2.4. ROOT Framework

Exercises are located at Exercises/4_ROOT/
To compile and run exercise programs use the line given in the head-comments in the code.
The results given here are obtained on Intel E7-4860 CPU with gcc4.7.3.

ROOT Introduction

ROOT is an object-oriented framework for data analysis.[1] ROOT includes abundant functionality for data processing: input, output, plots, curve fitting, minimisation, random generators, matrix computations, MIMD and SIMD.

This manual covers very basic usage of histogram plots and curve fitting and random generators.

ROOT is interpreter based environment, this means that ROOT reads and executes statements one by one. For comparison, compiler read full file, check for semantic errors first, create and executable, which can be executed afterwards separately. ROOT interpreter is called CINT, it is C++ based, so basically all standard C++ statements are allowed in CINT.

ROOT can be used in two different ways:

1. Start ROOT environment at system prompt (terminal):

root -l

then ROOT prompt (you will see “root[0]”) appears and you can input and execute statements from keyboard in the same way as with system prompt. To exit ROOT prompt use **.q** command.

2. Create a ROOT macros and process it with ROOT using:

root -l RootMacrosName.C

then ROOT will look for a function in the macros, which name is same as file name “RootMacrosName()”, and all statements in the function will be executed with interpreter, then ROOT will stop, allowing you to enter additional commands or to exit with .q command.

4_ROOT/FittingDemoSimple_0: description

This exercise show you basic usage of histograms. Histogram is a plot, which shows how many data of each type you have, it is called distribution. For example, it can shows number of days in 12 months, or number of students with each possible heights in cm. In the last case, since height is continuous scale (each student has it’s own height if we measured it precise enough) so called binning should be chosen. In the example above we chose bin size equal to 1 cm and put all students with same height within 1 cm to the one bin, that allows us to calculate height distribution. Bins with size of 10, 3 and 3.3 can be chosen too, for example.

The given macros **FittingDemoSimple_0.cpp** includes required parts of ROOT, describes a function to fit histogram **fitFunction** with and the main function **FittingDemoSimple_0**.

Instead of usual types like **int, float, double**, which can have different precision on different machines ROOT introduces machine independent types: **Int_t, Float_t, Double_t**.

To use graphics with ROOT one needs to declare **TCanvas** object, which has the following constructor:

TCanvas(const char* name, const char* title, Int_t wtopx, Int_t wtopy, Int_t ww, Int_t wh)

here **wtopx, wtopy** are the pixel coordinates of the top left corner of the canvas, **ww** is the canvas size in pixels along X, **wh** is the canvas size in pixels along Y.

Canvas should be allocated dynamically variable, otherwise it would be deleted upon function finish and will not be seen.

---

[1] http://root.cern.ch

The histogram object is called **TH1F**, where H states for histogram, 1 for 1-dimensional, F - for **float** type:

TH1F(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)

here **nbins** is number of bins, **xlow** is low edge of first bin, **xup** is upper edge of last bin.

To generate distribution random generator is used **TRandom3** class provides the most optimal random generator in ROOT, its member function **Uniform** provides floating point numbers uniformly distributed from 0 to 1. The generated numbers is added one by one to histogram using **Fill** function.

The histogram is fitted with quadratic function. For this TF1 object is created:

TF1(const char* name, void* fcn, Double_t xmin, Double_t xmax, Int_t npar)

here **fcn** is C++ function name, **xmim**, **xmas** is range of the function argument, **npar** is number of parameters to be fitted. The **fcn** must have two arguments: array of arguments and array of parameters. C++ function **fitFunction** is given to the object here, it has one argument and 3 parameters. The fitting procedure is executed by call of **Fit** function on histogram variable.

The task for this exercise is to :

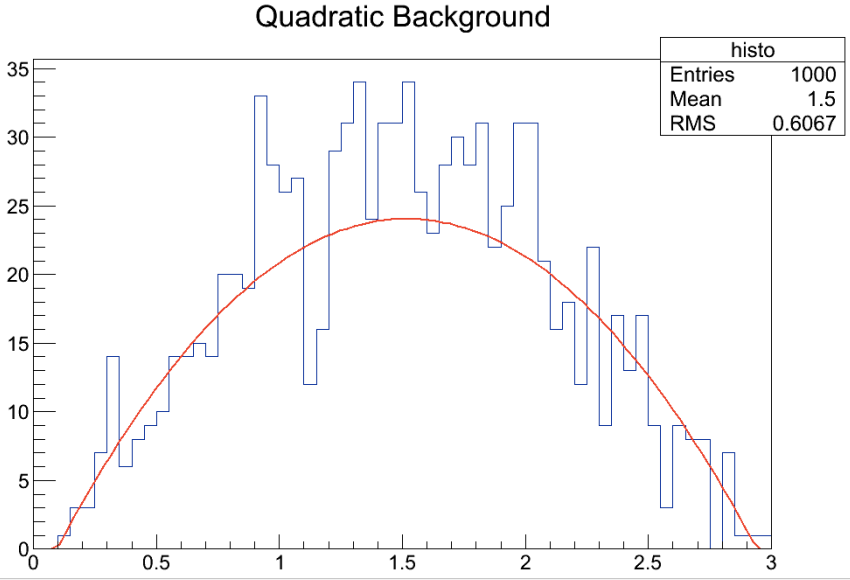
1) Increase the number of entries to the histogram up to 1 000 000.
2) Chose a proper function to fit obtained distribution and perform fit.

Part of the source code of FittingDemoSimple_0.cpp

```
9  #include "TH1.h"
10 #include "TMath.h"
11 #include "TF1.h"
12 #include "TLegend.h"
13 #include "TCanvas.h"
14 #include "TRandom3.h"
15
16   // Quadratic function
17   // x   - array of arguments
18   // par - array of parameters
19 Double_t fitFunction(Double_t *x, Double_t *par) {
20    return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
21 }
22
23 void FittingDemoSimple_0() {
24
25    const int nBins = 60;
26
27    TCanvas *c1 = new TCanvas("c1","Fitting Demo",10,10,700,500);
28
29    TH1F *histo = new TH1F("histo", "Quadratic Background",60,0,3);
30
31    TRandom3 rand;
32    for(int i=0; i < 1000; i++) histo->Fill( (rand.Uniform()+rand.Uniform())*1.5
   ); // fill with triangular distribution
33
34      // create a TF1 with the range from 0 to 3 and 3 parameters
35    TF1 *fitFcn = new TF1("fitFcn",fitFunction,0,3,3);
36
37      // fit histo by fitFcn
38    histo->Fit("fitFcn");
39 }
```

Typical output

```
Processing FittingDemoSimple_0.C...
 FCN=93.5605 FROM MIGRAD    STATUS=CONVERGED      54 CALLS          55 TOTAL
                     EDM=1.47795e-17    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           -3.32978e+00   8.53859e-01   1.65270e-03   7.48073e-09
   2  p1            3.60739e+01   1.71118e+00   7.72138e-04  -2.53062e-09
   3  p2           -1.18800e+01   5.35181e-01   2.79446e-04  -1.69088e-08
```



4_ROOT/FittingDemo_1: description

This exercise show you more advanced usage canvas options.

Here we use data from a physics experiment, it is saved as a constant array (line 35). The given data contains measurement, which defined by two processes: background process and signal process. Background process gives us quadratic distribution, signal - gives Lorenz distribution. Therefore the data should be fitted by sum of this functions (line 27). Since the function has 6 parameters fitting procedure has no idea about, to help fitting procedure it is better to initialise them by some values, **SetParameters** and **SetParameter** member-functions is used for this. Different options are used for fitting: **0** to do not plot fit immediately, **V** for verbose mode, **+** to save previous fit results, , all list of options and they description can be found at ROOT manual[2]. Creating in addition **background** and **lorentzianPeak** functions we can draw all three of them, showing background and signal components separately (lines 82-89).

The macro also shows how to change colours of canvas (lines 42-44) and style of histogram (lines 48-50), and fitting curves (lines 56-58,76,78,79).

In addition legend is drawn, it contains short description of each object on the picture.

The task is to:

Change background function to a higher order polynomial (order 3,4,6)

[2] http://root.cern.ch/root/html534/TH1.html#TH1:Fit

Part of the source code of FittingDemoSimple_0.cpp

```
12 // Quadratic background function
13 Double_t background(Double_t *x, Double_t *par) {
…
18 // Lorenzian Peak function
19 Double_t lorentzianPeak(Double_t *x, Double_t *par) {
…
25 // Sum of background and peak function
26 Double_t fitFunction(Double_t *x, Double_t *par) {
27   return background(x,par) + lorentzianPeak(x,&par[3]);
28 }
29
30 void FittingDemo_1() {
31  //Bevington Exercise by Peter Malzacher, modified by Rene Brun
32
33    const int nBins = 60;
34
35    Double_t data[nBins] = { 6, 1,10,12, 6,13,23,22,15,21,
36                             23,26,36,25,27,35,40,44,66,81,
37                             75,57,48,45,46,41,35,36,53,32,
38                             40,37,38,31,36,44,42,37,32,32,
39                             43,44,35,33,33,39,29,41,32,44,
40                             26,39,29,35,32,21,21,15,25,15};
41    TCanvas *c1 = new TCanvas("c1","Fitting Demo",10,10,700,500);
42    c1->SetFillColor(33);
43    c1->SetFrameFillColor(41);
44    c1->SetGrid();
45
46    TH1F *histo = new TH1F("histo",
47       "Lorentzian Peak on Quadratic Background",60,0,3);
48    histo->SetMarkerStyle(21);
49    histo->SetMarkerSize(0.8);
50    histo->SetStats(0);
51
52    for(int i=0; i < nBins;  i++) histo->SetBinContent(i+1,data[i]);
53
54    // create a TF1 with the range from 0 to 3 and 6 parameters
55    TF1 *fitFcn = new TF1("fitFcn",fitFunction,0,3,6);
56    fitFcn->SetNpx(500);
57    fitFcn->SetLineWidth(4);
58    fitFcn->SetLineColor(kMagenta);
…
65    fitFcn->SetParameters(1,1,1,1,1,1);
66    histo->Fit("fitFcn","0");
67
68    // second try: set start values for some parameters
69    fitFcn->SetParameter(4,0.2); // width
70    fitFcn->SetParameter(5,1);   // peak
71
72    histo->Fit("fitFcn","V+");
73
```

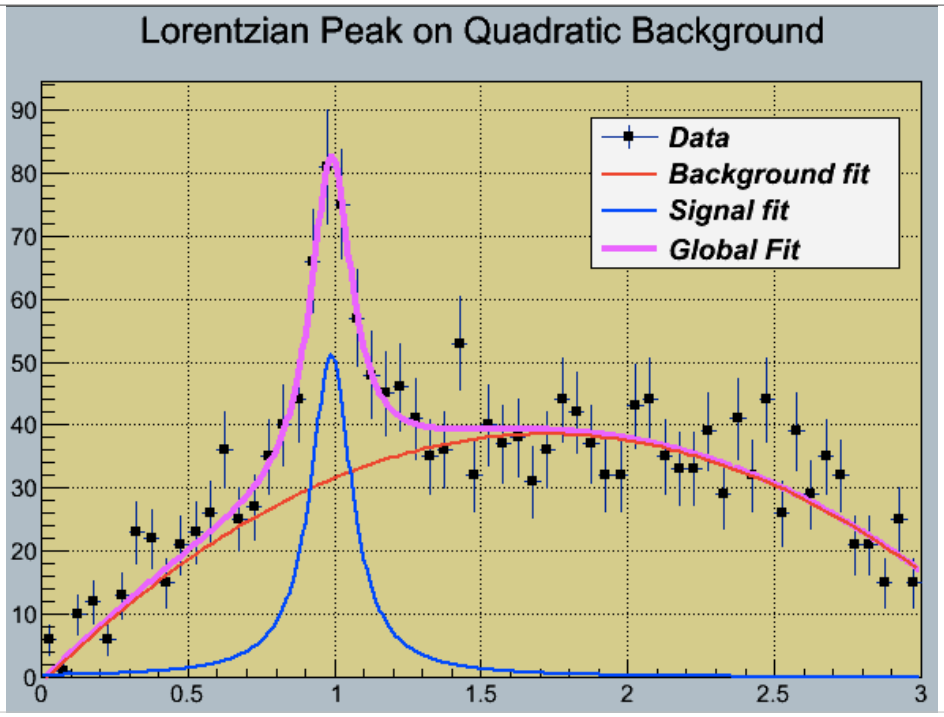
Part of the source code of FittingDemoSimple_0.cpp

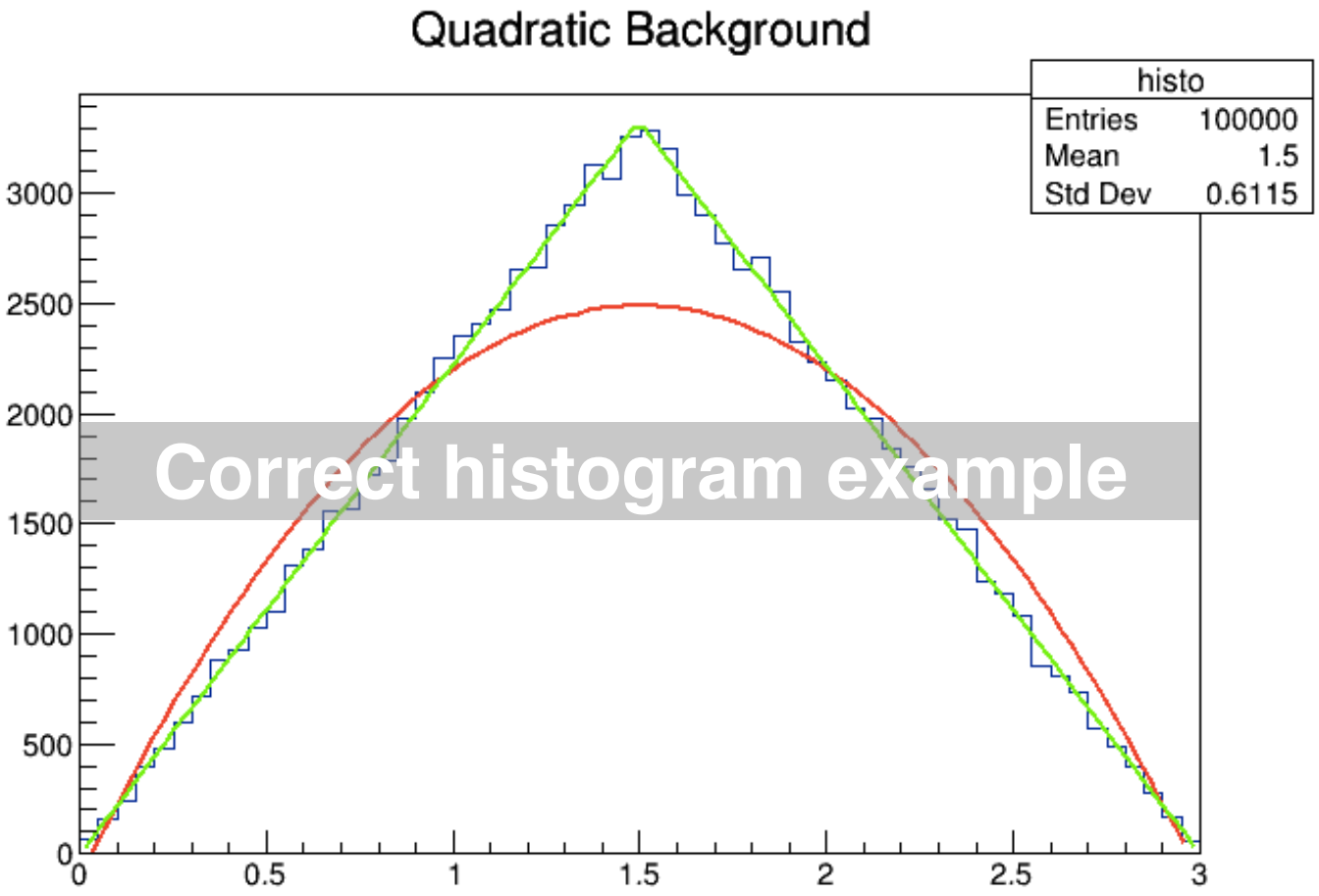
```
74    // improve the picture:
75    TF1 *backFcn = new TF1("backFcn",background,0,3,3);
76    backFcn->SetLineColor(kRed);
77    TF1 *signalFcn = new TF1("signalFcn",lorentzianPeak,0,3,3);
78    signalFcn->SetLineColor(kBlue);
79    signalFcn->SetNpx(500);
80    Double_t par[6];
81
82    // writes the fit results into the par array
83    fitFcn->GetParameters(par);
84
85    backFcn->SetParameters(par);
86    backFcn->Draw("same");
87
88    signalFcn->SetParameters(&par[3]);
89    signalFcn->Draw("same");
90
91    // draw the legend
92    TLegend *legend=new TLegend(0.6,0.65,0.88,0.85);
93    legend->SetTextFont(72);
94    legend->SetTextSize(0.04);
95    legend->AddEntry(histo,"Data","lpe");
96    legend->AddEntry(backFcn,"Background fit","l");
97    legend->AddEntry(signalFcn,"Signal fit","l");
98    legend->AddEntry(fitFcn,"Global Fit","l");
99    legend->Draw();
100
101 }
```

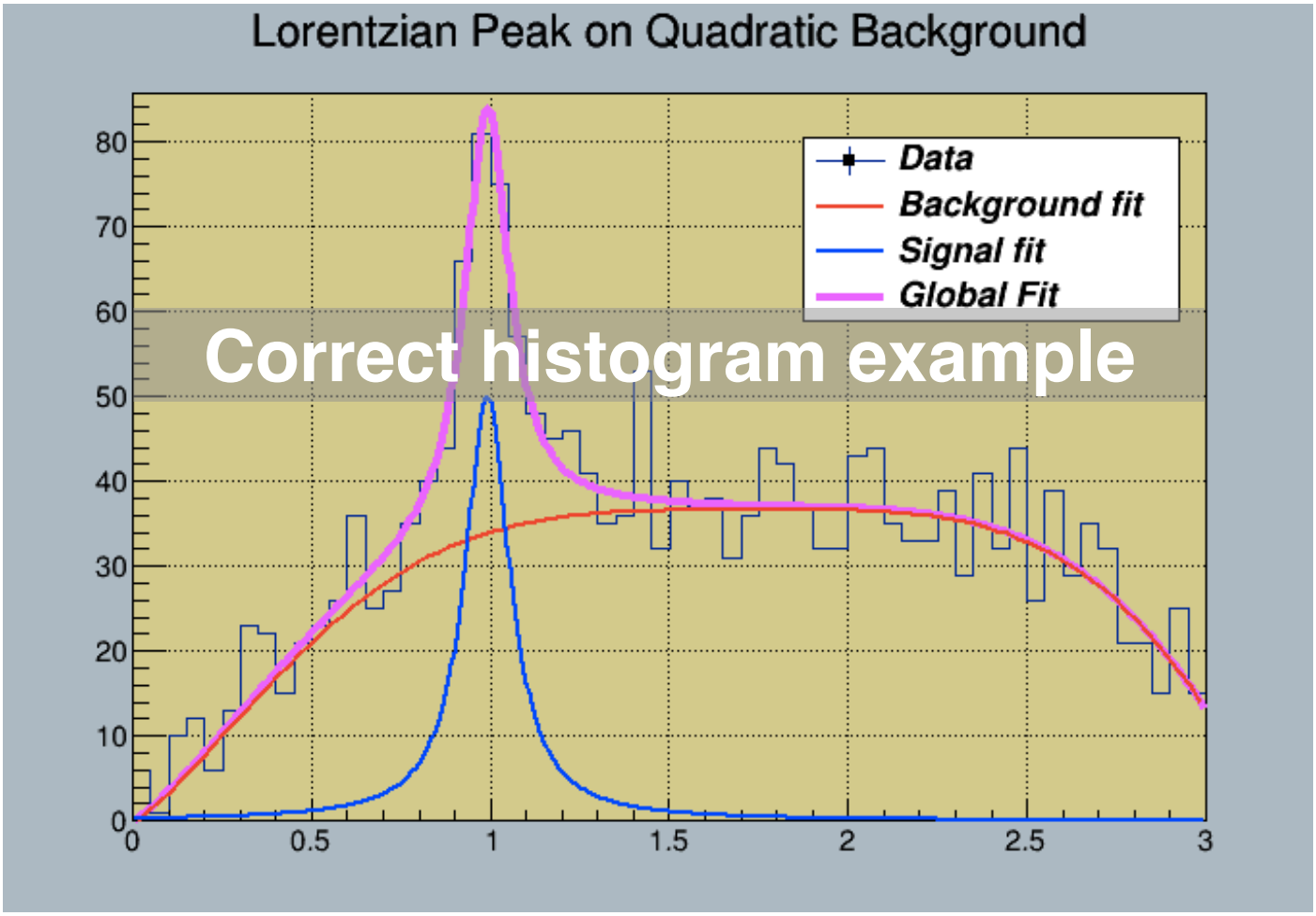
Begin of typical output

```
Processing FittingDemo_1.C...
 FCN=58.9284 FROM MIGRAD    STATUS=CONVERGED     618 CALLS         619 TOTAL
                     EDM=1.54329e-09    STRATEGY= 1  ERROR MATRIX UNCERTAINTY
1.2 per cent
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0          -8.64715e-01   8.87889e-01   3.02210e-05  -3.15277e-06
...
```

```
Begin of typical output
```




Quadratic Background
histo
Entries 100000
Mean 1.5
Std Dev 0.6115
3000
2500
2000
1500
1000
500
0
0
0.5
1
1.5
2
2.5
3
Correct histogram example


Lorentzian Peak on Quadratic Background
Data
Background fit
Signal fit
Global Fit
Correct histogram example
0
10
20
30
40
50
60
70
80
0
0.5
1
1.5
2
2.5
3