Introduction to ROOT (and some C++)

Bjoern Penning

Introduction

- Elementary techniques for data analysis and visualization with ROOT are discussed.
- This tutorial was until recently all C++,
  - Code: https://dl.dropboxusercontent.com/u/1882892/example_code_py.tar.gz
  - C++ examples are included as well
- ROOT utilizes C++ syntax but programming is fairly easy since ROOT offers a built-in compiler/interpreter.
- Even beginners should be able to gain the necessary skills quickly…
- When discussion code the name of the file is in the title

About ROOT

- ROOT is an object orientated software package for data analysis and visualization. It's based on the C/C++.
- Resource at `http://root.cern.ch`
  - Executables and source course for several OS (Linux, Windows, Mac OS X)
  - User's Guide
  - Reference Guide
  - Tutorials, Howto's etc
  - Mailing list

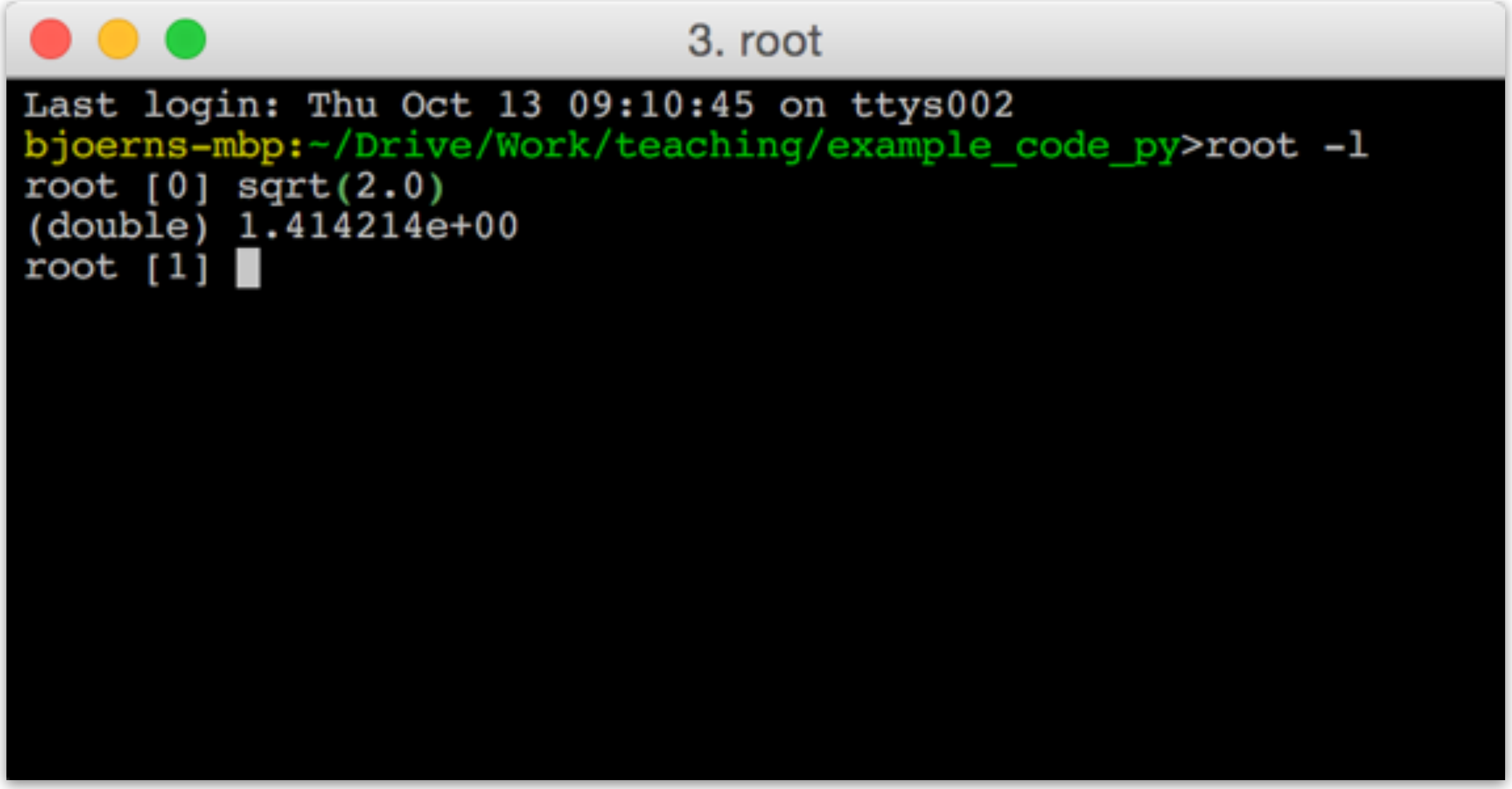
The ROOT shell

- After installing root on your machine execute in a shell:
  `#>root`

- The `ROOT` shell (CINT) appears, this is `ROOT's` built in interactive C/C++ interpreter

  - Exit using "`.q`"

- Use CINT to enter commands, e.g `sqrt(2)` (see sample left)



First steps

- We will discuss a few simple and short programs that illustrate quite some of the built in power
- You can edit these files with any editor, (e.g. emacs,vi, pico)
- Execute the code by executing '`#>python example.py`

Import libraries to the corresponding commands become available

Print text and output of simple calc. Note how we reference the library

```
{
  gRoot->Reset();
  cout << "Example 1: "<<endl;
  cout << "Sqrt of 2 = "<<sqrt(2.)<<endl;
}
```

Basic histogram operations

- Create a histogram myH1 with 10 bins and a range from 0. to 1. by using:

```
TH1F* myH1=new TH1F("myHisto","Distribution 0. to 1.",10,0.,1.);
```

name

title

number of bins and range

- Insert value x to the histogram by the `Fill()` method

```
myH1->Fill(x);
```

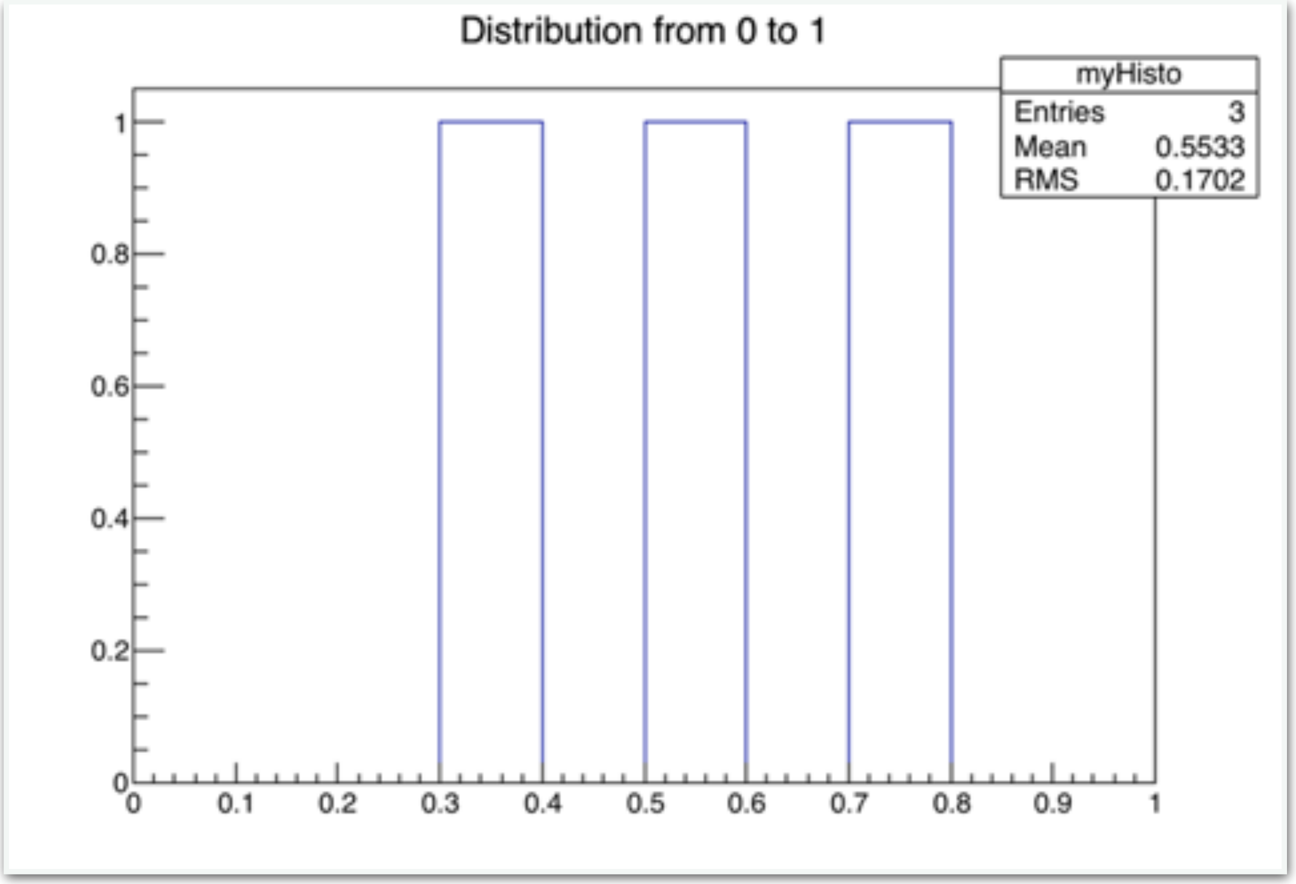
- Draw the histogram by calling the `Draw()` method

```
myH1->Draw();
```

histogram.C

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain"); //cleaner style
  TH1F* myH1 = new TH1F("myHisto","Distribution from 0 to 1",10,0.,1.);
  myH1->Fill(0.37);
  myH1->Fill(0.78);
  myH1->Fill(0.51);
  myH1->Draw();
}
```

- The following examples creates a histogram, fill, and draws is
- Output on the right:



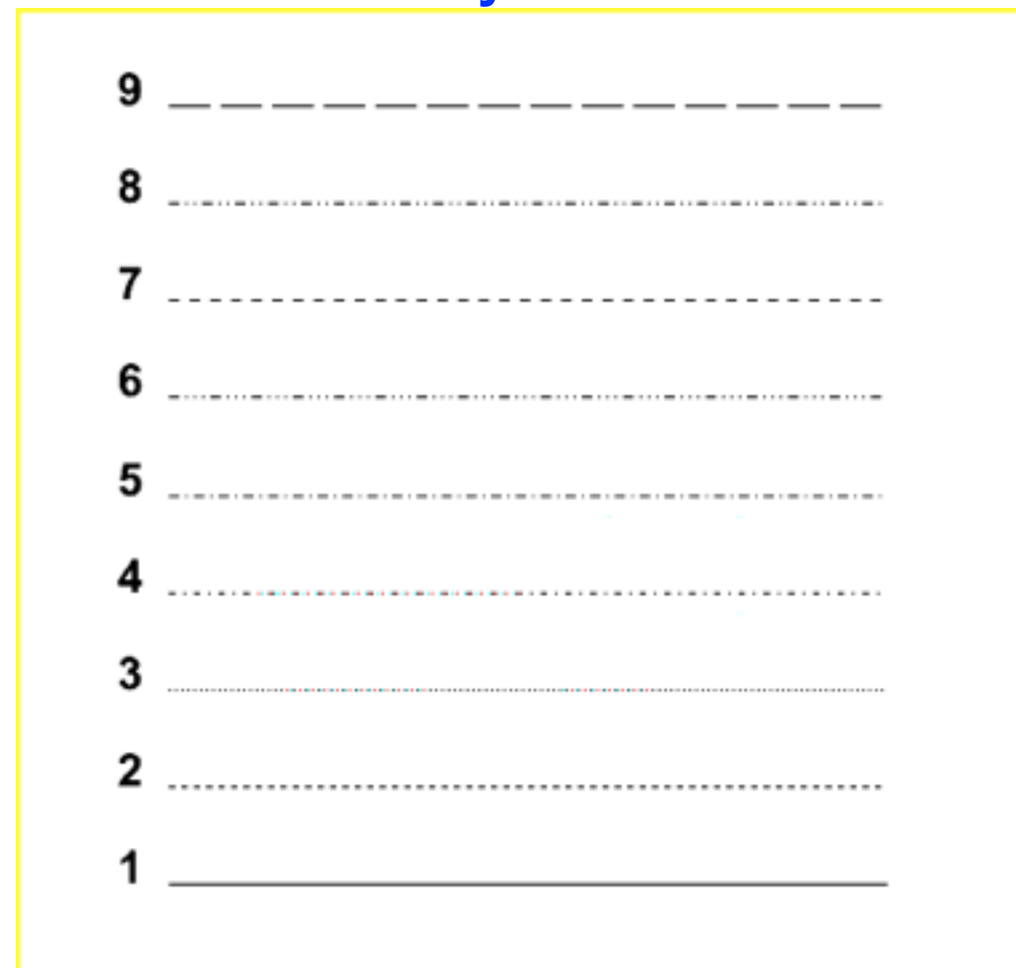
Drawing Options

- There are many arguments associated with `Draw()`. The most important are the following, more in the root manual
  - "`E`" draw the error bars
  - "`SAME`" overlay an already drawn histo with another one
  - "`C`" connect the data points with a smooth curvature
- most options can be combined, e.g.: "`SAME,E`" "`SAMEE`" respectively
- Many options are common for several ROOT objects, e.g. mathematical functions TF1
- other useful methods for histograms are:
  - `myH1->SetLineColor(4)`
  - `myH1->SetLineWidth(3)`
  - `myH1->SetFillColor(2)`
  - `myH1->SetFillStyle(3005)`
  - `myH1->SetLineStyle(2)`

Styleguide

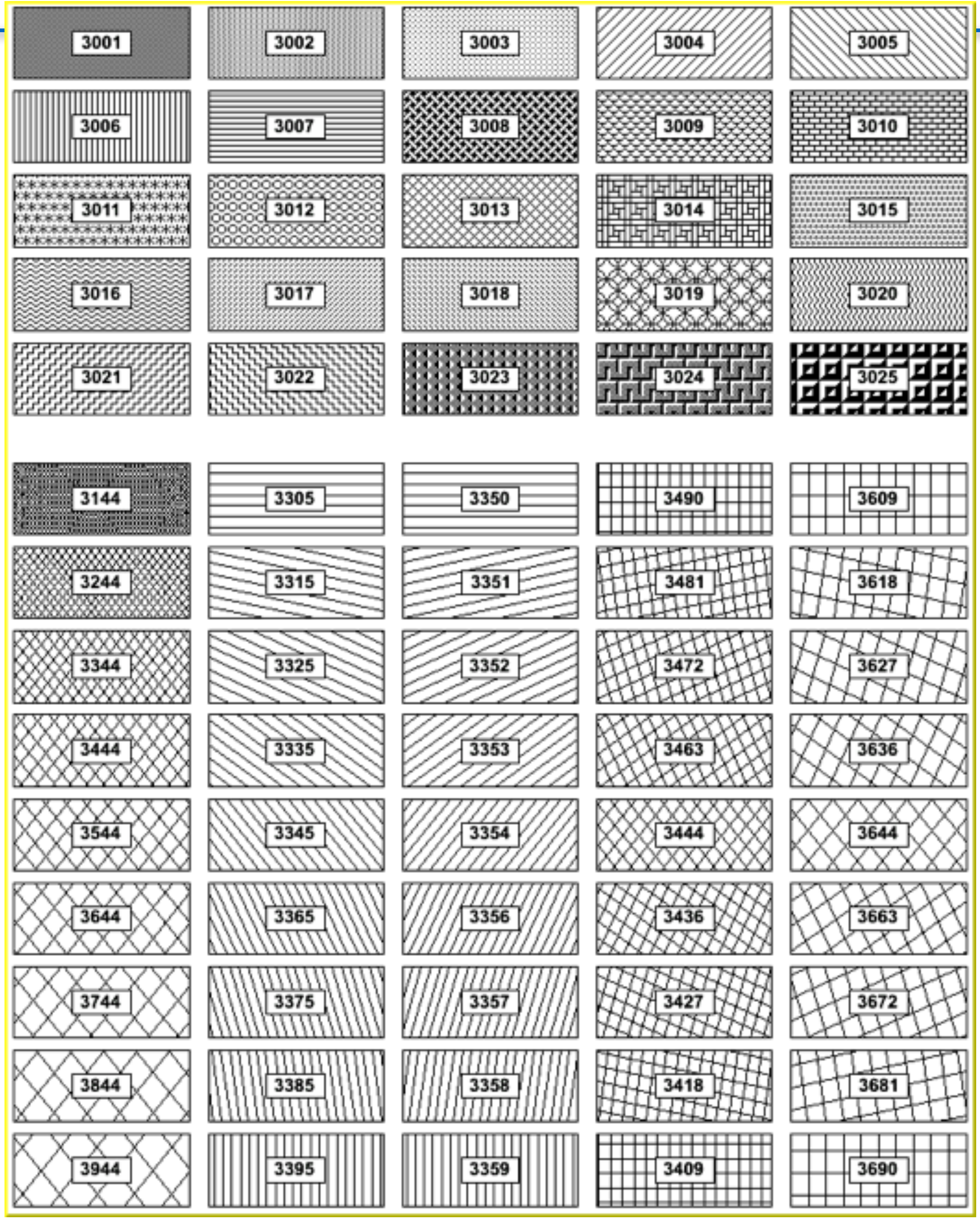
colors

line styles



marker styles

Fill Patterns

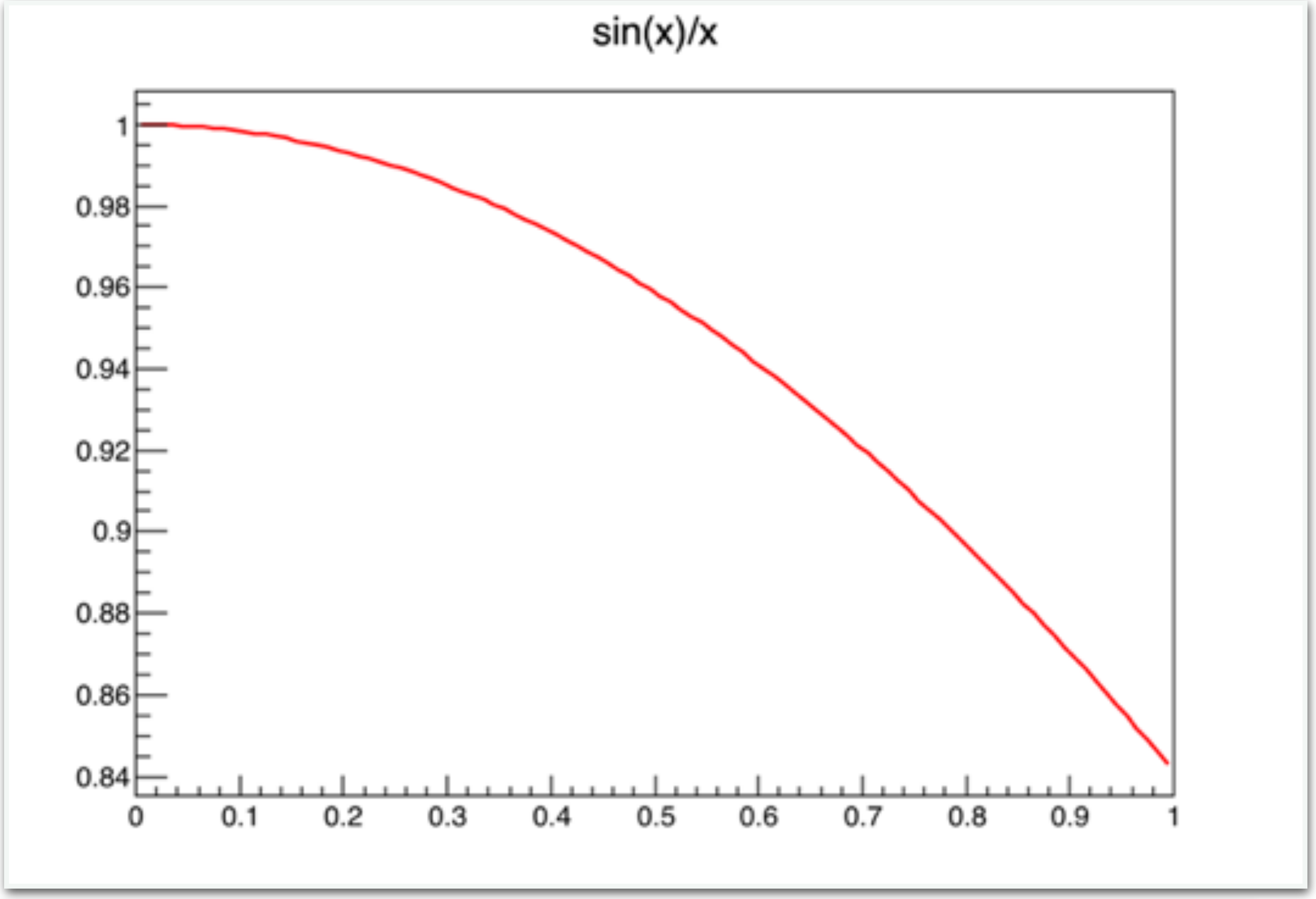


function.C

- ROOT is able to display mathematical functions and graphs as well. The following macros draws the function sin(x)/x within a range of 0 to 1

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TF1 *myFunc = new TF1("myFunction","sin(x)/x",0.,1.);
  myFunc->Draw();
}
```

name function range



- TF1 are useful to fit parameters

TFunction Options

- There are many functions predefined in TMath. Those are particularly useful for fitting, a small selection:
  - `pol1, pol2, pol3....`
  - `gaus`
  - `Landau`
  - `BreitWigner`
  - `sin, cos, ...`
  - `sqrt`
  - `exp`
  - `log`
  - `...`
- Functions can be combined with each other whereas the number of parameters and initial parameters can be chosen (almost) freely. More information regarding this can be found here:

  http://root.cern.ch/root/html402/TFormula.html

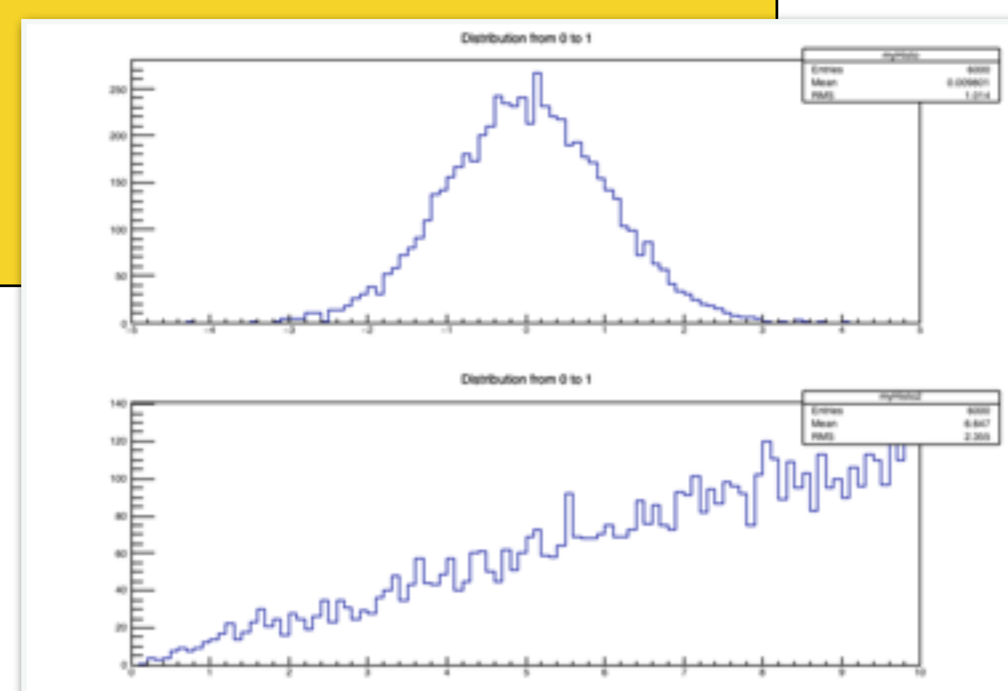
histogram2.C

- The `Canvas` represents the windows on which histograms and graphs are drawn. Calling `Draw()` automatically creates a canvas. Creating a canvas manually allows to apply various changes, e.g. drawing separate histograms, adjusting ist size, saving etc.

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TH1F* myH1 = new TH1F("myHisto","Distribution from -5 to 5",
100,-5.,5.);
  myH1->FillRandom("gaus",6000);
  TH1F* myH2 = new TH1F("myHisto2","li. Distribution",100,0.,10.);
  TF1* f1=new TF1("f1","2*x",0,10);
  myH2->FillRandom("f1",6000);
  TCanvas* c1=new TCanvas("myCanvas");
  c1->Divide(1,2);
  c1->cd(1);
  myH1->Draw();
  c1->cd(2);
  myH2->Draw();
}
```

split canvas in 1 column, 2 lines

select 1st and 2nd subcanvas



fit.C

- This examples illustrates how to perform a fit to data using the `Fit()` function. To do so an appropriate function within a sensible range has to be used.

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TH1F* myH1 = new TH1F("myHisto","gaussian distribution",100,-5.,5.);
  TF1* myGaus = new TF1("myGaus","gaus",-5,5);
  myH1->FillRandom("gaus",6000);
  myH1->SetMarkerColor(2);
  myH1->SetMarkerStyle(20);
  myH1->Fit("myGaus");
  myH1->Draw("E");
  cout<<" --------------------------------" <<endl;
  cout<<" chi2/dof: "<< myGaus->GetChisquare()/myGaus->GetNDF()<<endl;
  cout<<" mean: "<< myGaus->GetParameter(1) <<"+/-"<<myGaus->GetParError(1)<<endl;
  cout<<" width: "<< myGaus->GetParameter(2) <<"+/-"<<myGaus->GetParError(2)<<endl;
}
```

fit range

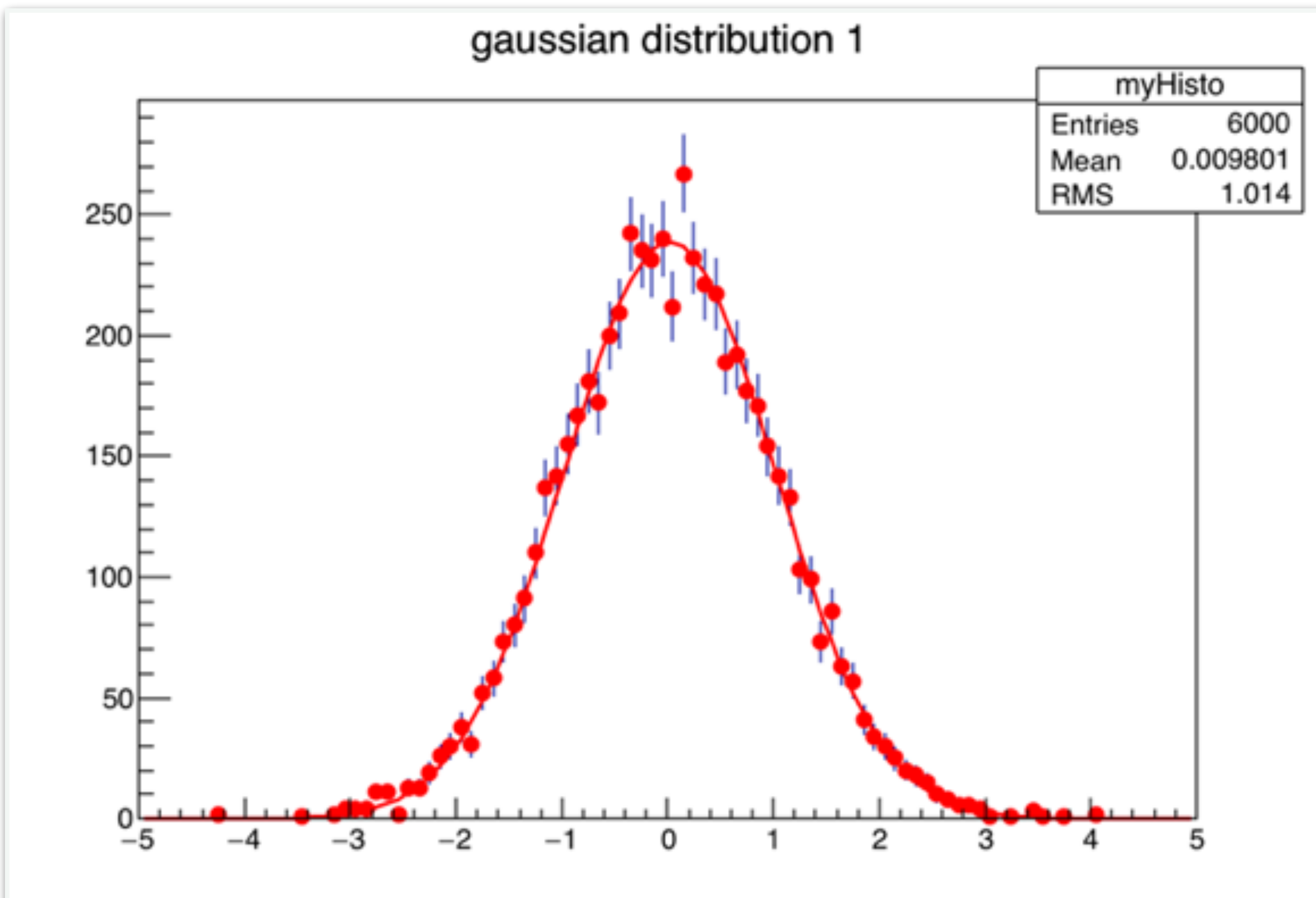
change marker & color

draw only data points with error bars

print fit parameters and fit quality

Output

```
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
 FCN=65.2581 FROM MIGRAD    STATUS=CONVERGED      62 CALLS          63 TOTAL
                     EDM=9.11306e-10    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     2.38594e+02   3.80928e+00   1.23098e-02  -6.27631e-06
   2  Mean         2.05919e-02   1.29916e-02   5.15403e-05   2.31062e-03
   3  Sigma        9.93349e-01   9.33666e-03   1.00053e-05   2.46256e-03
Info in <TCanvas::Print>: file fit.png has been created
 --------------------------------
 chi2/dof: 0.974002076002
 mean: 0.0205918892051+/-0.0129916016934
 width: 0.993349327318+/- 0.00933665930944
```



saveHisto.C

- Another useful tool is saving histograms to files and retrieving them again. In order to do so we have to use the TFile object:

```
TFile* file=new TFile("file.root", "RECREATE")
```

file name

if file exists it will be re-created

- ROOT stores the object in the last TFile object which has been accessed (use `cd()` to change there if necessary)

```
myH1->Draw()
```

- One has to close the file properly before exiting the program:

```
file->Close()
```

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TH1F* myH1 = new TH1F("myHisto","gaussian distribution",100,-5.,5.);
  TFile* _file=new TFile("histo.root","RECREATE");
  myH1->FillRandom("gaus",6000);
  myH1->Draw();
  myH1->Write();
  _file->Close();
}
```

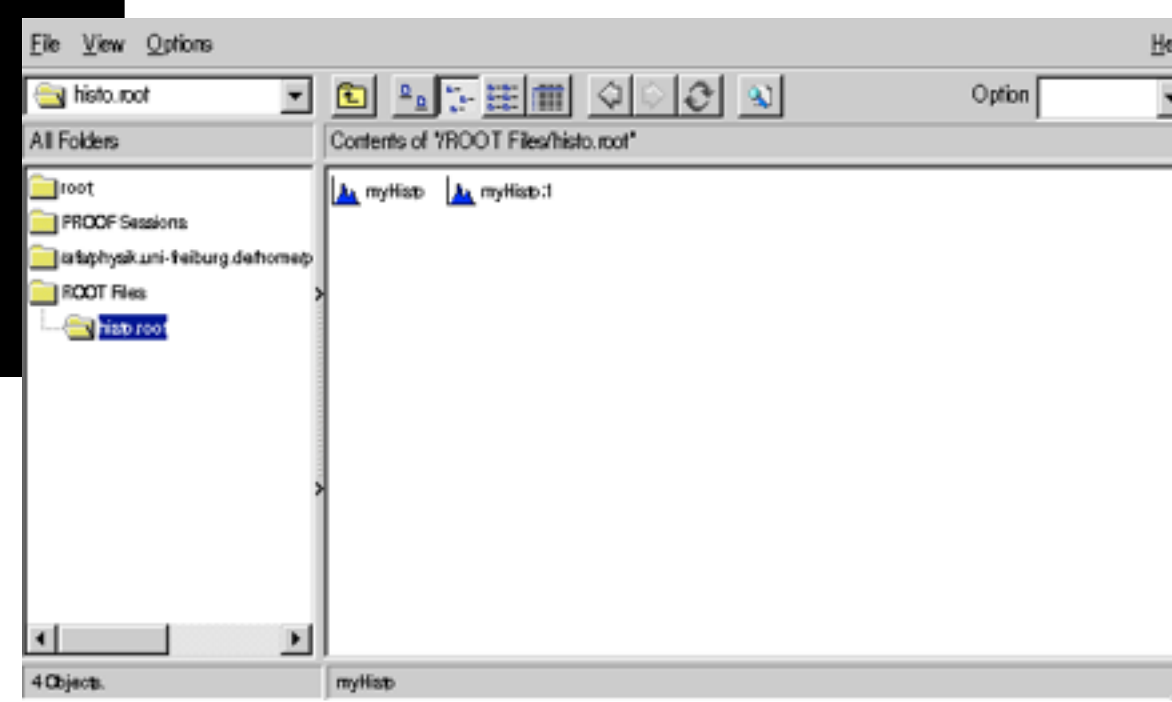
TBrowser

- Objects in `ROOT` files can be either read manually using the `TBrowser` or by using a script and reuse them.
- The `TBrowser` is an interactive browser for `ROOT` objects.
- To open the file in a simple `ROOT` session it's most easy to pass the file name as argument when calling the shell command:

```
#>root filename.root
```

```
penning@haco05:~fp/root/myprogs>root histo.root
root [0]
Attaching file histo.root as _file0...
root [1] new TBrowser
(class TBrowser*)0x8cd8950
```

starting TBrowser



getHisto.C

- It's often easier to load an object into a macro and use it later again.

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TFile* _file=new TFile("histo.root","OPEN");
  TH1F* _myH1 = (TH1F*)_file->Get("myHisto");
  _myH1->Draw();
}
```

- The procedure, consisting of `TObject->Write()`

  and `File->Gget('myHisto')`

  can can be used for many ROOT objects, (TF1, TH1, Canvas, etc).

readFile.C

- Read in data from a text file like this:

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  #include "Riostream.h"
  ifstream in;
  in.open("peaks.dat");
  Float_t xi;
  Int_t nlines = 0;
  TFile* _file = new TFile("readData.root","RECREATE");
  TH1F* _histo = new TH1F("_histo","Peaks", 1250, 0., 1250 );

  while (1) {
   in >> xi;
    if (!in.good()) break;
    _histo->SetBinContent( nlines, xi );
    nlines++;
  }
  cout<<"found "<<nlines<<" data points"<<endl;
  in.close();
  _histo->Draw();
  _file->Write();
}
```

open the file

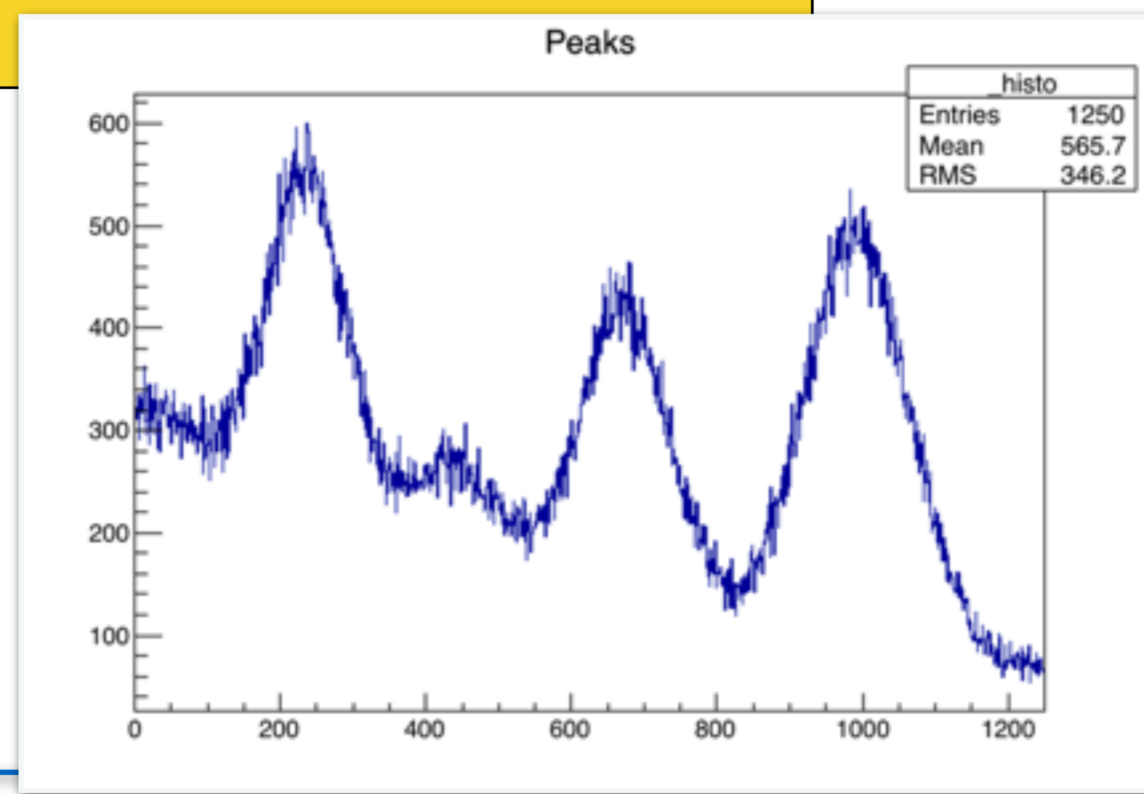
read in elements

filling histogram,
each line i corresponds to bin i

Input file:

```
322
323
322
312
314
335
291
331
329
...
```

Output:



More Fitting

- In the next example we are going to fit a Gaussian peak on top of an falling background. We will used a combination of Gaussian and the linear function to fit this examples

```
TF1* fitFunc = new TF1("fitFunc","pol1(0)+gaus(2)",0,300)
```

- Here

```
pol(0)+gaus(2) =
[0]+[1]*x+[2]*exp(-0.5*((x-[3])/[4])**2)
```

$$= \quad a + b \cdot x \cdot + c \cdot e^{-0.5\left(\frac{x-d}{e}\right)^2}$$

- In the expression "`pol1(0)+gaus(2)`" `(0)` resp. `(2)` correspond to the variables of the formula
- Useful functions:
  - `func.SetParameter(indices,value);`
  - `func.SetParLimits(indice, lower_boundary,  upper_boundary);`
  - `r.gStyle.SetOptFit();`

fit2.C

```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  gStyle->SetOptFit();
  #include "Riostream.h"
  ifstream in;
  in.open("peak.dat");
  Int_t xi;
  Float_t yi;
  Int_t nlines = 0;
  TH1F* _histo = new TH1F("_histo","Peaks", 350, 0., 350 );
  while (1) {
    in >> yi >> xi;
    if (!in.good()) break;
    _histo->SetBinContent( yi, xi );
    nlines++;
  }
  cout<<"found "<<nlines<<" data points"<<endl;
  TF1* fitFunc = new TF1("fitFunc","pol1(0)+gaus(2)",0,300);
  fitFunc->SetParameter(3,175);
  fitFunc->SetParameter(4,20);
  in.close();
  _histo->Rebin(3);
  _histo->Fit("fitFunc");
  _histo->Draw("E");
}
```

display fit parameters in statistics box

set value of bin x_i directly

set meaningful initial value of mean and width of the Gaussian

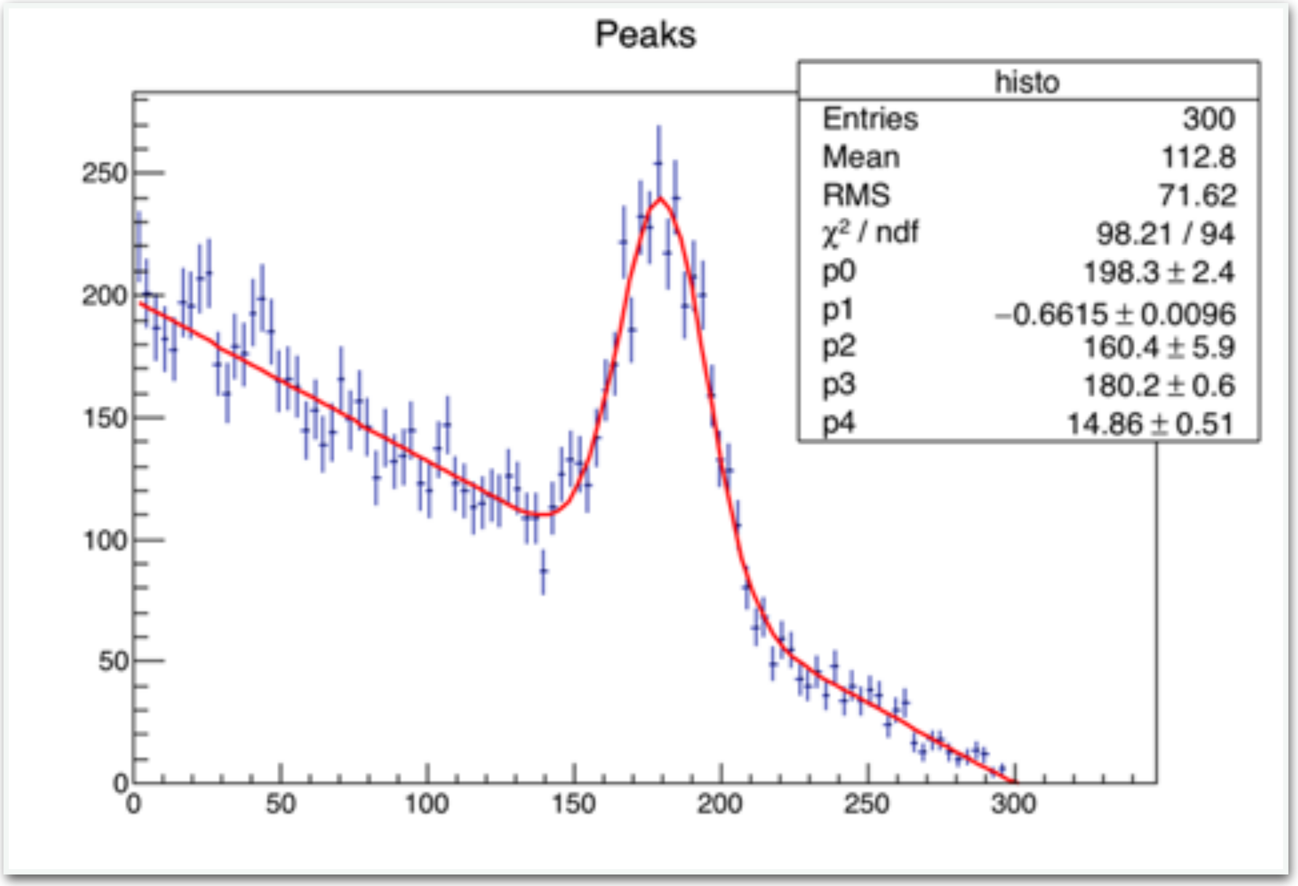
change binning of histogram

fit2.C

- Input file contains now two columns:

```
0        0
1        63
2        79
3        78
4        66
5        65
6        70
7        70
...
```

- Result:



linRegression.py

- Ge the covariance matrix as follows:

access the fitter object

```
TVirtualFitter *fitter = TVirtualFitter::GetFitter();
TMatrixD *matrix = new TMatrixD(2,2,fitter->GetCovarianceMatrix());
matrix->Print();
```

access the cov. matrix

dim. of matrix

- One has to pass the correct dimensions when initializing the matrix:
- (n x n), n = number of fit parameter

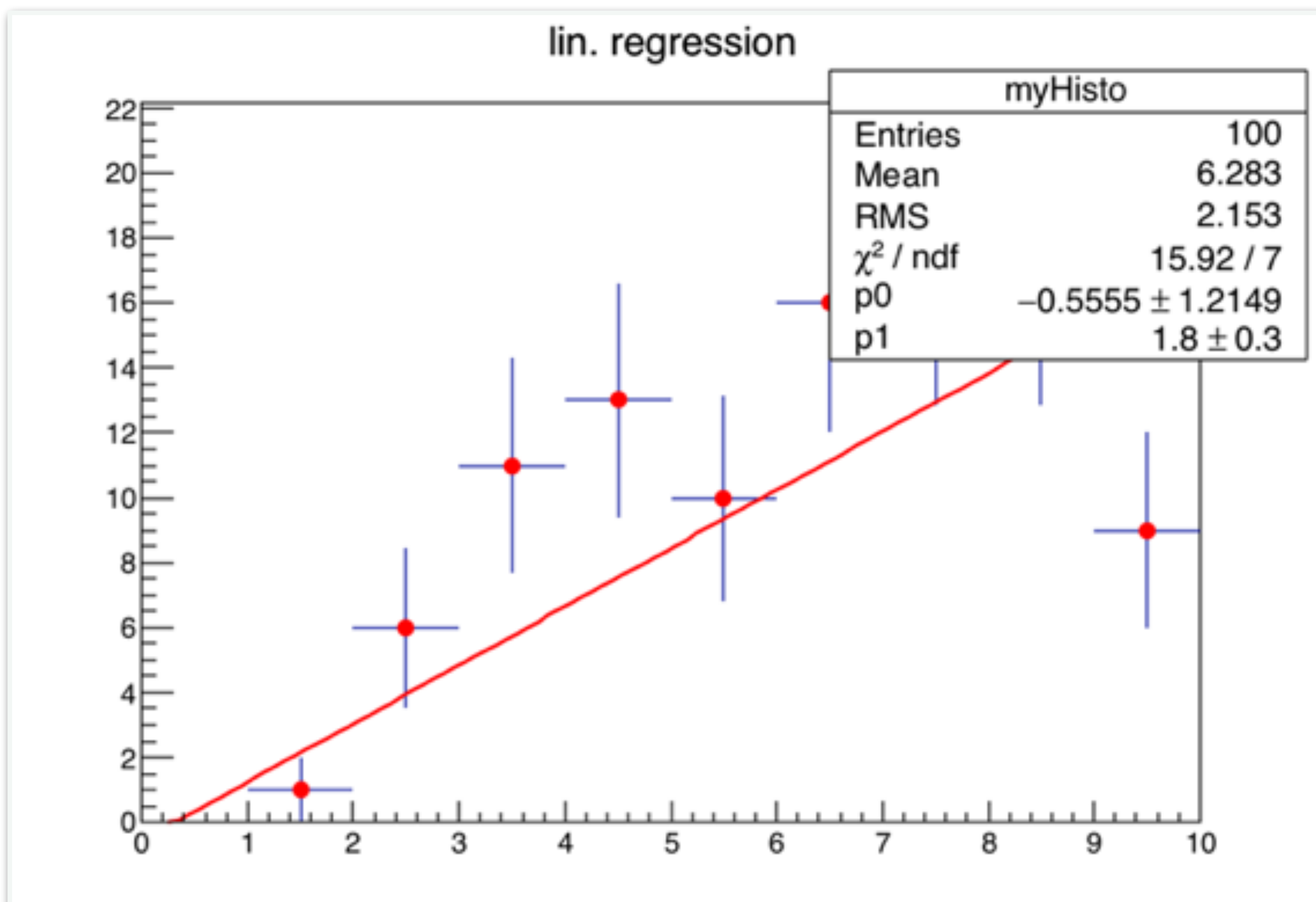
```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  gStyle->SetOptFit();
  TH1F* myH1 = new TH1F("myHisto","lin. regression",10,0.,10.);
  TF1* myPol1 = new TF1("myPol1","2*x",0.,10.);
  myH1->FillRandom("myPol1",100);
  myH1->SetMarkerColor(2);
  myH1->SetMarkerStyle(20);
  myH1->Fit("pol1");
  myH1->Draw("E");
  TVirtualFitter *fitter = TVirtualFitter::GetFitter();
  TMatrixD *matrix = new TMatrixD(2,2,fitter-
>GetCovarianceMatrix());
  matrix->Print();
}
```

fit with polynomial of 1st order, e.g. lin. regression

linRegression.py

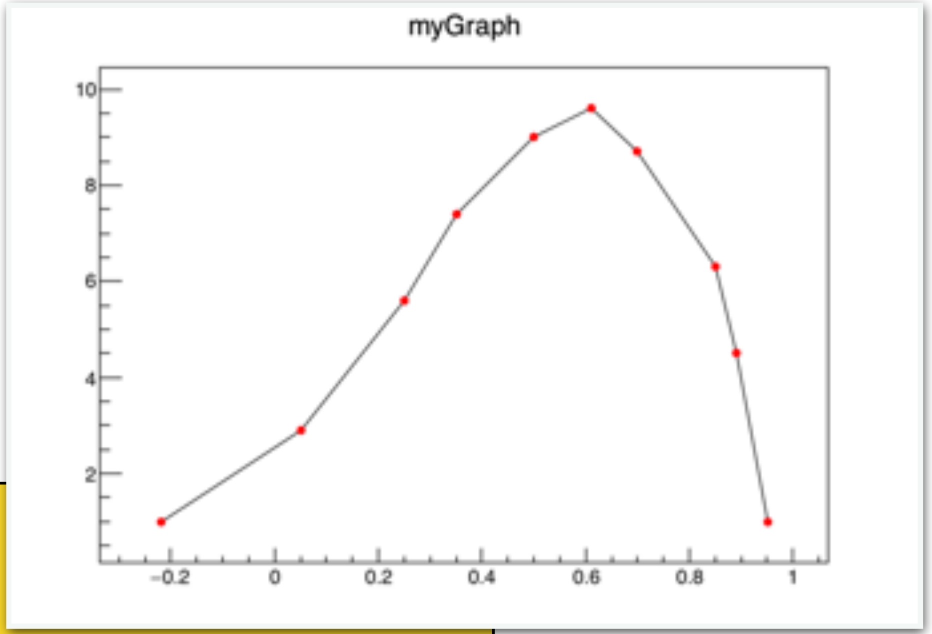
- Output:

```
****************************************
Minimizer is Linear
Chi2                      =     15.922
NDf                       =          7
p0                        =    -0.5555   +/-    1.21491
p1                        =    1.79955   +/-    0.295277
Info in <TCanvas::Print>: file linRegression.png has been created

2x2 matrix is as follows

     |      0      |      1      |
-------------------------------
   0 |      1.476      -0.2796
   1 |    -0.2796      0.08719
```



Graph.C

- `TGraphs` are useful to create draws from set of coordinates
- `Draw()` options:
  - “`a`“ draw axis
  - “`p`“ draw marker
  - “`l`“ connect with line




```
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  gStyle->SetOptFit();

  const int n = 10;
  float x[n]  = {-0.22, 0.05, 0.25, 0.35, 0.5,
0.61,0.7,0.85,0.89,0.95};
  float y[n]  = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};

  TGraph *myGraph = new TGraph(n,x,y);

  myGraph->SetTitle("myGraph");
  myGraph->SetMarkerColor(2);
  myGraph->SetMarkerStyle(20);
  myGraph->SetMarkerSize(0.7);
  myGraph->Draw("apl");
}
```

data rows x, y

Define graph with n data points

Set marker color, size and style

Draw axis, marker dots and connect with a line

grapherrors.C

```
{
  gROOT->Reset();  gROOT->SetStyle("Plain");  gStyle->SetOptFit();

  const int n = 10;
  float x[n]  = {-0.22, 0.05, 0.25, 0.35, 0.5,
0.61,0.7,0.85,0.89,0.95};
  float y[n]  = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
  float ex[n] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
  float ey[n] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};

 TGraphErrors *myGraph = new TGraphErrors(n,x,y,ex,ey);

  myGraph->Draw("ap");
  TF1 *func = new TF1("func","[0]+[1]*x+[2]*pow(x,2)+[3]*pow(x,
3)",-0.3,1.);
  func->SetParameter(0,1);
  func->SetParameter(1,1);
  func->SetParameter(2,1);
  func->SetParameter(3,-10);
  func->SetLineColor(4);

  myGraph->Fit(func,"R");
  myGraph->SetMaximum(13);
  myGraph->GetHistogram()->SetXTitle("X-Axis");
  myGraph->GetHistogram()->SetYTitle("Y-Axis");
}
```

x, y data

x, y uncertainties

Graph with n data points & error

define polynomial of 3rd order

set initial values

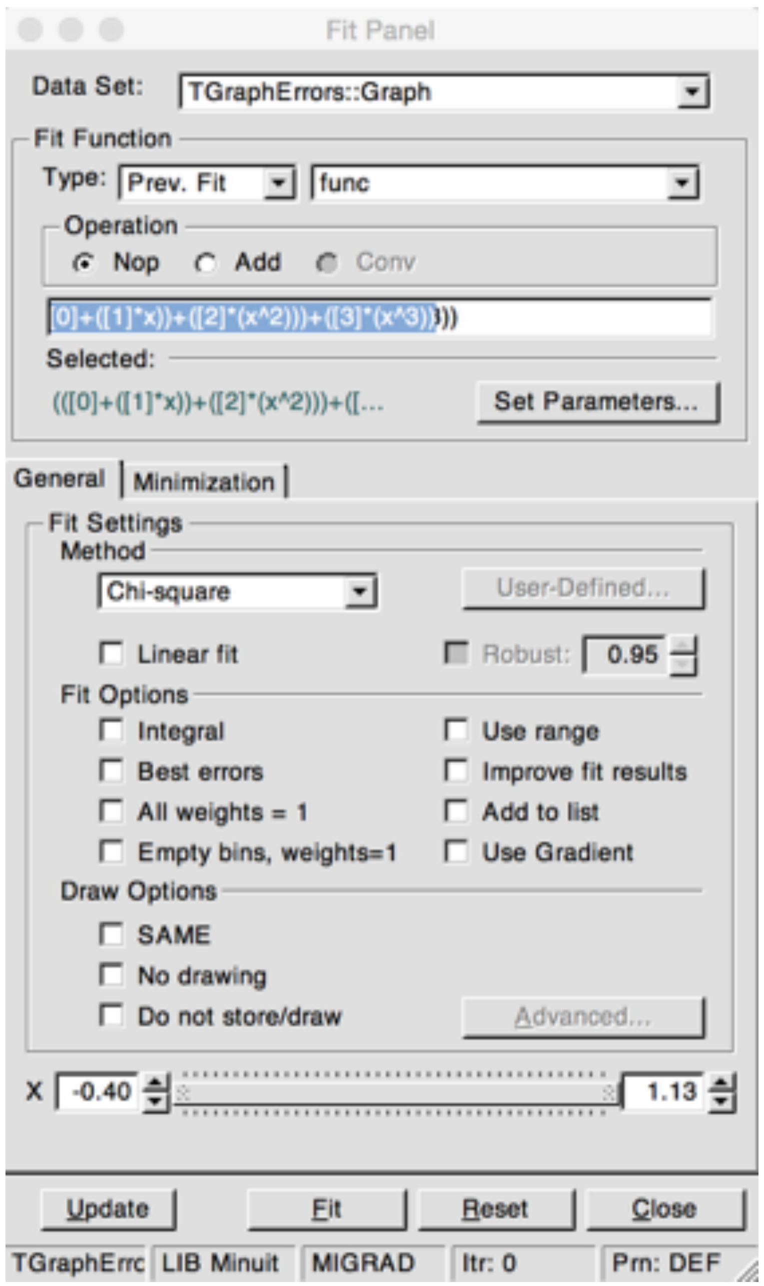
fit, 'R'ange: only within [-0.3,1.0]

maximum of y-axis

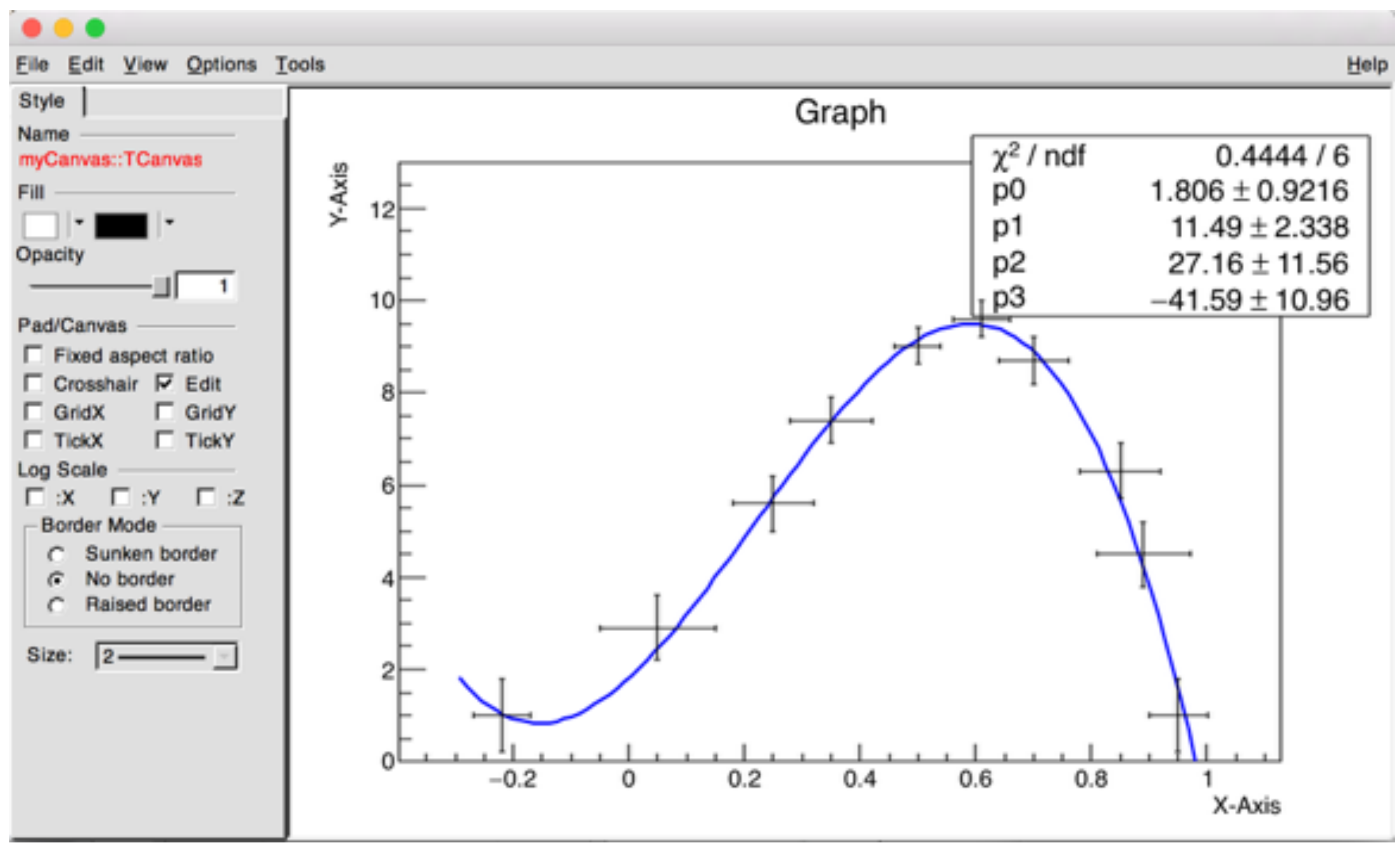
axis labels

grapherrors.C

- Execute with '`rootgrapherrors.C`'
- Many operations be modified interactively by selecting and 'right click' of the object and choose some action… e.g. change fit and draw options.



- Useful trick: Saving then your canvas manually as '`.C`' file allows to extract the changed options manually and use them in your program.



View → Editor allows to change e.g. Canvas

'Tools → Fit Panel' allows to rerun fit

University of BRISTOL

Drawing Options

- ROOT offers the possibility to define 2 dimensional histograms. A 2d histogram, function resp. is defined as:

  - `TH2F* _myH2 = new TH2F(“my2Dhisto”,”2d histo”,100,0.,1.,100,.0.,1.);`
  - `TF2* _myFunc2 = new TF2(“my2Dfunc”,”2d func”,”x^2+y^2”,-1.,1.,-1.,1.);`

- The `Draw()` method for 2d histos and functions offers many additional options, either in a planar or 3D view:

  - 2D: `“BOX”, “COL”, “COLZ”, “TEXT”, “CONT0”, “CONT1”, “CONT2”, “CONT3”, “CONT4”`

  - 3D: `“LEGO”, “LEGO1”, “LEGO2”, “SURF”, “SURF1”, “SURF2”, “SURF3”, “SURF4”`

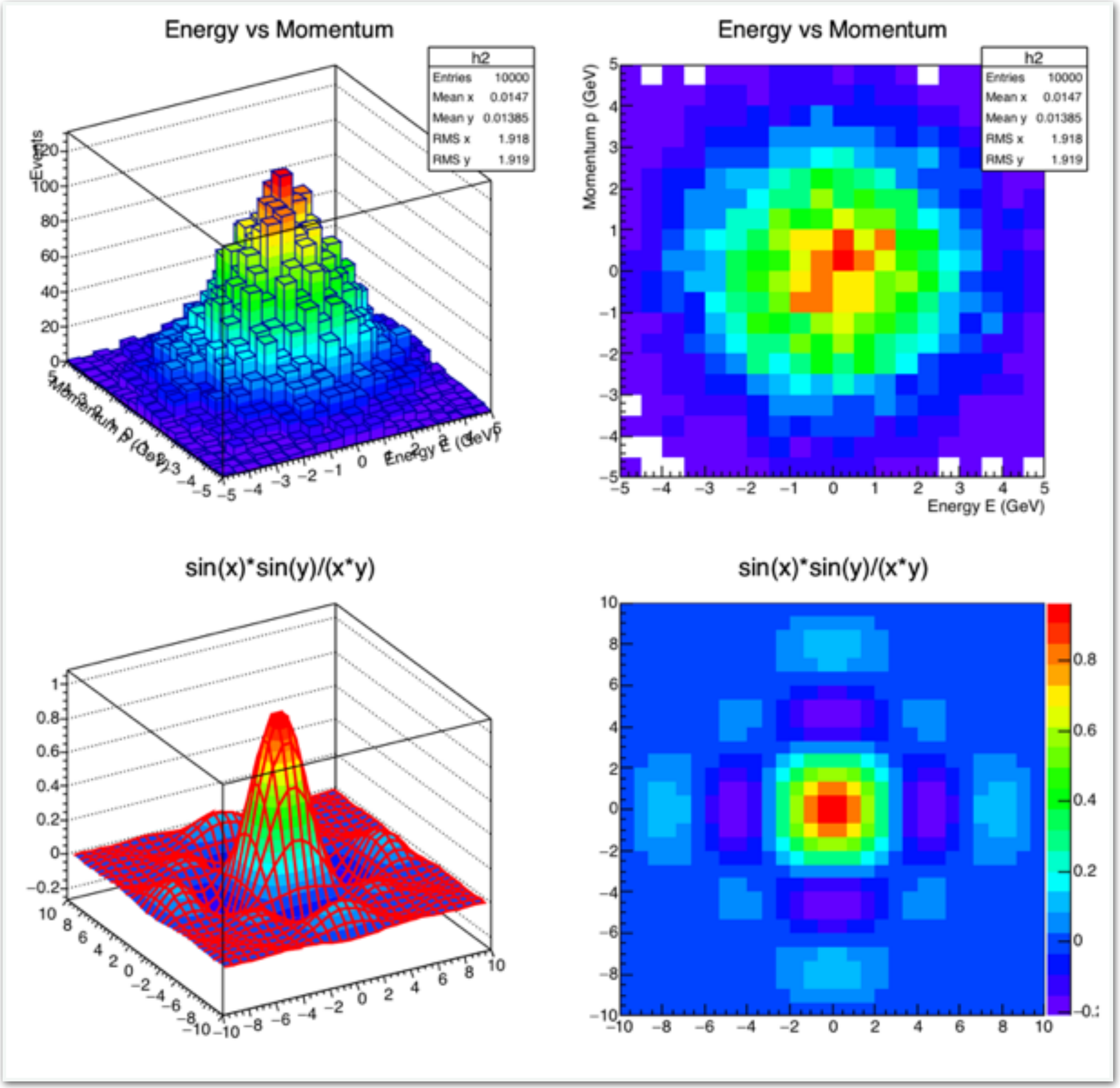
2d_histos.C

- Let's put things together:

```
{
  gROOT->Reset();
  gStyle->SetPalette(1);
  TCanvas *c1 = new Tcanvas("c1","canvas for many histos",800,800);
  c1->Divide(2,2);
  TH2F *h2 = new TH2F("h2","Energy vs Momentum",40,-5.,5.,40,-5.,5.);
  h2->FillRandom("gaus",6000);
  h2->GetXaxis()->SetTitle("Energy E (GeV)");
  h2->GetYaxis()->SetTitle("Momentum p (GeV)");
  h2->GetZaxis()->SetTitle("Events");
  TF2* f2=new TF2("func2","sin(x)*sin(y)/(x*y)",-10.,10.,-10.,10.);
  c1->cd(1);
  h2->Draw("LEGO2");
  c1->cd(2);
  h2->Draw("COL");
  c1->cd(3);
  f2->Draw("SURF1");
  c1->cd(3);
  f2->Draw("SURF1");
  c1->cd(4);
  f2->Draw("COLZ");
}
```

2d_histos.C

- Output:



Summary

- This tutorial enables you to start doing analysis
- Keep in mind I am not a python expert
- Example code: (including c++ code in subdir)
  - https://dl.dropboxusercontent.com/u/1882892/example_code_py.tar.gz
- Python allows to start programming quickly, but it is slow for resource intensive tasks
- Let me know if there are problems