Table of Contents

Fitting Using RooFit

Fitting using RooFit

Welcome to the hands-on session dedicated on fitting using RooFit. The aim is to start familiarizing with RooFit and trying understand the basic syntax of creating models using the workspace factory. We will also see how to save a workspace in a ROOT file which, allowing to perform the fitting analysis at a later stage or to share the models with other people.

RooFit provides also a separate User Guide (unfortunately it does not cover yet the workspace factory syntax), but it exists also a reduced booklet, showing the main functionality, which you can find here.

Exercise 1: Gaussian model and fit it to random generated data

We will start with a similar exercise we did for Root fitting. We will create a Gaussian model from which we will generate a pseudo-data set and then we will fit this data set.

Start directly creating the Gaussian model using the workspace factory, thus the syntax introduced in the lecture slide 15. One you have created the model, use the `generate()` method of the `RooAbsPdf` class to generate 1000 events. Try to plot the data set using `RooPlot` as shown in slide 12.

After, fit the model to the data and show the resulting fitted function as in slide 14.

At the end save the `RooWorkspace` object in a file, but before remember to import, by calling `RooWorkspace::import(data)`, the data set you have generated in the workspace. The workspace does not contains only the model, but also the data, allowing then to re-perform the analysis later on.

Hint Hide

- Use the syntax of the `RooWorkspace` factory to create first the variables (observables and parameters) of the Gaussian probability density function (p.d.f.), as shown in slide 15.

Every variable in RooFIt, when created needs to be created with a value a min and a max allowed range. Use 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 as half the range. To avoid undesired side effect the value given should be defined between the min and max of the given range.

- 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.

See also slide 21 to create the p.d.f.

- 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.
- You can then do as in slide 12 to generate the data set and plot it.
- To fit the data set with the pdf, you need to call the `RooAbsPdf::fitTo`. See slide 14 on how to fit and on how to plot the function after the fit.
- To save the model call, `RooWorkspace::write(file_name)`. See slide 19.

Below is the code solution:

Solution Hide

```
// make a simple Gaussian model
```

```
#include "RooWorkspace.h"
#include "RooRealVar.h"
#include "RooAbsPdf.h"
#include "RooDataSet.h"
#include "RooPlot.h"

#include "TCanvas.h"

using namespace RooFit;

void GaussianModel(int n = 1000) {


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

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


   // generate n gaussian measurement for a Gaussian(x, 1, 1);
   RooDataSet * data = pdf->generate( *x, n);

   data->SetName("data");

   // RooFit plotting capabilities
   RooPlot * pl = x->frame();
   data->plotOn(pl);
   pl->Draw();


   // remove thsi line if you want to fit the data
   return;

   // now fit the data set
   pdf->fitTo(*data);


   // plot the pdf on the same RooPlot object we have plotted the data
   pdf->plotOn(pl);

   pl->Draw();

   // import data in workspace (IMPORTANT for saving it )
   w.import(*data);

   w.Print();

   // write workspace in the file (recreate file if already existing)
   w.writeToFile("GaussianModel.root", true);

   cout << "model written to file " << endl;

}
```

Exercise 2: Reading a workspace from a file

Open the file you have just created in the previous exercise and get the `RooWorkspace` object ffrom the file. Get a pointer to the p.d.f describing your model, and a pointer to the data. Re-fit the data, but this time in the range [0,10] and plot the result.

Hint Hide

To read and analyse the workspace you need to do:

- Open the `TFile` object and use `TFile::Get` to get a pointer to the workspace using its name
- Once you have the workspace in memory, retrieve from it the p.d.f. and the data set with their names using `RooWorkspace::pdf` and `RooWorkspace::data`.
- Re-issue again the call to `RooAbsPdf::fitTo`. You can set the fit range using the `RooFit::Range(xmim,xmas)` command arg option in `fitTo`. See the reference documentation for all the possible options that you can pass (some are shown in the solution code).

Here is the solution. This macro (apart form the Range fit) can work and fit whatever workspace you have in the file. You just need to set the right names for file, workspace, global p.d.f. and data set.

Solution Hide

```
#include "RooWorkspace.h"
#include "RooAbsPdf.h"
#include "RooRealVar.h"
#include "RooPlot.h"
#include "RooDataSet.h"
#include "RooFitResult.h"

#include "TFile.h"


// roofit tutorial showing how to fit whatever model we get from a file

// we assume the name of the workspace is w
// the name of the pdf is pdf
// the name of the data is data
const char * workspaceName = "w";
const char * pdfName = "pdf";
const char * dataName = "data";


using namespace RooFit;



void fitModel(const char * filename = "GaussianModel.root" ) {

   // read file:
   // following lines are for reading workspace
   // and to check that is fine

   // Check if example input file exists
   TFile *file = TFile::Open(filename);

   // if input file was specified but not found, quit
   if(!file ){
      cout <<"file " << filename << " not found" << endl;
      return;
   }

   // get the workspace out of the file
   RooWorkspace* w = (RooWorkspace*) file->Get(workspaceName);
   if(!w){
      cout <<"workspace with name " << workspaceName << " not found" << endl;
      return;
   }

   // fit a pdf from workspace with name pdfName

   RooAbsPdf * pdf = w->pdf(pdfName);
   if (!pdf) {
```

```
      w->Print();
      cout << "pdf with name " << pdfName << " does not exist in workspace " << endl;
      return;
   }

   // get the data out of the file
   RooAbsData* data = w->data(dataName);

   if(!data ){
      w->Print();
      cout << "data " << dataName << " was not found" <<endl;
      return;
   }

   //------------------------------------------------------------------------
   //// real code starts here

   // get variable x (is the first of the data)
   RooRealVar * x = w->var("x");
   RooPlot * plot = x->frame();

   data->plotOn(plot);
   plot->Draw();

   // fit pdf - (example using option: save result and using diffferent minimizer

   // global fit

   pdf->fitTo( *data );

   // for doing a reduce fit in a Range (plus other options)

   RooFitResult * r = pdf->fitTo( *data, Save(true), Minimizer("Minuit2","Migrad"),Range(0.,10.)

   pdf->plotOn(plot);
   pdf->paramOn(plot);

   plot->Draw();

   // if we have a result we can do

   r->Print();

   r->correlationMatrix().Print();


}
```

Exercise 3: Fitting the spectrum histogram with background and gaussian peaks

We are going now to re-use the sub-histogram we have created with the IRMM data yesterday. We need first to import the histogram in a RooFit data object, the `RooDataHist`. See the lecture slide 13 on how to do this.

For fitting, we need to build the background plus a signal model (for example two peaks). We need first to build the pdf's for the signals (Gaussians) and then the p.d.f for the background (Polynomial). It is recommended to use the Chebyshev polynomial, which are orthogonal and make the fit more robust (see slide 24 on how to do it). Again it is better to fit the single components first and then the overall model. Note that the parameters in RooFit are shared between all the functions in the workspace. After fitting a single component p.d.f you don't need to set the parameters in the big sum p.d.f.

Note that Roofit automatically parametrizes also the pdf as normalized ones, so the number of peak signal events is always fitted from the data.

Compare what you obtain in terms of signal events with what we obtained with the exercise 1 yesterday.

Hint Hide

- Create each p.d.f component (polynomial, Gaussian1, and Gaussian2) using the factory. Add different parameters name for parameters which are not in common.
- When you introduce a parameter add also a sensible value and range.
- Do again first the separate fits of the various components, as we did yesterday, in order to have good initial parameters for the global fit.
- use the Chebyshev polynomial to make the fit more robust (replace `Polynomial` with `Chebychev` in slide 21).
- Build the overall model using the operator `SUM`. See slide 28.
- You can also plot the separate components using the `Components` option (see slide 29)

Solution Hide
Here below is the macro showing how to do this with RooFit:

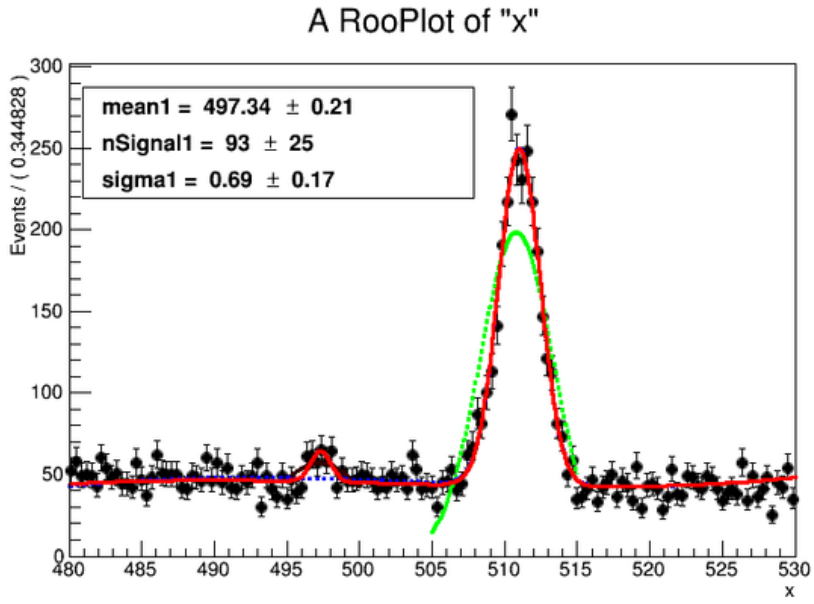
```
// fit spectrum histogram using RooFit

#include "RooWorkspace.h"
#include "RooAbsPdf.h"
#include "RooRealVar.h"
#include "RooPlot.h"
#include "RooDataSet.h"
#include "RooFitResult.h"
#include "RooDataHist.h"

#include "TFile.h"
#include "TH1.h"
#include "TPad.h"

// roofit totorial showing how to fit whatever model we get from a file

// we assume the name of the workspace is w
// the name of the pdf is pdf
// the name of the data is data
const char * workspaceName = "w";
const char * pdfName = "pdf";
const char * dataName = "data";
using namespace RooFit;

TH1D *SubHisto(TH1D *h0, char *name, Double_t xmin, Double_t xmax)
{

   int ifirst = h0->FindBin(xmin);
   int ilast = h0->FindBin(xmax);
   double axmin = h0->GetXaxis()->GetBinLowEdge(ifirst);
   double axmax = h0->GetXaxis()->GetBinUpEdge(ilast);

   int nbin = ilast - ifirst + 1;
   TString title = TString::Format("%s (from %g to %g)",h0->GetTitle(),axmin,axmax);
   TH1D * h1 = new TH1D(name,title, nbin, xmin,xmax);

   for (int i = ifirst; i<= ilast; ++i) {
      h1->SetBinContent( i-ifirst+1, h0->GetBinContent(i) );
   }


   return h1;
```

```
}


void RooFitSpectrum() {

   // make a model :
   // two gaussian + polynomial background

   RooWorkspace w;
   w.factory("x[480,530]");
   w.factory("Gaussian:g1( x, mean1[497,490,500],sigma1[1,0,100] )");
   w.factory("Gaussian:g2( x, mean2[510,480,530],sigma2[1,0,100] )");

   w.factory("a[0,0,1000]");
   w.factory("b[0,-100,100]");
   w.factory("c[0,-100,100]");
   //w.factory("Polynomial:bkg( x,{a,b})");
   // use a 2-degree polynomial
   w.factory("Chebychev:bkg( x,{a,b,c})");

   // define number of events
   w.factory("nSignal1[100,0,1000000]");
   w.factory("nSignal2[1000,0,1000000]");
   w.factory("nBackg[10000,0,1000000]");

   w.factory("SUM::pdf(nSignal1*g1,nSignal2*g2, nBackg*bkg)");

   RooRealVar * x = w.var("x");
   RooAbsPdf * pdf = w.pdf("pdf");

   TFile * file = TFile::Open("Spectrum.root");

   TH1D *h0 = (TH1D*) file->Get("h1");

   TH1D *h1 = SubHisto(h0, "hzoom_h1", x->getMin(), x->getMax());

   h1->Draw();

   // make a RooDataHist to be used by RooFit for fitting

   RooDataHist data("data","data",*x, Import(*h1) );

   RooPlot * pl = x->frame();
   data.plotOn(pl);


   // now we fit

   // fit first the background
   w.pdf("bkg")->fitTo(data);
   w.pdf("g1")->fitTo(data, RooFit::Range(495,500));
   w.pdf("g1")->plotOn(pl, LineColor(kRed+4), LineStyle(kDashed) );
   w.pdf("g2")->fitTo(data, RooFit::Range(505,515) );
   w.pdf("g2")->plotOn(pl, LineColor(kGreen), LineStyle(kDashed) );


   // fit with only g2+bkg
   w.factory("SUM::pdf0(nSignal2*g2, nBackg*bkg)");
   RooFitResult * r0 = w.pdf("pdf0")->fitTo(data, Save(true),Minimizer("Minuit2"));
   r0->Print();
   w.pdf("pdf0")->plotOn(pl, LineColor(kBlue), LineStyle(kDashed));
   pl->Draw();

   double minNLL0 = r0->minNll();


   RooFitResult * r = pdf->fitTo(data, Save(true),Minimizer("Minuit2"));
```

```
    r->Print();


    pdf->plotOn(pl, LineColor(kRed));

    double minNLL = r->minNll();

    std::cout << "Number of peak (497) events is : " << w.var("nSignal1")->getVal() << " +/- " <<
    std::cout << "significance = " << sqrt( (minNLL0 - minNLL) ) << std::endl;

    // import data in ws and write to file
    w.import(data);
    w.writeToFile("RooFitSpectrum.root",true);

    // plot also the fitted number of events , mean and sigma of the peak
    w.pdf("pdf")->paramOn(pl,Parameters(RooArgList(*w.var("nSignal1"),*w.var("mean1"),*w.var("sigm
    pl->Draw();


}
```

At the end you can also save the workspace in the file to re-use later, without worrying to re-create the model

Here is the plot that we get at the end using RooFit. Note also that we display the parameter values in the plot. See at the end of the code on how to do this.



Fit of the Spectrum using RooFit

This topic: Main > RootIRMMTutorial2013RooFitExercises
Topic revision: r4 - 2013-03-01 - LorenzoMoneta