       3.39032e+01   1.18977e+01   3.03808e-03  -7.50002e-03
                               ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  4    ERR DEF=0.5
 1.589e-01 -3.890e-05  1.463e-01 -1.478e-01
-3.890e-05  8.836e-06 -2.020e-04  2.038e-04
 1.463e-01 -2.020e-04  1.073e+03 -1.072e+02
-1.478e-01  2.038e-04 -1.072e+02  1.420e+02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2      3      4
       1  0.04519   1.000 -0.033  0.011 -0.031
       2  0.03317  -0.033  1.000 -0.002  0.006
       3  0.27465   0.011 -0.002  1.000 -0.275
       4  0.27610  -0.031  0.006 -0.275  1.000
**********
**   18 **HESSE        2000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=-2327.96 FROM HESSE     STATUS=OK             23 CALLS          91 TOTAL
                     EDM=1.18354e-05    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   INTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR       STEP SIZE       VALUE
  1  MH           1.24628e+02   3.98102e-01   5.33087e-04  -7.45138e-02
  2  alpha       -4.07708e-02   2.97195e-03   2.20185e-04   5.42722e-01
  3  norm_b       9.66098e+02   3.25983e+01   2.38684e-04   1.20043e+00
  4  norm_s       3.39032e+01   1.19046e+01   1.21523e-04  -1.04393e+00
                               ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  4    ERR DEF=0.5
 1.588e-01 -3.888e-05  1.304e-01 -1.305e-01
-3.888e-05  8.836e-06 -1.953e-04  1.954e-04
 1.304e-01 -1.953e-04  1.074e+03 -1.082e+02
-1.305e-01  1.954e-04 -1.082e+02  1.421e+02
```

```
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2      3      4
        1  0.04275   1.000 -0.033  0.010 -0.027
        2  0.03314  -0.033  1.000 -0.002  0.006
        3  0.27693   0.010 -0.002  1.000 -0.277
        4  0.27805  -0.027  0.006 -0.277  1.000
[#1] INFO:Minization -- RooMinuit::optimizeConst: deactivating const optimization
```

Notice now the result for the fitted MH is not 125 and gets added to the fitted parameters since now it is floating.

We can get more information about the fit, via the `RooFitResult`, using the option `Save()`.

```
In [27]: RooFitResult *fit_res = (RooFitResult*) model.fitTo(*hgg_data,RooFit::Extended(),RooFit
         fit_res->Print("v");

[#1] INFO:Minization -- RooMinuit::optimizeConst: activating const optimization
[#1] INFO:Minization --  The following expressions will be evaluated in cache-and-track mode: (s
 **********
 **   13 **MIGRAD        2000           1
 **********
 FIRST CALL TO USER FUNCTION AT NEW START POINT, WITH IFLAG=4.
 START MIGRAD MINIMIZATION.  STRATEGY  1.  CONVERGENCE WHEN EDM .LT. 1.00e-03
 FCN=-2327.96 FROM MIGRAD    STATUS=INITIATE        8 CALLS           9 TOTAL
                     EDM= unknown      STRATEGY= 1      NO ERROR MATRIX
  EXT PARAMETER               CURRENT GUESS       STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  MH           1.24628e+02   3.98102e-01   7.99285e-02   2.51085e-02
   2  alpha       -4.07708e-02   2.97195e-03   3.30666e-02   8.32829e-02
   3  norm_b       9.66098e+02   3.25983e+01   2.24704e-01  -1.97473e-03
   4  norm_s       3.39032e+01   1.19046e+01   9.61797e-02  -6.87727e-03
                               ERR DEF= 0.5
 MIGRAD MINIMIZATION HAS CONVERGED.
 MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
 COVARIANCE MATRIX CALCULATED SUCCESSFULLY
 FCN=-2327.96 FROM MIGRAD    STATUS=CONVERGED      49 CALLS          50 TOTAL
                     EDM=3.9026e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                       STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  MH           1.24627e+02   3.98145e-01   2.66086e-03   2.47732e-04
   2  alpha       -4.07785e-02   2.97195e-03   1.10099e-03   2.75281e-03
   3  norm_b       9.66108e+02   3.25801e+01   5.79852e-03   3.35061e-03
   4  norm_s       3.39100e+01   1.18981e+01   3.03569e-03  -3.90940e-04
                               ERR DEF= 0.5
 EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  4    ERR DEF=0.5
  1.589e-01 -3.890e-05  1.439e-01 -1.453e-01
 -3.890e-05  8.836e-06 -2.012e-04  2.029e-04
  1.439e-01 -2.012e-04  1.073e+03 -1.072e+02
```

```
-1.453e-01  2.029e-04 -1.072e+02  1.420e+02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2      3      4
       1  0.04483   1.000 -0.033  0.011 -0.031
       2  0.03318  -0.033  1.000 -0.002  0.006
       3  0.27472   0.011 -0.002  1.000 -0.275
       4  0.27612  -0.031  0.006 -0.275  1.000
**********
**   18 **HESSE        2000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=-2327.96 FROM HESSE     STATUS=OK             23 CALLS          73 TOTAL
                    EDM=3.91404e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   INTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR       STEP SIZE       VALUE
  1  MH           1.24627e+02   3.98091e-01   1.06434e-04  -7.46654e-02
  2  alpha       -4.07785e-02   2.97195e-03   4.40396e-05   5.42636e-01
  3  norm_b       9.66108e+02   3.26023e+01   2.31941e-04   1.20049e+00
  4  norm_s       3.39100e+01   1.19046e+01   1.21428e-04  -1.04388e+00
                               ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  4    ERR DEF=0.5
 1.588e-01 -3.887e-05  1.247e-01 -1.248e-01
-3.887e-05  8.836e-06 -1.928e-04  1.928e-04
 1.247e-01 -1.928e-04  1.075e+03 -1.082e+02
-1.248e-01  1.928e-04 -1.082e+02  1.421e+02
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2      3      4
       1  0.04198   1.000 -0.033  0.010 -0.026
       2  0.03314  -0.033  1.000 -0.002  0.005
       3  0.27693   0.010 -0.002  1.000 -0.277
       4  0.27796  -0.026  0.005 -0.277  1.000
[#1] INFO:Minization -- RooMinuit::optimizeConst: deactivating const optimization

 RooFitResult: minimized FCN value: -2327.96, estimated distance to minimum: 3.91404e-07
               covariance matrix quality: Full, accurate covariance matrix
               Status : MIGRAD=0 HESSE=0

   Constant Parameter    Value
 -------------------  ------------
           resolution    1.0000e+00

   Floating Parameter  InitialValue    FinalValue +/-  Error     GblCorr.
 --------------------  ------------  --------------------------  --------
                   MH    1.2463e+02    1.2463e+02 +/-  3.98e-01  <none>
                alpha   -4.0771e-02   -4.0779e-02 +/-  2.97e-03  <none>
              norm_b    9.6610e+02    9.6611e+02 +/-  3.26e+01  <none>
              norm_s    3.3903e+01    3.3910e+01 +/-  1.19e+01  <none>
```

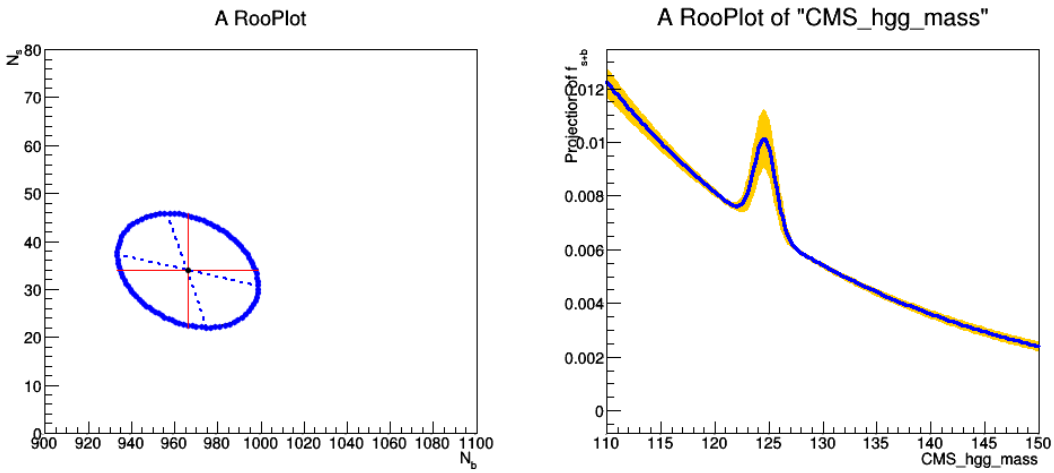
For example, we can get the Correlation Matrix from the fit result... Note that the order of the parameters are the same as listed in the "Floating Parameter" list above

```
In [28]: TMatrixDSym cormat = fit_res->correlationMatrix();
         cormat.Print();
```

```
4x4 matrix is as follows

     |      0    |      1    |      2    |      3    |
-------------------------------------------------------
   0 |          1    -0.03282    0.009548    -0.02626
   1 |   -0.03282           1   -0.001979     0.00544
   2 |   0.009548   -0.001979           1     -0.2769
   3 |   -0.02626     0.00544     -0.2769           1
```

And there even some fancy ways to visualise these results ...

```
In [29]: TCanvas *can2 = new TCanvas("c","c",1000,460);
         can2->Divide(2);

         can2->cd(1);

         RooPlot plot_err_mat(norm_b,norm_s,900,1100,0,80);
         fit_res->plotOn(&plot_err_mat,norm_b,norm_s,"MEVH12");
         plot_err_mat.Draw();

         can2->cd(2);

         plot = hgg_mass->frame();
         model.plotOn(plot,RooFit::VisualizeError(*fit_res,1),RooFit::FillColor(kOrange));
         model.plotOn(plot);
         plot->Draw();

         can2->Draw();
```



2 Removing Nuisance Parameters

In HEP, we often have the case where there are some parameters of the model which are physics parameters of interest, while others are known as nuisance parameters. In some cases, those parameters may have external constraints, coming from other data or some theoretical reasoning. This is how we often treat systematic uncertainties but we won't cover those today.

There are two schools of thought for removing nuisance parameters

- Frequentists use profiling
- Bayesians use marginalisation

In our case, the hypothesis mass M_H might be the thing which we can write papers on and win Nobel prizes for while we're not nearly so interested in the values of N_s, α or N_b.

The profile likelihood is one which removes the nuisances parameters by fitting them away. Note this is a major divide for Bayesians and Freqeuentist since removing nuisance parameters for Bayesians involves integrating (or marginalising) over them.

The profile likeihood is written as,

$$\mathcal{L}_p(M_H) = \mathcal{L}_{s+b}(M_H, \hat{\hat{N}}_s, \hat{\hat{N}}_b, \hat{\hat{\alpha}})$$

where the double hat notation, $\hat{\hat{\cdot}}$, denotes the values of the nuisance parameters which maximise L_{s+b} at a given value of M_H.

RooFit has a useful class under the RooStats libraries for profile likelihoods: `ProfileLikelihoodCalculator` : https://root.cern.ch/doc/v606/classRooStats_1_1ProfileLikelihoodCalculator.html

```
In [30]: MH.setRange(124,126);
         RooStats::ProfileLikelihoodCalculator plc(*hgg_data,model,RooArgSet(MH));
```

According to Wilks' theorem, -2 times the log of profile likelihood ratio,

$$-2\log\lambda = -2\log\frac{\mathcal{L}(M_H)}{\mathcal{L}(\hat{M}_H)}$$

should be distributed as a χ^2 with the number degrees of freedom equal to the number of parameters (in our case 1). This means that we can obtain the 1σ (or 68%) uncertainty by finding the interval for which

$$-\log\lambda(M_H) < 0.5$$

We can tell RooFit to find that interval for us,

```
In [31]: plc.SetConfidenceLevel(0.68);
         RooStats::LikelihoodInterval *interval = plc.GetInterval();

         std::cout << "\n\t hat{MH} = "
             << ((RooRealVar*)interval->GetBestFitParameters()->find("MH"))->getVal()
             << std::endl;

         std::cout << "\t 68% interval = ("
             << interval->LowerLimit(MH) << "," << interval->UpperLimit(MH) << ")"
             << std::endl;
```

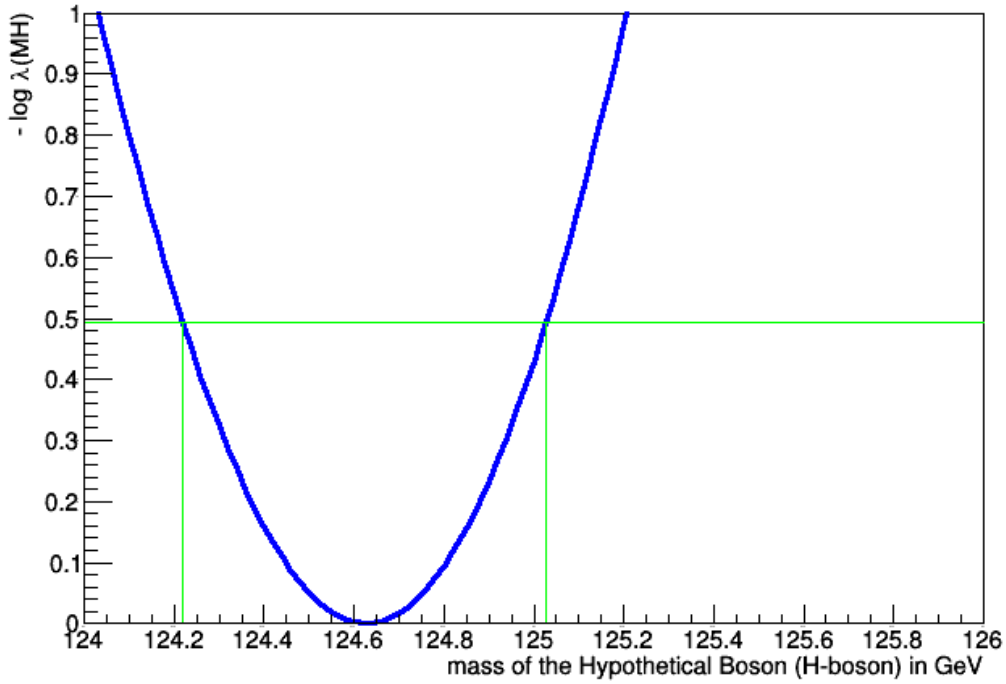
```
[#1] INFO:Minization -- p.d.f. provides expected number of events, including extended term in li
[#0] PROGRESS:Minization -- ProfileLikelihoodCalcultor::DoGLobalFit - find MLE
[#0] PROGRESS:Minization -- ProfileLikelihoodCalcultor::DoMinimizeNLL - using Minuit / Migrad wi
[#1] INFO:Minization -- RooMinimizer::optimizeConst: activating const optimization
[#1] INFO:Minization --  The following expressions will be evaluated in cache-and-track mode: (s
[#1] INFO:Minization --
  RooFitResult: minimized FCN value: -2327.96, estimated distance to minimum: 5.13199e-08
                covariance matrix quality: Full, accurate covariance matrix
                Status : MINIMIZE=0

    Floating Parameter    FinalValue +/-  Error
  --------------------  --------------------------
                    MH    1.2463e+02 +/-  3.86e-01
                 alpha   -4.0779e-02 +/-  2.97e-03
                norm_b    9.6609e+02 +/-  3.26e+01
                norm_s    3.3910e+01 +/-  1.19e+01


         hat{MH} = 124.627
         68% interval = (124.219,125.027)
```

We can see where these values come from by drawing the negative of the profile likelihood ratio.

```
In [32]: RooStats::LikelihoodIntervalPlot plotLI(interval);
```

```
        TCanvas *canLI = new TCanvas();
        plotLI.Draw();
        canLI->Draw();
```

```
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_model_dataset_Profile[MH]) Creating instance
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_model_dataset_Profile[MH]) determining minimu
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_model_dataset_Profile[MH]) minimum found at (
.
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_model_dataset_Profile[MH]) Creating instance
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_model_dataset_Profile[MH]) determining minimu
[#0] ERROR:InputArguments -- RooArgSet::checkForDup: ERROR argument with name MH is already in t
[#1] INFO:Minization -- RooProfileLL::evaluate(nll_model_dataset_Profile[MH]) minimum found at (
...
```



..

We can also obtain a Bayesian posterior density by using Bayes' theorem ...

$$P(M_H|\text{data}) = \frac{\int\int\int P(\text{data}|M_H, N_s, N_b, \alpha)\pi(M_H, N_s, N_b, \alpha)dN_s dN_b d\alpha}{P(\text{data})}$$

where $\pi(M_H, N_s, N_b, \alpha)$ is the prior probability density for our parameters. You might notice that $P(\text{data}|M_H, N_s, N_b, \alpha)$ is nothing more than our likelihood function $\mathcal{L}(M_H, N_s, N_b, \alpha)$!

Notice that I can use $=$ here because we normalise according to $P(\text{data})$, which is the integral over the likelihood (multiplied by the prior) with respect to all of its parameters.

We can get RooFit to calculate this posterior using the `BayesianCalculator` class but first, for this time, we are going to create a `ModelConfig` which keeps track of which parameters are POIs and which are nuisances/observables. This is useful when sharing your model between collegues.

```
In [33]: RooStats::ModelConfig mconfig("mymodel");
         mconfig.SetWorkspace(*wspace);

         mconfig.SetPdf("model");
         mconfig.SetParametersOfInterest("MH");
         mconfig.SetObservables("CMS_hgg_mass");
```

For Bayesian results, we need to set the prior density function (π). For now, let's assume no prior knowledge of the parameters - i.e the prior is constant within the range of the parameters and 0 elsewhere.

Note that choosing a suitable prior is a debated subject amongst Bayesians since the results can depend on what you choose as a prior. Worse still, if a flat prior is chosen, say for M_H, should it be flat in M_H, or flat in M_H^2 (eg since $M_H = 2\lambda v^2$ and maybe λ is the fundamental thing)?

```
In [34]: wspace->var("MH")->setRange(122,126);

         RooUniform flat_p("flat_prior","flat_prior",RooArgSet(MH,norm_s,norm_b,alpha));
         wspace->import(flat_p);

         mconfig.SetPriorPdf("flat_prior");

         mconfig.Print("v");
```

```
[#1] INFO:ObjectHandling -- RooWorkspace::import(workspace) importing RooUniform::flat_prior

=== Using the following for mymodel ===
Observables:             RooArgSet:: = (CMS_hgg_mass)
Parameters of Interest:  RooArgSet:: = (MH)
PDF:                     RooAddPdf::model[ norm_s * signal + norm_b * exp ] = 0.00164611
Prior PDF:               RooUniform::flat_prior[ x=(MH,norm_s,norm_b,alpha) ] = 1
```

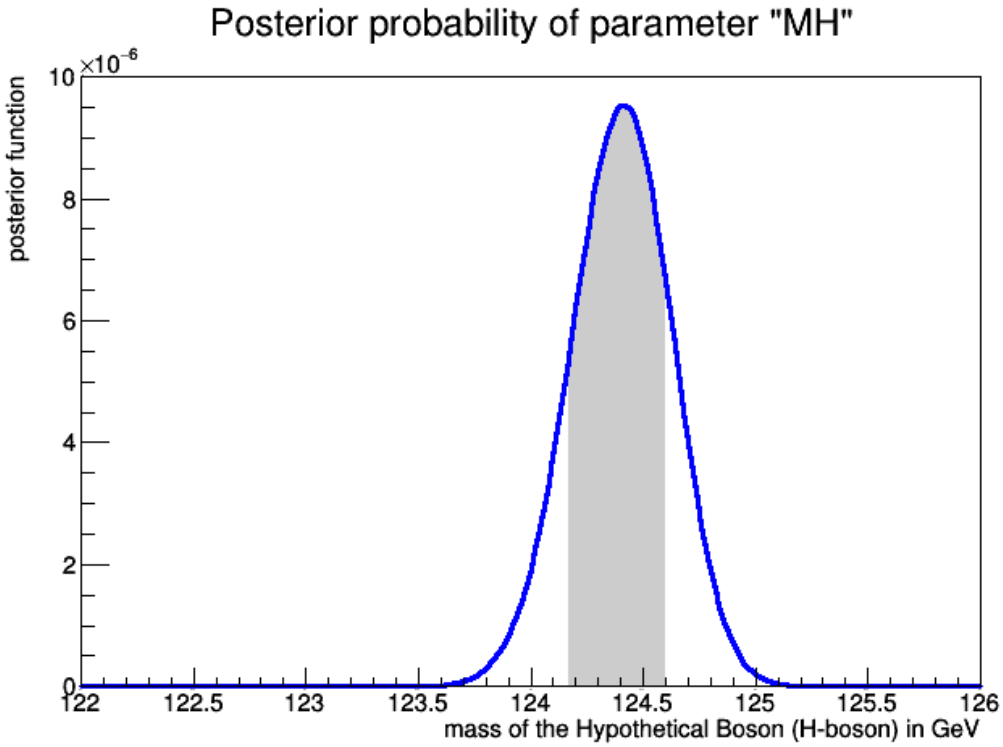
Now we can create the calculator and plot the posterior pdf...

```
In [35]: RooStats::BayesianCalculator bCalculator(*hgg_data,mconfig);
         bCalculator.SetConfidenceLevel(0.683);

         RooPlot *plotBC = bCalculator.GetPosteriorPlot();
```

```
[#1] INFO:Minization -- p.d.f. provides expected number of events, including extended term in li
[#1] INFO:Eval -- BayesianCalculator::GetPosteriorFunction :  nll value -2120.78 poi value = 125
[#1] INFO:Eval -- BayesianCalculator::GetPosteriorFunction : minimum of NLL vs POI for POI =  12
[#1] INFO:Minization -- p.d.f. provides expected number of events, including extended term in li
```

```
[#1] INFO:Minization --  Including the following contraint terms in minimization: (flat_prior)
[#1] INFO:Eval -- BayesianCalculator: Compute interval using RooFit:  posteriorPdf + createCdf +
[#1] INFO:NumericIntegration -- RooRealIntegral::init(_posteriorPdf_likelihood_times_prior_produ
[#1] INFO:NumericIntegration -- RooRealIntegral::init(_posteriorPdf_likelihood_times_prior_produ
[#1] INFO:Eval -- BayesianCalculator::GetInterval - found a valid interval : [124.172 , 124.601
```

```
In [36]: TCanvas *cBC = new TCanvas();
         plotBC->Draw();
         cBC->Draw();
```



This was the "central" interval, but we can also find the "shortest?such interval

```
In [37]: bCalculator.SetShortestInterval();

         RooStats::SimpleInterval *intervalBayes
             = (RooStats::SimpleInterval*)bCalculator.GetInterval();

         std::cout << "\n\t 68% (Bayes shortest) interval \t= ("
             << intervalBayes->LowerLimit() << "," << intervalBayes->UpperLimit() << ")"
             << std::endl;

         std::cout << "\t 68% (frequentist) interval \t= ("
             << interval->LowerLimit(MH) << "," << interval->UpperLimit(MH) << ")"
             << std::endl;
```

```
[#0] WARNING:Eval -- BayesianCalculator::GetInterval - recomputing interval for the same CL and
[#1] INFO:Eval -- BayesianCalculator - computing shortest interval with CL = 0.683
[#1] INFO:Eval -- BayesianCalculator - scan posterior function in nbins = 100
[#1] INFO:Eval -- BayesianCalculator::GetInterval - found a valid interval : [124.2 , 124.64 ]

        68% (Bayes shortest) interval         = (124.2,124.64)
        68% (frequentist) interval          = (124.219,125.027)
```

Note that the Bayesian interval isn't the same as our frequentist interval!

You will learn a more about Likelihoods and their use in Hypothesis Testing (Frequentist/Basyesian intervals, coverage and credibility, setting limits, discovery...) in future statistics Lectures. There are many more useful toos for performing them with the RooStats library https://twiki.cern.ch/twiki/bin/view/RooStats/WebHome - check them out!