Imperial College London

RooFit Tutorial

Matthew Kenzie
Imperial College London

Introduction

Imperial College London

- Computing lectures are dull
  - I will try to be brief and quick
  - These slides should serve more as reference
  - However as we talk through them I will provide many examples and explanations which are not in the slides
- The Basics
  - Observables and variables
    - RooAbsReal -> RooRealVar / RooFormulaVar
  - Filling datasets
    - RooAbsData -> RooDataSet / RooDataHist
  - Fitting datasets
    - RooAbsPdf -> RooGaussian / RooExponential / RooCBShape etc.
  - Integrating PDFs
  - Throwing toys (generating datasets)
  - RooWorkspace
  - RooPlot
- Advanced topics
  - MinNll and chi2
  - Component pdfs
    - RooAddPdf
  - More complex variables (RooFormulaVar)
  - Constraining fits across categories or to a control sample
    - RooSimultaneous / RooCategory

What I assume you already know

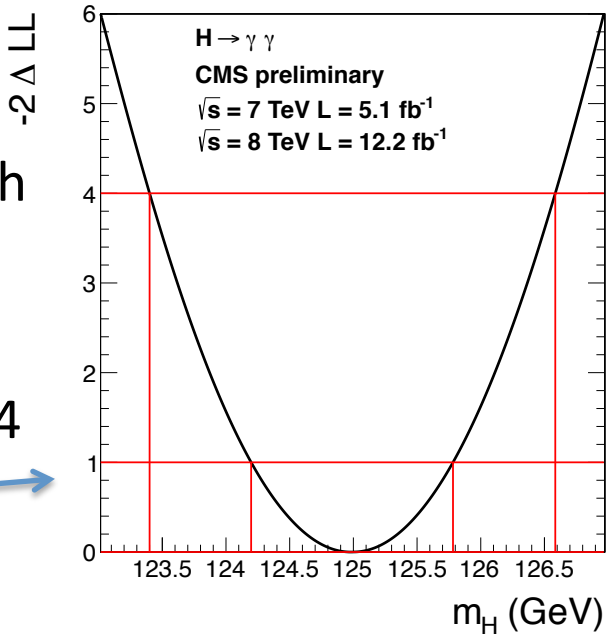
Imperial College London

- bash stuff (see appendix slides)
  - makefiles
  - simple .sh scripts to allow submission to batch
- ROOT stuff
  - TH1F, TTree, TGraph, TFile
- C++ stuff
  - Reading from .dat files (ifstream / ofstream)
    - Useful for reading and dumping parameter values and limits
  - Polymorphism
    - Virtual functions which allow a pointer to base class
  - vector
  - map
    - Very useful in RooFit for storing parameters which you may need to manipulate to coerce the fit
- Python
  - Not in this tutorial but it is useful for making quick scripts (plotting etc.) and RooFit works quite well in pyROOT
    - Quick to write
    - Slow to run

If you're not familiar with something here – ask!

Where do we begin?

- Usually with a TTree (what we call an ntuple)
  - A very stripped down version of the original datasets containing only the essentials you need for the analysis
  - Might have a RooWorkspace (more later) if someone else has already run it
- We want to fit some dataset of an observable (maybe more than one – multidimensional fit) with a pdf (probability distribution function)
  - Involves a minimization (either the χ^2 or the NLL)
  - RooFit uses likelihood because it can deal much better with low statistics and can be unbinned
  - You can force RooFit to use χ^2
  - Get 1σ and 2σ errors from where ΔNLL=1 and 4
  - E.g. measure Higgs mass in H→ϒϒ



About RooFit

Imperial College London

- Wrapper for Minuit
  - RooFit doesn't actually do the minimization itself. For this it uses TMinuit.
  - It uses Hesse and Migrad to get the errors
- RooFit is an "easy" to use wrapper with many built in operations and functions
- There are bugs in RooFit. If you are experiencing one then you may have to update to a newer version or revert to an older version
  - Be sure you can replicate it and submit a bug report
- RooFit fits PDFs not arbitrary functions
  - Probability Distribution Functions cannot go negative (can't have negative events)
  - So it isn't useful for fitting arbitrary functions
  - It is useful for fitting physics distributions
    - the most common in HEP is the invariant mass of something
- If it's not this or something simple use TH1::Fit as it's a lot simpler

Tips, tricks and comments

- Basically RooFit figures out all the parameters the fit depends on it scans across them and works out the likelihood.
- It puts the best fit result where the negative log likelihood is minimized and it uses a covariance matrix (how correlated are the input parameters) to work out the errors
- It can get stuck in local minima
- It can wonder off into ridiculous bits of phase space
- You will have a lot more success if you use sensible starting values and set sensible ranges for your fit parameters
  - As a consequence try and use physically motivated parameters and models in fits (as it's helps set constraints on the parameters)
- Often the fit will work but RooFit will warn you that it thinks the error matrix is inaccurate
  - If this is the case then do your own likelihood scan for the parameter of interest (POI) to get the errors.

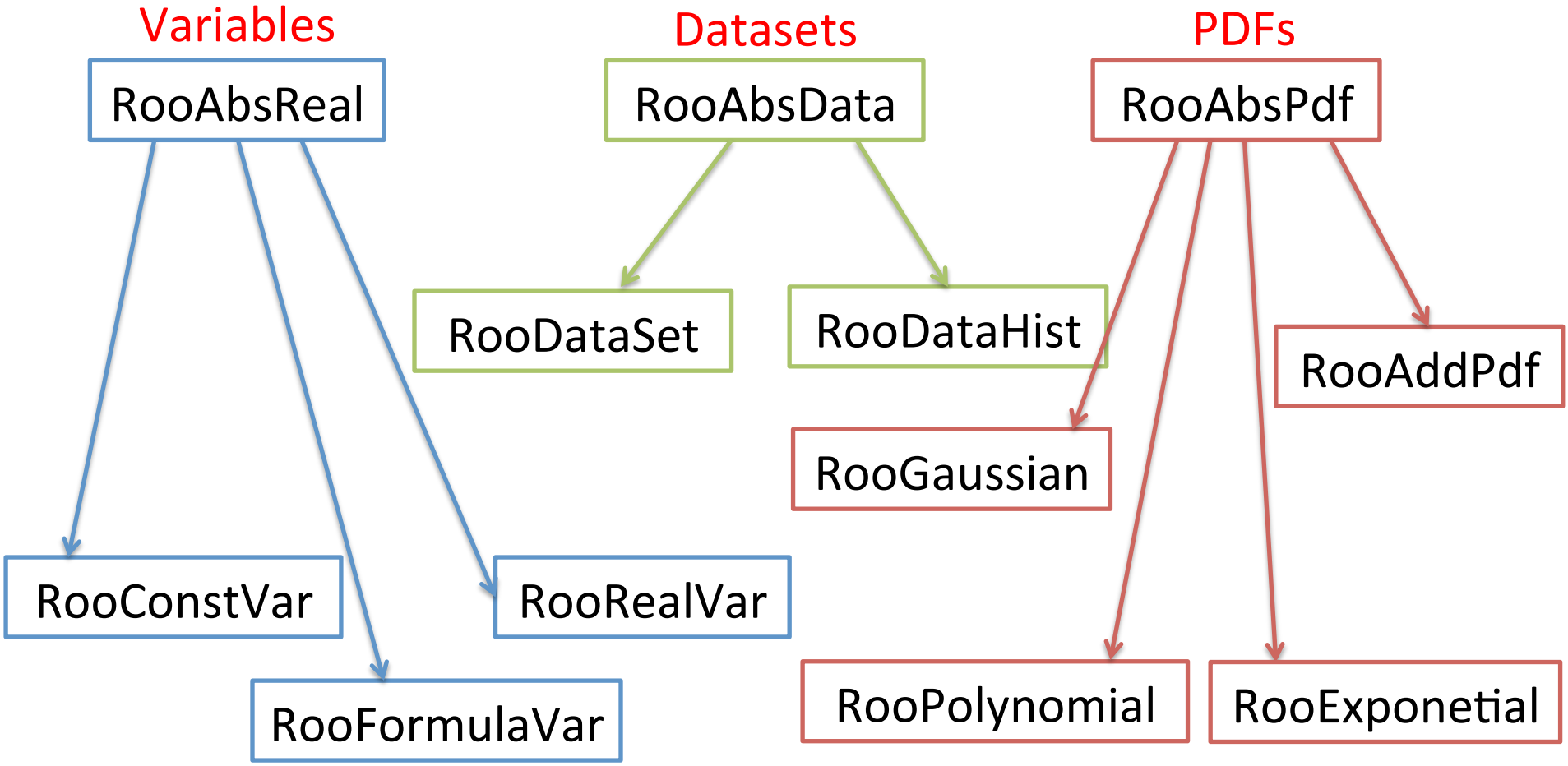
Documentaion

Imperial College London

- Google the class name!
  - E.g TH1, RooDataSet and use the html page
    http://root.cern.ch/root/html/TH1.html
    http://root.cern.ch/root/html/RooDataSet.html
- ROOT
  - Tutorials:
    http://root.cern.ch/root/html/tutorials/
- RooFit
  - Tutorials:
    http://root.cern.ch/root/html/tutorials/roofit/index.html

The Basics

The basic inheritance structure

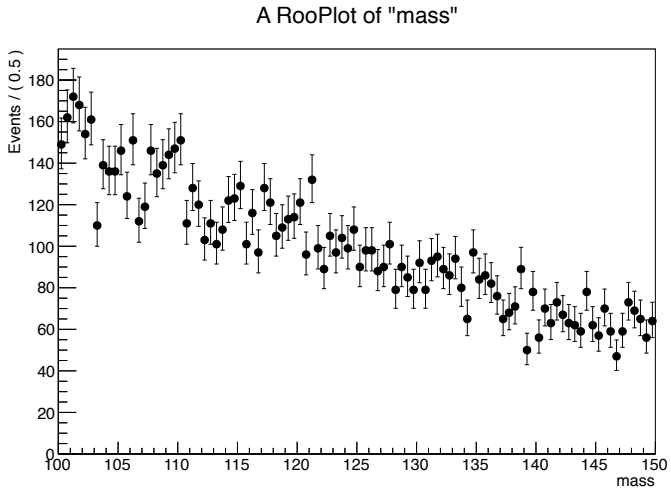


I think easiest to learn by example

In the next few slides I will work through a very simplified example

Filling a dataset

- We can store data in two ways
  - Save every single event (unbinned) → **RooDataSet**
  - Collect events into bins (binned) → **RooDataHist**
- Usually we have a single observable (POI) we are interested in
  - E.g $m_{\gamma\gamma}$, m_{ee}, $m_{\mu\mu}$ - we fit with a signal and background component and determine significance or measure XS
  - In more complex fits there may be additional observables (e.g MELA in H->ZZ->4l at CMS) which give more discriminating power
- If it is MC we also want to save a weight for each event
  - Usually scale by xs * BR
  - Also have scale factors from theory and from measured MC / Data differences
- Usually you will be in an event loop and adding events to the dataset (like you would with a TH1)
- Sometimes someone will already have made the RooDataSet you need so you will just need to pull it from a RooWorkspace



Filling a **RooDataSet** (unbinned)

Imperial College London

- Define our observable of interest
  - invariant diphoton mass – $m_{\gamma\gamma}$
  - Call it mass
  - Specify that it can only take values between 100 and 180

```
RooRealVar *mass = new RooRealVar("mass","mass",100.,180.)
```

name

title

Can also define a starting value here

range

- Define a weight variable

```
RooRealVar *weight = new RooRealVar("weight","weight",0.,100.)
```

- Define our dataset
  - Dependent on mass

Could be more than one POI

optional

```
RooDataSet *data= new RooDataSet("data","data",RooArgSet(*mass),"weight")
```

- Fill our dataset
  - Inside event loop

```
mass->setVal(some_number);
data->add(RooArgSet(*mass),some_weight_value);
```

Filling a **RooDataHist** (binned)

Imperial College London

- Set number of bins and range

```
mass->setRange(100,180);
mass->setBins(80);
```

- Proceed exactly as for **RooDataSet**
- Alternatively you can make a **RooDataHist** from an existing **RooDataSet** or **TH1**

```
RooDataHist *hist = new RooDataHist("hist","hist",RooArgSet(*mass),*data);
RooDataHist *hist = new RooDataHist("hist","hist",RooArgSet(*mass),*TH1);
RooDataHist *hist = data->binnedClone("hist");
```

- Get number of bins

```
data->numEntries();
```

  - for a RooDataHist this returns the number of bins
  - for a RooDataSet this returns the number of entries
  - Which if you think about are equivalent

- Get number of events

```
data->sumEntries();
data->sumEntries("mass>=110. && mass<=120.")
```

  - This returns the weighted sum of entries
  - Or the weighted sum of entries given cut criteria

TCut

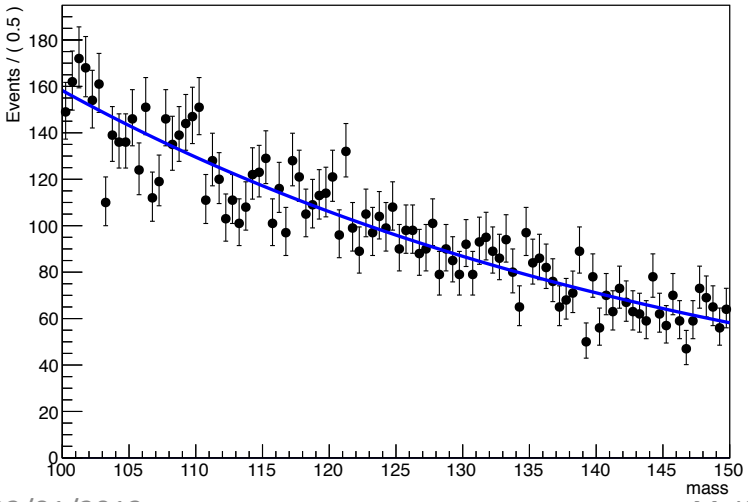
Fitting a dataset - **RooAbsPdf**

Imperial College London

RooAbsPdf

- Once we have a dataset we want to fit it with a distribution
- RooFit has many built in classes that represent PDFs
  - You can also define your own (more later)
  - These all inherit from **RooAbsPdf**
- A **RooAbsPdf** depends on a number of **RooRealVar**
  - These are the free parameters in the fit
  - E.g an exponential (e^{px}) has one free parameter, p, where *x* is our observable
  - A gaussian has two free parameters, mean and sigma.
- Lets make a RooExponential to fit some data

```
RooRealVar *exp_p0 = new RooRealVar("exp_p0","exp_p0",-2.,0.)
RooExponential *exp = new RooExponential("exp","exp",*mass,*exp_p0);
exp->fitTo(*data);
```



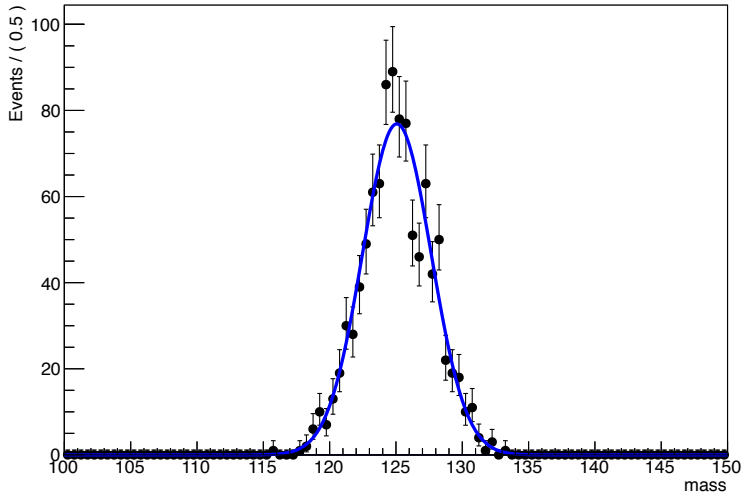
Common built in pdf classes:

- RooGaussian
- RooCBShape
- RooBreitWigner
- RooPolynomial (don't use this!)
- RooChebychev
- RooBernstein

Fitting a dataset - **RooAbsPdf**

- On the last slide we fitted a background like distribution to data
- As another example we can fit a signal like distribution to signal MC – a weighted dataset

```
RooRealVar *mean = new RooRealVar("mean","mean",120.,130.);
RooRealVar *sigma = new RooRealVar("sigma","sigma",0.,10.);
RooGaussian *gaus = new RooGaussian("g","g",*mass,*mean,*sigma);
gaus->fitTo(*data,SumW2Error(true));
```



Fitting with normalization term

Imperial College London

RooExtendPdf

- By default **RooAbsPdf**s have no normalization term
- When plotting or integrating the pdf it gets scaled by the number of events in the dataset you have fit to
- You can add a normalization term if you want to with **RooExtendPdf**

```
RooRealVar *exp_p0 = new RooRealVar("exp_p0","exp_p0",-2.,0.)
RooExponential *exp = new RooExponential("exp","exp",*mass,*exp_p0);
RooRealVar *norm = new RooRealVar("norm","norm",100,10000);
RooExtendPdf *ext = new RooExtendPdf("ext","ext",*exp,*norm);
ext->fitTo(*data,Extended());
```

Fitting a dataset (ii)

Imperial College London

To fit a function to a dataset call the **fitTo()** method

- In it's simplest form

```
exp->fitTo(*data);
```

- To save result in a **RooFitResult**

```
RooFitResult *fitRes = exp->fitTo(*data,Save(true));
```

- Fitting to weighted datasets

```
exp->fitTo(*data,SumW2Error(true));
```

- Fit in range

```
exp->fitTo(*data,Range(low,high));
```

```
exp->fitTo(*data,Range("rangename"));
```

- If also fitting a normalization term

```
exp->fitTo(*data,Extended());
```

- Split NLL calculations over multiple cores – BEWARE!

```
exp->fitTo(*data,NumCPU(int));
```

Alternatively you can calll the **chi2FitTo()** method – must be to a RooDataHist

Fitting a dataset (iii)

Imperial College London

The **RooFitResult**

- When you call fit a pointer to a **RooFitResult** is returned
- Often you're not interested in this so to save memory a NULL pointer is returned unless you set the Save(true) option when you can fitTo()
- The **RooFitResult** contains a lot of useful of information (the parameters before the fit, after the fit, the minNll etc.)

```
RooFitResult *fitRes = exp->fitTo(*data,Save(true));
RooArgSet finalParams = fitRes->floatParsFinal();
// print the final params
finalParams->Print("v");

double minNll = fitRes->minNll();
```

Useful functions for RooRealVar

Imperial College London

- Get current value, and max and min allowed range

```
var->getVal(); var->getMin(); var->getMax()
```

- Set value and set range

```
var->setVal(val); var->setMin(min); var->setMax(max);
```

- Specify a particular range with a name – useful if you only want to plot or fit in this range

```
var->setRange(min,max);
```

```
var->setRange("name",min,max);
```

- Get the error on the value

```
var->getError();
```

- Get the number of bins

```
var->getBins();
```

- Set the number of bins

```
var->setBins(nbins);
```

- Specify a binning with this name
- Useful if you need to plot with different binnings

```
var->setBins(nbins,"name");
```

Plotting

Imperial College London

- Have a class called **RooPlot**
  - Create a **RooPlot** by "framing" a **RooRealVar**

```
RooPlot *myPlot = mass->frame();
```

- Has much of the recognised functionality

```
myPlot->SetTitle(); myPlot->GetXaxis()->SetTitle() etc...
```

- Plotting datasets:

```
data->plotOn(myPlot);
data->plotOn(myPlot,Binning(80),MarkerStyle(kFullCircle),MarkerColor(kBlue));
```

- Plotting pdfs:
  - Default is o normalizae PDF to number of events in dataset

```
pdf->plotOn(myPlot);
pdf->plotOn(myPlot,Range("name"),NormRange("range"));
pdf->plotOn(myPlot,Norm(RooAbsReal::numEvent,nevents));
pdf->plotOn(myPlot,LineColor(kRed),LineStyle(kDashed),LineWidth(2));
```

Integrating a PDF

- Obviously this is very useful
  - E.g getting the number of predicted background events, in a particular region, from the fit
- Set a range for the variable of interest

```
mass->setRange("signalWindow",min,max);
```

- Create a RooAbsArg for the integral and get is' value.
- This returns the fraction of the PDF in that range. As no normalization is included in RooFit – integrating over the entire range will return 1. So to scale the integral to the number of events you will need to multiply up by the number of events in the dataset

```
exp->fitTo(*data);
RooAbsArg *integral =
    exp->createIntegral(*mass,NormSet(*mass),Range("signalWindow"));
double eventsInWindow = integral->getVal()*data->sumEntries();
```

Throwing a toy

Imperial College London

- Throwing "toy" datasets from PDFs is highly desirable

Unbinned

```
exp->fitTo(*data);
RooDataSet *toyData = exp->generate(RooArgSet(*mass),10000);
```

Observable(s) of toy dataset

nEntries in toy

Binned

```
exp->fitTo(*data);
RooDataHist *toyDataHist = exp->generateBinned(RooArgSet(*mass),10000);
```

Optional arguments

Throw Poisson from nEntries first

```
exp->fitTo(*data);
RooDataSet *toyData = exp->generate(RooArgSet(*mass),10000,Extended());
```

RooRandom (more in lecture)

```
RooRandom::randomGenerator->SetSeed();
RooRandom::randomGenerator->Poisson();
RooRandom::randomGenerator->PoissonD();
```

Saving the output

Imperial College London

RooWorkspace

- Objects in RooFit are interlinked
  - A dataset depends on observables (e.g mass)
  - A pdf depends on parameters (mean, sigma etc.)
- When saving objects in RooFit you need to use a RooWorkspace which stores all of these dependencies
- Use the import() functions of a RooWorkspace to save an object.
  - It will save the current state of that object (i.e it's dependent values parameters)
- Print the contents of a workspace using Print().
- Get objects out of a workspace using:
  - var() for **RooRealVar**
  - data() for **RooDataSet**s and **RooDataHist**s
  - pdf() for any **RooAbsPdf**.

End of Basics section

Imperial College London

- Have covered the real basics of RooFit with a simplified example
- The first task I will set you is to write a script which emulates this and makes a few plots
  - See handout
  - An example of this script can be found at:

`/vols/cms04/mk1009/RooFitTutorial/test/RooFitBasics.cpp`

- Now we move on to the more advanced topics
  - These are summarised with an example script:

`/vols/cms04/mk1009/RooFitTutorial/test/RooFitAdvanced.cpp`

  - These skills will be need for the exercises described in the handout

Advanced Topics

RooAbsPdf::minNll()

Imperial College London

- It is often vary useful to know the value of the minNll as likelihoods (and their ratios) are used in many statistical tests:
  - Setting limits, extracting pvalues, finding best fit mass, xs, couplings etc.
- Remember the minimization needs to have already happened - you need to have fit the PDF to the data first – i.e. minimized the Nll

Option 1

```
RooFitResult *fitRes = exp->fitTo(*data,Save(true));
double minNll = fitRes->minNll();
```

- This second option seems unnecassry but is much more robust as it allows you to pass several options to createNLL – e.g. Extended, FixParamters etc. (see RooFit documentation)

Option 2

```
exp->fitTo(*data,Save(true));
RooAbsReal *expMinNll = exp->createNLL(*data);
double minNll = expMinNll->getVal()
```

RooAbsPdf::createChi2()

Imperial College London

- As for the Nll but returns a chi2 (data must be binned of course)
- Useful for goodness of fit tests etc.
- You must have already done the fit

Option 1

```
exp->fitTo(*dataHist);
RooChi2Var *chi2var = new RooChi2Var("name","title",*exp,*dataHist);
double chi2 = chi2var->getVal();
```

Option 2

```
RooAbsReal *chi2var = exp->createChi2(*dataHist);
double chi2 = chi2var->getVal();
```

- Often useful to convert this to a chi2 probability (which penalises for different degrees of freedom)

```
RooFitResult *fitRes = exp->fitTo(*dataHist,Save(true));
// RooDataHist::numEntries() returns number of bins
// RooFitResult::floatParsFinal::getSize() returns number of free params in fit
int nDof = dataHist->numEntries()-fitRes->floatParsFinal().getSize()
double chi2prob = TMath::Prob(chi2,nDof);
```

Component PDFs - **RooAddPdf**

Imperial College London

- An incredibly useful class which allows you to add pdfs together
- E.g fitting data with a signal and background component

```
RooExponential *bkgPdf = new RooExponential("bkg","bkg",*mass,*exp_p0);
RooGaussian *sigPdf = new RooGaussian("sig","sig",*mass,*mean,*sigma);
RooRealVar *bkgYield = new RooRealVar("bkgY","bkgY",100,1.e6);
RooRealVar *sigYield = new RooRealVar("sigY","sigY",-100,100);
RooAddPdf *sandbPdf = new
RooAddPdf("sb","sb",RooArgList(*bkgPdf,*sigPdf),RooArgList(*bkgYield,*sigYield));
sandbPdf->fitTo(*data);
```

- Can do this by fraction instead

```
RooExponential *bkgPdf = new RooExponential("bkg","bkg",*mass,*exp_p0);
RooGaussian *sigPdf = new RooGaussian("sig","sig",*mass,*mean,*sigma);
RooRealVar *fracOfBkg = new RooRealVar("f","f",100,1.e6);
RooAddPdf *sandbPdf = new
RooAddPdf("sb","sb",RooArgList(*bkgPdf,*sigPdf),RooArgList(*fracOfBkg));
sandbPdf->fitTo(*data);
```

RooFormulaVar

- A RooRealVar is a linear variable but in real life one of your fit parameters may depend on multiple variables.
  - E.g instead of fitting for the absolute mean of a signal gaussian you fit for it's distance from the known mass
- After the usual name and title pass a **TFormula** and then a list of dependents which can be referenced in the formula by @0, @1, @2 etc…

```
// as the bd_mass isn't going to change we can use a RooConstVar (like a
// RooRealVar but cannot vary – has no range and no error)
// if you want to propagate the error use a RooRealVar
RooConstVar *bd_mass = new RooConstVar("bd_mass","bd_mass",5279);
RooRealVar *delta_mass = new RooRealVar("dmass","dmass",-200,200);

RooFormulaVar *mean = new
    RooFormulaVar("mean","mean","@0+@1",RooArgList(*bd_mass,*delta_mass"));
```

A TFormula where dependents are listed as @x where x is the location in the dependent list

RooSimultaneous / RooCategory

Imperial College London

- Very powerful tool for constraining fits between categories or between an analysis sample and a control samples
- For example you may want to constrain a particular parameter to be the same between two datasets but allow others to be different
  - E.g fixing the width of a gaussian in two datasets (one analysis sample, one control sample) but allowing them different means

```
RooCategory *cat = new RooCatrgory("sample","sample");
cat->defineType("analysis");
cat->defineType("control");
RooRealVar *mean1 = new RooRealVar("m1","m1",120,125,135);
RooRealVar *mean2 = new RooRealVar("m2","m2",120,125,135);
RooRealVar *sigma = new RooRealVar("s","s",5,2,7);

RooGaussian *g1 = new RooGaussian("g1","g1",*mass,*mean1,*sigma);
RooGaussian *g2 = new RooGaussian("g2","g2",*mass,*mean2,*sigma);

RooDataSet *combData = new
RooDataSet("d","d",RooArgSet(*mass),Import("analysis",*dataAn),Import("control"
,*dataCon));
RooSimultaneous *simPdf = new RooSimultaneous();
simPdf->addPdf(*g1,"analysis");
simPdf->addPdf(*g2,"sample");

simPdf->fitTo(*combData);
```

Print Out

- RooFit by default prints lots of information
  - can be quite useful when you start
  - but after a while you will want to supress most of it
- When you call fitTo() there is further print out which comes from the underlying minimization process (Minuit, Hesse, Migrad)
- It will print
  - the starting values,
  - the best fit values
  - then the error matrix and correlation coefficients
  - all this info can be extracted in your code using **RooFitResult**
  - A lot of this can also be visualized using plot options of **RooFitResult**
- It will print other info along the way
  - covariance matrix converged
  - error matrix accurate

```
[#1] INFO:Minization -- RooMinuit::optimizeConst: activating const optimization
**********
**   13 **MIGRAD       1000         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=2374.16 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  m         1.25000e+02   1.00000e+00   2.01358e-01  -5.36028e+01
  2  s         2.50000e+00   1.00000e+00   2.35352e-01  -1.34855e+02
                           ERR DEF= 0.5
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=2372.38 FROM MIGRAD    STATUS=CONVERGED     31 CALLS          32 TOTAL
                EDM=3.2896e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  m         1.25067e+02   8.20454e-02   5.50771e-04   3.26712e-02
  2  s         2.59462e+00   5.80154e-02   4.43167e-04  -1.54180e-02
                           ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 6.732e-03  3.683e-06
 3.683e-06  3.366e-03
PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL     1     2
        1  0.00077   1.000  0.001
        2  0.00077   0.001  1.000
**********
**   18 **HESSE       1000
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=2372.38 FROM HESSE     STATUS=OK             10 CALLS          42 TOTAL
                EDM=3.2908e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                INTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR       STEP SIZE       VALUE
  1  m         1.25067e+02   8.20454e-02   1.10154e-04   1.34099e-02
  2  s         2.59462e+00   5.80154e-02   1.77267e-05  -5.01882e-01
                           ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 6.732e-03  8.293e-07
 8.293e-07  3.366e-03
PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL     1     2
        1  0.00017   1.000  0.000
        2  0.00017   0.000  1.000
[#1] INFO:Minization -- RooMinuit::optimizeConst: deactivating const optimization
(class RooFitResult*)0x0
```

Supressing Print Out

Imperial College London

- After a while you will realise RooFit spits out a lot of text. If you are running lots and lots of fits this is really annoying
  - Also if you're running on the batch and saving the output somewhere it has to stash a lot of stuff in a .txt file
- Turn off output from fit (output comes from underlying minimization)

```
exp->fitTo(*data,Verbose(false),PrintLevel(-1),PrintEvalErrors(-1))
```

- You can use RooMsgService to suppress other output as well

```
RooMsgService::instance().setGlobalKillBelow(RooFit::INFO);
RooMsgService::instance().setGlobalKillBelow(RooFit::WARNING);
RooMsgService::instance().setGlobalKillBelow(RooFit::ERROR);
```

- RooMsgService also allow rerouting different bits of output to different locations

Imperial College London

The End

But only just the beginning….

Hands on tasks

Imperial College London

See handout and explanation
at the end of the lecture

Appendix

Things I would like to have covered with more time

Imperial College London

- RooStats
  - Whole statistical package – very useful for complex model building, combining analyses, dealing with systematics and nuisance parameters

makefiles

Imperial College London

- Don't know why I've included it as my knowledge is limited
- The makefile I have provided is fairly generic and will automatically compile
  - all headers (.h) in interface/ with the corresponding .cc in src/ as standalone objects (.o) and put them in obj/
  - all the objects (.o) into a shared library (.so) and put it in lib/
  - all .cpp files in test/ as executables in bin/
- You can make ROOT dictionaries so that your own classes and datatypes can be used in CINT.
  - this also allows you to load your classes with python as ROOT objects
    - very useful for configuration and option parsing
    - many large frameworks use this
  - see ROOT documentation for examples

shell scripts

- If you want to submit jobs to the batch queues then you need to submit an executable .sh script
  - I usually add commands which help you quickly see if a jobs has been sucessful or not

An example shell script – sub.sh

```
#!/bin/bash
dir = /vols/cms04/mk1009/RooFitTutorial
touch $dir/sub.sh.run
cd $dir
if ( $dir/bin/RooFitAdvanced $dir/files/RooFitBasicsOutput.root )
    then touch $dir/sub.sh.done
    else touch $dir/sub.sh.fail
done
rm $dir/sub.sh.run
```

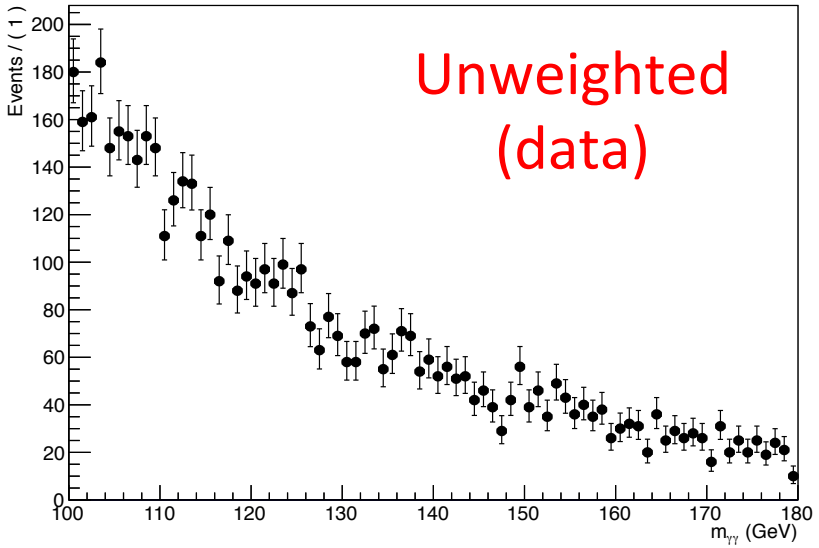
- Submit to the batch by doing:

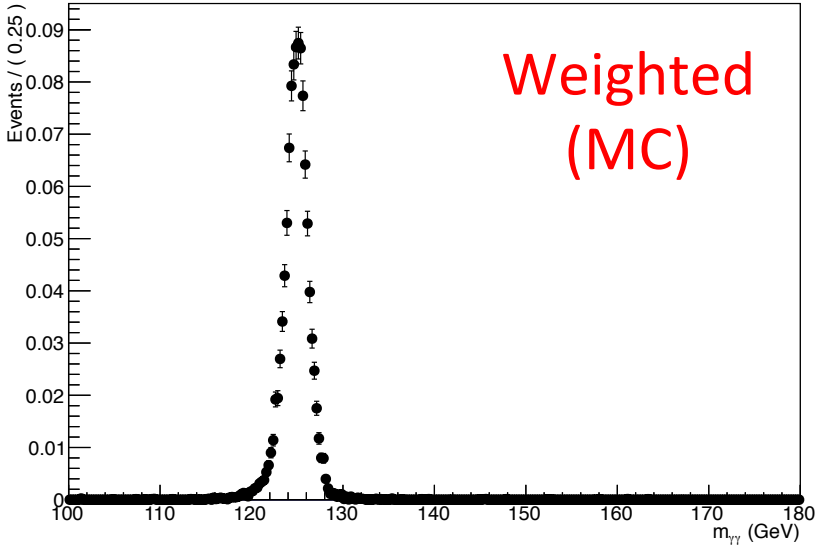
```
$ chmod +x sub.sh  # make script executable
$ qsub –q hepmedium.q –o $PWD/sub.sh.log < $PWD/sub.sh  # to sub at IC
$ bsub –q 1nh –o $PWD/sub.sh.log $PWD/sub.sh  # to sub on lxplus (CERN)
```

BACK UP

RooDataSet (unbinned data)

- A class which stores the value of your observable for each event in your data or MC
- It depends on at least one observable
  - It's a distribution that you will fit with a pdf
- It's unbinned
  - If you have a 1000 events there will be 1000 entries each with a value of $m_{\gamma\gamma}$
  - Each value can also have a weight (i.e. for MC)





RooRealVar

Imperial College London

- A RooRealVar is variable or parameter which could be
  - An observable (POI)
    - e.g. $m_{\gamma\gamma}$, m_{ee}, $m_{\mu\mu}$ which is something we would usually fit with a the distribution
  - A free parameter in our fit (a variable which floats)
- These are the basic building blocks of RooFit representing real values

```
RooRealVar *var = new RooRealVar(name,title,startVal,minVal,maxVal)
```

- It has both a value and a minimum and maximum value it can move to
  - For an observable it will take on many values (events) in a dataset.
  - For a fit parameter you give it a value and a min/max and when you ask RooFit to fit something it will vary the parameter within it's min and max and find the value that minimizes the neg. likelihood and set this as it's value
- From now on lets assume our observable is diphoton mass ($m_{\gamma\gamma}$)

Useful functions for RooRealVar

Imperial College London

- getVal(), getMin(), getMax()
  - Get current value and maximum and minimum allowed values
- setVal(), setMin(), setMax()
  - Set above
- getError()
  - Get the error on the current value
- setBins()
  - Set the number of bins (will be used for RooDataHists of this variable and when plotting distributions of this variable)
- getBins()
- setRange()
  - You can define ranges (e.g. if you only want to fit in a certain range)
- frame()
  - Used when plotting this variable

RooDataHist (binned data)

Imperial College London

- A class analogous to a TH1. It has a binning (defined by the observable it depends on)
- It depends on at least one observable
  - It's a distribution that you will fit with a pdf
- It's binned
  - If you have a 100 events there will be 100 entries each with a weight corresponding to the number of entries in that bin
  - These can have an overall weight
- When RooDataSets are plotted you have to define a binning (or RooFit will pick one for you.
- When RooDataHists are plotted the binning is already defined
- RooDataSets (i.e. unbinned) contain more information, however fitting them takes a lot longer (the number of degrees of freedom = nevents)
- RooDataHists (i.e. binned) are much quicker to fit but can be less accurate if the statistics are very low
- Make a sensible decision about which to use

Useful functions for RooAbsData

Imperial College London

- sumEntries()
  - Weighted sum of entries, i.e. number of events
- numEntries()
  - Number of independent bins
  - So for unbinned data this is the number of entries
- add(RooAbsReal*)
  - Add an event to the dataset
- append(RooAbsData*)
  - Add another dataset to the current dataset
- plotOn(RooPlot*)
  - plot this distribution on a RooPlot

RooAbsPdf

Imperial College London

- These are PDFs i.e. functions you will fit to data
- They come in several already made formats:
  - RooExponential
  - RooGaussian
  - RooCBShape
  - RooBreitwigner
  - RooPolynomial
    - very bad at convergeing
    - Use RooBernstein or RooChebychev instead
  - RooAddPdf
    - Very useful for composite models (e.g. signal + background)
  - RooSimultaneous
    - Very useful for constraining pdf shapes across categories or to a control sample
  - RooGenericPdf
    - Can define any valid TFormula i.e. any function you want
    - These are also very bad at converging as their integral has to be done numerically
    - If you want to use your own shape a lot you should write your own PDF class where the analytical integral is defined (see later)

Useful functions for RooAbsPdf

Imperial College London

- fitTo(RooAbsData*)
  - Fit to data!
- createIntegral()
  - Integrate the pdf over a range
- createNll()
  - Get the value of the NLL
- createChi2()
  - Get the value of the chi2
- generate()
  - Throw a toy dataset from this PDF
- plotOn(RooPlot*)
  - plot the fit on a RooPlot

RooWorkspace

Imperial College London

- Stores RooFit objects
- This is where everything you want to keep should be saved
- RooFit objects are dependent on each other
  - A PDF will depend on several variables
  - Some objects may have shared dependence
- A RooWorkspace keeps all of this intact
- Saving objects:

```
RooWorkspace *work = new RooWorkspace("ws_name");
work->import(adataset);
work->import(apdf);
TFile *outFile = new TFile("Out.root","RECREATE");
outFile->cd();
work->Write();
outFile->Close();
```

- Retrieving objects:

```
TFile *inFile = new TFile("Out.root");
RooWorkspace *work = (RooWorkspace*)inFile->Get("ws_name");
RooRealVar *var = (RooRealVar*)work->var("varname");
RooDataSet *dat = (RooDataSet*)work->data("dataname");
```