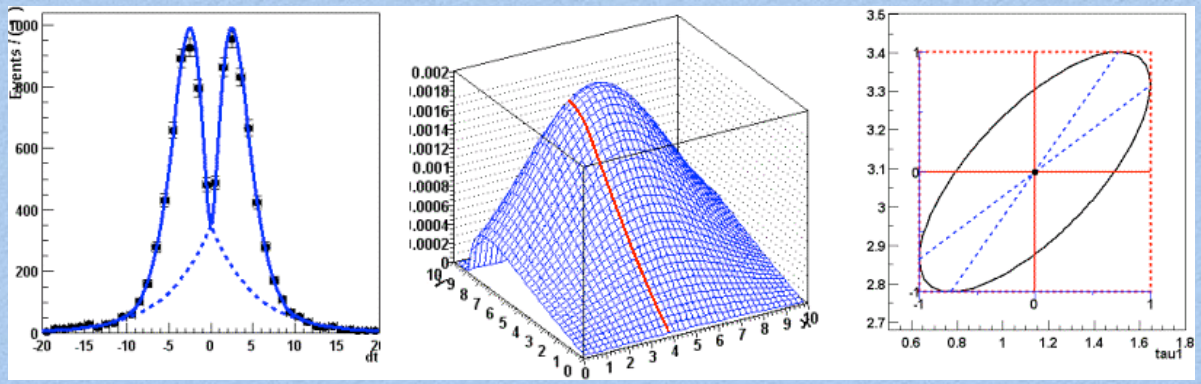
RooFit



Based on L. Moneta lessons at Terascale Statistics School 2015

Outline

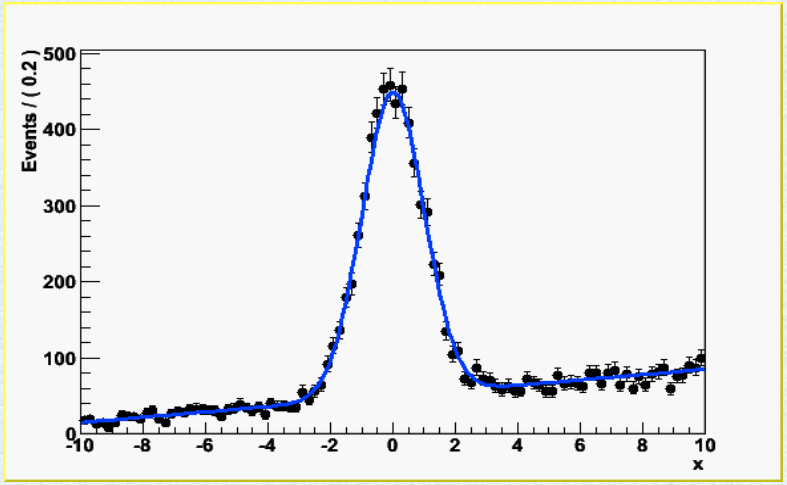
- Introduction to RooFit
  - Basic functionality
  - Model building using the workspace
  - Composite models

Material based on slides from W. Verkerke (author of RooFit)

- Exercises on RooFit:
  - building and fitting model

RooFit

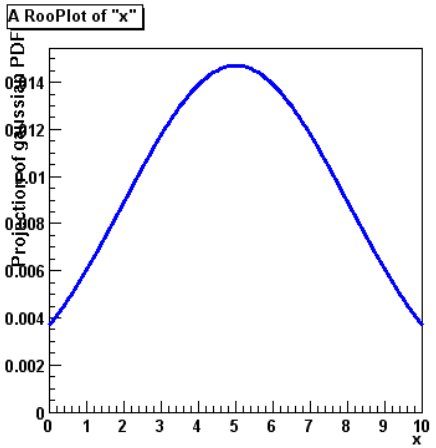
- Toolkit for data modeling
  - developed by *W. Verkerke and D. Kirkby*
- model distribution of observable ***x*** in terms of parameters ***p***
  - probability density function (pdf): P `(x;p)`
- pdf are normalized over allowed range of observables ***x*** with respect to the parameters ***p***

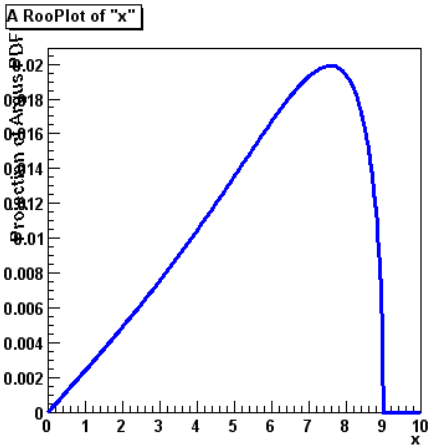


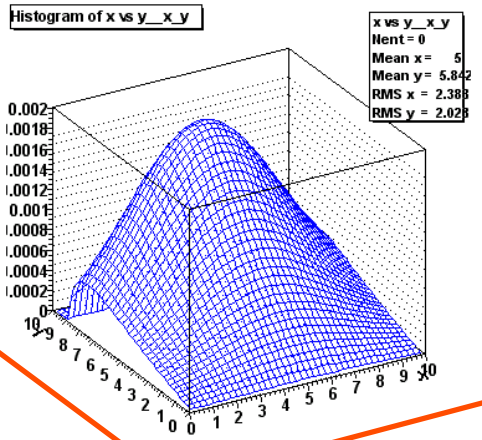
Mathematic – Probability density functions

- Probability Density Functions describe probabilities, thus
  - All values most be >0
  - The total probability must be 1 *for each* *p*, i.e.
  - Can have any number of dimensions

$$\int_{\vec{x}_{\min}}^{\vec{x}_{\max}} g(\vec{x}, \vec{p})\, d\vec{x} \equiv 1$$







$$\int F(x)\,dx \equiv 1$$

$$\int F(x, y)\,dx\,dy \equiv 1$$

- Note distinction in role between *parameters* (p) and *observables* (x)
  - Observables are measured quantities
  - Parameters are degrees of freedom in your model

Why RooFit ?

- ROOT function framework can handle complicated functions
  - but require writing large amount of code
- Normalization of p.d.f. not always trivial
  - RooFit does it automatically
- In complex fit, computation performance important
  - need to optimize code for acceptable performance
  - built-in optimization available in RooFit
    - evaluation only when needed
- Simultaneous fit to different data samples
- Provide full description of model for further use

RooFit

- RooFit provides functionality for building the pdf's
  - complex model building from standard components
  - composition with addition product and convolution
- All models provide the functionality for
  - maximum likelihood fitting
  - toy MC generator
  - visualization

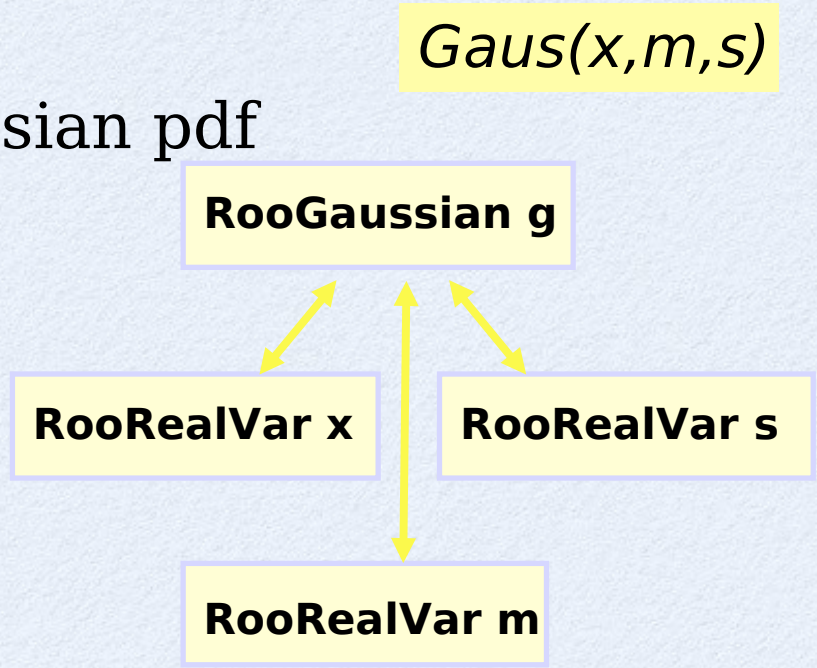
RooFit Modeling

Mathematical concepts are represented as C++ objects

Mathematical concept		RooFit class
variable	x	**RooRealVar**
function	$f(x)$	**RooAbsReal**
PDF	$f(x)$	**RooAbsPdf**
space point	$\vec{x}$	**RooArgSet**
integral	$\int_{x_{\min}}^{x_{\max}} f(x)dx$	**RooRealIntegral**
list of space points		**RooAbsData**

RooFit Modeling

Example: Gaussian pdf



RooFit code:

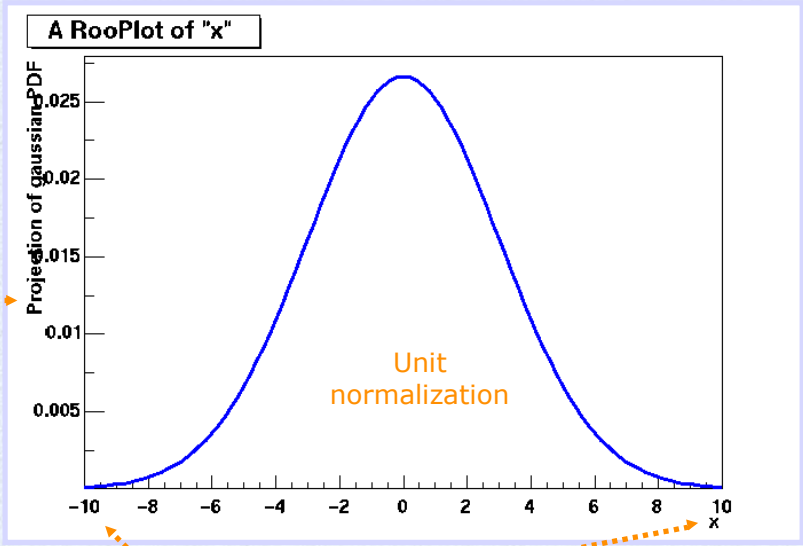
```
RooRealVar x(“x”,”x”,2,-10,10)
RooRealVar s(“s”,”s”,3) ;
RooRealVar m(“m”,”m”,0) ;
RooGaussian g(“g”,”g”,x,m,s)
```

- Represent relations between variables and functions as client/server links between objects

RooFit Functionality

- pdf visualization

```
RooPlot * xframe = x->frame();
pdf->plotOn(xframe);
xframe->Draw();
```



Axis label from **gauss** title

A **RooPlot** is an empty frame capable of holding anything plotted versus it variable

Plot range taken from limits of **x**

RooFit Functionality

- Toy MC generation from any pdf

Generate 10000 events from Gaussian p.d.f and show distribution

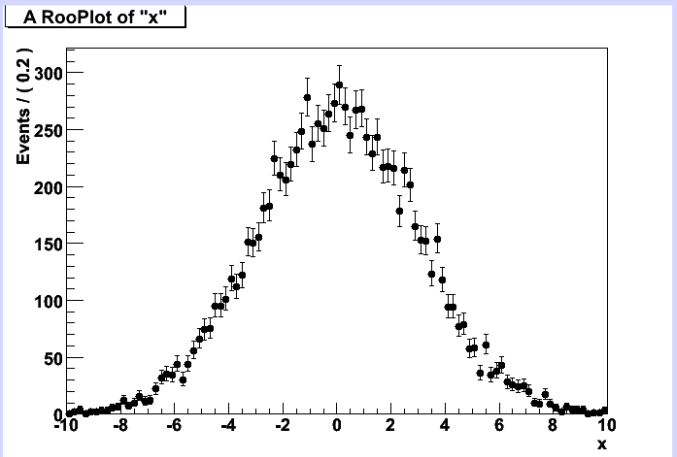
```
RooDataSet * data = pdf->generate(*x,10000);
```

- data visualization

```
RooPlot * xframe = x->frame();
data->plotOn(xframe);
xframe->Draw();
```

Note that dataset is ***unbinned*** (vector of data points, x, values)

Binning into histogram is performed in **data->plotOn()** call



RooFit Functionality

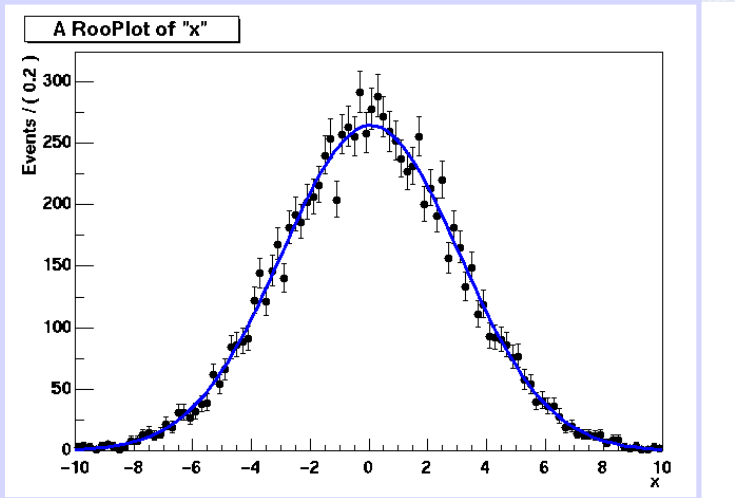
- Fit of model to data
  - e.g. unbinned maximum likelihood fit

```
pdf = pdf->fitTo(data);
```

- data and pdf visualization after fit

```
RooPlot * xframe = x->frame();
data->plotOn(xframe);
pdf->plotOn(xframe);
xframe->Draw();
```

PDF automatically normalized to dataset



RooFit Workspace

- **RooWorkspace** class: container for all objected created:
  - full model configuration
    - PDF and parameter/observables descriptions
    - uncertainty/shape of nuisance parameters
  - (multiple) data sets
- Maintain a complete description of all the model
  - possibility to save entire model in a ROOT file
  - all information is available for further analysis
- Combination of results joining workspaces in a single one
  - common format for combining and sharing physics results

```
RooWorkspace workspace("w");
workspace.import(*data);
workspace.import(*pdf);
workspace.writeToFile("myWorkspace.root")
```

RooFit Factory

```
RooRealVar x(“x”,”x”,2,-10,10)
RooRealVar s(“s”,”s”,3) ;
RooRealVar m(“m”,”m”,0) ;
RooGaussian g(“g”,”g”,x,m,s)
```

The workspace provides a factory method to auto-generates objects from a math-like language (the p.d.f is made with 1 line of code instead of 4)

```
RooWorkspace w;
w.factory(“Gaussian::g(x[2,-10,10],m[0],s[3])”)
```

In the tutorial we will work using the workspace factory to build models

Using the workspace

- Workspace
  - A generic container class for all RooFit objects of your project
  - Helps to organize analysis projects
- Creating a workspace

```
RooWorkspace w("w") ;
```

- Putting variables and function into a workspace
  - When importing a function or pdf, all its components (variables) are automatically imported too

```
RooRealVar x("x","x",-10,10) ;
RooRealVar mean("mean","mean",5) ;
RooRealVar sigma("sigma","sigma",3)  ;
RooGaussian f("f","f",x,mean,sigma) ;

// imports f,x,mean and sigma
w.import(f) ;
```

Using the workspace

- Looking into a workspace

```
w.Print() ;

variables
---------
(mean,sigma,x)

p.d.f.s
-------
RooGaussian::f[ x=x mean=mean sigma=sigma ] = 0.249352
```

- Getting variables and functions out of a workspace

```
// Variety of accessors available
RooRealVar * x = w.var("x");
RooAbsPdf * f = w.pdf("f");
```

- Writing workspace and contents to file

```
w.writeToFile("wspace.root") ;
```

Factory syntax

- Rule #1 – Create a variable

```
x[-10,10]   // Create variable with given range
x[5,-10,10] // Create variable with initial value and range
x[5]        // Create initially constant variable
```

- Rule #2 – Create a function or pdf object

```
ClassName::Objectname(arg1,[arg2],...)
```

  – Leading 'Roo' in class name can be omitted
  – Arguments are names of objects that already exist in the workspace
  – Named objects must be of correct type, if not factory issues error
  – Set and List arguments can be constructed with brackets {}

```
Gaussian::g(x,mean,sigma)
      ! RooGaussian("g","g",x,mean,sigma)

Polynomial::p(x,{a0,a1})
      ! RooPolynomial("p","p",x",RooArgList(a0,a1));
```

Factory syntax

- Rule #3 – Each creation expression returns the name of the object created
  - Allows to create input arguments to functions 'in place' rather than in advance

```
Gaussian::g(x[-10,10],mean[-10,10],sigma[3])
   !  x[-10,10]
          mean[-10,10]
          sigma[3]
          Gaussian::g(x,mean,sigma)
```

- Miscellaneous points
  - You can always use numeric literals where values or functions are expected
  - It is not required to give component objects a name, e.g.

```
Gaussian::g(x[-10,10],  0,3)
```

```
SUM::model(0.5*Gaussian(x[-10,10],0,3),   Uniform(x)) ;
```

Model building – (Re)using standard components

- List of most frequently used pdfs and their factory spec

Gaussian	**Gaussian::g(x,mean,sigma)**
Breit-Wigner	**BreitWigner::bw(x,mean,gamma)**
Landau	**Landau::l(x,mean,sigma)**
Exponential	**Exponential::e(x,alpha)**
Polynomial	**Polynomial::p(x,{a0,a1,a2})**
Chebychev	**Chebychev::p(x,{a0,a1,a2})**
Kernel Estimation	**KeysPdf::k(x,dataSet)**
Poisson	**Poisson::p(x,mu)**
Voigtian (=BW$\otimes$G)	**Voigtian::v(x,mean,gamma,sigma)**

Factory syntax – using expressions

- Customized p.d.f from interpreted expressions

```
w.factory("EXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- Customized class, compiled and linked on the fly

```
w.factory("CEXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- re-parametrization of variables (making functions)

```
w.factory("expr::w('(1-D)/2',D[0,1])") ;
```

  – note using expr (builds a function, a RooAbsReal)
  – instead of EXPR (builds a pdf, a RooAbsPdf)

This usage of upper vs lower case applies also for other factory commands (SUM, PROD,....)

Factory syntax: Adding p.d.f.

- Additions of PDF (using fractions)

```
SUM::name(frac1*PDF1,PDFN)
```

```
SUM::name(frac1*PDF1,frac2*PDF2,...,PDFN)
```

  – Note that last PDF does not have an associated fraction

$$F(x) = f \times S(x) + (1-f)B(x) \quad ; \quad N_{\exp} = N$$

- PDF additions (using expected events instead of fractions)

```
SUM::name(Nsig*SigPDF,Nbkg*BkgPDF)
```

$$F(x) = \frac{N_S}{N_S + N_B} \times S(x) + \frac{N_B}{N_S + N_B} B(x) \quad ; \quad N_{\exp} = N_S + N_B$$

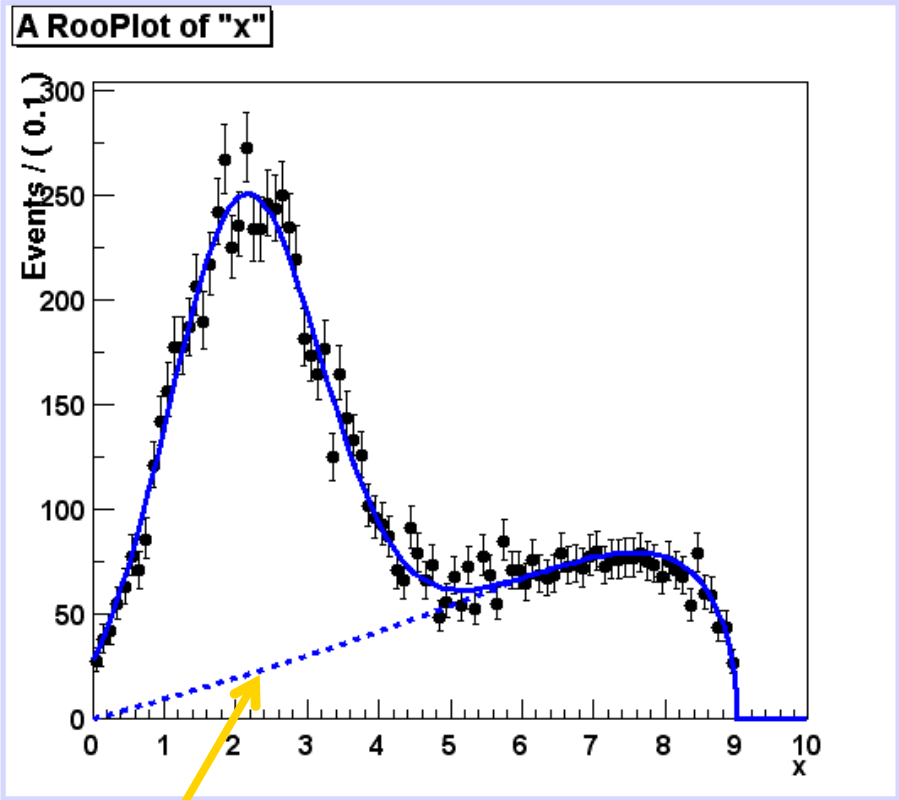
  – the resulting model will be extended

  – the likelihood will contain a Poisson term depending on the total number of expected events (Nsig+Nbkg)

$$L(x \mid p) \rightarrow L(x|p)\mathrm{Poisson}(N_{obs}, N_{exp})$$

Component plotting - Introduction

- Plotting, toy event generation and fitting works identically for composite p.d.f.s
  - Several optimizations applied behind the scenes that are specific to composite models (e.g. delegate event generation to components)
- Extra plotting functionality specific to composite pdfs
  - Component plotting




```
// Plot only argus components
w::sum.plotOn(frame,Components("argus"),LineStyle(kDashed)) ;

// Wildcards allowed
w::sum.plotOn(frame,Components("gauss*"),LineStyle(kDashed)) ;
```

Uncorrelated products – Mathematics and constructors

- Mathematical construction of products of uncorrelated p.d.f.s is straightforward

2D

$$H(x,y) = F(x) \times G(y)$$

– No explicit normalization required ! product is also unit normalized

nD

$$H(x^{\{i\}}) = \prod_i F^{\{i\}}(x^{\{i\}})$$

If input p.d.f.s are unit normalized,

– (Partial) integration and toy MC generation automatically uses factorizing properties of product, e.g. is deduced from structure.

$$\int H(x,y)dx \equiv G(y)$$

- Corresponding factory operator is PROD

```
w.factory("Gaussian::gx(x[-5,5],mx[2],sx[1])") ;
w.factory("Gaussian::gy(y[-5,5],my[-2],sy[3])") ;

w.factory("PROD::gxy(gx,gy)") ;
```

Introducing correlations through composition

- RooFit pdf building blocks do not require variables as input, just real-valued functions
  - Can substitute any variable with a function expression in parameters and/or observables

$$f(x;p) \Rightarrow f(x,p(y,q)) = f(x,y;q)$$

  - Example: Gaussian with shifting mean

```
w.factory("expr::mean('a*y+b',y[-10,10],a[0.7],b[0.3])") ;
w.factory("Gaussian::g(x[-10,10],mean,sigma[3])") ;
```

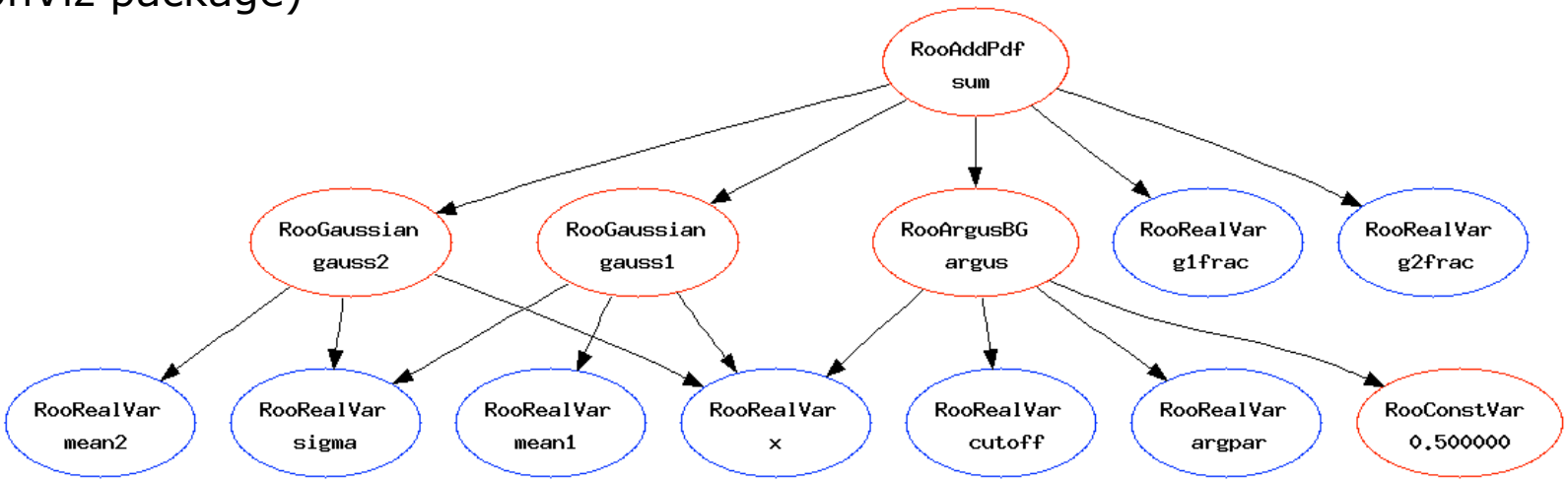
  - No assumption made in function on a,b,x,y being observables or parameters, any combination will work

Operations on specific to composite pdfs

- Tree printing mode of workspace reveals component structure – **w.Print("t")**

```
RooAddPdf::sum[ g1frac * g1 + g2frac * g2 + [%] * argus ] = 0.0687785
   RooGaussian::g1[ x=x mean=mean1 sigma=sigma ] = 0.135335
   RooGaussian::g2[ x=x mean=mean2 sigma=sigma ] = 0.011109
   RooArgusBG::argus[ m=x m0=k c=9 p=0.5 ] = 0
```

  - Can also make input files for GraphViz visualization (**w.pdf("sum")->graphVizTree("myfile.dot")**)
  - Graph output on ROOT Canvas in near future (pending ROOT integration of GraphViz package)



Constructing joint pdfs (RooSimultaneous)

- Operator class SIMUL to construct joint models at the pdf level
  - need a discrete observable (category) to label the channels

```
// Pdfs for channels 'A' and 'B'
w.factory("Gaussian::pdfA(x[-10,10],mean[-10,10],sigma[3])") ;
w.factory("Uniform::pdfB(x)") ;

// Create discrete observable to label channels
w.factory("index[A,B]") ;

// Create joint pdf (RooSimultaneous)
w.factory("SIMUL::joint(index,A=pdfA,B=pdfB)") ;
```

- Construct joint datasets
  - contains observables ("x") and category ("index")

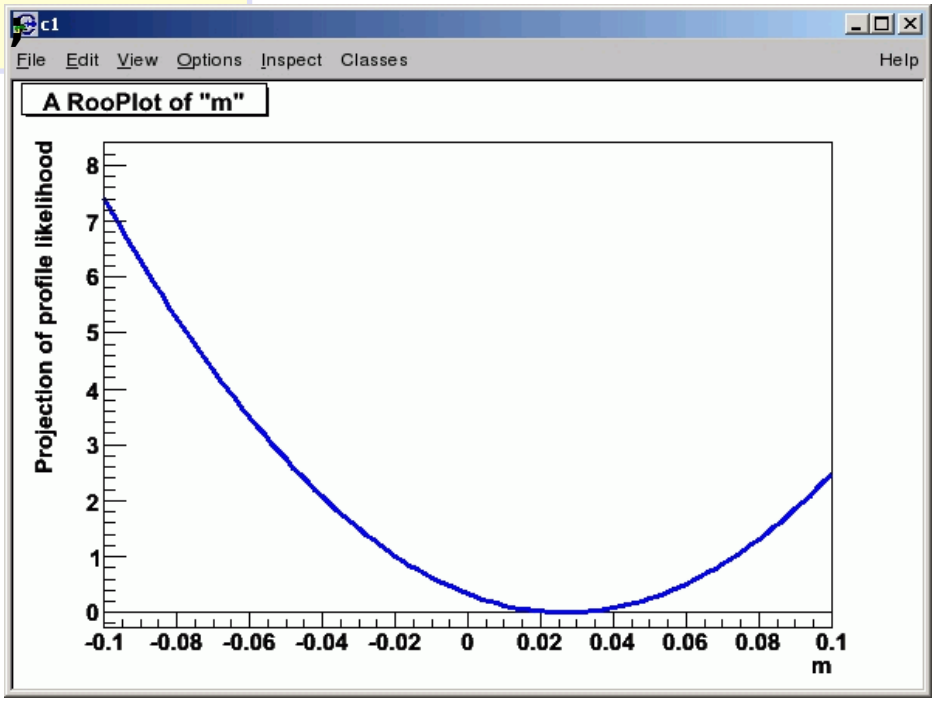
```
RooDataSet *dataA, *dataB ;
RooDataSet dataAB("dataAB","dataAB",
                  RooArgSet(*w.var("x"),*w.cat("index")),
                  Index(*w.cat("index")),
                  Import("A",*dataA),Import("B",*dataB)) ;
```

Constructing the likelihood

- So far focus on construction of pdfs, and basic use for fitting and toy event generation
- Can also explicitly construct the likelihood function of and pdf/ data combination
  - Can use (plot, integrate) likelihood like any RooFit function object

```
RooAbsReal* nll = pdf->createNLL(data) ;

RooPlot* frame = parameter->frame() ;
nll->plotOn(frame,ShiftToZero())
```



Constructing the likelihood

- Example – Manual MIMIZATION using MINUIT
  - Result of minimization are immediately propagated to RooFit variable objects (values and errors)

```
// Create likelihood (calculation parallelized on 8 cores)
RooAbsReal* nll = w::model.createNLL(data,NumCPU(8)) ;


 RooMinimizer m(*nll) ;            // create Minimizer class
m.minimize("Minuit2","Migrad");  // minimize using Minuit2
m.hesse() ;                       // Call HESSE
m.minos(w::param) ;               // Call MINOS for 'param'

RooFitResult* r = m.save() ; // Save status (cov matrix etc)
```

  - Also other minimizers (Minuit, GSL etc) supported
  - N.B. Different minimizer can also be used from RooAbsPdf::fitTo

```
//fit a pdf to a data set using Minuit2 as minimizer
pdf.fitTo(*data, RooFit::Minimizer("Minuit2","Migrad")) ;
```

Basics – Importing data

- Unbinned data can also be imported from ROOT **TTrees**

```
// Import unbinned data
RooDataSet data("data","data",x,Import(*myTree)) ;
```

  - Imports **TTree** branch named "x".
  - Can be of type **Double_t**, **Float_t** , **Int_t** or **UInt_t** .
    All data is converted to Double_t internally
  - Specify a **RooArgSet** of multiple observables to import multiple observables

- Binned data can be imported from ROOT **THx** histograms

```
// Import binned data
RooDataHist data("data","data",x,Import(*myTH1)) ;
```

  - Imports values, binning definition *and* SumW2 errors (if defined)
  - Specify a **RooArgList** of observables when importing a TH2/3.

Basics – Importing data

- Unbinned data can also be imported from ROOT **TTrees**

```
// Import unbinned data
RooDataSet data("data","data",x,Import(*myTree)) ;
```

  - Imports **TTree** branch named "x".
  - Can be of type **Double_t**, **Float_t** , **Int_t** or **UInt_t** .
    All data is converted to Double_t internally
  - Specify a **RooArgSet** of multiple observables to import multiple observables

- Binned data can be imported from ROOT **THx** histograms

```
// Import binned data
RooDataHist data("data","data",x,Import(*myTH1)) ;
```

  - Imports values, binning definition *and* SumW2 errors (if defined)
  - Specify a **RooArgList** of observables when importing a TH2/3.

RooFit Summary

- Overview of RooFit functionality
  - not everything covered
  - not discussed on how it works internally (optimizations, analytical deduction, etc..)
- Capable to handle complex model
  - scale to models with large number of parameters
  - being used for many analysis at LHC
- Workspace:
  - easy model creation using the factory syntax
  - tool for storing and sharing models (analysis combination)

RooFit Documentation

- Starting point: http://root.cern.ch/drupal/content/roofit
- Users manual (134 pages ~ 5 year old)
- Quick Start Guide (20 pages, recent)
- Link to 84 tutorial macros (also in $ROOTSYS/tutorials/roofit)
- More than 200 slides from *W. Verkerke* documenting all features are available at the *French School of Statistics 2008*
  - http://indico.in2p3.fr/getFile.py/access?contribId=15&resId=0&materialId=slides&confId=750
- On the internet you can find plenty of tutorials!
  If what you are looking for in not on these slide … Ask Google ;)

Esercitazione 13

Exercise 1

The aim of this exercise is to show the basic fitting functionality of ROOT and RooFit.

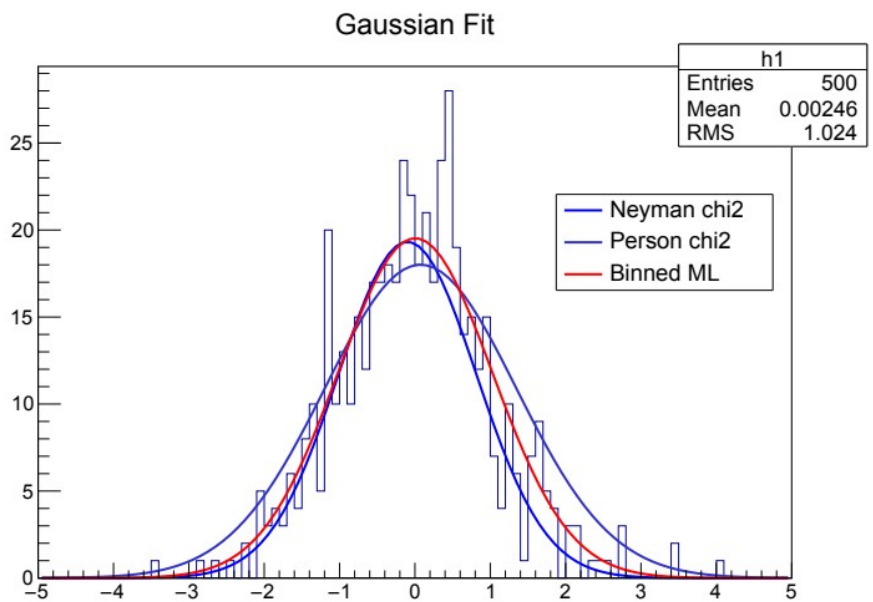
- Using ROOT
  - Create an histogram with 100 bins between-5 and 5 and fill with it 500 gaussian distributed numbers with mean 0 and sigma 1. Then fit the histogram with a gaussian function (possibly using its normalized form) using ROOT to estimate the number of events, the mean and the sigma of the underlined gaussian distribution.
  - Fit using the Neyman chi-square → Look at the "fit options"
  - Fit using the Pearson chi-square
  - Fit using a binned maximum likelihood
  - Plot the resulting fit function on top of the data
- Using RooFit
  - Create a Gaussian model (using RooWorkspace w; w.factory(...)), which include the number of events (i.e. an extended model) and a RooFit binned data object (RooDataHist – RooDataHist * data = new RooDataHist("data","data",*x,h1);
  - Fit the RooDataHist object with

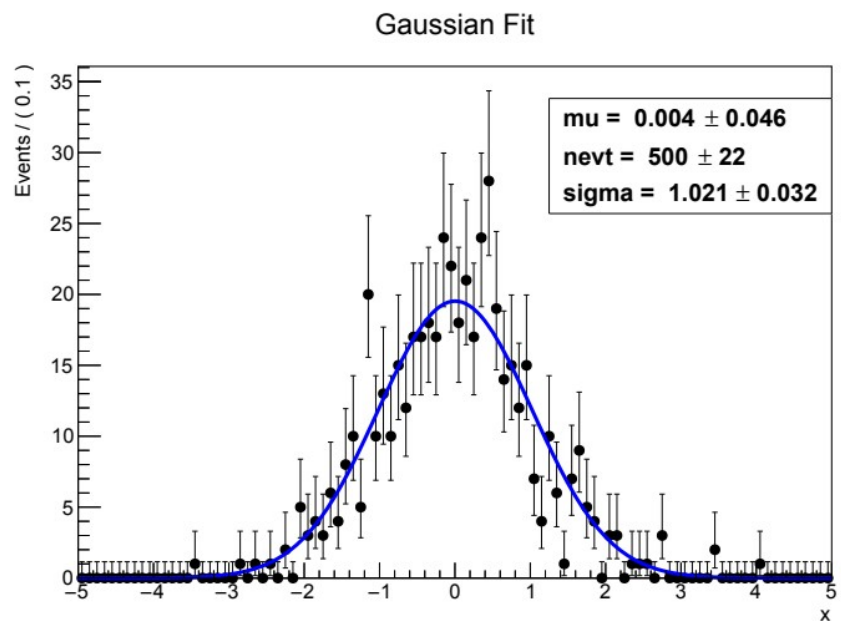
    ```
    pdf.fitTo(*data, RooFit::Minimizer("Minuit2","Migrad")) ;
    ```

  - Look at $ROOTSYS/tutorial/roofit to look for "inspiration"

https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13 – Exercise 1





https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13 – Exercise 1

Hint for fitting with RooFit

- For using RooFit you need first to create the RooDataHist class from an histogram.
- For creating the RooFit model you need to first to create a RooGaussian pdf which has the mean and the sigma as parameters. For adding the number of events you need to create the RooExtendPdf class. You can create the classes directly or by using the factory functionality of the RooWorkspace.
- How to create a RooFit model
  - Every variable in RooFit, needs to be created with a defined range [xmin,xmax]. Use a very large value if you don't know the range. If you provided only a value, the variable is considered constant. If you provide only the minimum and the maximum, the initial value will be taken in the middle of the range. The initial value must be within the given range.
  - Using the workspace factory syntax any existing RooFit pdf class by using the class name stripped by the Roo suffix. The exact signature required (variables to be passed) is defined in the class constructor
  - You need to define the [value, min, max] of a variable only the first time you create in the factory. Afterwards you can reference the variable by its name.

```
// create a Gaussian Pdf and an extended pdf

RooWorkspace w("w");

w.factory("Gaussian:pdf(x[-10,10],mu[1,-1000,1000],s[2,0,1000])");

w.factory("ExtendPdf:model(g,n[100,0,10000])");
```

https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13 – Exercise 1

- How to use the model from the workspace
- After you have created the variable and p.d.f in the workspace, you can access their pointers, by using RooWorkspace::var for variables or RooWorkspace::pdf for p.d.f. Example:

```
// retrieving model components

RooAbsPdf * pdf = w.pdf("model");   // access object from workspace

RooRealVar * x = w.var("x");   // access object from workspace
```

- Plotting the data:

```
// create a RooPlot object and plot the data
RooPlot * plot = x->frame();
data->plotOn(plot);
```

- Fitting the data using RooAbsPdf::fitTo.

```
// fit the data and save its result. Use the optional =Minuit2= minimiser
RooFitResult * res = pdf->fitTo(*data, Minimizer("Minuit2","Migrad"), Save(true) );
```

- Plotting the fit function resulting from the fit:

```
pdf->plotOn(plot);
plot->Draw();   // to show the RooPlot in the current ROOT Canvas
```

https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

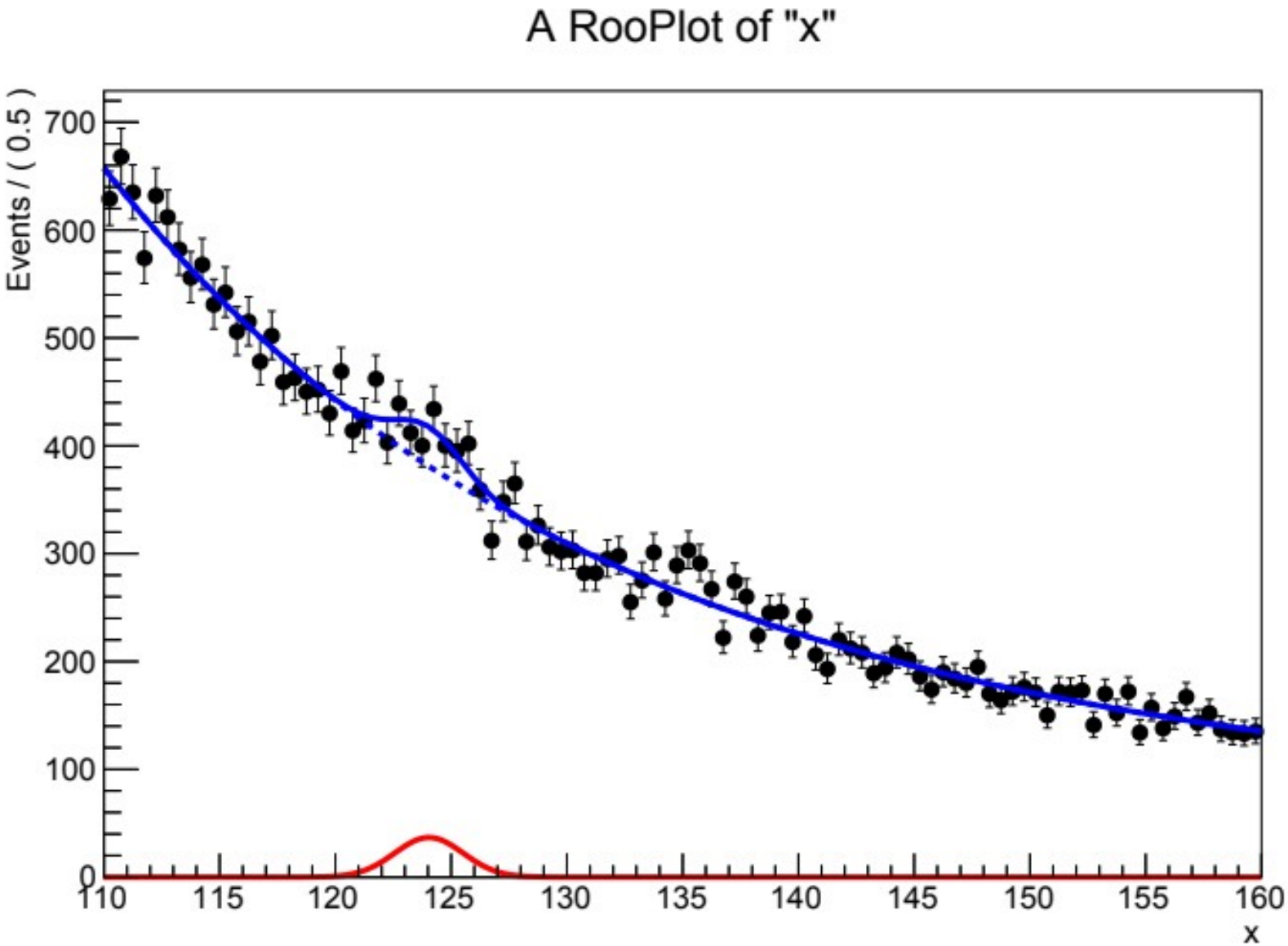
Esercitazione 13

Exercise 2

- Create first the composite model formed by a Gaussian signal over a falling exponential background.
- Read the data (in text format) from the file (Hgg.txt → To be downloaded) and create a RooDataSet class with all the data.
- Perform an extended unbinned fit to the data to extract the Higgs signal strength.
- Plot the resulting fit function from the fit with separate signal and background components.

https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13 – Exercise 2



https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13

- Read first the attached file (Hgg.txt in a RooDataSet using RooDataSet::read or in a ROOT TTree using TTree::ReadFile ()

```
TTree tree("tree","tree");
int nevt = tree.ReadFile("Hgg.txt","x");
RooDataSet data("data","data",*w.var("x"),Import(tree) );
```

- Create the model using the RooWorkspace factory
  - create the Exponential pdf and the Gaussian signal p.d.f. function. We create the exponential as two different components to reproduce better the data.
  - create then a RooAddPdf using the Gaussian and the Exponential and the relative number of events. Note that we could instead of the number of events a relative fraction. In this last case we would fit only for the shape and not the normalisation.
  - The RooAddPdf is created using the SUM operator of the factory syntax.

```
// create model
   RooWorkspace w("w");
   w.factory("x[110,160]");  // invariant mass

   // create exponential model as two components
   w.factory("a1[ 7.5, -500, 500]");
   w.factory("a2[-1.5, -500, 500]");
   w.factory("expr::z('-(a1*x/100 + a2*(x/100)^2)', a1, a2, x)");
   w.factory("Exponential::bmodel(z, 1)");

    // signal model
   w.factory("mass[130, 110, 150]");
   w.factory("width[1, 0.5, 5]");
   w.factory("Gaussian::smodel(x, mass, width)");
   // create RooAddPdf in extended mode
   w.factory("nbackground[10000, 0, 1000000]");
   w.factory("nsignal[100, 0.0, 1000.0]");
   w.factory("SUM::model(nbackground*bmodel, nsignal*smodel)");
   //w.factory("SUM:model(nsig[100,0,10000]*sig_pdf, nbkg[1000,0,10000]*bkg_pdf)");  // for extended model
```

Esercitazione 13

- Fit the data: you need to call the RooAbsPdf::fitTo.

```
// fit the data and save its result. Use eventually the optional =Minuit2= minimiser
RooFitResult * res = pdf->fitTo(*data, Minimizer("Minuit2","Migrad"), Save(true) );
```

- Plot the resulting fit function in the same plot.
- Plot also the signal and background fit components with different colour and style

```
pdf->plotOn(plot);
//draw the two separate pdf's
pdf->plotOn(plot, RooFit::Components("bkg_pdf"), RooFit::LineStyle(kDashed) );
pdf->plotOn(plot, RooFit::Components("sig_pdf"), RooFit::LineColor(kRed),
RooFit::LineStyle(kDashed) );
plot->Draw();   // to show the RooPlot in the current ROOT Canvas
```

Esercitazione 13

Exercise 3

- Aim: create a combined model based on two channels and perform a simultaneous fit on both channels.
- The first channel is a Gaussian signal over an Exponential decay background. We assume that the location and width of the signal are known. The "mass" observable is distributed in range [0- 10] and the signal has mean ~ 2 and sigma 0.3. The exponential decay of the background is 2. We assume we have 1000 background events.
- The second channel will be similar to the first one, we will have the same signal model, but a different background model. We assume we have less and a flatter background. We have an exponential decay of 4 and 100 background events in total.
- We assume the number of signal events in the first channel is 50 and in the second channel is 30. We introduce a common rate parameter mu expressing the signal strength.
- You have to follow to following steps:
  - Create the models for the two channels
  - Create a combined model using simultaneous pdf (class RooSimultaneous).
  - Generate a combined data set, fit and plot the result

https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13

https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13

- Create first the two separate background models and a common Gaussian signal model
- Express the number of signal events in term of the signal strength mu: the number of signal events is the product of mu with a fixed number (50 for the first channel and 30 for the second).

```
RooWorkspace w("w");
w.factory("Exponential:bkg1_pdf(x[0,10], a1[-0.5,-2,-0.2])");
w.factory("Exponential:bkg2_pdf(x, a2[-0.25,-2,-0.2])");
w.factory("Gaussian:sig_pdf(x, mass[2], sigma[0.3])");
// express number of events as the
w.factory("prod:nsig1(mu[1,0,5],xsec1[50])");
w.factory("prod:nsig2(mu,xsec2[30])");

w.factory("SUM:model1(nsig1*sig_pdf, nbkg1[1000,0,10000]*bkg1_pdf)");  // for extended model
w.factory("SUM:model2(nsig2*sig_pdf, nbkg2[100,0,10000]*bkg2_pdf)");  // for extended model
```

- Make a simultaneous pdf * : create first the category label to identify the channels (RooCategory) * use the factory operator SIMUL for creating the RooSimultaneous pdf.

```
// Create discrete observable to label channels
w.factory("index[channel1,channel2]");
// Create joint pdf (RooSimultaneous)
w.factory("SIMUL:jointModel(index,channel1=model1,channel2=model2)");
```

https://twiki.cern.ch/twiki/bin/view/RooStats/RooFitTutorialMarch2015

Esercitazione 13

- Generate the events. We can generate directly a combined data sets from the model, with the number of events fluctuating around the number of expected events in each channel.

```
// generate combined data set
RooDataSet * data = pdf->generate( RooArgSet(*x,*index));
```

- Plot the data. We make two plots representing the two channels. We need then to tell to the combined data set to select the channel we want

```
RooPlot * plot1 = x->frame();
RooPlot * plot2 = x->frame();
data->plotOn(plot1,RooFit::Cut("index==index::channel1"));
data->plotOn(plot2,RooFit::Cut("index==index::channel2"));
```

- Fit the data is as before using the fitTo method.
- Plot the resulting model on the two data sets. We need to project the simultaneous pdf on the corresponding channel. This is done as following:

```
// fit
 RooFitResult * r = pdf->fitTo(*data, RooFit::Save(true), RooFit::Minimizer("Minuit2","Migrad"));
 r->Print();
 // plotting (draw the two separate pdf's )
 pdf->plotOn(plot1,RooFit::ProjWData(*data),RooFit::Slice(*w.cat("index"),"channel1"));
 pdf->plotOn(plot2,RooFit::ProjWData(*data),RooFit::Slice(*w.cat("index"),"channel2"));
 // draw the two plots in two separate pads:
 c1 = new TCanvas();
 c1->Divide(1,2);
 c1->cd(1); plot1->Draw();
 c1->cd(2); plot2->Draw();
```