ROOT

An Object-Oriented
Data Analysis Framework

Basic Tutorial

Topics

What is ROOT?
Interactive ROOT session
  - command line vs. macros vs. user-compiled code
Opening files/ accessing information
Trees and histograms
Fitting
Other useful things...

Exercises

ROOT

What is it?

Very versatile software package for performing analysis on HEP data

- develop and apply cuts on data
- perform calculations & fits
- make plots
- save results in ROOT files

ROOT can be used in many ways:

Command line – good for quickly making plots, checking file contents, etc.

Unnamed macros – execute commands as if you typed them on the command line

list of commands enclosed by one set of { }.

execute from ROOT command line: ".x file.C"

Named macros – best for analysis, can be compiled and run outside of ROOT, or loaded and executed during interactive session

Interactive ROOT uses a C++ interpreter (CINT) which allows (but does not require) you to write *pseudo-C++*

Be careful! This will make your programming much more difficult later in life!

It's best if you try to use standard C++ syntax, instead of the CINT shortcuts.

ROOT CINT syntax allows the following sloppy things:

"." and "–>" are interchangeable

";" is optional at the end of single commands

Many commands may be accessed interactively (point and right-click in plots)

Interactive ROOT

```
> root
  *******************************************
  *                                         *
  *        W E L C O M E  to  R O O T       *
  *                                         *
  *   Version   5.12/00      10 July 2006   *
  *                                         *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch            *
  *                                         *
  *******************************************

FreeType Engine v2.1.9 used to render TrueType fonts.
Compiled on 11 July 2006 for macosx with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.13, June 8, 2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.

root [0] TFile* f1 = new TFile("histogram.root");
root [1] f1->ls();
TFile**         histogram.root  Histograms for ROOT class
 TFile*         histogram.root  Histograms for ROOT class
  KEY: TH1F     hist1;1 Function to be fit
  KEY: TH1F     hist2;1 Another function to be fit
root [2] .q
>
```

Load a file

List contents of file

Quit ROOT

This file contains 2 1-dimensional histograms, named “hist1” and “hist2”

Canvases

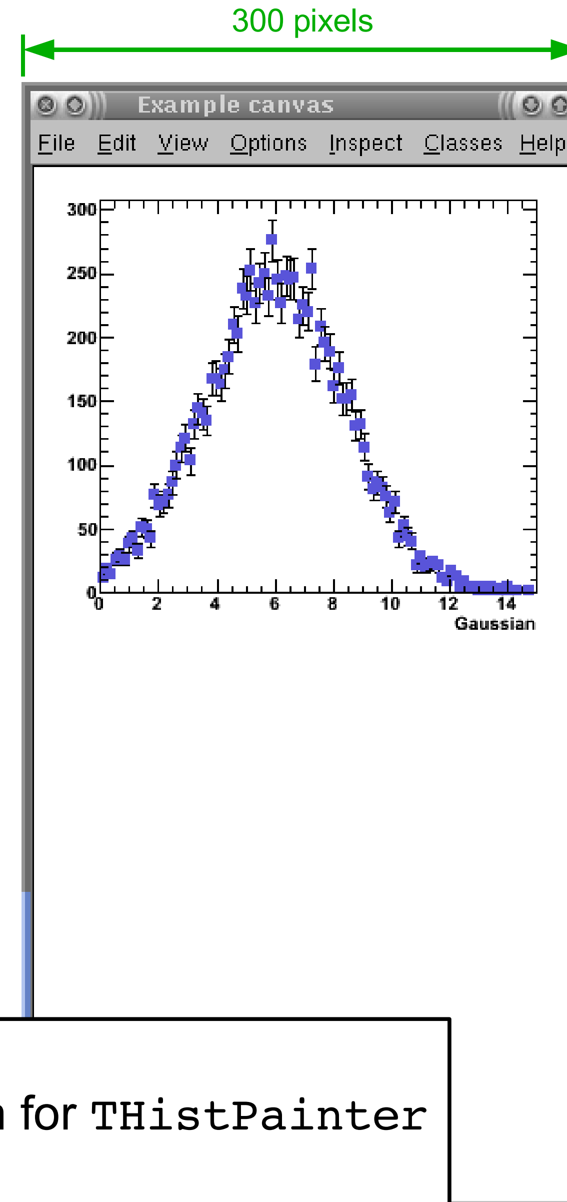
ROOT will automatically create a canvas for you if you try to draw something, but you can define your own (e.g., if you need a particular size, or you want equal-sized sub-divisions).

canvas name — canvas title — x,y dimensions

```
TCanvas* c1 = new TCanvas("c1","Example canvas",300,600);
```

```
c1->Divide(1,2);
```

← Divide the canvas into 2 areas (1 column, 2 rows)



Set various attributes of histogram “h1”:

```
h1->GetXaxis()->SetTitle(“Gaussian”);
h1->SetMarkerColor(4);
h1->SetMarkerStyle(21);
```

Change to the 1st canvas area and draw histogram “h1”:

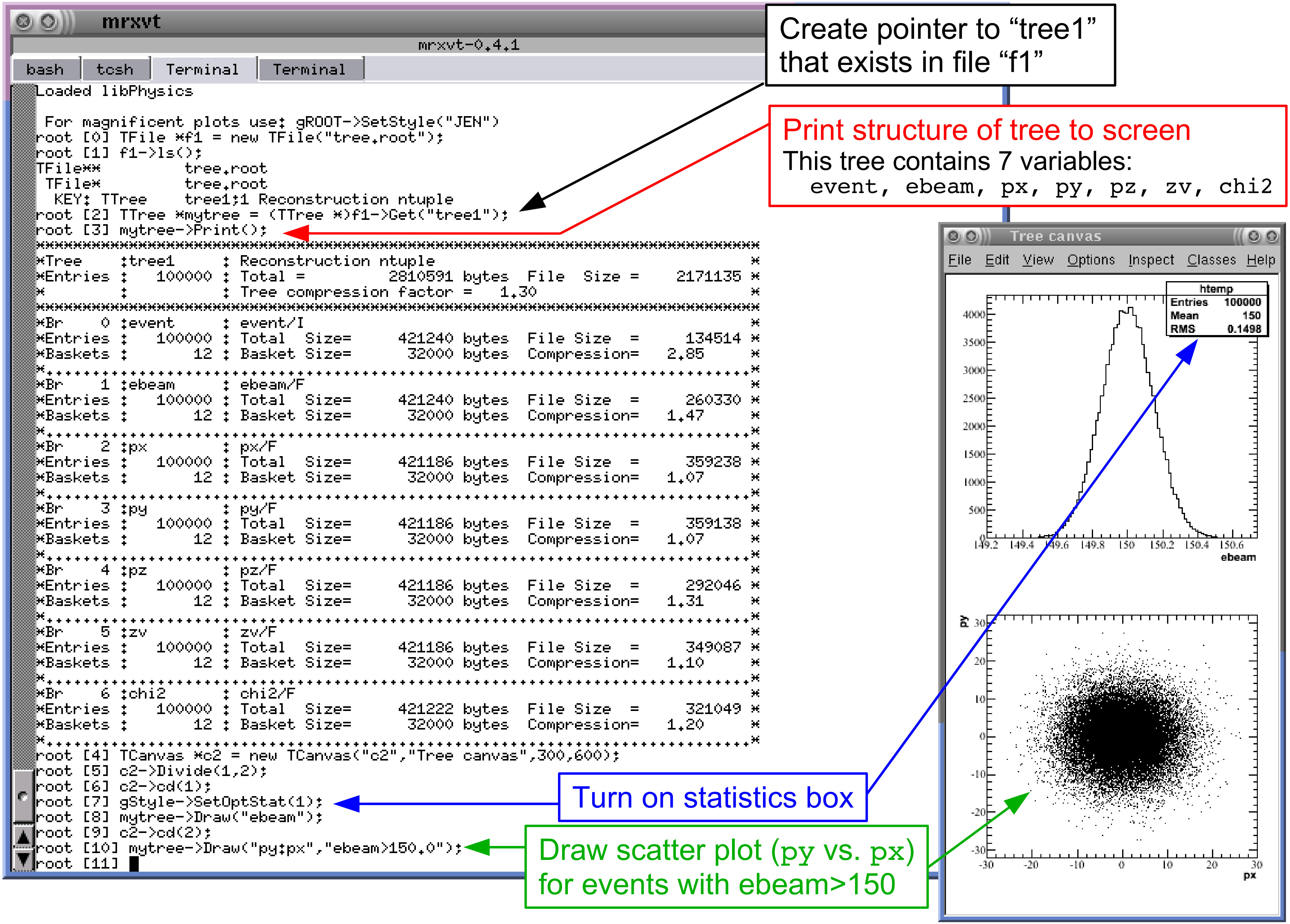
```
c1->cd(1);
h1->Draw(“ep”);
```

e = draw with error bars
p = draw with points (instead of line)

Histograms are drawn via the `THistPainter` class in ROOT.
You can find all drawing options by looking at the web documentation for `THistPainter`
http://root.cern.ch/root/html/THistPainter.html

Options for `TCanvas`: http://root.cern.ch/root/html/TCanvas.html

Trees


mrxvt
mrxvt-0.4.1
bash
tcsh
Terminal
Terminal
Loaded libPhysics
For magnificent plots use: gROOT->SetStyle("JEN")
root [0] TFile *f1 = new TFile("tree.root");
root [1] f1->ls();
TFile** tree.root
TFile* tree.root
KEY: TTree tree1;1 Reconstruction ntuple
root [2] TTree *mytree = (TTree *)f1->Get("tree1");
root [3] mytree->Print();
****
*Tree :tree1 : Reconstruction ntuple *
*Entries : 100000 : Total = 2810591 bytes File Size = 2171135 *
* : : Tree compression factor = 1.30 *
****
*Br 0 :event : event/I *
*Entries : 100000 : Total Size= 421240 bytes File Size = 134514 *
*Baskets : 12 : Basket Size= 32000 bytes Compression= 2.85 *
..
*Br 1 :ebeam : ebeam/F *
*Entries : 100000 : Total Size= 421240 bytes File Size = 260330 *
*Baskets : 12 : Basket Size= 32000 bytes Compression= 1.47 *
..
*Br 2 :px : px/F *
*Entries : 100000 : Total Size= 421186 bytes File Size = 359238 *
*Baskets : 12 : Basket Size= 32000 bytes Compression= 1.07 *
..
*Br 3 :py : py/F *
*Entries : 100000 : Total Size= 421186 bytes File Size = 359138 *
*Baskets : 12 : Basket Size= 32000 bytes Compression= 1.07 *
..
*Br 4 :pz : pz/F *
*Entries : 100000 : Total Size= 421186 bytes File Size = 292046 *
*Baskets : 12 : Basket Size= 32000 bytes Compression= 1.31 *
..
*Br 5 :zv : zv/F *
*Entries : 100000 : Total Size= 421186 bytes File Size = 349087 *
*Baskets : 12 : Basket Size= 32000 bytes Compression= 1.10 *
..
*Br 6 :chi2 : chi2/F *
*Entries : 100000 : Total Size= 421222 bytes File Size = 321049 *
*Baskets : 12 : Basket Size= 32000 bytes Compression= 1.20 *
..
root [4] TCanvas *c2 = new TCanvas("c2","Tree canvas",300,600);
root [5] c2->Divide(1,2);
root [6] c2->cd(1);
root [7] gStyle->SetOptStat(1);
root [8] mytree->Draw("ebeam");
root [9] c2->cd(2);
root [10] mytree->Draw("py:px","ebeam>150.0");
root [11]
Create pointer to “tree1” that exists in file “f1”
Print structure of tree to screen
This tree contains 7 variables:
event, ebeam, px, py, pz, zv, chi2
Turn on statistics box
Draw scatter plot (py vs. px) for events with ebeam>150
Tree canvas
File Edit View Options Inspect Classes Help
htemp
Entries 100000
Mean 150
RMS 0.1498
ebeam
py
px

Trees, cont'd.

To project something from a tree into a histogram, first define a histogram:

Name — Title — Number of bins — Low edge, High edge

```
TH1F* h_ebeam = new TH1F("h_ebeam", "Beam Energy", 100, 149.0, 151.0);
```

Then use the `TTree` class member “`Project`” to put the tree contents into the histogram:

```
mytree->Project("h_ebeam","ebeam","(px > 10.0) || (py <= 5.0)");
```

cuts: optional argument

To define complicated or often-used cuts:

```
TCut* cut1 = new TCut("x > 0");
TCut* cut2 = new TCut("y == sqrt(2+x**2)”);
TCut* cut3 = new TCut(*cut1 && *cut2);
```

Then use the `TCut` object when you draw (or project) the variable. Note that `""` are not used with `TCut` objects.

```
mytree->Draw("ebeam", *cut3);
```

Cuts are specified using C logic

&&	AND
||	OR
==	equal
!=	NOT equal
>	greater than
<	less than
>=	greater/equal
<=	less/equal

Fitting

Often you will need to fit distributions to determine the best parameters (e.g., particle mass, width, lifetime, etc.)

Several ways to define a fitting function in ROOT:

- Use ROOT's pre-defined functions (see `TF1` and `TFormula` class descriptions)

```
TF1* func1 = new TF1("func1", "gaus");
TF1* func2 = new TF1("func2", "gaus(0) + expo(3)");
```

- Define your own function within the `TF1` constructor

```
TF1* func1 = new TF1("func1", "[0]*x*exp([1]*x)");
func1->SetParameters(5.0, -0.5);
```

- Define your own function outside of the `TF1` class

Low edge | High edge | Number of parameters

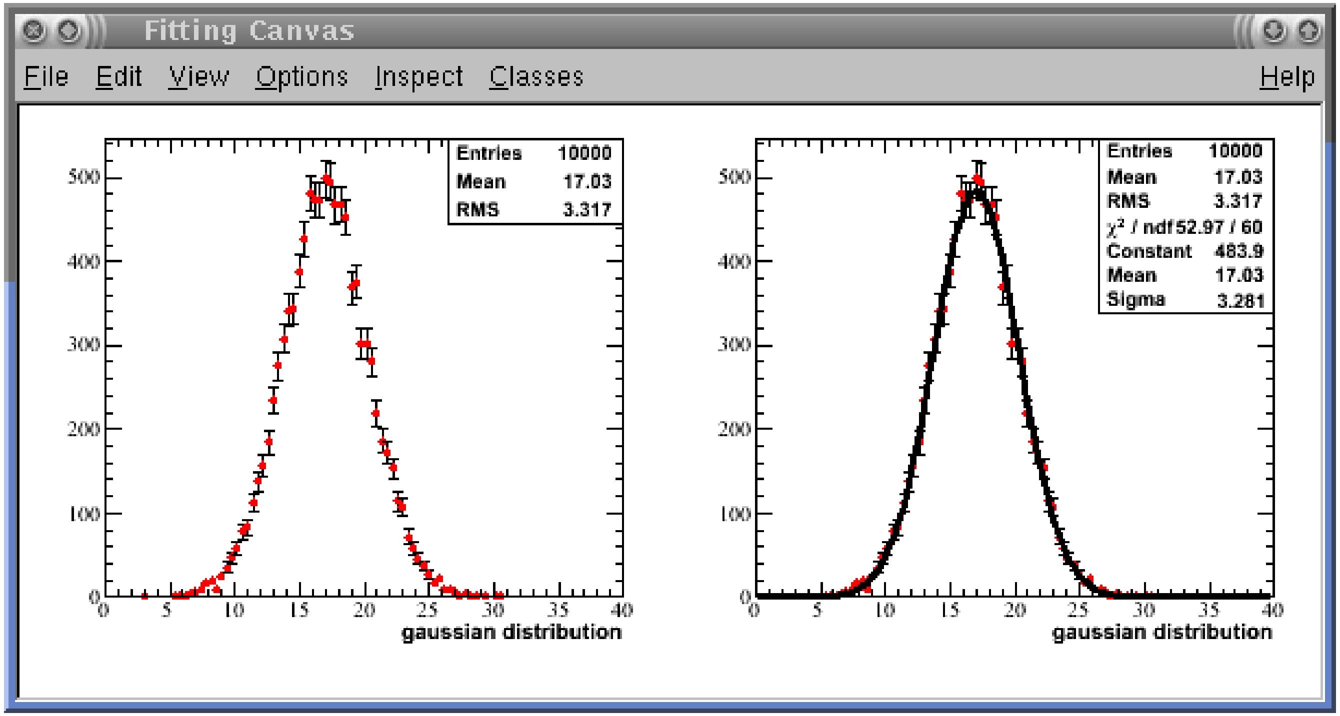
```
TF1* func1 = new TF1("func1", crazy_function, 0.0, 10.0, 2);

Double_t crazy_function(Double_t *x, Double_t *par){
    Float_t xx = x[0];
    Double_t function = abs(par[0]*sin(par[1]*xx)/xx);
}
```

Then fit the desired histogram or tree variable:

```
h1->Fit("func1");
tree1->Fit("func1","ebeam",*cut3);
```

Simple fitting example



Available ROOT fit functions may be found in `TFormula` class.

A few examples:

```
gaus
expo
polN   (N=0,1,2,...)
```

```
root [12] h1->Fit("gaus")
 FCN=52.9686 FROM MIGRAD    STATUS=CONVERGED      56 CALLS           57 TOTAL
                    EDM=5.33373e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                  STEP         FIRST
  NO.   NAME      VALUE           ERROR          SIZE      DERIVATIVE
   1  Constant     4.83945e+02   5.94585e+00   1.74050e-02  -4.07788e-05
   2  Mean         1.70284e+01   3.29593e-02   1.18187e-04  -6.96677e-03
   3  Sigma        3.28077e+00   2.33731e-02   6.90977e-06  -1.08841e-01
(Int_t)0
root [13] h1->GetXaxis()->SetTitle("gaussian distribution");
root [14] h1->Draw("e1");
root [15] TF1 *fitinfo = h1->GetFunction("gaus");
root [16] float gaus_constant = fitinfo->GetParameter(0);
root [17] float gaus_mean     = fitinfo->GetParameter(1);
root [18] float gaus_sigma    = fitinfo->GetParameter(2);
root [19] float gaus_chi2     = fitinfo->GetChisquare();
root [20] int n_events = h1->GetEntries();
```

Skeleton code for analysis

ROOT can create files for you that contain a code structure for analyzing trees

```
root [0] TFile *f1 = new TFile("tree.root");
root [1] TTree *t = (TTree *)f1->Get("tree1");
```

Load a file and tree

Use the `MakeClass` method to create code

```
root [2] t->MakeClass("TreeAnalysis");
```

This will create
TreeAnalysis.C and
TreeAnalysis.h

Add your analysis code to the .C file, then execute in ROOT

```
root [0] .L TreeAnalysis.C
root [1] TreeAnalysis mytreeanalysis;
root [2] mytreeanalysis.Loop();
```

"L"oad the file
Create an object of type TreeAnalysis
Access the "Loop" method
(where your analysis code is)

`MakeClass` is a quick way to create a framework for analyzing your data

– a good way to start, but... it can be fairly slow, especially in the case of trees with many variables
– for long projects (like your thesis work!), probably better to write your own code...

Other useful ROOT classes

`TLorentzVector`

A general 4-vector class with implemented functionality to do almost everything you typically need to do with 4-vectors.

- dot products
- rotations
- boosting
- angle between vectors
- magnitude...

`TFractionFitter`

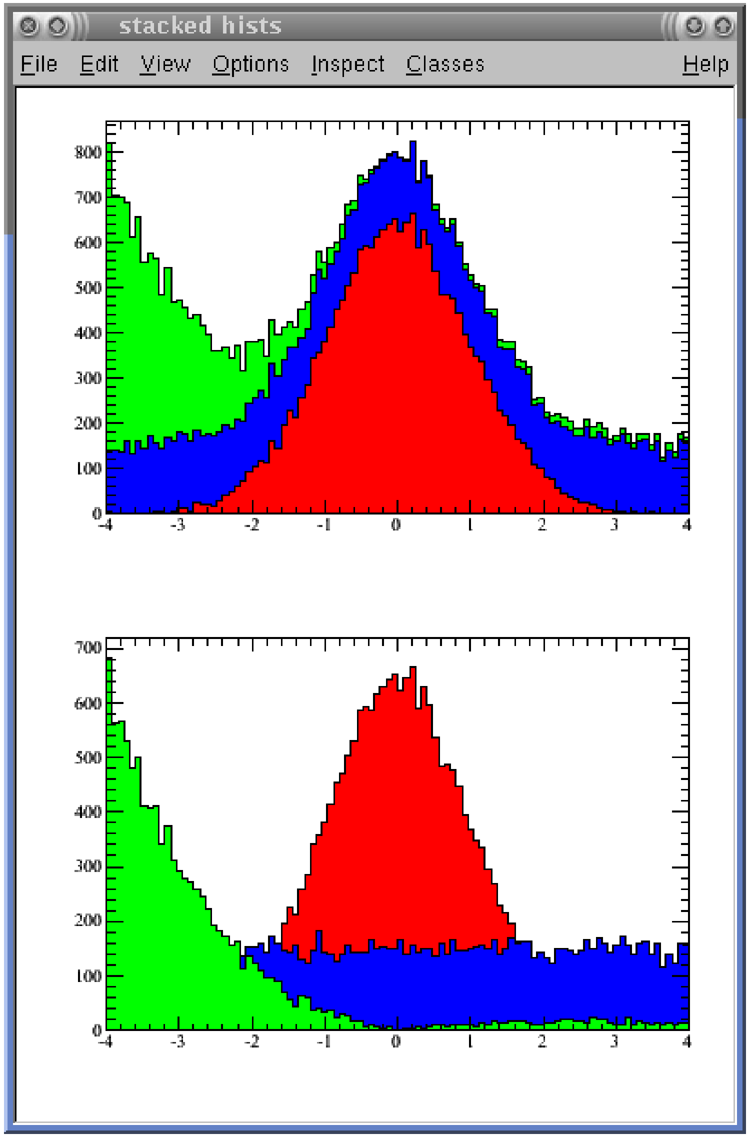
Fits data histogram using multiple MC histograms (instead of a defined function)

`TFitter, TMinuit`

Classes for fitting

`THStack`

Takes a collection of histograms and draws them "stacked" on each other.



Remember, the ROOT web documentation is your friend!

http://root.cern.ch/root/Reference.html

Useful links

ROOT Classes http://root.cern.ch/root/Categories.html
ROOT Tutorials http://root.cern.ch/root/Tutorials.html
ROOT Discussion Forum http://root.cern.ch/phpBB2/

BaBar ROOT tutorials
http://www.slac.stanford.edu/BFROOT/www/doc/workbook/root1/root1.html
http://www.slac.stanford.edu/BFROOT/www/doc/workbook/root2/root2.html

Nevis ROOT tutorial
http://www.nevis.columbia.edu/~seligman/root-class/

Exercise 1

Download the files in the directory http://hep.bu.edu/~jlraaf/NEPPSR/basic/

Write a macro to open the file “neppsr_basictutorial1.root”

What is in the file?

Create a canvas and draw one of the things in the file

- Try changing the line color
- Try drawing with error bars
- How many bins are there? (Hint: Look at `TH1` class description)
- What is the bin width?

Draw 2 of the things in the file on the same plot

Perform a fit on the first object contained in the file

- What type of shape did you use for the fit?
- What is the fitted width? fitted mean? true mean?
- What is the χ^2 for the fit? Is it a good fit?

Perform a fit on the second object contained in the file

- Try fitting with the built-in “expo” function first
  Does it give a good fit?
- Define your own `TF1` with the form par0*x*exp(par1*x)
  Perform the fit again. Does it give a good fit?
  Set reasonable starting parameters for your function.
  Perform the fit again. What are the fitted values of the 2 parameters?

If you have time, try to figure out what function would fit well for the third object.

Exercise 2

Read through the named macro “make_tree.C” to learn what it will do.

Run the code once, then look at the output file.
- Draw the histogram
- Draw one of the tree variables

Modify the code:
- Add the missing variables in “itree”
- Create some new branches for “newtree”
- Make a new histogram
- Run the code and verify that your new additions worked properly

Using `MakeClass`:
- Start ROOT, load the file “neppsr_basictutorial2.root” and make a pointer to the tree
- Use `MakeClass` to create skeleton code
  - Modify the `Loop()` method to perform the same actions as make_tree.C
    - (be sure to change the name of your output file!)
- Open both output files simultaneously in ROOT and compare them.