ROOT Tutorial

Tulika Bose
Brown University
NEPPSR 2007

ROOT

- What is ROOT ?
  - ROOT is an object-oriented C++ analysis package
  - User-compiled code can be called to produce 1-d, 2-d, and 3-d graphics and histograms…
- ROOT uses a language called CINT (C/C++ Interpreter) which contains several extensions to C++
  - CINT allows the use of a dot "." while an arrow "->" is used in C++
  - CINT commands always start with "."

Useful Links

ROOT web-page: http://root.cern.ch/

ROOT can be downloaded from
http://root.cern.ch/twiki/bin/view/ROOT/Download

ROOT Tutorials:

- http://root.cern.ch/root/Tutorials.html
- Babar ROOT Tutorial I
- Babar ROOT Tutorial II
- Nevis tutorial:
  http://www.nevis.columbia.edu/~seligman/root-class/

All files used in this tutorial can be found at
http://home.fnal.gov/~tulika/NEPPSR/

ROOT Basics

Start ROOT:

- type “root”
- (to skip the introductory welcome type “root -l”)

```
[tulika@cmslpc04 ~]$ root
  *******************************************
  *                                         *
  *        W E L C O M E  to  R O O T       *
  *                                         *
  *   Version   5.14/00f       29 May 2007  *
  *                                         *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch            *
  *                                         *
  *******************************************

FreeType Engine v2.1.9 used to render TrueType fonts.
Compiled on 29 June 2007 for linux with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.16, November 24, 2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

For help: type “.?”, “.h”

Quit ROOT: type “.q”

ROOT analysis

A typical ROOT analysis could be:

- Take variables in an n-tuple, make histograms, calculate quantities, and perform fits…
  - How does one make a histogram ?
  - What is an n-tuple ?
  - How are calculations done ?
  - How does one fit ?

Histograms

Making your first histogram:

- Histograms can be 1-d, 2-d and 3-d
- Declare a histogram to be filled with floating point numbers:

TH1F *histName = new TH1F("histName", "histTitle", num_bins,x_low,x_high)

```
TH1F *my_hist = new TH1F("my_hist", "My First Histogram", 100, 2, 200)
```

Note: Bin 0 → underflow (i.e. entries < x_low)

Bin (num_bins+1) → overflow (i.e. entries > x_high)

2-d and 3-d histograms can be booked similarly…

```
TH2F *myhist = new TH2F("myhist", "My Hist", 100, 2, 200, 200, 0,500)
```

Drawing Histograms

- To draw:
  - my_hist->Draw();
- To fill a histogram:
  - my_hist->Fill(50);
  - my_hist->Fill(100, 3); // the number 100 has weight=3
- Update the histogram:
  - my_hist->Draw();

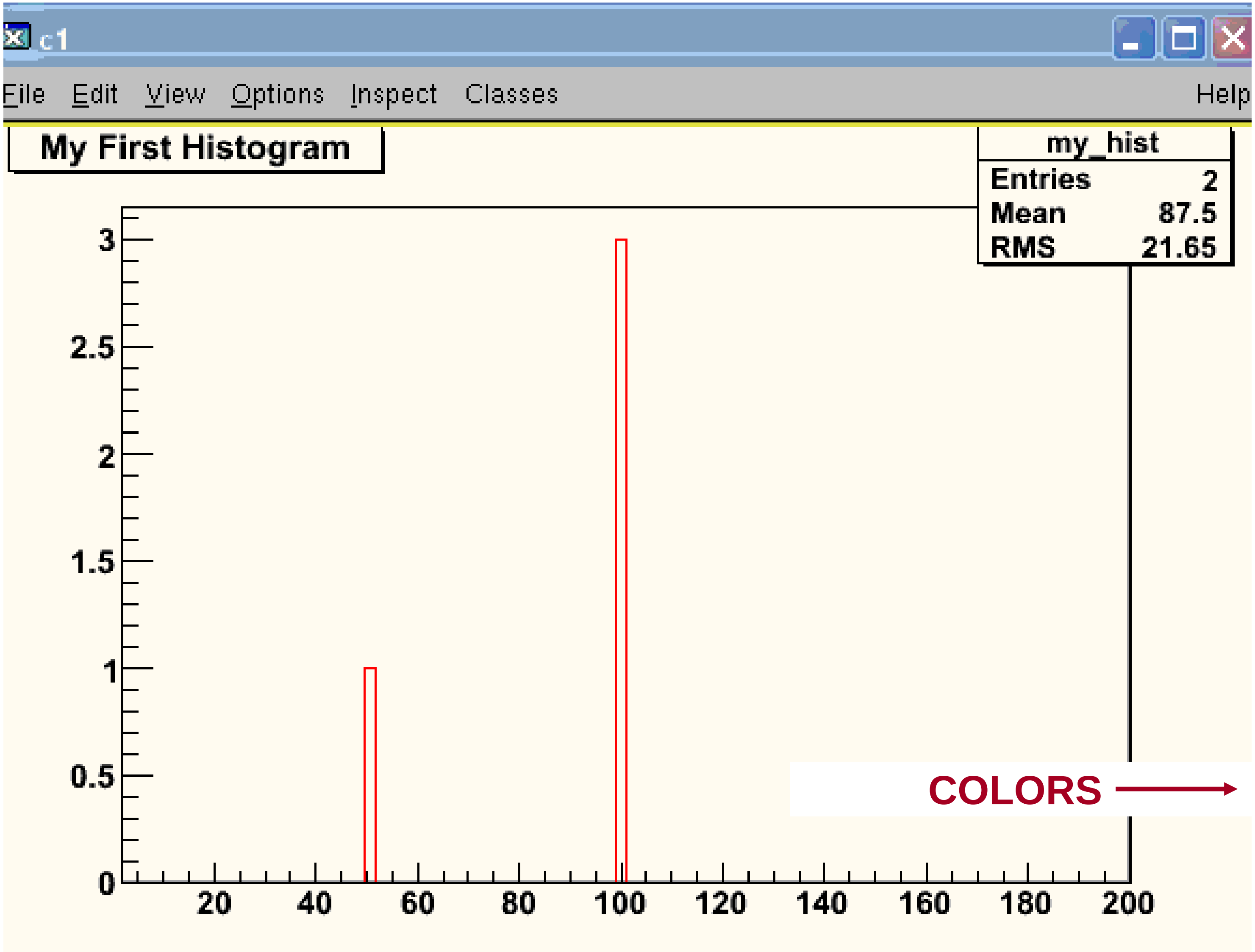
Histogram attributes:

- Change line color:
  - my_hist->SetLineColor(2); //red
  - or my_hist->SetLineColor(kRed);
  - my_hist->Draw();

Look at the official documentation for the different drawing options

```
[tulika@cmslpc04 ~]$ root -l
root [0]
root [0] TH1F *my_hist = new TH1F("my_hist", "My First Histogram", 100, 2, 200);
root [1] my_hist->Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [2] my_hist->Fill(50);
root [3] my_hist->Fill(100,3);
root [4] my_hist->Draw();
root [5] my_hist->SetLineColor(2);
root [6] my_hist->Draw();
root [7]
```

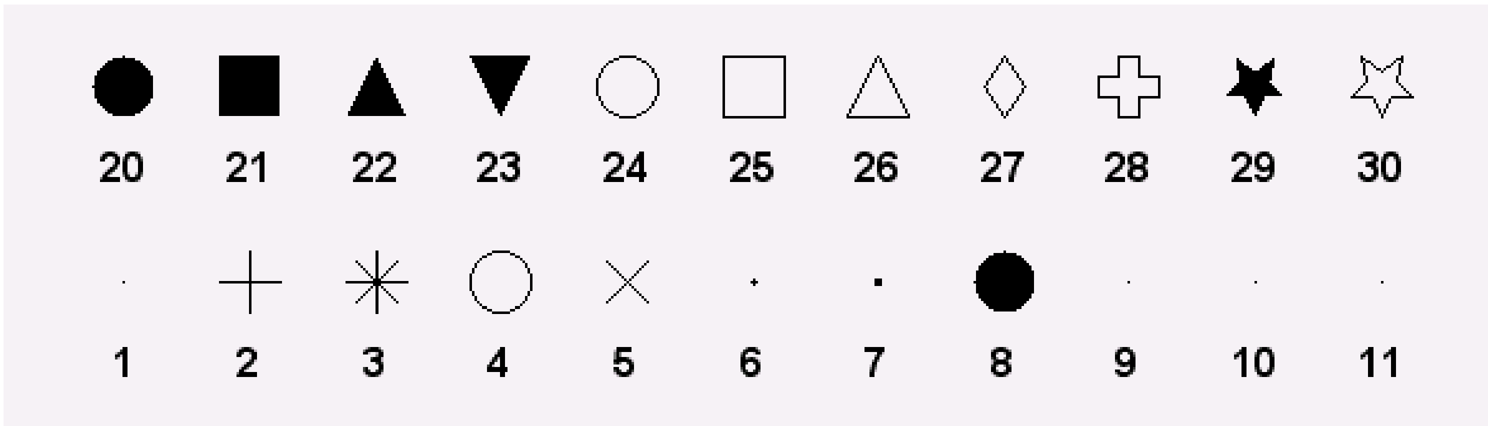
ROOT Session



1	Black
2	Red
3	Light green
4	Blue
5	Yellow
6	Magenta
7	Cyan
8	Green

Drawing Options

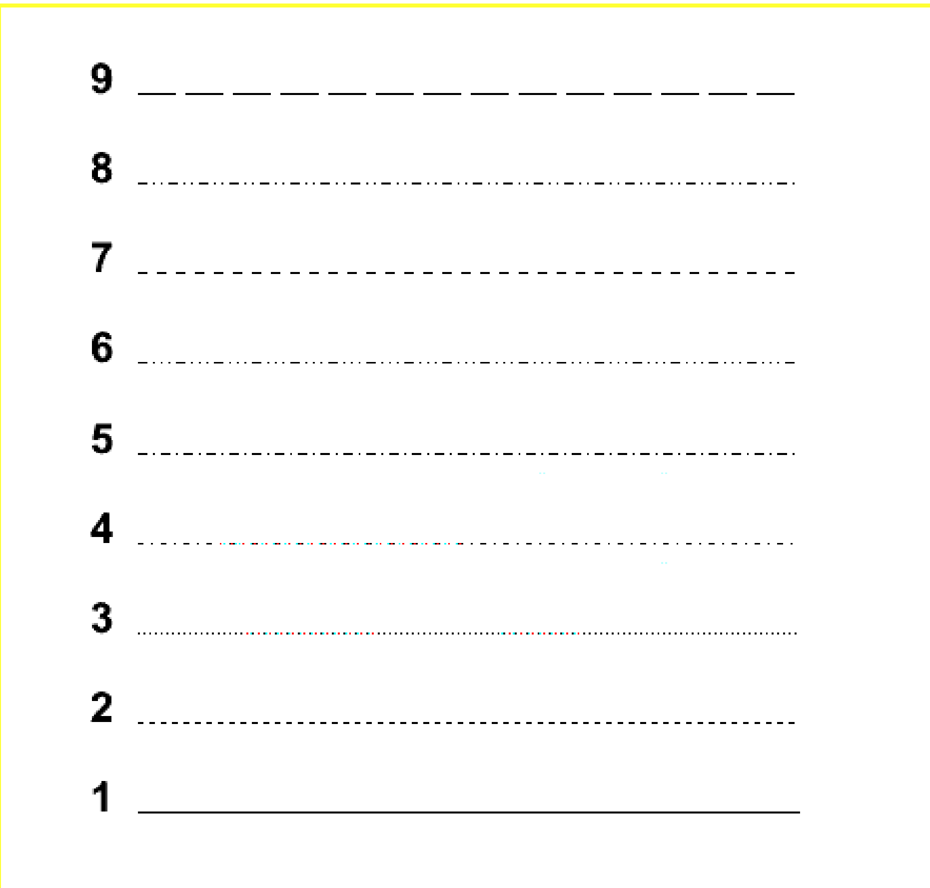
Marker Styles



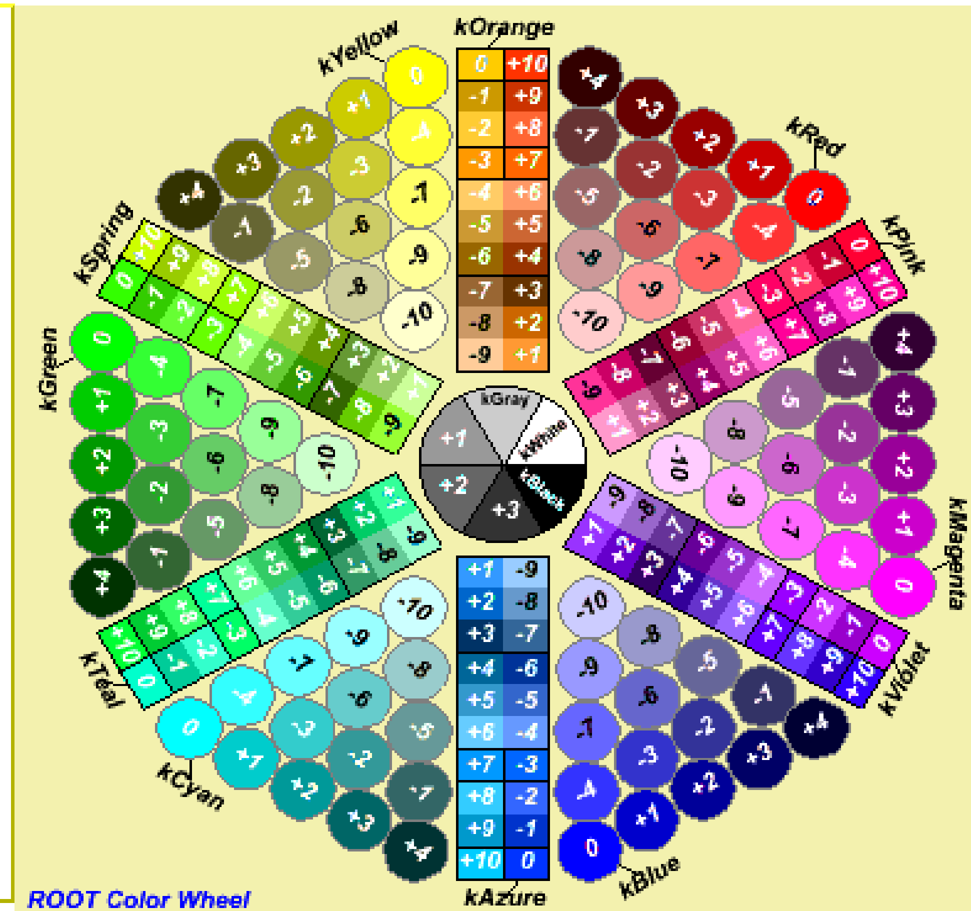
Colors

Drawing Options

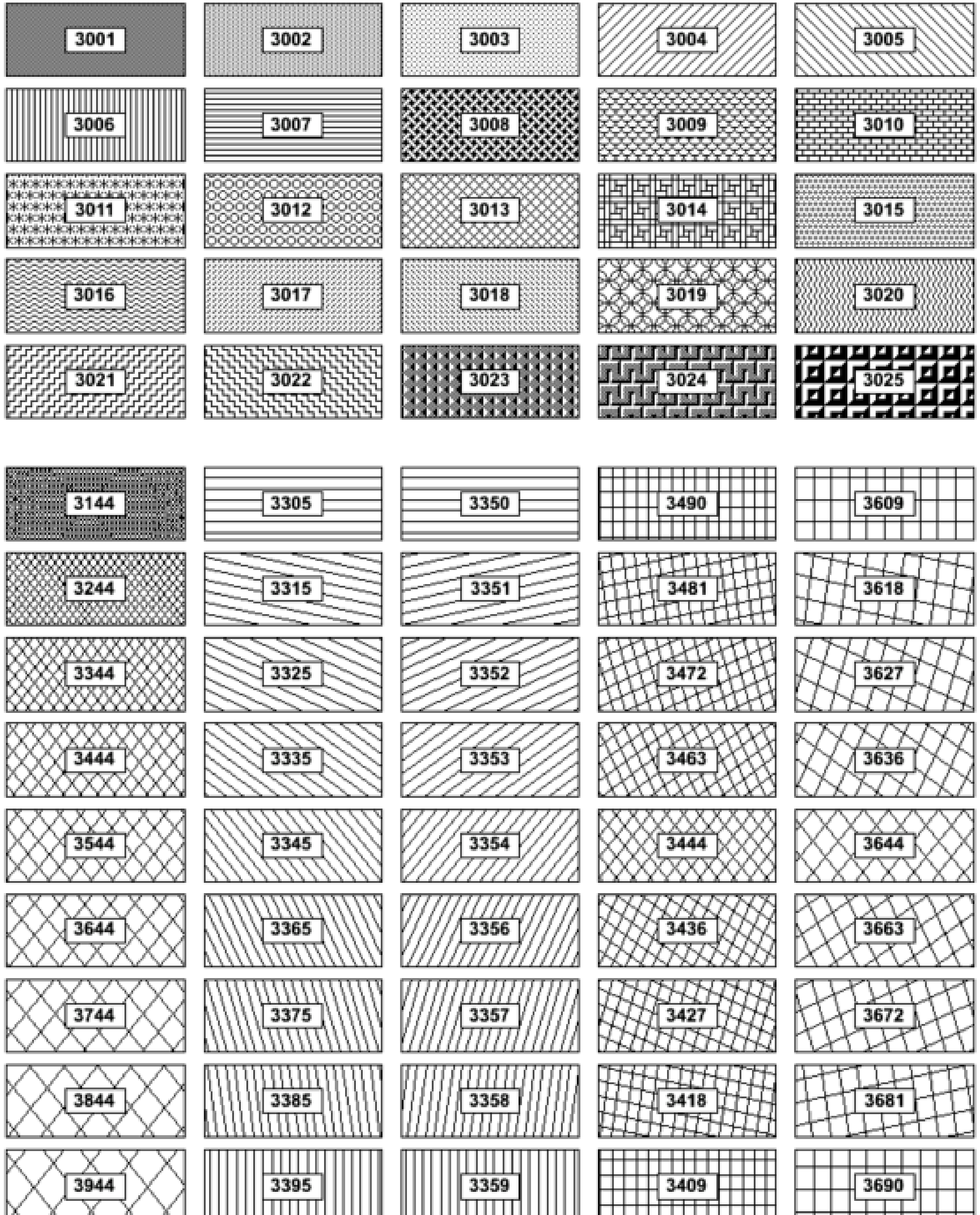
LINE STYLES



COLOR WHEEL



FILL STYLES


3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3144
3305
3350
3490
3609
3244
3315
3351
3481
3618
3344
3325
3352
3472
3627
3444
3335
3353
3463
3636
3544
3345
3354
3444
3644
3644
3365
3356
3436
3663
3744
3375
3357
3427
3672
3844
3385
3358
3418
3681
3944
3395
3359
3409
3690

LaTeX in ROOT

≤	#leq	/	#/	∞	#infty	ƒ	#voidb
♣	#club	♦	#diamond	♥	#heart	♠	#spade
↔	#leftrightarrow	←	#leftarrow	↑	#uparrow	→	#rightarrow
↓	#downarrow	°	#circ	±	#pm	″	#doublequote
≥	#geq	×	#times	∝	#propto	∂	#partial
•	#bullet	÷	#divide	≠	#neq	≡	#equiv
≈	#approx	…	#3dots	∣	#cbar	—	#topbar
↵	#downleftarrow	ℵ	#aleph	ℑ	#Jgothic	ℜ	#Rgothic
℘	#voidn	⊗	#otimes	⊕	#oplus	∅	#oslash
∩	#cap	∪	#cup	⊃	#supset	⊇	#supseteq
⊄	#notsubset	⊂	#subset	⊆	#subseteq	∈	#in
∉	#notin	∠	#angle	∇	#nabla	®	#oright
©	#ocopyright	™	#trademark	∏	#prod	√	#surd
·	#upoint	¬	#corner	∧	#wedge	∨	#vee
⇔	#Leftrightarrow	⇐	#Leftarrow	⇑	#Uparrow	⇒	#Rightarrow
⇓	#Downarrow	♦	#diamond	〈	#LT	®	#void1
©	#copyright	™	#void3	∑	#sum	⎛	#arctop
∣	#lbar	⎝	#arcbottom	—	#topbar	∣	#void8
⌊	#bottombar	⎜	#arcbar	⎨	#ltbar	Å	#AA
å	#aa		#void06	〉	#GT	∫	#int

The "Stats Box"

The Statistics Box :

- Setup with:
  - gStyle->SetOptStat(mode)
  - (ksiourmen)

- Default is (000001111):

- To show overflows and underflows:
  - gStyle->SetOptStat(111111);

- To remove entirely:
  - gStyle->SetOptStat(0);

k=1	kurtosis
K=2	Kurtosis+error
s=1	Skewness
S=2	Skewness+error
i=1	Integral
o=1	Overflow
u=1	Underflow
r=1	RMS
R=2	RMS+error
m=1	Mean
M=2	Mean+error
e=1	Entries
n=1	Name

More on histograms

- Draw a second histogram on top of the first:
  - First book the histogram
  - Use another color to differentiate
  - Fill the histogram
  - Draw the histogram
  - my_hist_2->Draw("same")

- Errors:
  - my_hist_2->Draw("esame")
- Default : errors = sqrt(entries)
- To get error as sqrt(sum of weights), enter
  - my_hist_2->Sumw2() before filling the histogram

Exercises

Exercises:

- Add axis labels
- Add a legend
- Add a text box

Save Histograms:

```
c1->SaveAs("myhisto.eps");
c1->SaveAs("myhisto.ps");
c1->SaveAs("myhisto.gif");
```

Also can save source code for histogram:

```
c1->SaveAs("myhisto.C");
```

Recreate histogram in a brand new ROOT session:

```
.x myhisto.C
```

Functions and Histograms

- Define a function:

```
TF1 *myfunc = new TF1("myfunc", "gaus", 0, 3);
myfunc->SetParameters(10.,1.5,0.5);
myfunc->Draw();
```

- Generate histograms from functions:

```
(myfunc->GetHistogram())->Draw();
```

- Generate histograms with random numbers from a function:

```
TH1F *myhist = new TH1F("myhist", "Histo from gaus", 50, 0, 3);
myhist->FillRandom("myfunc", 10000);
myhist->Draw();
```

- Write histogram to a root file:

```
TFile *myfile = new TFile("fillrandom.root", "RECREATE");
myhist->Write();
myfile->Close();
```

Fitting Histograms

Let us try to fit the histogram created by the previous step:

Interactively:

- Open root file containing histogram:

  root -l fillrandom.root
- Draw histogram:

  myhist->Draw()
- Right click on the histogram and select "Fit Panel"
- Check to ensure:
- "gaus" is selected in the Function->Predefined pop-up menu
- "Chi-square" is selected in the Fit Settings->Method menu
- Click on "Fit" at the bottom

[Display fit parameters: Select Options-> Fit Parameters]

Fitting contd:

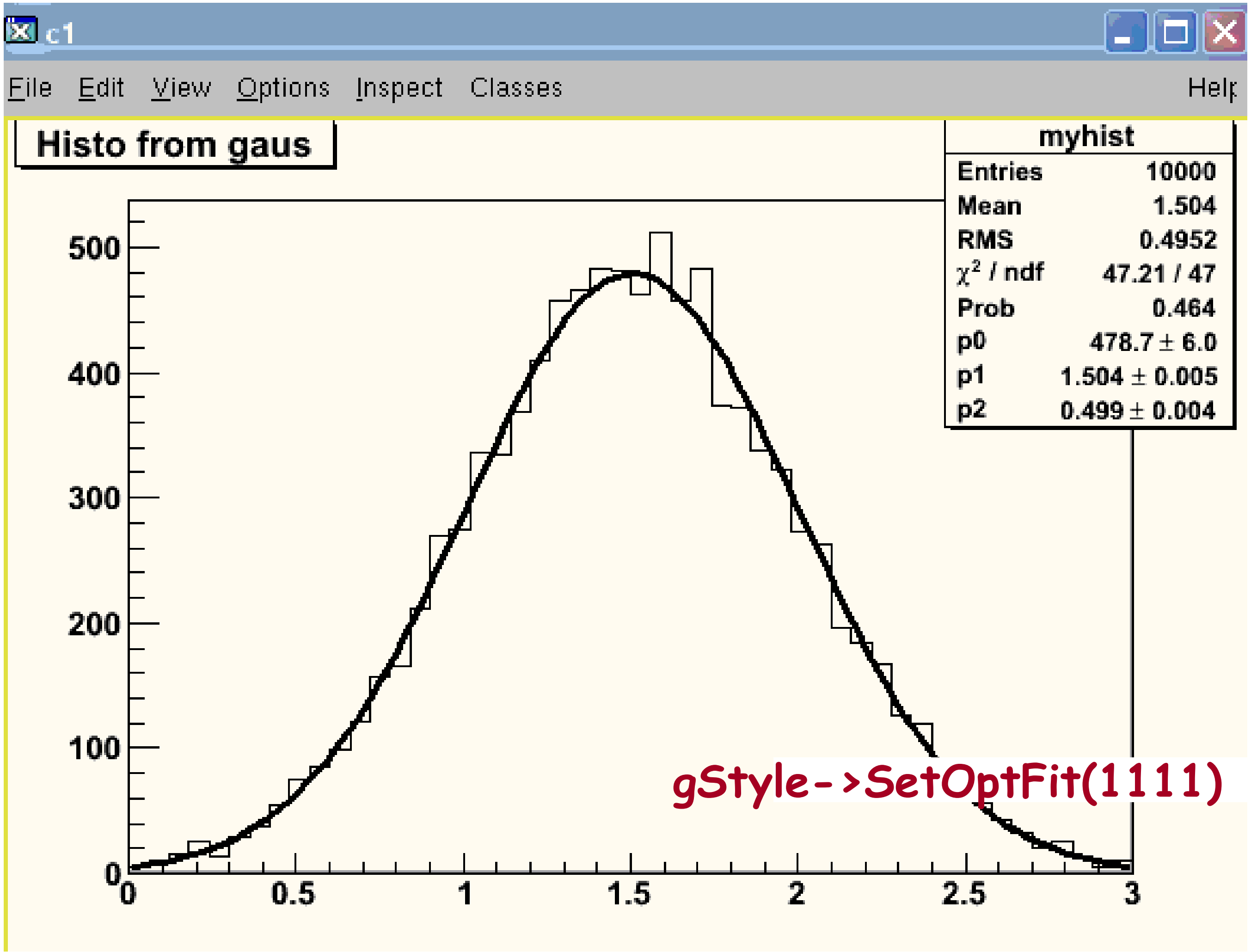
Using user defined functions:

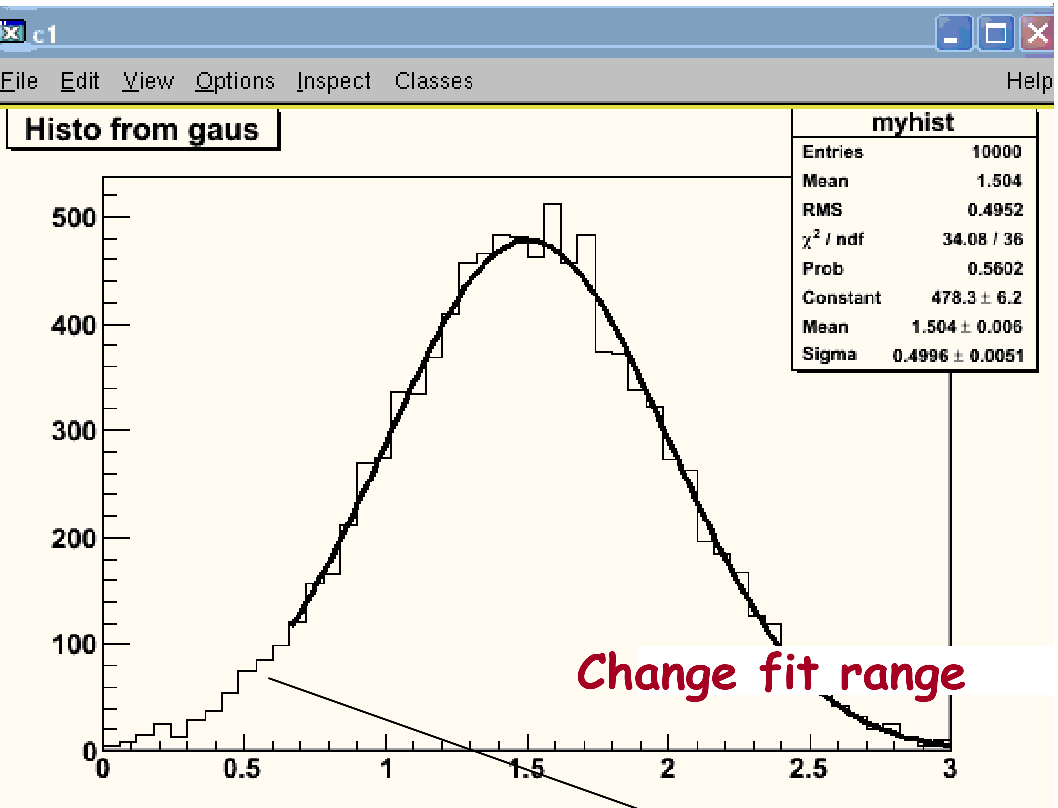
- Create a file called user_func.C with the following contents:

```
double user_func(double *x, double *par) {
double arg = 0;
if (par[2]) arg = (x[0] - par[1])/par[2];
return par[0]*TMath::Exp(-0.5*arg*arg);
}
```

```
[tulika@cmslpc04 NEPPSR]$ root -l fillrandom.root
root [0]
Attaching file fillrandom.root as _file0...
root [1] .L user_func.C
root [2] TF1 *f1 =  new TF1("f1", user_func, 0, 3, 3);
root [3] myhist->Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [4] f1->SetParameters(10,myhist->GetMean(), myhist->GetRMS());
root [5] myhist->Fit("f1");
```

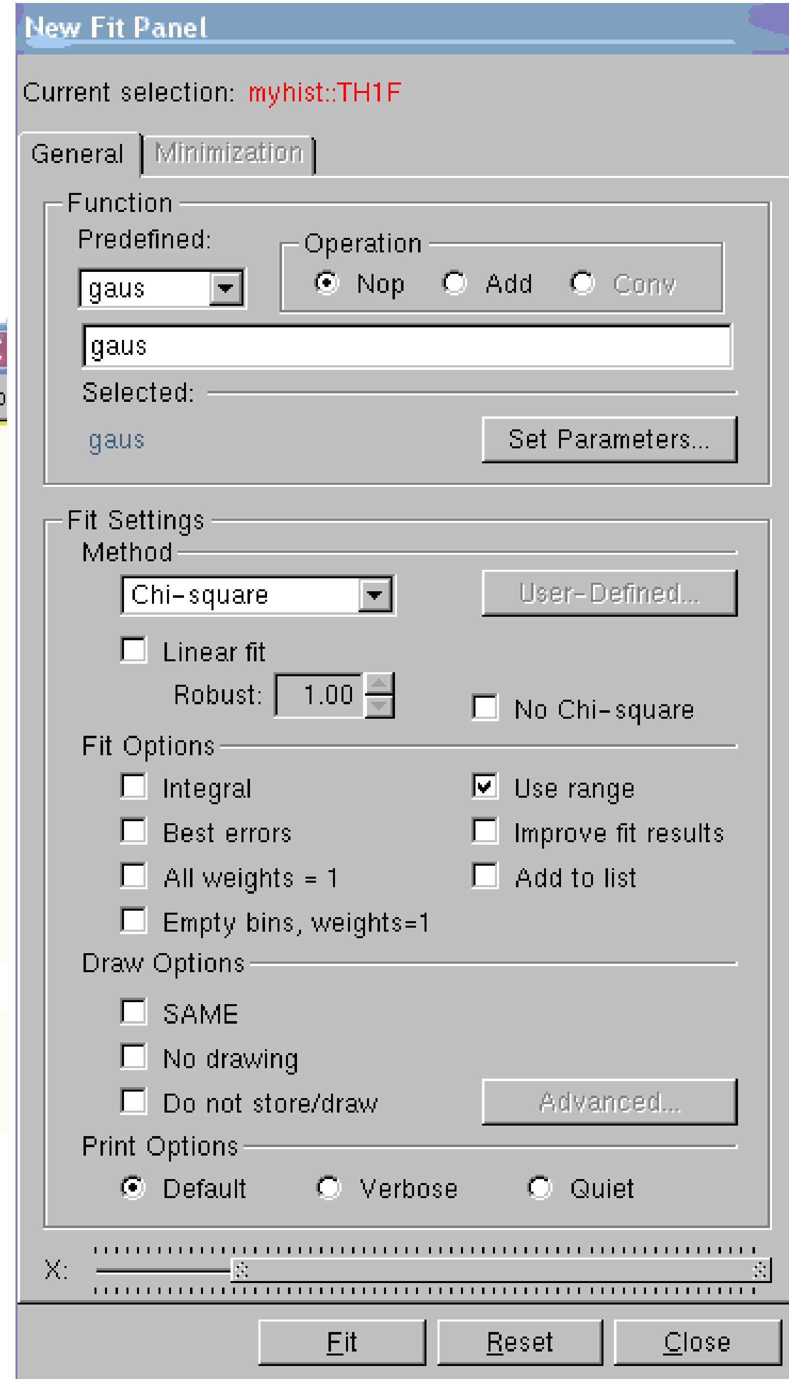
Fitting contd:





Change fit range

Move the slider



N-tuples and trees

- An n-tuple is an ordered list of numbers

Row	event	ebeam	px	py	pz	zv	chi2
0	0	150.14	14.33	-4.02	143.54	22.26	0.94
1	1	149.79	0.05	-1.37	148.60	0.61	1.02
2	2	150.16	4.01	3.89	145.69	16.57	0.89
3	3	150.14	1.46	4.66	146.71	11.47	1.02
4	4	149.94	-10.34	11.07	148.33	0.37	0.85
5	5	150.18	17.08	-12.14	143.10	22.09	0.90
6	6	150.02	5.19	7.79	148.59	2.28	1.06
7	7	150.05	7.55	-7.43	144.45	21.40	0.97
8	8	150.07	0.23	-0.02	147.78	6.96	0.93
9	9	149.96	1.21	7.27	146.99	7.17	1.02
10	10	149.92	5.35	3.98	140.70	38.81	1.08
11	11	149.88	-4.63	-0.08	147.91	4.01	0.86
12	12	150.11	-1.96	11.46	147.41	6.76	1.08
13	13	150.02	-4.97	4.29	145.06	17.79	0.92
14	14	149.86	0.26	0.10	144.69	22.26	0.93

- A ROOT Tree can be an ordered list of any collections of C++ objects
  - It has branches which can hold many other objects, for eg. vectors
- An n-tuple in ROOT is just a simple Tree, where each branch contains floating point data

Getting started with trees

Download the file at
http://home.fnal.gov/~tulika/NEPPSR/histo-174.root

Use the following commands to open the file and list contents
The root-file consists of a tree called "ttbar".

To display the names of the variables stored in the tree
use the "Print()" command

```
[tulika@cmslpc04 NEPPSR]$ root -l histo-174.root
root [0]
Attaching file histo-174.root as _file0...
root [1] .ls
TFile**         histo-174.root
 TFile*         histo-174.root
  KEY: TTree    ttbar;1
root [2] ttbar->Print();
```

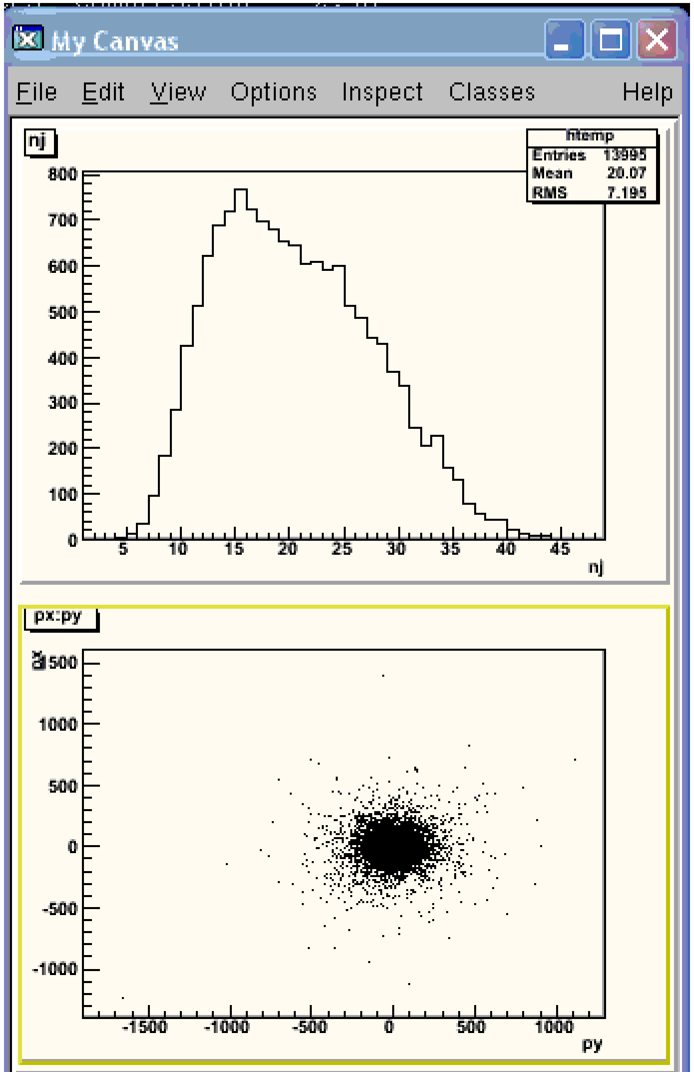
Plotting tree variables

```
root [3] TCanvas *c2= new TCanvas ("c2", "My Canvas",400, 600)
```

```
root [4] c2->Divide(1,2);
root [5] c2->cd(1);
root [6] ttbar->Draw("nj")
root [7] c2->cd(2);
root [8] ttbar->Draw("px:py")
```

To apply cuts:

```
root[9] TCanvas c3;
root [10] ttbar->Draw("nj", "nj>15");
```



Analyzing trees

```
[tulika@cmslpc04 NEPPSR]$ root -l histo-174.root
root [0]
Attaching file histo-174.root as _file0...
root [1] .ls
TFile**         histo-174.root
 TFile*         histo-174.root
  KEY: TTree    ttbar;1
root [2] ttbar->MakeClass("Analyze")
Info in <TTreePlayer::MakeClass>: Files: Analyze.h and Analyze.C
generated from TTree: ttbar
(Int_t)0
```

Add your analysis code to Analyze.C and then execute in ROOT:

`root [0] .L Analyze.C`	"L"oad the file
`root [1] Analyze t`	Create an object of type t
`root [2] t.Loop()`	Execute Loop command of object t

“Analyze” trees

```
#define Analyze_cxx
#include "Analyze.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void Analyze::Loop()
{
//   In a Root session, you can do:
//      Root > .L Analyze.C
//      Root > Analyze t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
//

//     This is the loop skeleton
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(i);  // read all branches
//by  b_branchname->GetEntry(i); //read only this branch
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntries();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
   }
}
```

Setup code goes here

Loop code goes here

Exercises

Look at the example code in
http://home.fnal.gov/~tulika/NEPPSR/AnalyzeExample.C

1. Modify it to plot "px", "py", and "pz" for all jets in the event

2. Calculate η of the jets

3. Calculate the invariant mass of all possible di-jet and three-jet combinations

4. Require pt > 20 GeV and abs(η) < 3.0 and repeat Step 3.

Search for new particles while learning ROOT :
http://www-clued0.fnal.gov/~tulika/brown/root-proj.htm