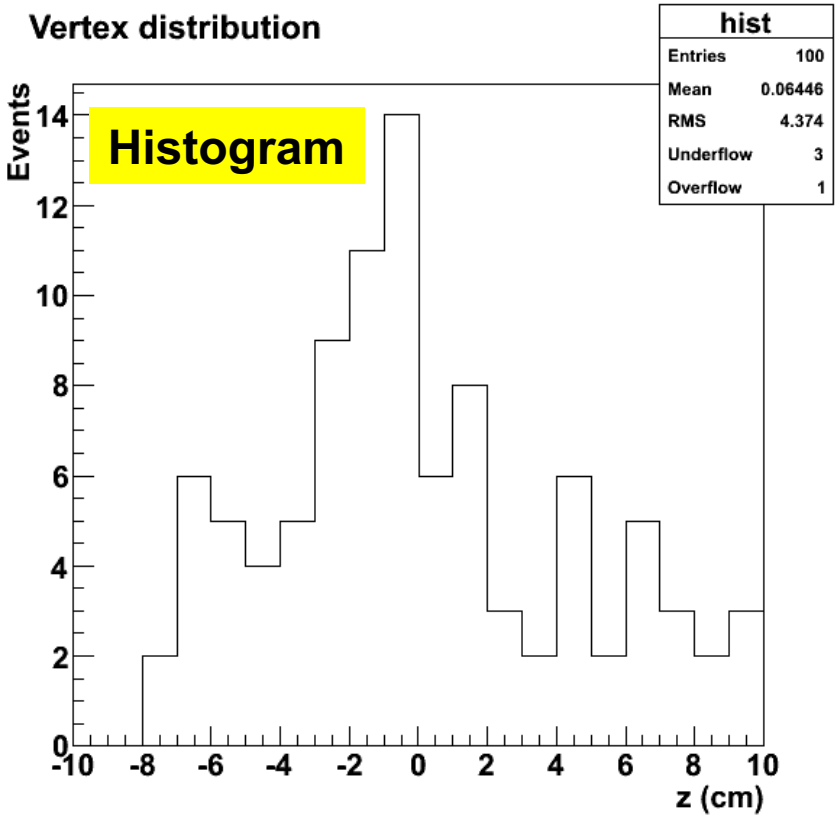
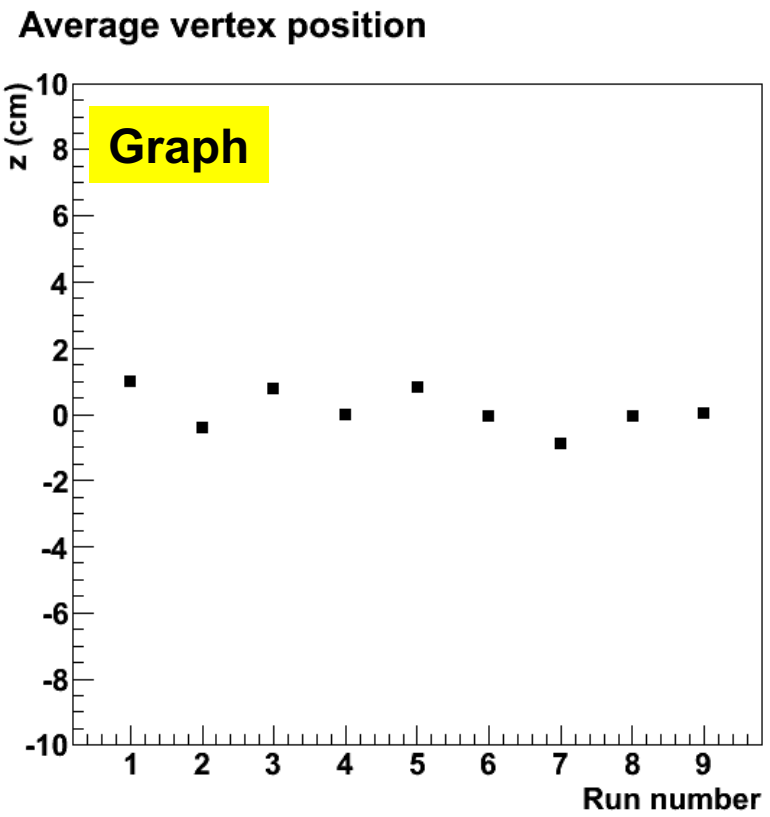
ROOT Framework2

Histograms & Graphs



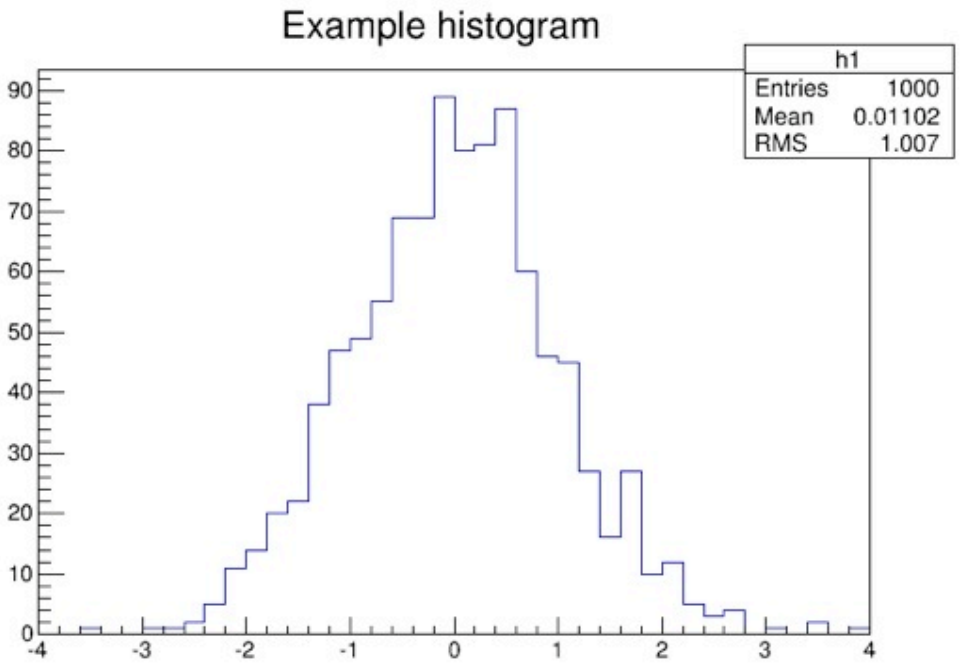
- Container for binned data
  - Most of HEP's distributions



- Container for distinct points
  - Calculation or fit results

What is a histogram?

- **What is a histogram?**
  - *from Wikipedia:*
  - a **histogram** is a graphical representation showing a visual impression of the distribution of data. It is an estimate of the probability distribution of a continuous variable and was first introduced by Karl Pearson.[1]
  - A histogram consists of tabular frequencies, shown as adjacent rectangles, erected over discrete intervals (bins), with an area equal to the frequency of the observations in the interval.
  - The height of a rectangle is also equal to the frequency density of the interval, i.e., the frequency divided by the width of the interval. The total area of the histogram is equal to the number of data entries.

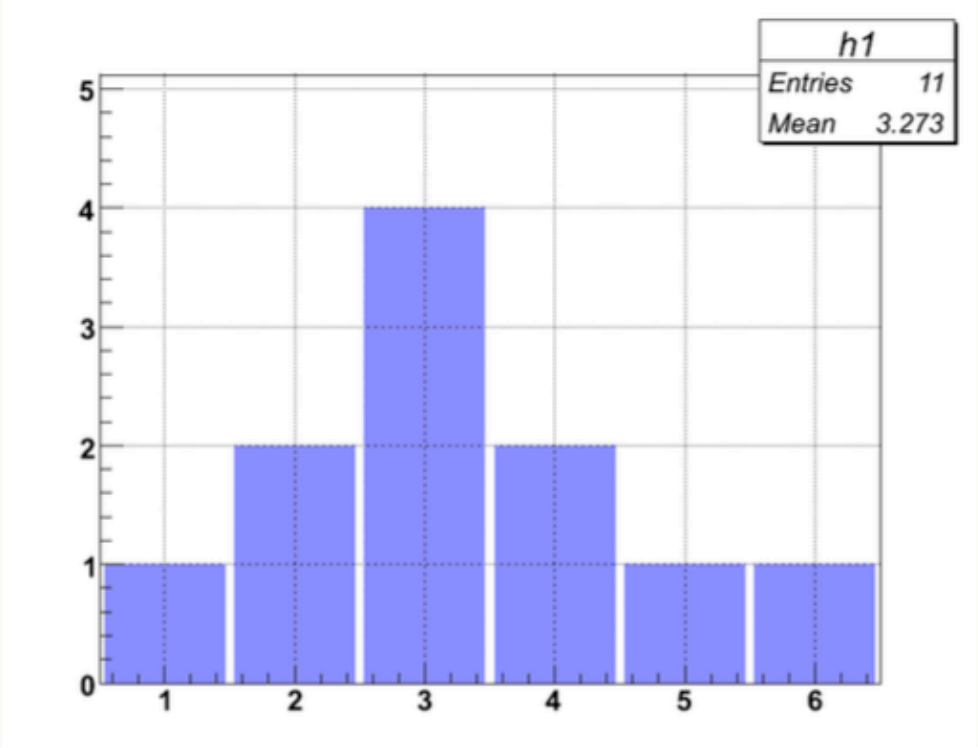


Histograms in ROOT

- Used to display and estimate the distribution of a variable (e.g. observed energy spectrum)
  - visualize number of events in a certain range, called *bin*
  - *bins* typically have equal widths, but not always
    - ROOT supports histograms with equal and variable bins
- Histograms can be used for further analysis
  - e.g to understand the underlying parent distribution
- ROOT provides various types of histograms depending on:
  - contained data type (double, float, integer, char)
  - choice of uniform or variable bins
  - dimension (1,2 or 3)

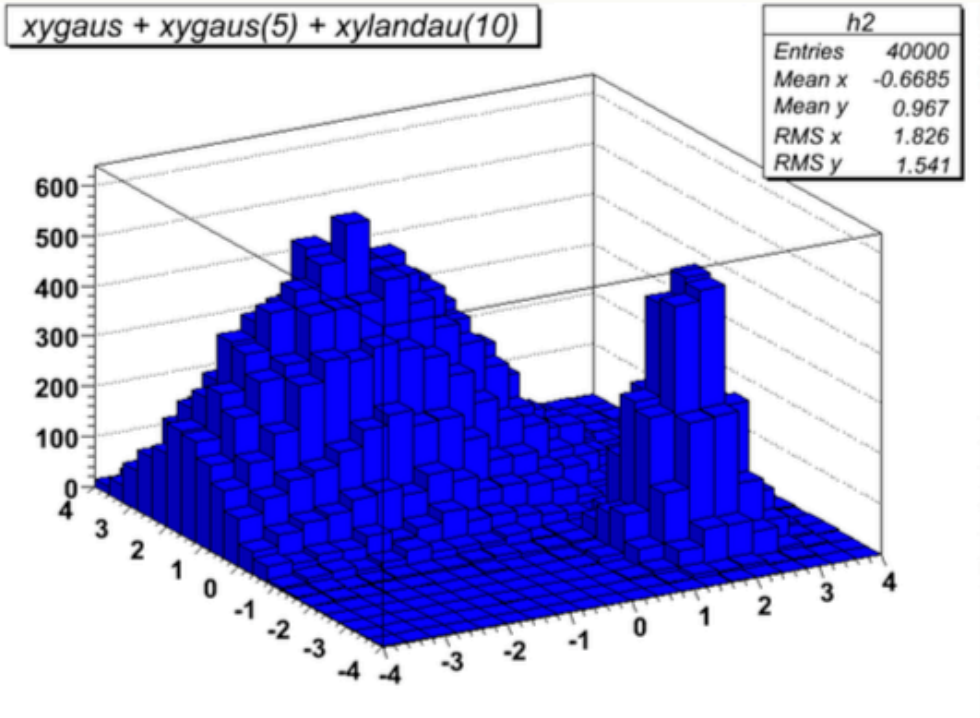
Histogramming

- Histogram is just occurrence counting, i.e. how often they appear
- Example: {1,3,2,6,2,3,4,3,4,3,5}

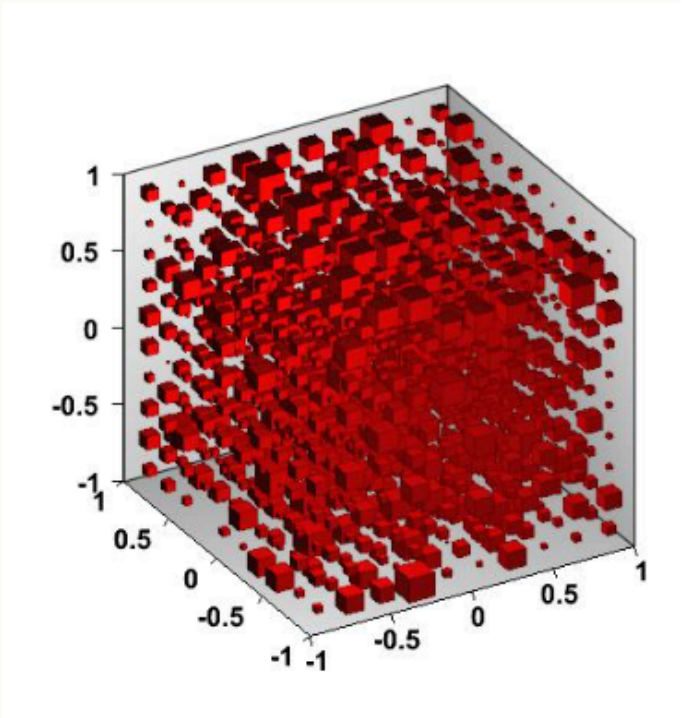


2D/3D Histogram

We have seen 1D histograms, but there are also histograms in more dimensions.



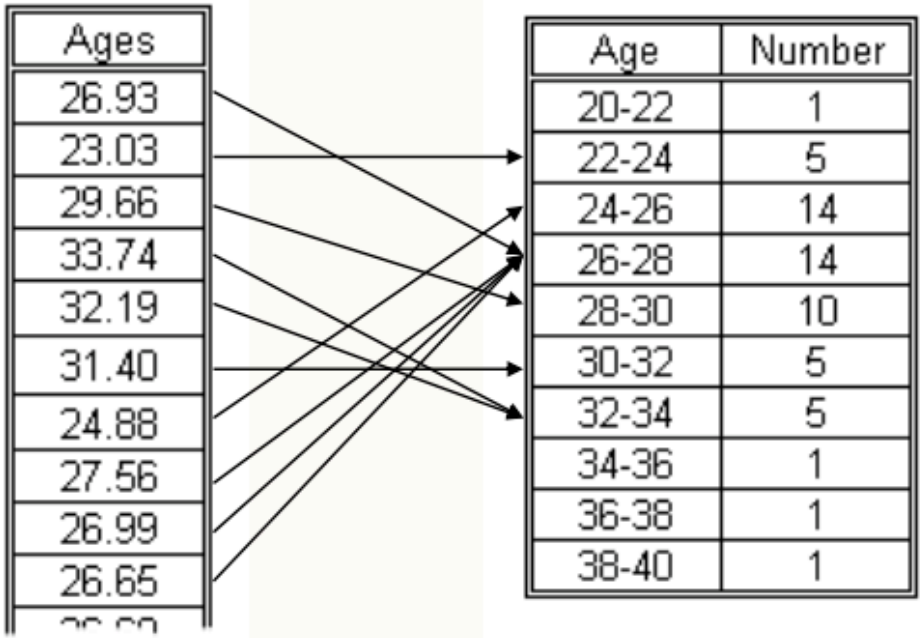
2D Histogram



3D Histogram

Histogramming

- **How is a Real Histogram Made?**
  Lets consider the age distribution of the CSC participants in 2008:



Ages
26.93
23.03
29.66
33.74
32.19
31.40
24.88
27.56
26.99
26.65

Age	Number
20-22	1
22-24	5
24-26	14
26-28	14
28-30	10
30-32	5
32-34	5
34-36	1
36-38	1
38-40	1

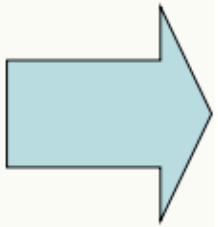
Binning:

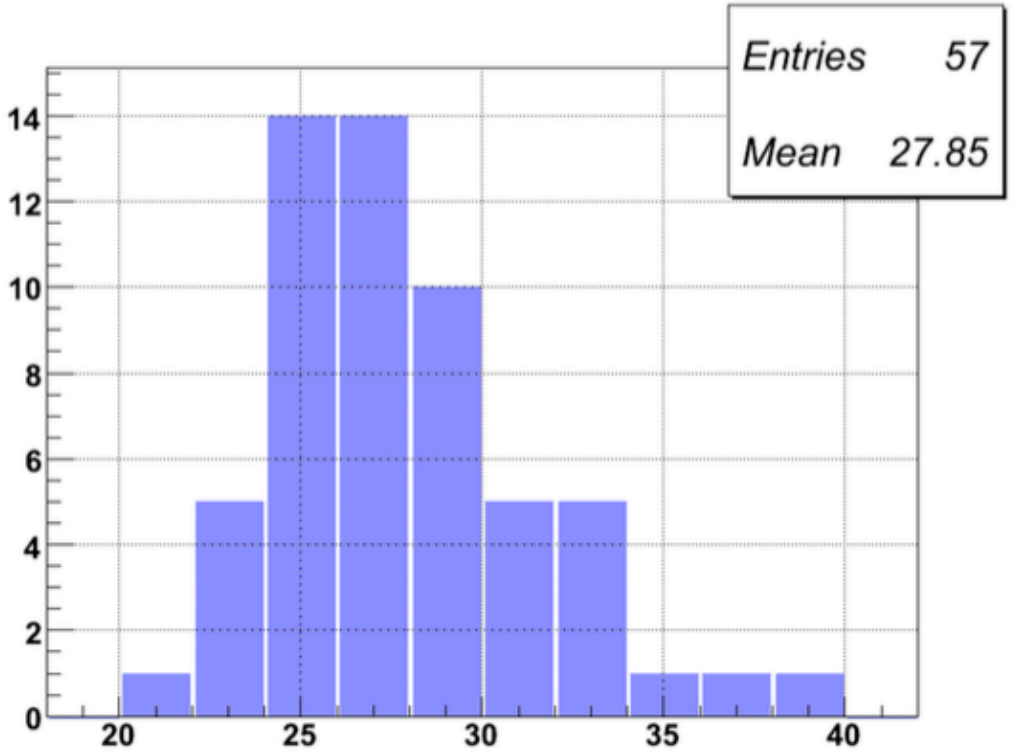
Grouping ages of participants in several categories (bins)

Histogramming

Table of Ages (binned)

Age	Number
20-22	1
22-24	5
24-26	14
26-28	14
28-30	10
30-32	5
32-34	5
34-36	1
36-38	1
38-40	1





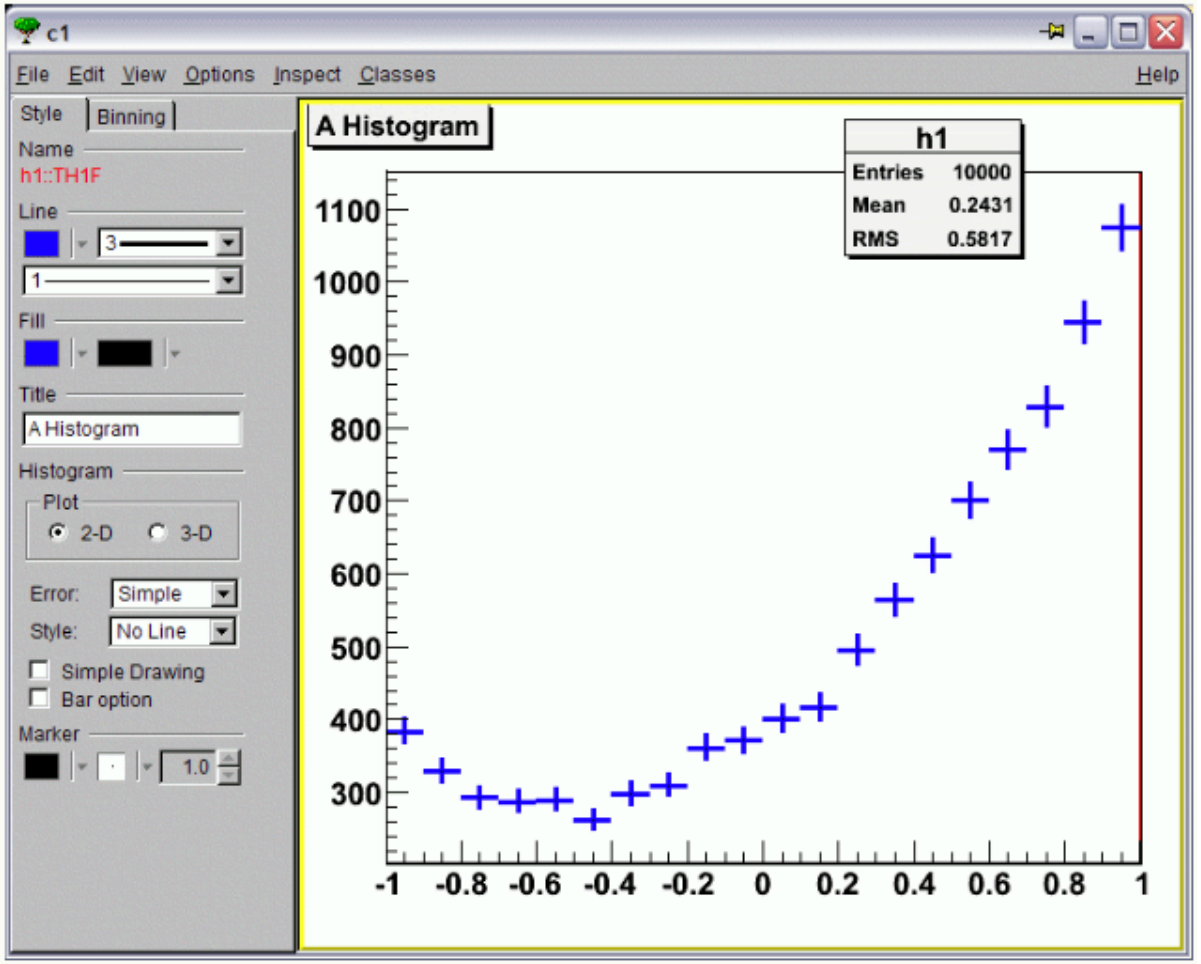
Shows distribution of ages, total number of entries (57 participants) and average: 27 years 10 months 6 days…

Histograms

Analysis result: often a histogram

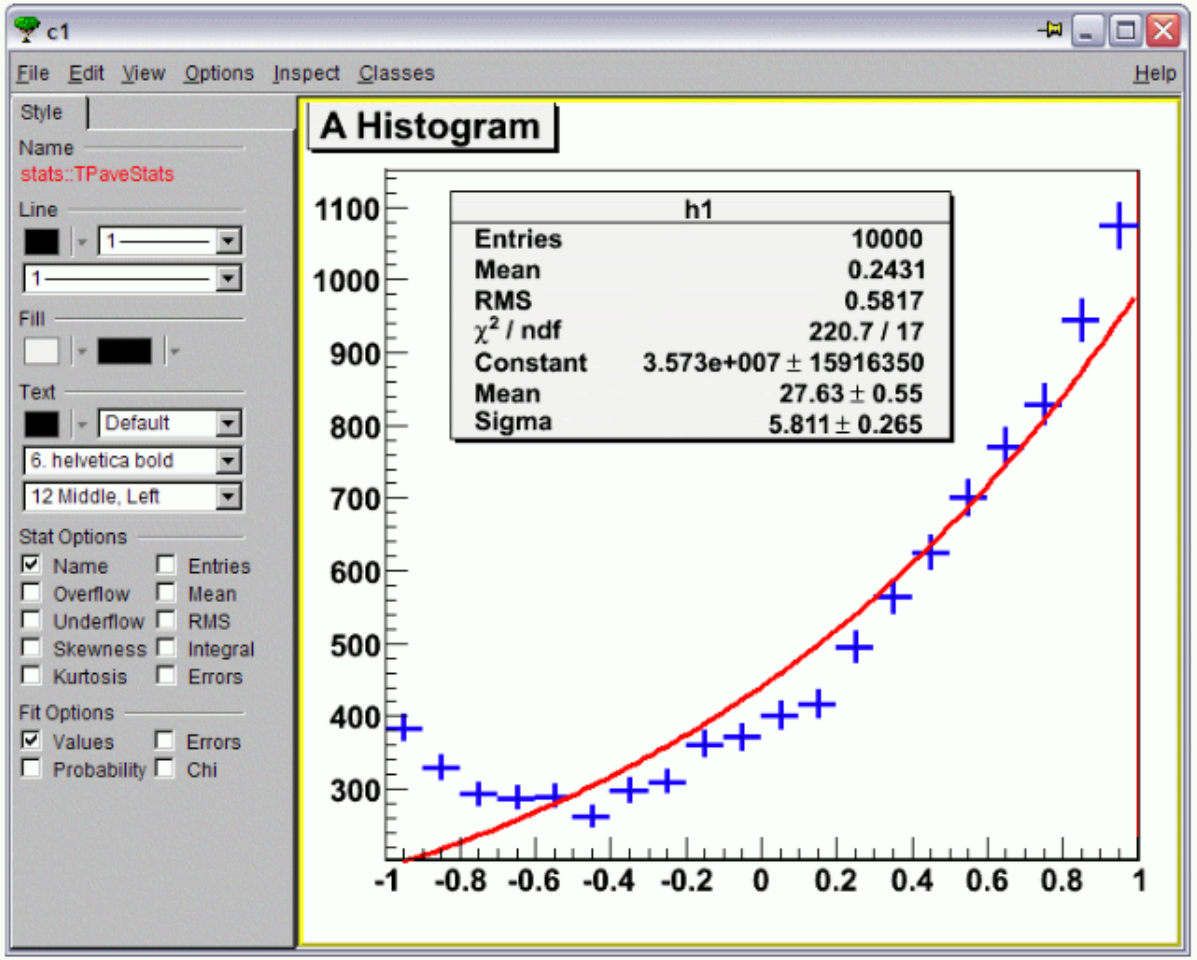
Menu:

View / Editor



Fitting

Analysis result: often a ***fit*** of a histogram



How to use ROOT Histograms

- Example of creating a one-dim. histogram:

```
TH1D * h1 = new TH1D("h1","Example histogram",40,-4.,4.);
```

histogram type
TH1D: one-dimension
using double types

name

title

axis settings

number of bins

min,max values

- Filling histogram:

```
h1->Fill(x);
```

Fill the histogram with one observation "

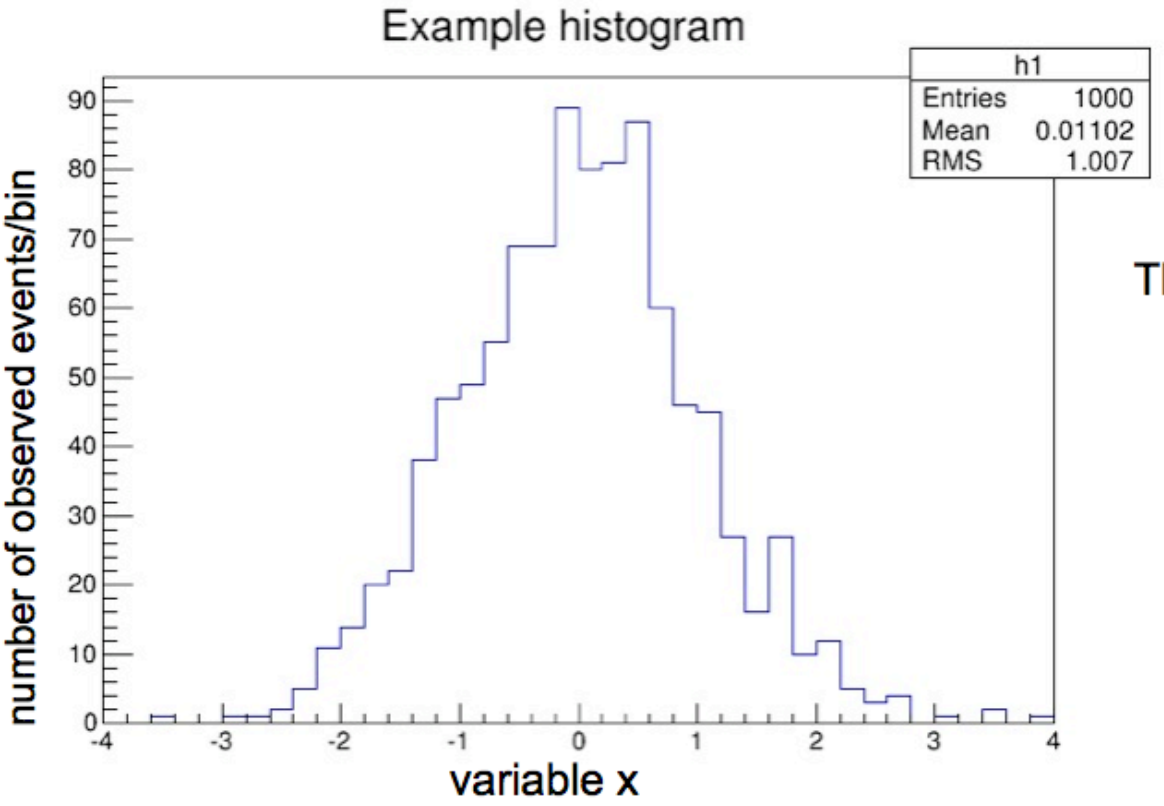
```
for (int i = 0; i<1000; ++i) {
  double x = gRandom->Gaus(0,1);
  h1->Fill(x);
}
```

Fill the histogram with 1000 gaussian distributed random numbers

Displaying ROOT histograms

- Drawing histograms in a ROOT canvas:

```
h1->Draw();
```



histogram name

The histogram statistics:

- Total number of entries
- Sample Mean
- Sample Standard Deviation (RMS)

Try the following commands in root prompt :

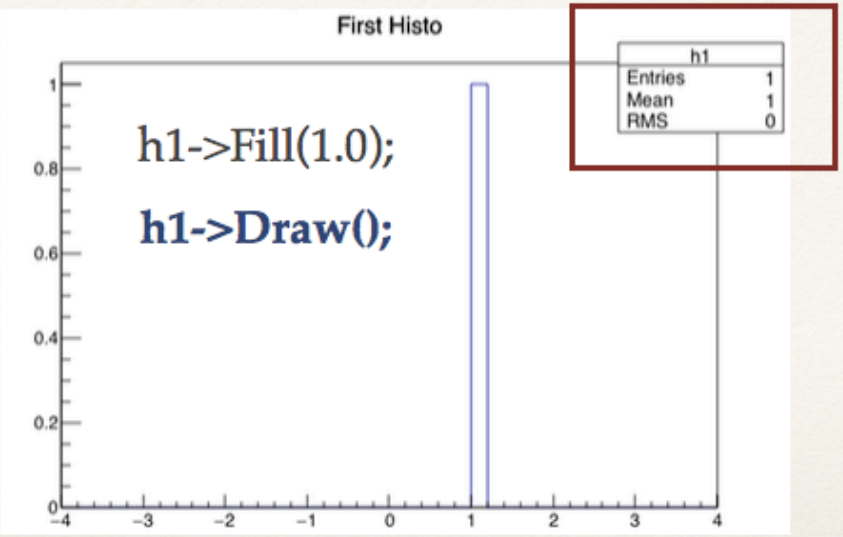
```
TH1D * h1 = new TH1D("h1", "First Histo",40,-4,4);

h1->Fill(1.0);
h1->Draw();
h1->Fill(1.1);
h1->Draw();
h1->Fill(1.15);
h1->Draw();
h1->Fill(1.19);
h1->Draw();
h1->Fill(1.2);
h1->Draw();
h1->Fill(1.3);
h1->Draw();
```

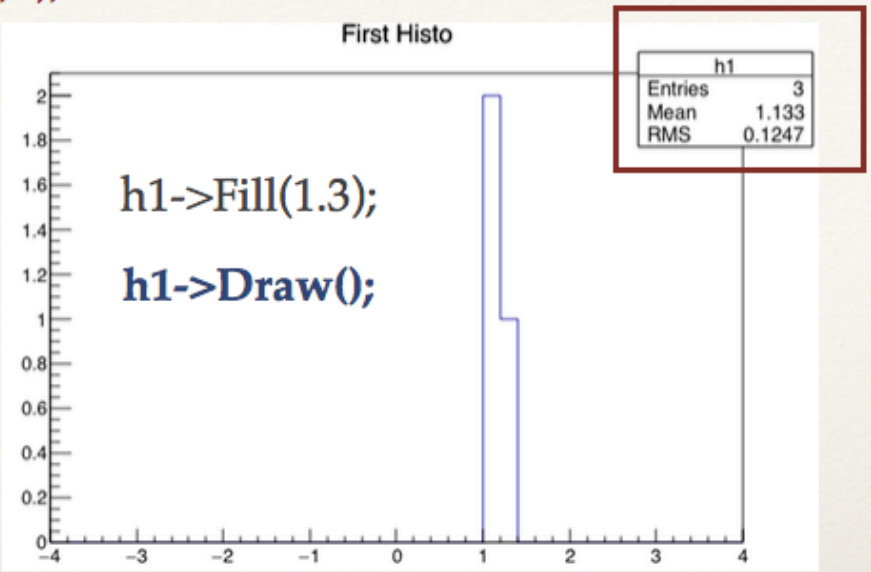
Redo the same exercise but now arrange bin size to 20;

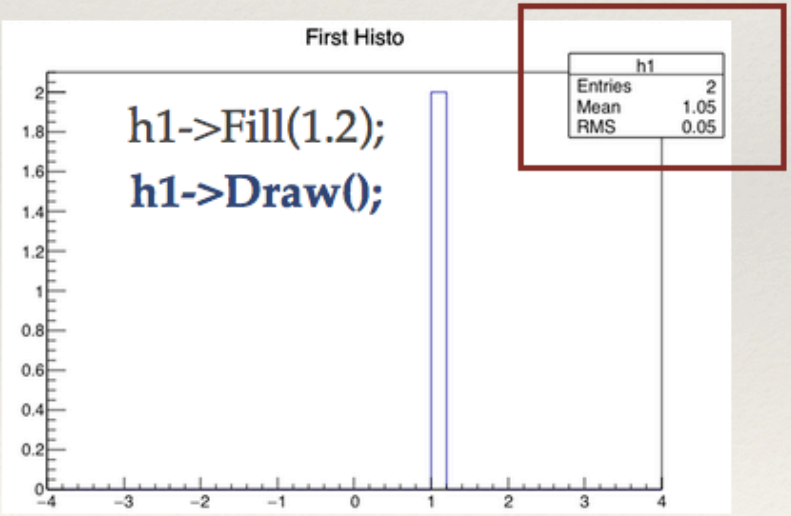
```
TH1D * h2 = new TH1D("h2", "my Second Histo",20,-4,4);

h2->Fill(1.0);
h2->Draw();
.
.
.
.
.
.
```

?? Look at the output graphs and comment on the graphs!! Understand the effects of bin size!!

TH1D * h1 = new TH1D("h1", "First Histo",40,-4,4);







Check the number of entries in each time

Check the distribution and number of occurrences (y axes) each time

Understand what is mean and RMS

Check manual !!

Histogram Statistics

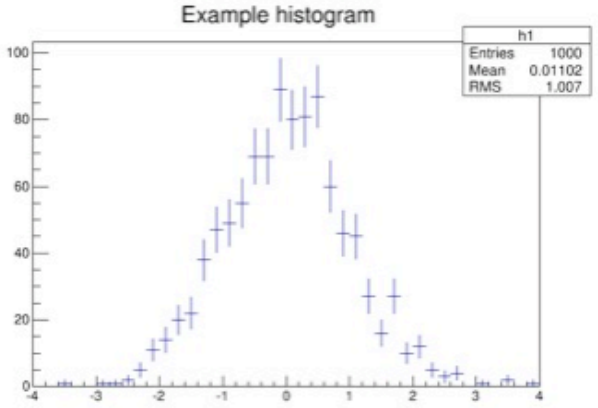
- To extract statistics information from an histogram:

```
root [] h1->GetEntries()
(const Double_t)1.0000000000000000e+03
root [] h1->Integral()
(const Double_t)1.0000000000000000e+03
root [] h1->GetMean()
(const Double_t)1.101727920359271 00e-02
root [] h1->GetMeanError()
(const Double_t)3.1831174486931387 8e-02
root [] h1->GetRMS()
(const Double_t)1.0065901197694480 1e+00
root [] h1->GetRMSError()
(const Double_t)2.2508039332841407 7e-02
root [] h1->GetSkewness()
(const Double_t)1.1782073846449019 1e-01
root [] h1->GetKurtosis()
(const Double_t)2.5896196835884000 0e-01
```

Describe what each of the term means;
prepare a one or two pages slide to explain these terms, with an example if possible

Histogram Drawing Options

- Various drawing options are available:
  - draw error bars on every bin




```
h1->Draw("E");
```

  - "SAME"

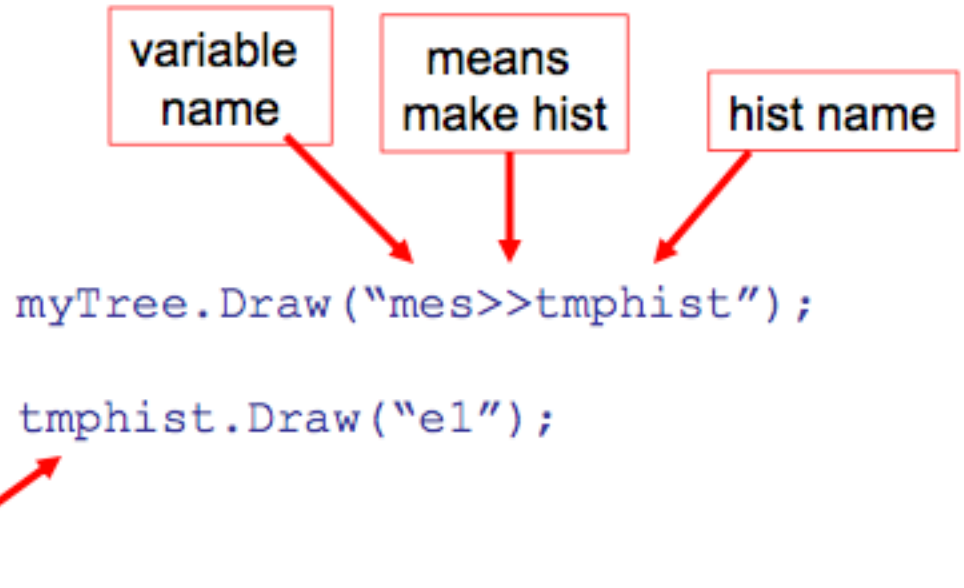
```
h1->Draw("SAME");
```

    - draw the histogram on the canvas without replacing what is already there
    - use to plot one histogram on top of another
- The default drawing option is "HIST" for histograms without errors (unweighted histograms) and "E" for weighted histograms
- For displaying the histogram in log scale in one axis, e.g. the y axis:

```
gPad->SetLogy();
```

Making histograms from a TTree

- When you draw a variable from a TTree you can fill a histogram

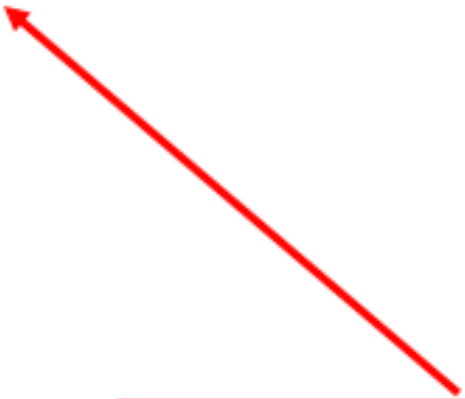


Now root knows you have a histogram of name `tmphist`

`tmphist` is a histogram made to have the content corresponding to that of the tree

Making histograms from a TTree

```
myTree.Draw("mes>>tmphist");
myTree1.Draw("mes>>+tmphist");
myTree2.Draw("mes>>+tmphist");

tmphist.Draw("e1");
```



>>+ means add to existing histogram

By default you get a histogram with 100 bins. If you want to change this you'll have to specify a histogram yourself; e.g.:

```
TH1F tmphist("tmphist", "", 25, 5.2, 5.3);
myTree->Draw("mes>>tmphist");
```

Useful commands for Histogram

- Change the line colour: `h1->SetLineColor(kRed);`
- Title: `h1->SetTitle("My title");`
- X axis title: `h1->SetXTitle("The x axis");`
- Change x-axis range: `SetAxisRange(4., 15);` //zoom
- Line colours: `SetMarkerColor(kBlue);` //etc
- Point size: `SetMarkerSize(1.);`
- Point style: `SetMarkerStyle(20);` //experiment!!
- Fill colour: (def: white) `SetFillColor(kGreen);`
- Draw a filled histo: `SetFillStyle(3004);` // diagonal lines
- Histo with error bars: `h1->Draw("e");` //error = sqrt[nentries]
- Print to screen: `h1->Print("all")` //can omit "all"
- Usually need to redraw histo after any changes:

```
h1->Draw();
```

- Draw a second histogram, `h2`, on the same canvas as another one:

```
h2->Draw("same");
```

TGraph: Plotting Measurements

- The Graph class (**`TGraph`**):
  - for plotting and analyzing 2-dimensional data (X,Y),
  - contains a set of N distinct points (X_i , Y_i) i = 1,..N.
  - can be constructed from a set of x,y arrays:

```
root [1] double x[] = { 1,2,3,4,5};
root [2] double y[] = { 0.5,2.,3.,3.2,4.7};
root [3] TGraph * g = new TGraph(5,x,y);
```

  - or directly from a text file containing rows of (X,Y) data

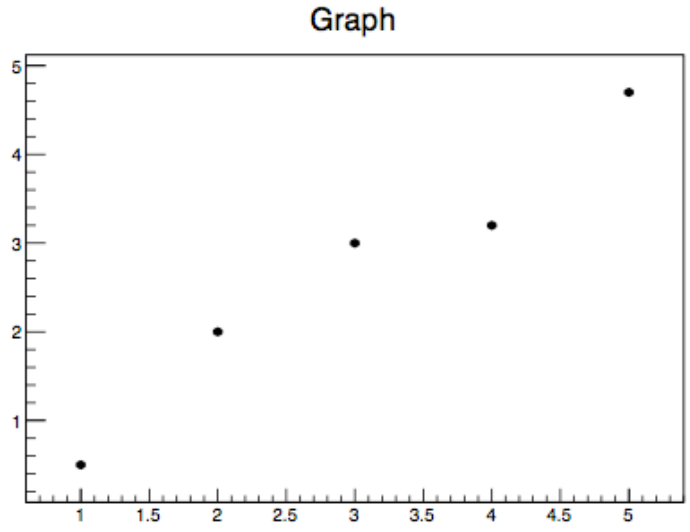
```
root [1] TGraph * g = new TGraph("XYData.txt");
```

Displaying a Graph

- To display the graph:

```
root [4] g->Draw("AP");
```

  - option "A" means displaying the axis,
  - option "P" means displaying the points.



To change the point markers do:

```
root [4] g->SetMarkerStyle(20);
```

Graphs Drawing Options

- The drawing of Graphs is done via the **`TGraphPainter`**
  - see http://root.cern.ch/root/html/TGraphPainter.html
  - the documentation lists all drawing options for the different types of graphs available in ROOT

Graphs' plotting options

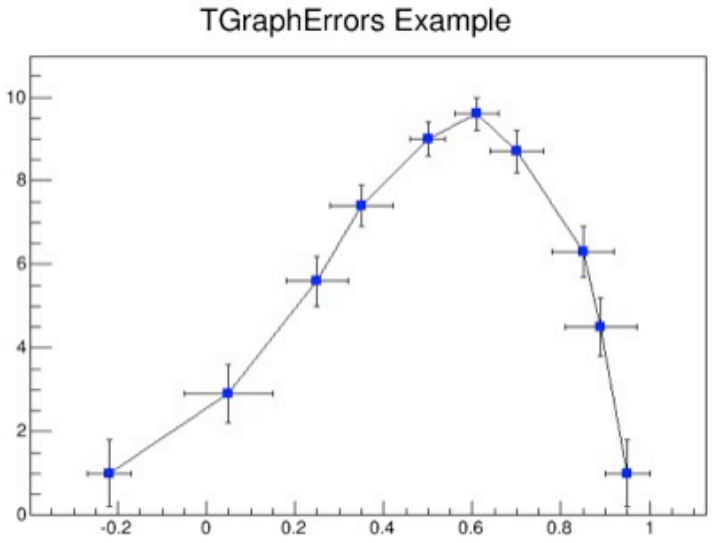
Graphs can be drawn with the following options:

"A"	Axis are drawn around the graph
"L"	A simple polyline is drawn
"F"	A fill area is drawn ('CF' draw a smoothed fill area)
"C"	A smooth Curve is drawn
"*"	A Star is plotted at each point
"P"	The current marker is plotted at each point
"B"	A Bar chart is drawn
"1"	When a graph is drawn as a bar chart, this option makes the bars start from the bottom of the pad. By default they start at 0.
"X+"	The X-axis is drawn on the top side of the plot.
"Y+"	The Y-axis is drawn on the right side of the plot.

Drawing options can be combined. In the following example the graph is drawn as a smooth curve (option "C") with markers (option "P") and with axes (option "A").

Types of Graphs

- ROOT provides various types of Graphs:
  - **TGraph**： (x,y) data points.
  - **TGraphErrors**:
    - (x,y) data points with error bars (σ_x,σ_y).
  - **TGraphAsymmErrors**:
    - (x,y) data points with asymmetric error bars [(σ^-_x,σ^+_x),(σ^-_y,σ^+_y)].




```
TGraphErrors *gr = new TGraphErrors(n,x,y,ex,ey);
gr->SetTitle("TGraphErrors Example");
gr->SetMarkerColor(4);
gr->SetMarkerStyle(21);
gr->Draw("ALP");
```

Use the drawing option “L” for displaying a line connecting the points

Plotting Measurements

- The Graph class (**`TGraph`**):
  - for plotting and analyzing 2-dimensional data (X,Y),
  - contains a set of N distinct points (X_i , Y_i) i = 1,..N.
  - can be constructed from a set of x,y arrays:

```
root [1] double x[] = { 1,2,3,4,5};
root [2] double y[] = { 0.5,2.,3.,3.2,4.7};
root [3] TGraph * g = new TGraph(5,x,y);
```

  - or directly from a text file containing rows of (X,Y) data

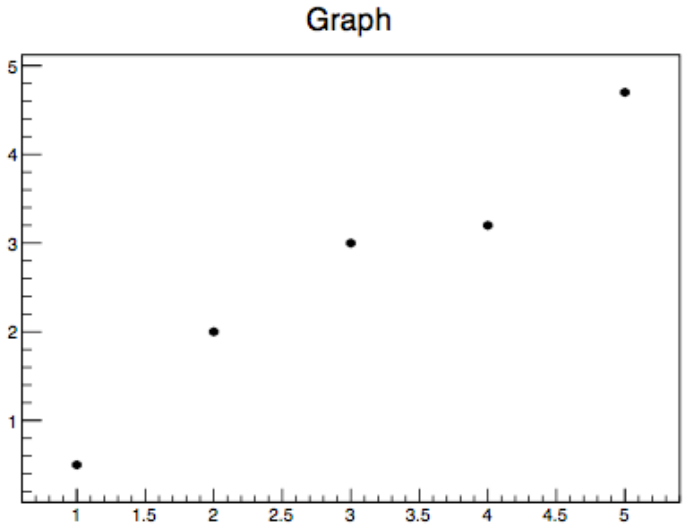
```
root [1] TGraph * g = new TGraph("XYData.txt");
```

Displaying a Graph

- To display the graph:

```
root [4] g->Draw("AP");
```

  - option "A" means displaying the axis,
  - option "P" means displaying the points.

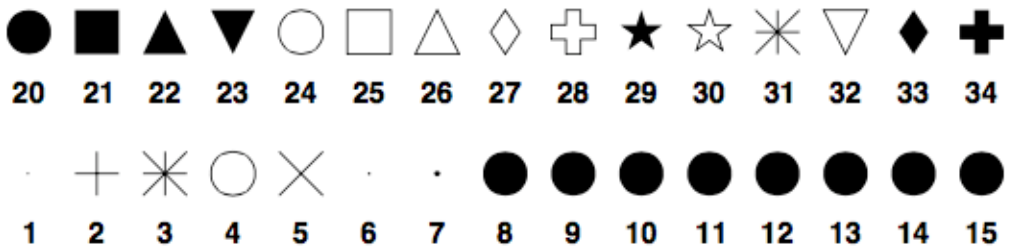


To change the point markers do:

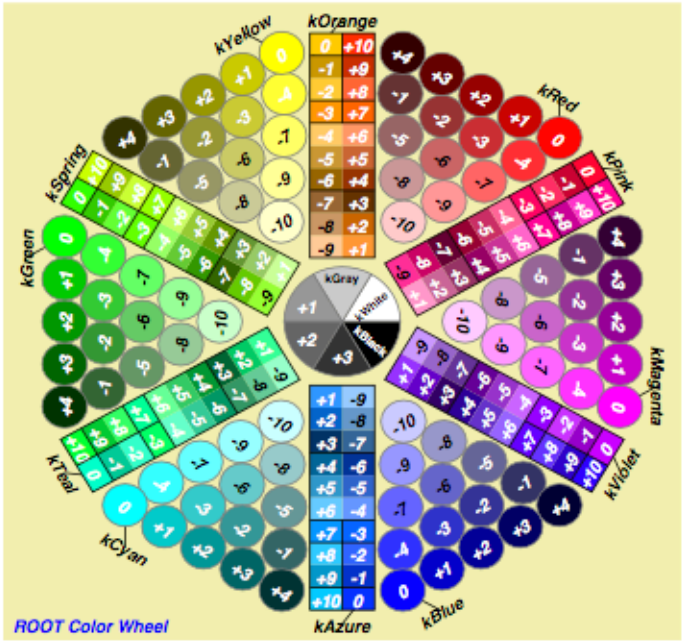
```
root [4] g->SetMarkerStyle(20);
```

Markers and Colors

- Available markers in ROOT



- Available Colors



you can access them from the Canvas View Menu

Graphs Drawing Options

- The drawing of Graphs is done via the **`TGraphPainter`**
  - see http://root.cern.ch/root/html/TGraphPainter.html
  - the documentation lists all drawing options for the different types of graphs available in ROOT

Graphs' plotting options

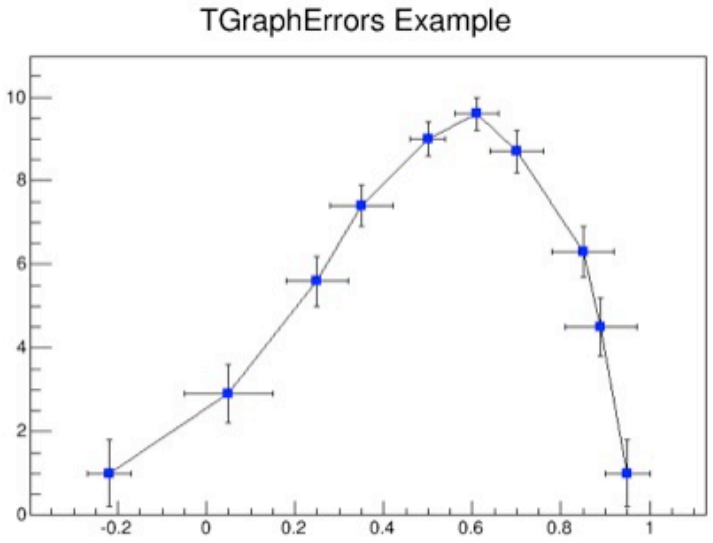
Graphs can be drawn with the following options:

"A"	Axis are drawn around the graph
"L"	A simple polyline is drawn
"F"	A fill area is drawn ('CF' draw a smoothed fill area)
"C"	A smooth Curve is drawn
"*"	A Star is plotted at each point
"P"	The current marker is plotted at each point
"B"	A Bar chart is drawn
"1"	When a graph is drawn as a bar chart, this option makes the bars start from the bottom of the pad. By default they start at 0.
"X+"	The X-axis is drawn on the top side of the plot.
"Y+"	The Y-axis is drawn on the right side of the plot.

Drawing options can be combined. In the following example the graph is drawn as a smooth curve (option "C") with markers (option "P") and with axes (option "A").

Types of Graphs

- ROOT provides various types of Graphs:
  - **TGraph** : (x,y) data points.
  - **TGraphErrors**:
    - (x,y) data points with error bars (σ_x,σ_y).
  - **TGraphAsymmErrors**:
    - (x,y) data points with asymmetric error bars [(σ^-_x,σ^+_x),(σ^-_y,σ^+_y)].




```
TGraphErrors *gr = new TGraphErrors(n,x,y,ex,ey);
gr->SetTitle("TGraphErrors Example");
gr->SetMarkerColor(4);
gr->SetMarkerStyle(21);
gr->Draw("ALP");
```

Use the drawing option “L” for displaying a line connecting the points

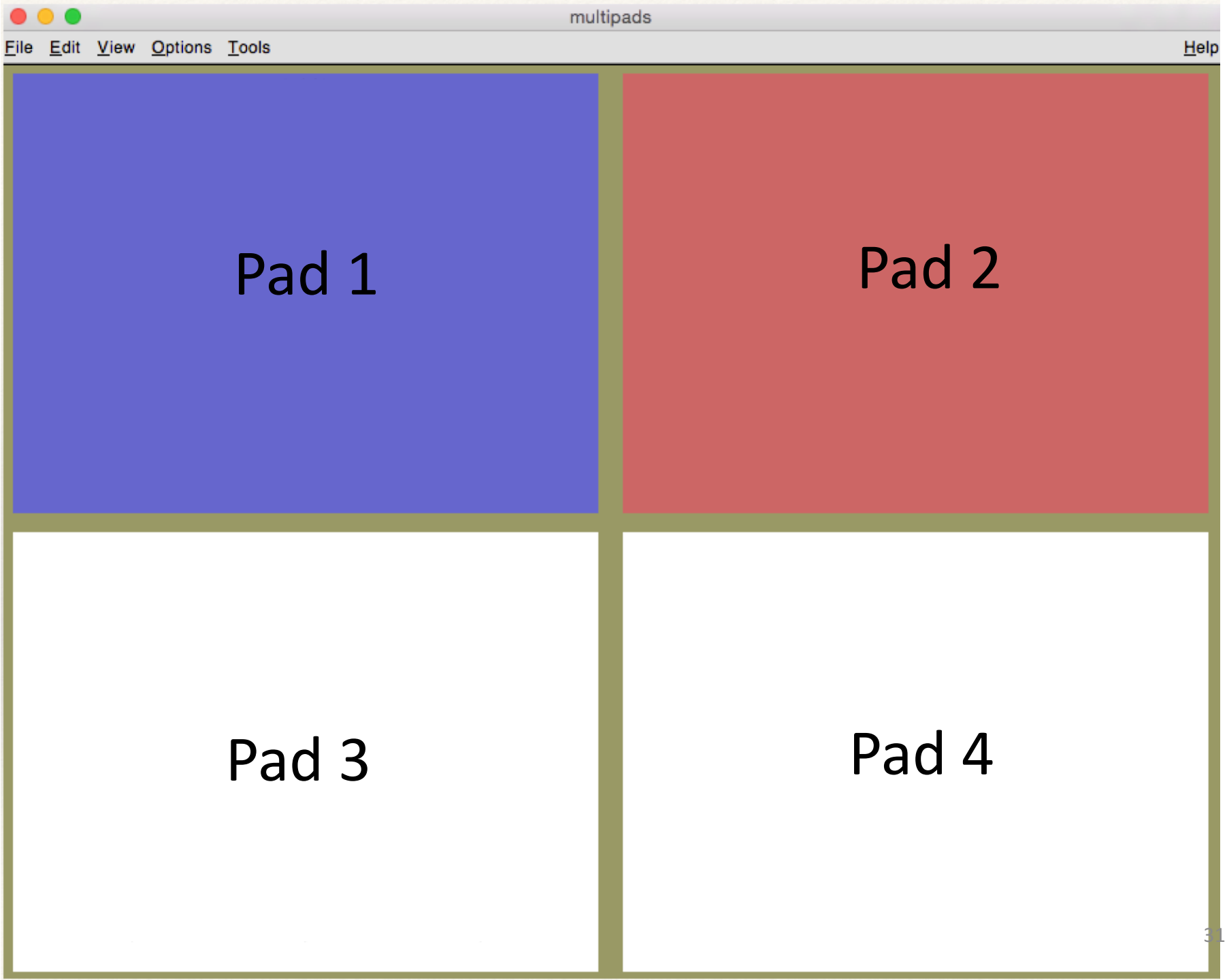
TCanvas and Tpad

$ROOTSYS/tutorials/graphs/zones.C

```
1  void zones() {
2  // example of script showing how to divide a canvas
3  // into adjacent subpads + axis labels on the top and right side
4  // of the pads.
5  //Author; Rene Brun
6
7     TCanvas *c1 = new TCanvas("c1","multipads",900,700);
8     gStyle->SetOptStat(0);
9     c1->Divide(2,2,0,0);
10    TH2F *h1 = new TH2F("h1","test1",10,0,1,20,0,20);
11    TH2F *h2 = new TH2F("h2","test2",10,0,1,20,0,100);
12    TH2F *h3 = new TH2F("h3","test3",10,0,1,20,-1,1);
13    TH2F *h4 = new TH2F("h4","test4",10,0,1,20,0,1000);
14
15    c1->cd(1);
16    gPad->SetTickx(2);
17    h1->Draw();
18
19    c1->cd(2);
20    gPad->SetTickx(2);
21    gPad->SetTicky(2);
22    h2->GetYaxis()->SetLabelOffset(0.01);
23    h2->Draw();
24
25    c1->cd(3);
26    h3->Draw();
27
28    c1->cd(4);
29    gPad->SetTicky(2);
30    h4->Draw();
31 }
32
```

Example: multipads

```
1  void our_macro(){
2    //We will create a canvas first
3
4    //Create a Canvas with name c1
5  TCanvas *c1 = new TCanvas("c1","multipads",900,700);
6
7  //Divide the canvas int0 2x2
8   c1->Divide(2,2);
9
10  //to activate the first pad
11  c1->cd(1);
12
13  //change the fill color of the first pad
14  c1->SetFillColor(kYellow-5);
15
16  //change the fill color of the Pad
17  gPad->SetFillColor(kBlue-6);
18
19  //change the fill color of the second pad
20 c1->cd(2);
21 gPad->SetFillColor(kRed-6);
22
23 //Draw to see the results
24 c1->Draw();
25 }
```


multipads
File Edit View Options Tools Help
Pad 1
Pad 2
Pad 3
Pad 4

root [0] root $ROOTSYS/tutorials/graphics/first.C

CODE:

```
1  //Show some basic primitives
2  //Author: Rene Brun
3  void first() {
4
5     TCanvas *nut = new TCanvas("nut", "FirstSession",100,10,700,900);
6     nut->Range(0,0,20,24);
7     nut->SetFillColor(10);
8     nut->SetBorderSize(2);
9
10    TPaveLabel *pl = new TPaveLabel(3,22,17,23.7,
11       "My first ROOT interactive session","br");
12    pl->SetFillColor(18);
13    pl->Draw();
14
15    TText t(0,0,"a");
16    t.SetTextFont(62);
17    t.SetTextSize(0.025);
18    t.SetTextAlign(12);
19    t.DrawText(2,20.3,"ROOT is based on CINT, a powerful C/C++ interpreter.");
20    t.DrawText(2,19.3,"Blocks of lines can be entered within {...}.");
21    t.DrawText(2,18.3,"Previous typed lines can be recalled.");
22
```

```
39
40    TPad *pad = new TPad("pad","pad",.2,.05,.8,.35);
41    pad->SetFillColor(42);
42    pad->SetFrameFillColor(33);
43    pad->SetBorderSize(10);
44    pad->Draw();
45    pad->cd();
46    pad->SetGrid();
47    TF1 *f1 = new TF1("f1","sin(x)/x",0,10);
48    f1->Draw();
49    nut->cd();
50 }
51
```

OUTPUT:

My first ROOT interactive session

ROOT is based on CINT, a powerful C/C++ interpreter.

Blocks of lines can be entered within {...}.

Previous typed lines can be recalled.

Root > float x=5; float y=7;

*Root > x*sqrt(y)*

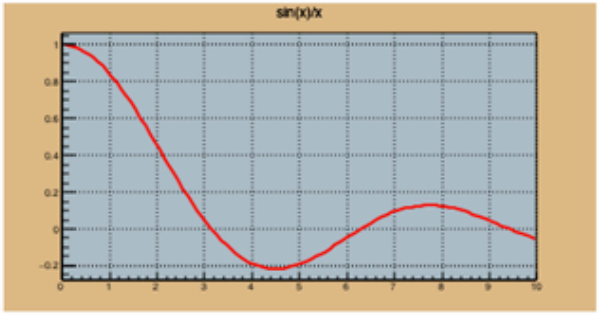
```
(double)1.322875655532e+01
```

Root > for (int i=2;i<7;i++) printf("sqrt(%d) = %f",i,sqrt(i));

```
sqrt(2) = 1.414214
sqrt(3) = 1.732051
sqrt(4) = 2.000000
sqrt(5) = 2.236068
sqrt(6) = 2.449490
```

Root > TF1 f1("f1","sin(x)/x",0,10)

Root > f1.Draw()



Exercise: user function

Pad 1

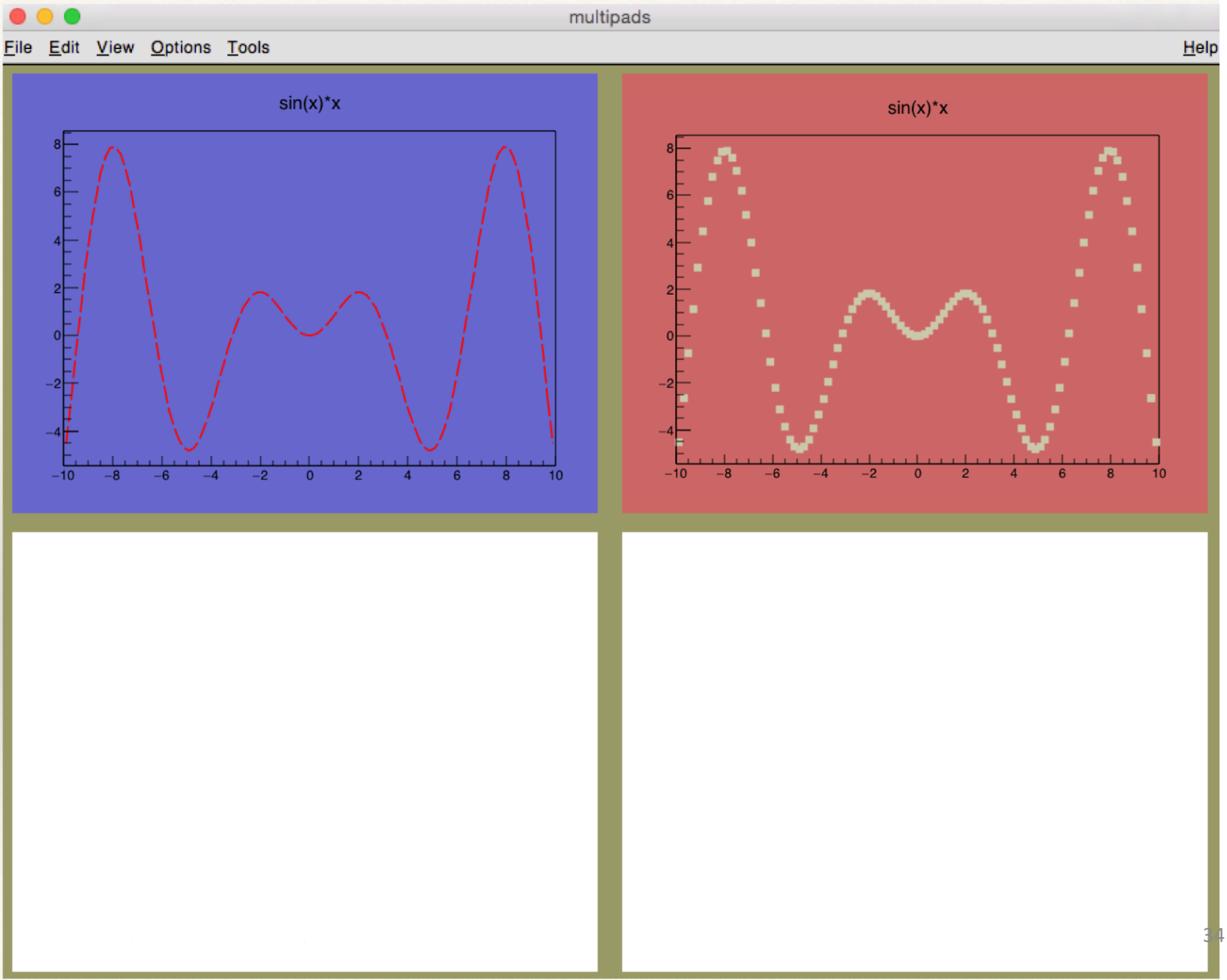
ex1

```
//Create our first function (Lets create x*sin(x))

TF1 * f1 = new TF1("xsinx","sin(x)*x",-10,10);
//Lets do some make up for the plot

f1->SetMarkerStyle(21);
f1->SetMarkerColor(21);

f1->SetLineStyle(9);
f1->SetLineColor(kRed);

//Choose the pad you want to draw
//Lets draw it to the first pad
c1->cd(1);
//If you want to draw with line
f1->Draw("L");
```

Pad 2

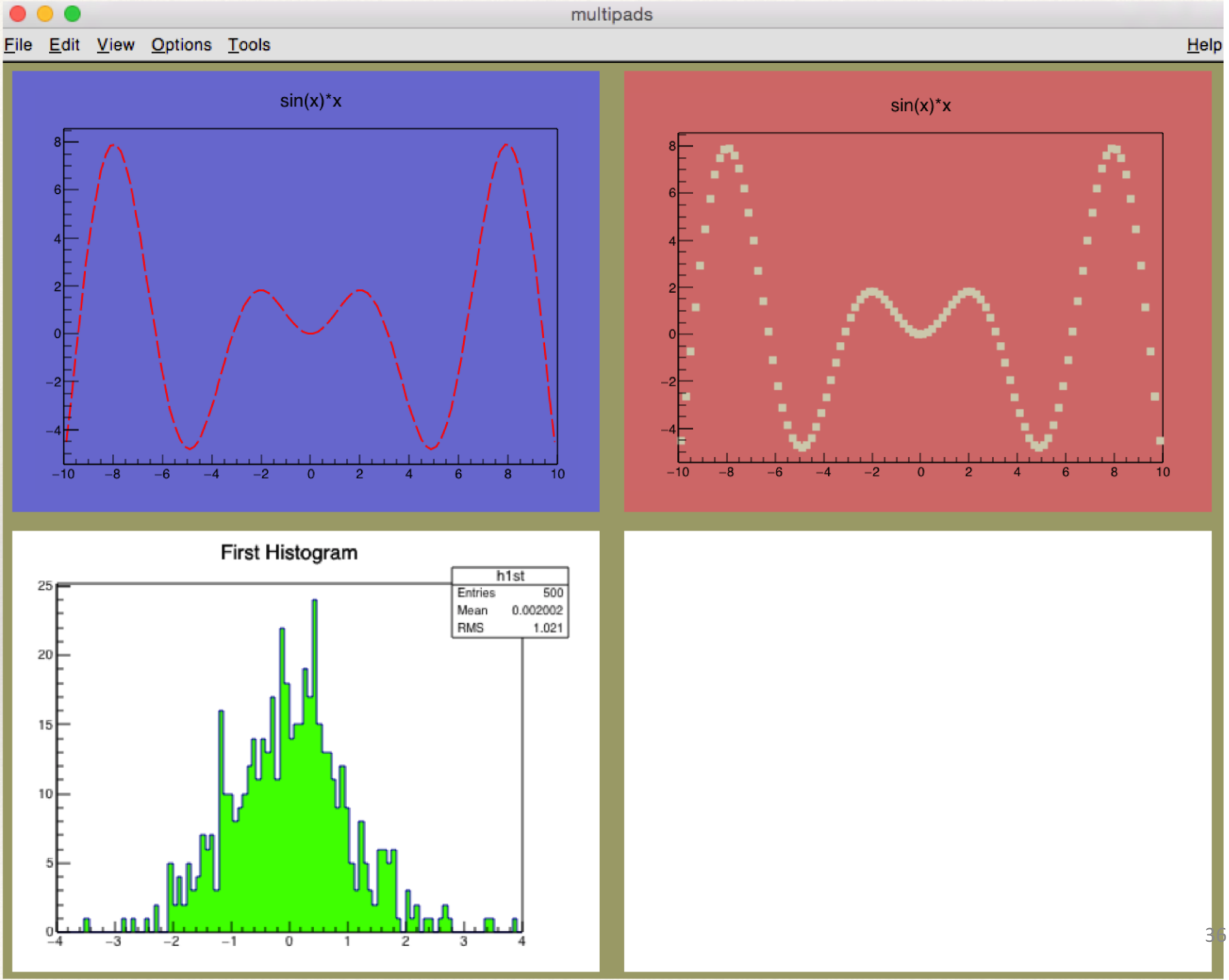
ex2

```
//Create our first function (Lets create x*sin(x))

 TF1 * f1 = new TF1("xsinx","sin(x)*x",-10,10);
 //Lets do some make up for the plot

 f1->SetMarkerStyle(21);
 f1->SetMarkerColor(21);

 f1->SetLineStyle(9);
 f1->SetLineColor(kRed);

 //Choose the pad you want to draw
 //Lets draw it to the first pad
 c1->cd(1);
 //If you want to draw with line
 // f1->Draw("L");

 //If you want to Draw with Points using marker style then;
 f1->Draw("P");


 // L option means line
 // P option means point
}
```


multipads
File Edit View Options Tools Help
sin(x)*x
sin(x)*x

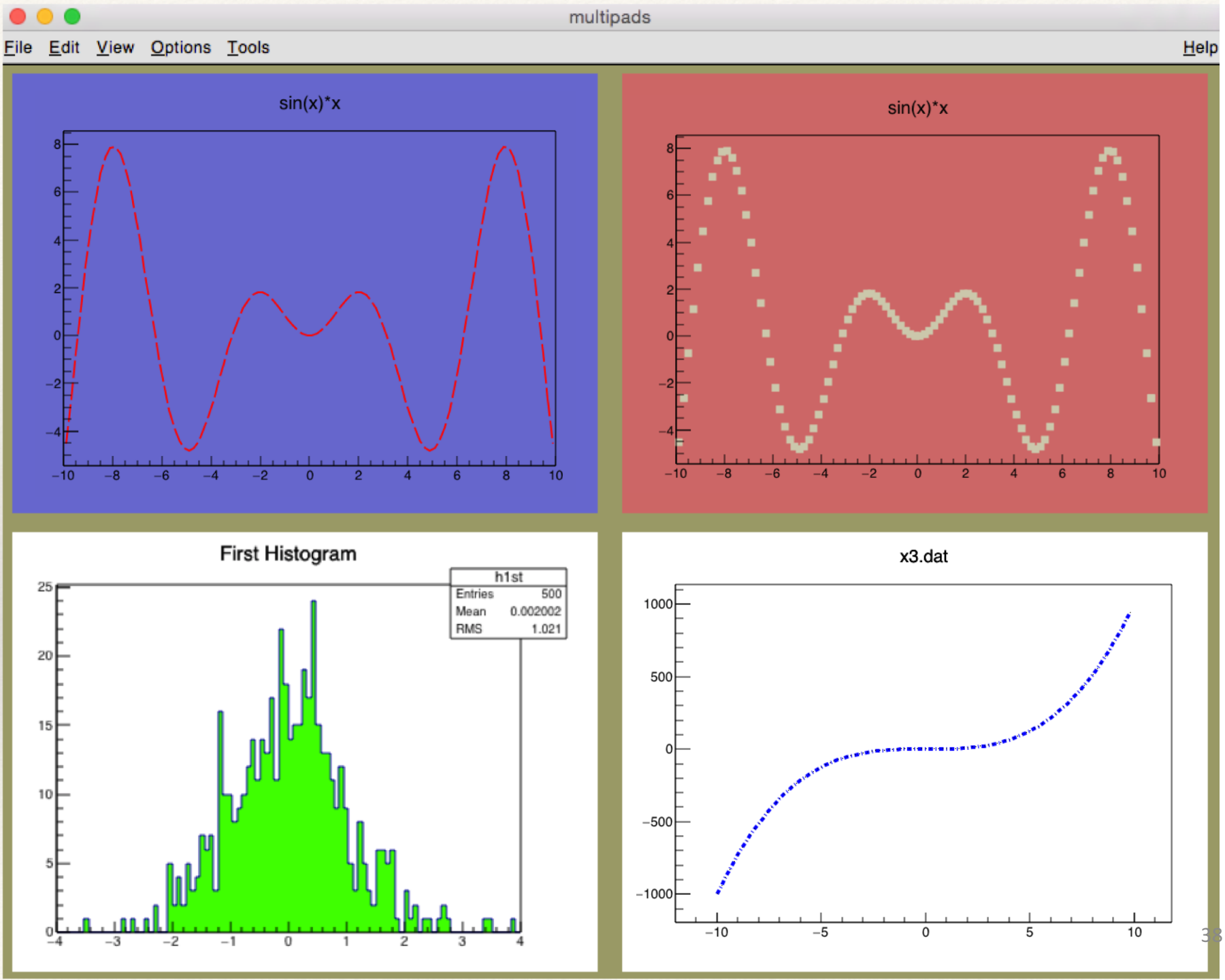
Exercise : Gaussian histogram

- Create a 1D Histogram
- Fill with Gaussian (As we did in previous lectures)
- Draw this histogram for pad 3


multipads
File Edit View Options Tools Help
sin(x)*x
sin(x)*x
First Histogram
h1st
Entries 500
Mean 0.002002
RMS 1.021

Add plot in pad 4

```
//Lets create a Graph

//cubicdata.dat file contains two column data; try to plot x vs y;

//Create a graph and some make up for the graph
TGraph * g1 = new TGraph("x3.dat");
g1->SetLineColor(kBlue);
g1->SetLineWidth(4);
g1->SetLineStyle(5);

//change your pad; lets plot it to the second pad
c1->cd(4);

//then draw
// g1->Draw("AL");
// A option draw the axes

// Even if you run g1->Draw(); you get the same result, since these are the
default options
}
```


multipads
File Edit View Options Tools Help
sin(x)*x
sin(x)*x
First Histogram
h1st
Entries 500
Mean 0.002002
RMS 1.021
x3.dat

To save your canvas to pdf,ps,img ……

c1->SaveAs(“first macro .pdf”)

```
//To be able to save your canvas as pdf,ps,jpeg....
c1->SaveAs("first macro.pdf");
```

Creating Legends

https://root.cern.ch/root/html/TLegend.html

Legends

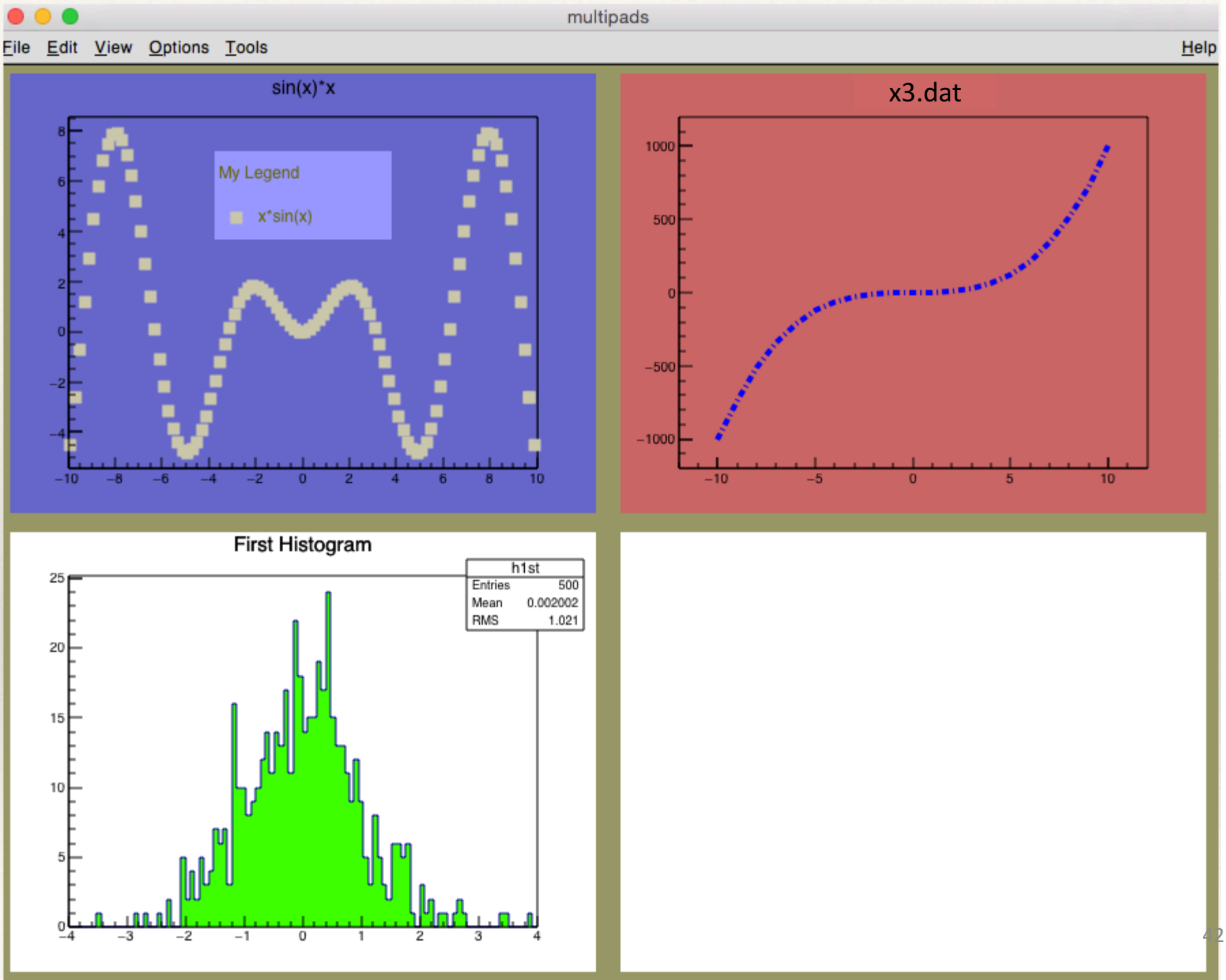
- ***TLegend*** – key to the lines on a plot
- E.g. for a two-line histo (**h1** and **h2**):

```
TLegend *myLegend=new TLegend(0.4,0.5,0.7,0.7,"My Legend");
                        //x1,y1,x2,y2,header
myLegend -> SetTextSize(0.04);
myLegend->AddEntry(&h2, "Energy B", "l");  //first arg must be pointer
myLegend->AddEntry(&h1, "Energy A", "l");
myLegend->Draw();
```

- "l" makes ROOT put a line in the entry

Example: Legend

```
 TLegend * l1 = new TLegend(0.4,0.5,0.7,0.7,"My Legend");
 //x1,y1,x2,y2,header (options of TLegend)

// You can easily arrange of the location of the legend using gui

//some options for legend

 l1->SetTextSize(0.04);
 l1->SetTextAlign(12);
 l1->SetFillColor(kBlue-9);
 l1->SetTextColor(kYellow+3);
 l1->SetBorderSize(0);
//what you need to write as a legend, p option for the point l option for the
line
 l1->AddEntry(f1,"x*sin(x)","p");
 l1->Draw();


 //To be able to save your canvas as pdf,ps,jpeg....
 c1->SaveAs("first macro.pdf");

}
```


multipads
File Edit View Options Tools Help
sin(x)*x
My Legend
x*sin(x)
x3.dat
First Histogram
h1st
Entries 500
Mean 0.002002
RMS 1.021

Saving Objects

Given a TFile:

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

Write an object deriving from TObject:

```
object->Write("optionalName")
```

"optionalName" or `TObject::GetName()`

Write any object (with dictionary):

```
f->WriteObject(object, "name");
```

TFile owns histograms, graphs, trees
(due to historical reasons):

```
TFile* f = new TFile("myfile.root");
TH1F* h = new TH1F("h","h",10,0.,1.);
h->Write();
TCanvas* c = new TCanvas();
c->Write();
delete f;
```

h automatically deleted: owned by file.

c still there. → *names unique!*

TFile acts like a scope for hists, graphs, trees!

Risks With I/O

Physicists can loop a lot:

For each particle collision

For each particle created

For each detector module

Do something.

Physicists can loose a lot:

Run for hours…

Crash.

Everything lost.

Reading a variable from root file

Reading is simple:

```
TFile* f = new TFile("myfile.root");
TH1F* h = 0;
f->GetObject("h", h);
h->Draw();
delete f;
```

Remember:

TFile owns histograms!

file gone, histogram gone!

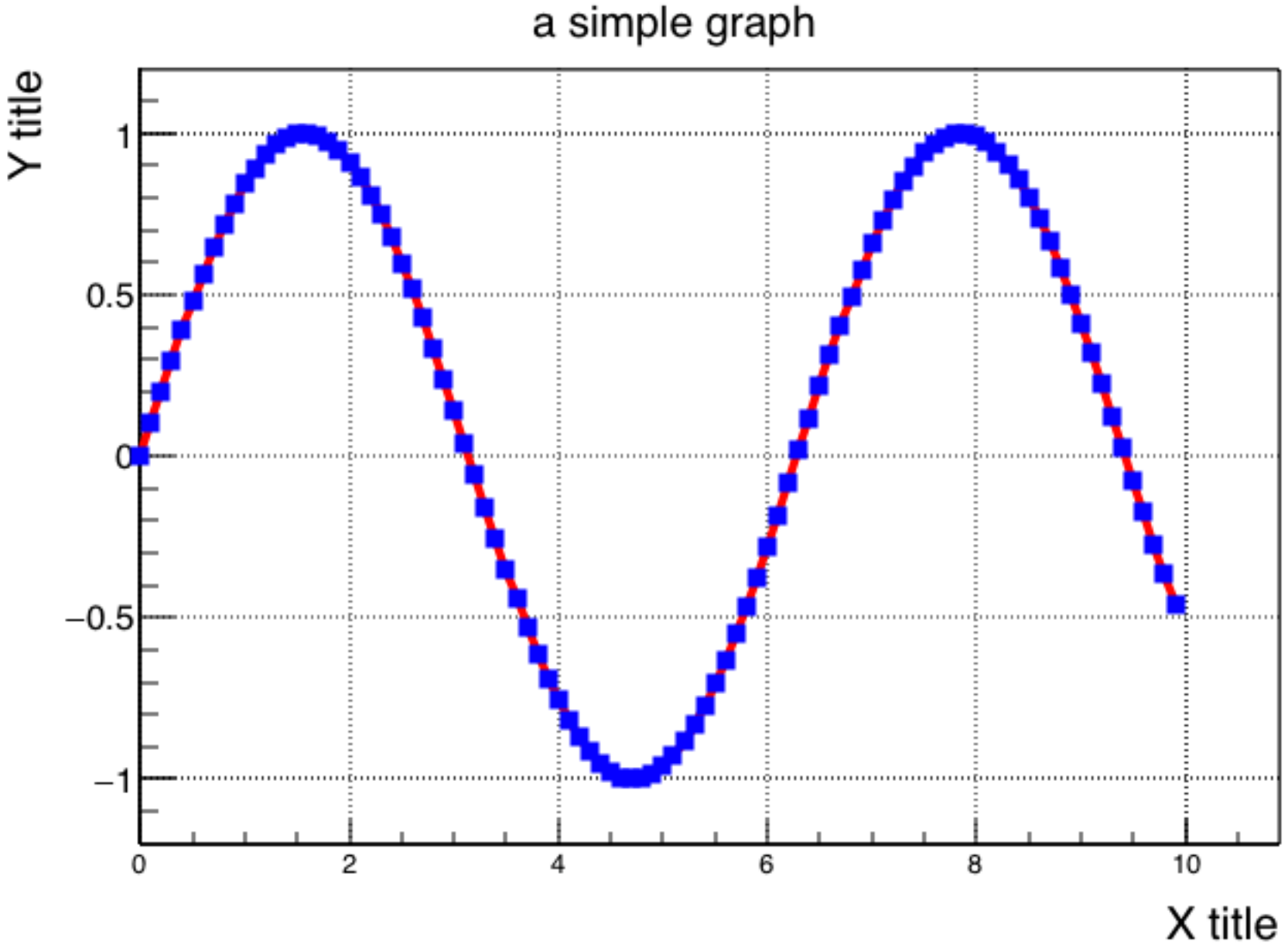
Exercise : TGraph

- Draw points on the sine function every 0.1 radian in 0<x<10 using TGraph.

Exercise : TGraph

```
{
   TCanvas *c1 = new TCanvas("c1","A Simple Graph Example",200,10,700,500);

   c1->SetGrid();

   const Int_t n = 100;
   Double_t x[n], y[n];
   for (Int_t i=0;i<n;i++) {
     x[i] = i*0.1;
     y[i] = sin(x[i]);
     printf(" i %i %f %f \n",i,x[i],y[i]);
   }
   TGraph *gr = new TGraph(n,x,y);
   gr->SetLineColor(2);
   gr->SetLineWidth(4);
   gr->SetMarkerColor(4);
   gr->SetMarkerStyle(21);
   gr->SetTitle("a simple graph");
   gr->GetXaxis()->SetTitle("X title");
   gr->GetYaxis()->SetTitle("Y title");
   gr->Draw("ACP");

   // TCanvas::Update() draws the frame, after which one can change it
   c1->Update();
   c1->GetFrame()->SetBorderSize(12);
   c1->Modified();
}
```

Exercise : TGraph



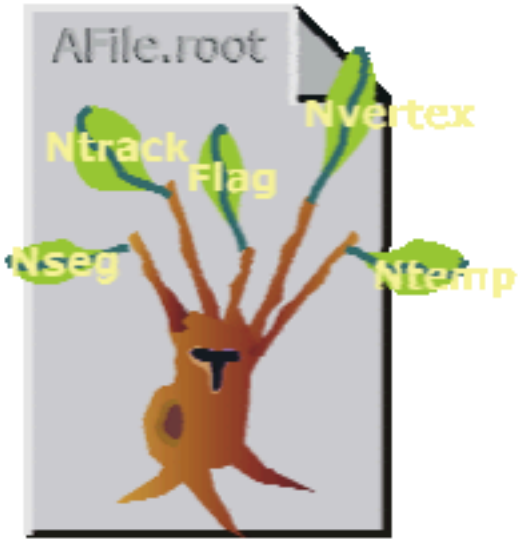
ROOT Trees

Ntuples and Trees

- Ntuples
  - support PAW-like ntuples and functions
  - PAW ntuples/histograms can be imported
- Trees
  - Extension of Ntuples for Objects
  - Collection of branches (branch has its own buffer)
  - Can input partial Event
  - Can have several Trees in parallel
- Chains = collections of Trees

Why Trees ?

- Any object deriving from TObject can be written to a file with an associated key with object.Write()
- However each key has an overhead in the directory structure in memory (about 60 bytes). Object.Write is very convenient for objects like histograms, detector objects, calibrations, but not for event objects.

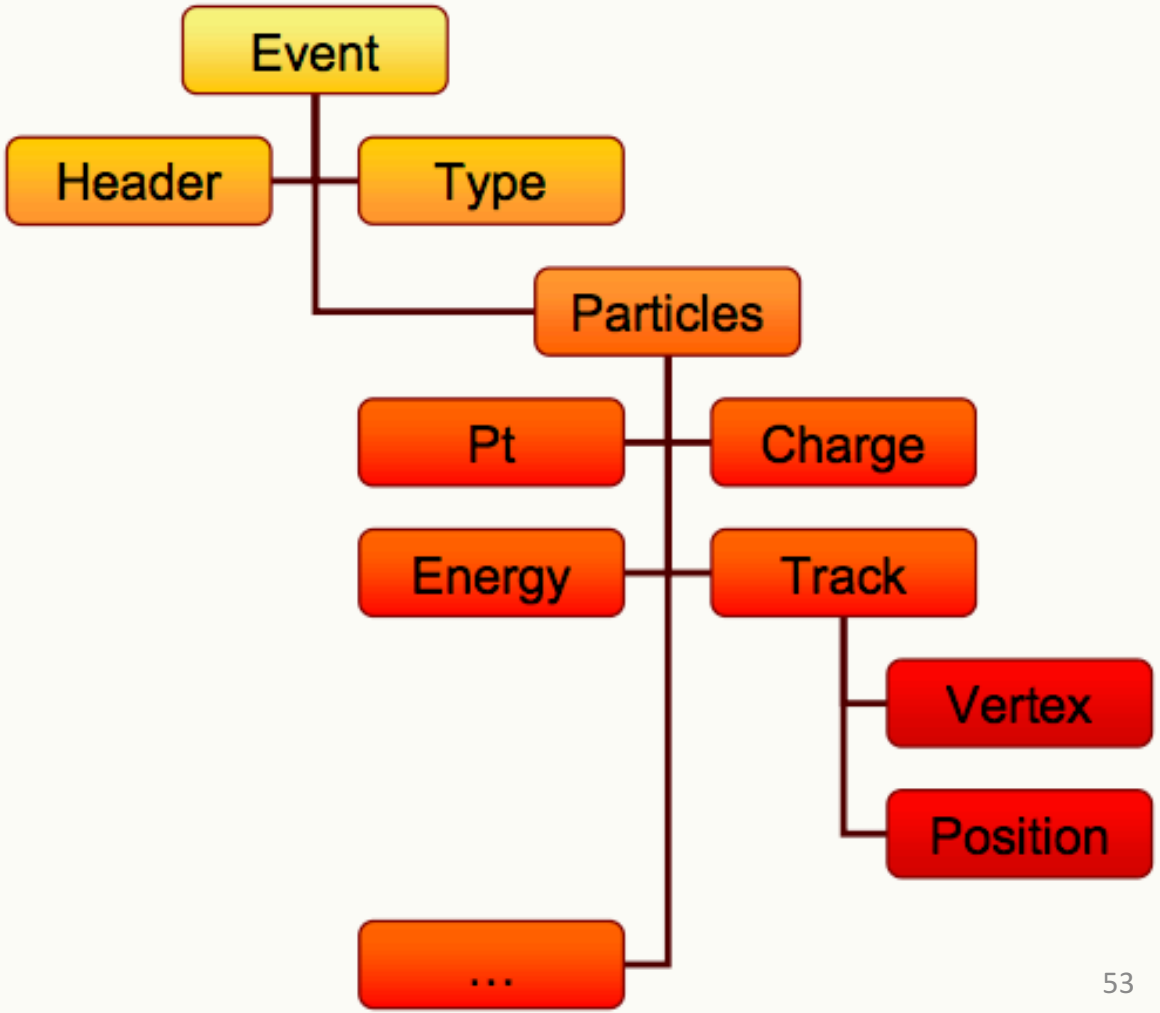


Trees

From:
Simple data types
(e.g. Excel tables)

x	y	z
-1.10228	-1.79939	4.452822
1.867178	-0.59662	3.842313
-0.52418	1.868521	3.766139
-0.38061	0.969128	1.084074
0.552454	-0.21231	0.350281
-0.18495	1.187305	1.443902
0.205643	-0.77015	0.635417
1.079222	-0.32739	1.271904
-0.27492	-1.72143	3.038899
2.047779	-0.06268	4.197329
-0.45868	-1.44322	2.293266
0.304731	-0.88464	0.875442
-0.71234	-0.22239	0.556881
-0.27187	1.181767	1.470484
0.886202	-0.65411	1.213209
-2.03555	0.527648	4.421883

To:
Complex data types
(e.g. Database tables)



Why Trees ?

- Trees have been designed to support very large collections of objects. The overhead in memory is in general less than 4 bytes per entry.
- Trees allow direct and random access to any entry (sequential access is the best)
- Trees have branches and leaves. One can read a subset of all branches. This can speed-up considerably the data analysis processes.

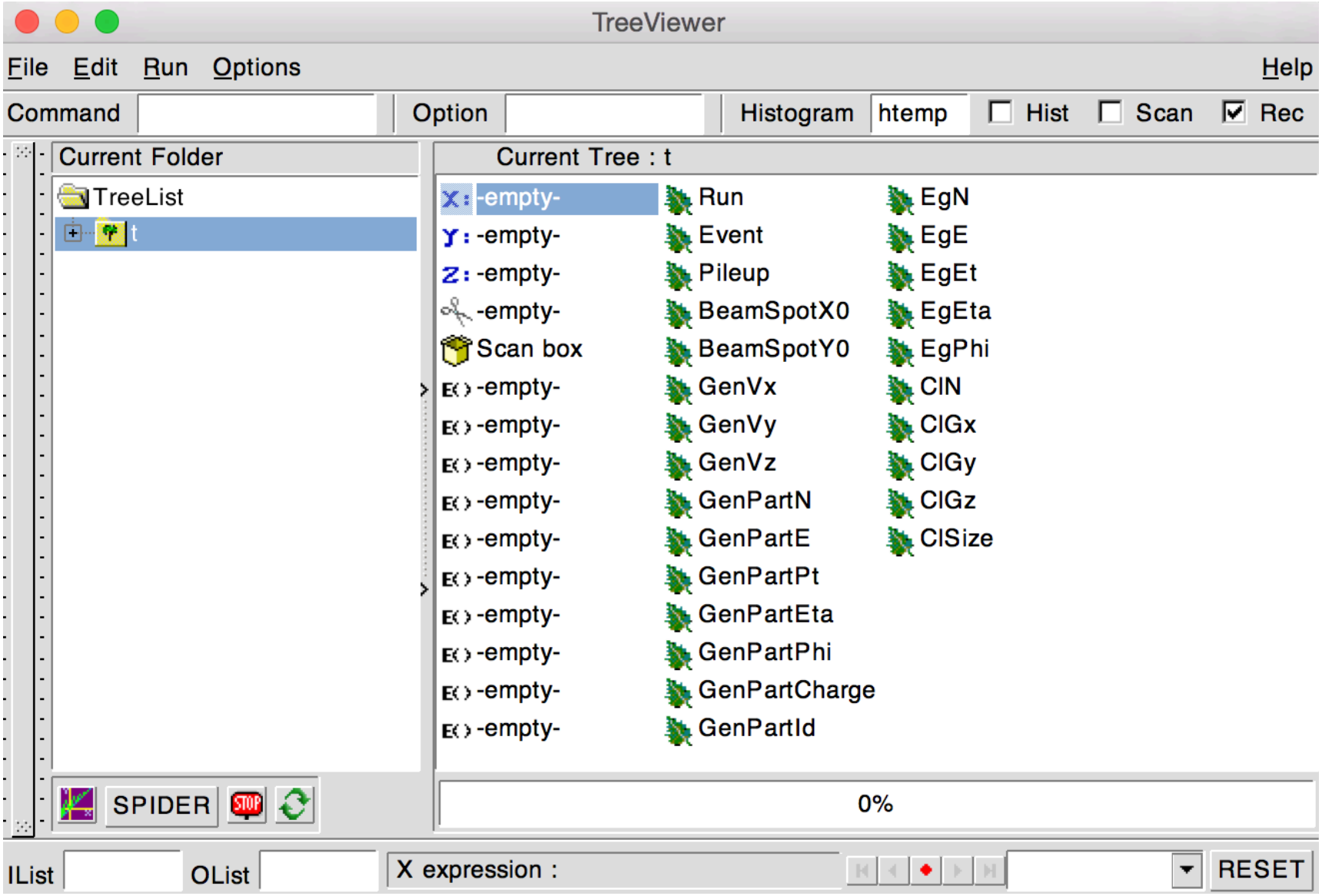
Why Trees ?

- PAW ntuples are a special case of Trees.
- Trees are designed to work with complex event objects.
- High level functions like TTree::Draw loop on all entries with selection expressions.
- Trees can be browsed via TBrowser
- Trees can be analized via TTreeViewer

The PROOF system is designed to process chains of Trees in parallel in a GRID environment

Tree structure

- Branches: directories
- Leaves: data containers
- Can read a subset of all branches – speeds up considerably the data analysis processes
- Tree layout can be optimized for data analysis
- The class for a branch is called **TBranch**
- Variables on **TBranch** are called leaf (yes - **TLeaf**)
- Branches of the same **TTree** can be written to separate files

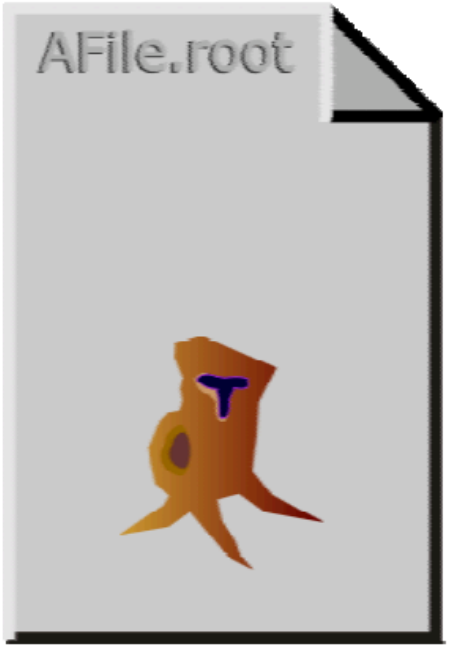

TreeViewer
File Edit Run Options Help
Command
Option
Histogram htemp Hist Scan Rec
Current Folder
TreeList
t
Current Tree : t
X: -empty-
Y: -empty-
Z: -empty-
-empty-
Scan box
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
-empty-
Run
Event
Pileup
BeamSpotX0
BeamSpotY0
GenVx
GenVy
GenVz
GenPartN
GenPartE
GenPartPt
GenPartEta
GenPartPhi
GenPartCharge
GenPartId
EgN
EgE
EgEt
EgEta
EgPhi
ClN
ClGx
ClGy
ClGz
ClSize
SPIDER
STOP
0%
IList
OList
X expression :
RESET

Create a TTree Object

A tree is a list of branches.

The TTree Constructor:

- Tree Name (e.g. "myTree")
- Tree Title

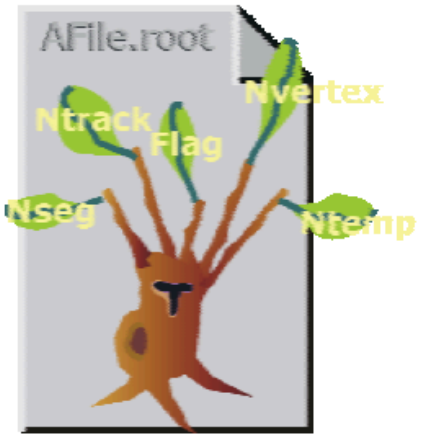



```
TTree *tree = new TTree("T","A ROOT tree");
```

Adding a Branch

Many Branch constructors Only a few shown here

- Branch name
- Class name
- Address of the pointer to the Object (descendant of TObject)
- Buffer size (default = 32,000)
- Split level (default = 1)

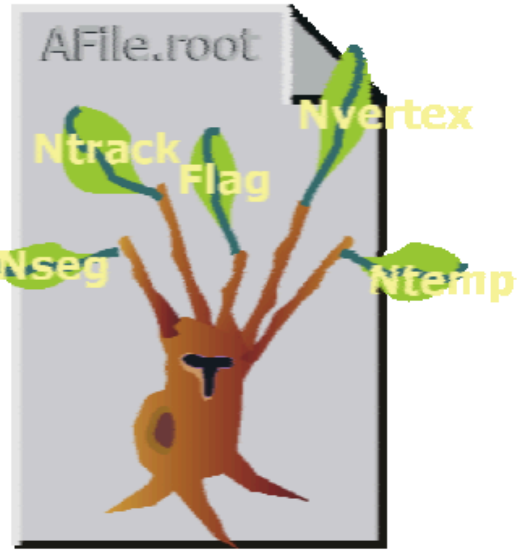



```
Event *event = new  Event();
myTree->Branch("eBranch","Event",&event,64000,1);
```

Splitting a Branch

Setting the split level (default = 1)

Split level = 0



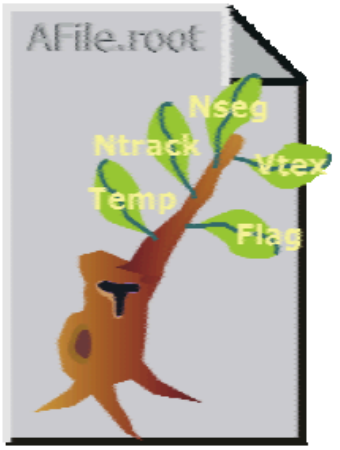
Split level = 1

Example:

```
tree->Branch("EvBr","Event",&ev,64000,0);
```

Adding Branches with a List of Variables

- Branch name
- Address: the address of the first item of a structure.
- Leaflist: all variable names and types
- Order the variables according to their size



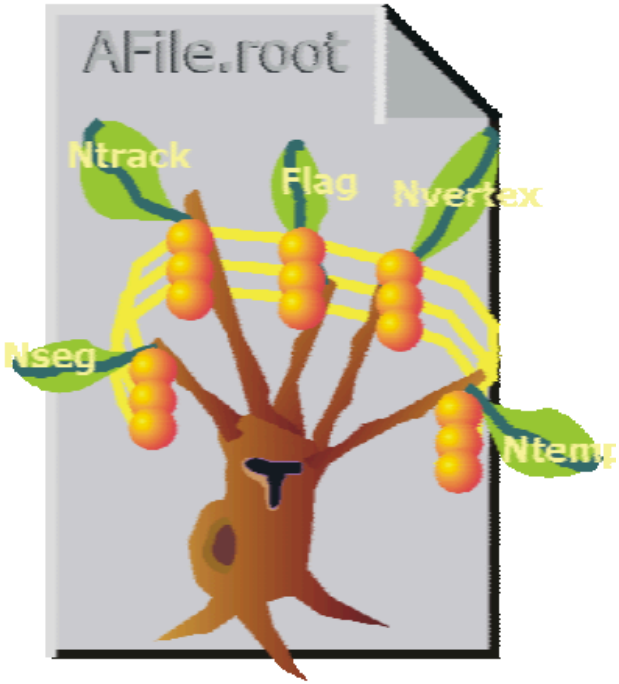
Example

```
TBranch *b = tree->Branch ("Ev_Branch",&event,
      "ntrack/I:nseg:nvtex:flag/i:temp/F");
```

Filling the Tree

- Create a for loop
- Create Event objects.
- Call the Fill method for the tree.

```
myTree->Fill()
```

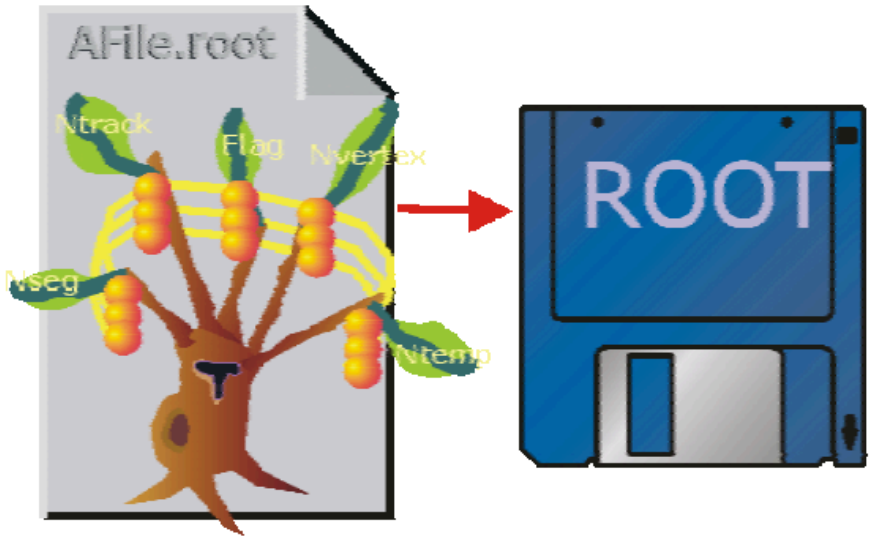


Write the Tree header

The Tree header contains a description of the Tree

- It owns the collection of branches
- Each branch has a buffer (TBasket) partially filled
- TTree::Write writes one single record on the file

```
tree->Write();
```



Exercise: Tree created

✓ Download tree.C file from LMS

```
#include "TROOT.h"
#include "TFile.h"
#include "TTree.h"
#include "TBrowser.h"
#include "TH2.h"
#include "TRandom.h"

void tree1w()
{
   //create a Tree file tree1.root

   //create the file, the Tree and a few branches
   TFile f("tree1.root","recreate");
   TTree *t1 = new TTree("t1","a simple Tree with simple variables");
   Float_t px, py, pz;
   Double_t random;
   Int_t ev;
   t1->Branch("px",&px,"px/F");
   t1->Branch("py",&py,"py/F");
   t1->Branch("pz",&pz,"pz/F");
   t1->Branch("random",&random,"random/D");
   t1->Branch("ev",&ev,"ev/I");

   //fill the tree
   for (Int_t i=0;i<10000;i++) {
     gRandom->Rannor(px,py);
     pz = (px + py)/2.;
     random = gRandom->Rndm();
     ev = i;
     t1->Fill();
  }
```

Exercise: Tree created

```
38 void treer()
39 {
40    //read the Tree generated by treew and fill two histograms
41
42    //note that we use "new" to create the TFile and TTree objects !
43    //because we want to keep these objects alive when we leave this function.
44    TFile *f = new TFile("tree.root");
45    TTree *t1 = (TTree*)f->Get("t1");
46    Float_t px, py, pz;
47    Double_t random;
48    Int_t ev;
49    t1->SetBranchAddress("px",&px);
50    t1->SetBranchAddress("py",&py);
51    t1->SetBranchAddress("pz",&pz);
52    t1->SetBranchAddress("random",&random);
53    t1->SetBranchAddress("ev",&ev);
54
55    //create two histograms
56    TH1F *hpx   = new TH1F("hpx","px distribution",100,-3,3);
57    TH2F *hpxpy = new TH2F("hpxpy","py vs px",30,-3,3,30,-3,3);
58
59    //read all entries and fill the histograms
60    Long64_t nentries = t1->GetEntries();
61    for (Long64_t i=0;i<nentries;i++) {
62      t1->GetEntry(i);
63      hpx->Fill(px);
64      hpxpy->Fill(px,py);
65   }
66
67   //we do not close the file. We want to keep the generated histograms
68   //we open a browser and the TreeViewer
69   if (gROOT->IsBatch()) return;
70   new TBrowser();
71   t1->StartViewer();
72   // in the browser, click on "ROOT Files", then on "tree.root".
73   //     you can click on the histogram icons in the right panel to draw them.
74   // in the TreeViewer, follow the instructions in the Help button.
75 }
```

```
77 void tree() {
78    treew();
79    treer();
80 }
```

Exercise: Data Analysis with TTree

1. Open a root file
2. Create source code (test.C) and its header file (test.h)

```
(143) ~ ➜ ››› root tree.root
root [0]
Attaching file tree.root as _file0...
(TFile *) 0x421ba10
root [1] TTree *myTree = (TTree*)_file0->Get("t1");
root [2] myTree->MakeClass("test");
Info in <TTreePlayer::MakeClass>: Files: test.h and test.C generated from TTree: t1
root [3] .q
```

3. Now ready to do your own programing.

In the source code (test.C)

Event Loop: Insert your algorithm here

```
#define test_cxx
#include "test.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void test::Loop()
{
//   In a ROOT session, you can do:
//      root> .L test.C
//      root> test 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 where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       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(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();

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

In the hear file (test.h)

```
1  //////////////////////////////////////////////////////////
2  // This class has been automatically generated on
3  // Tue Sep 12 14:33:47 2017 by ROOT version 6.06/01
4  // from TTree t1/a simple Tree with simple variables
5  // found on file: tree.root
6  //////////////////////////////////////////////////////////
7
8  #ifndef test_h
9  #define test_h
10
11 #include <TROOT.h>
12 #include <TChain.h>
13 #include <TFile.h>
14
15 // Header file for the classes stored in the TTree if any.
16
17 class test {
18 public :
19    TTree          *fChain;   //!pointer to the analyzed TTree or TChain
20    Int_t           fCurrent; //!current Tree number in a TChain
21
22 // Fixed size dimensions of array or collections stored in the TTree if any.
23
24    // Declaration of leaf types
25    Float_t         px;
26    Float_t         py;
27    Float_t         pz;
28    Double_t        random;
29    Int_t           ev;
30
31    // List of branches
32    TBranch        *b_px;   //!
33    TBranch        *b_py;   //!
34    TBranch        *b_pz;   //!
35    TBranch        *b_random;   //!
36    TBranch        *b_ev;   //!
37
38    test(TTree *tree=0);
39    virtual ~test();
40    virtual Int_t    Cut(Long64_t entry);
41    virtual Int_t    GetEntry(Long64_t entry);
42    virtual Long64_t LoadTree(Long64_t entry);
43    virtual void     Init(TTree *tree);
44    virtual void     Loop();
45    virtual Bool_t   Notify();
46    virtual void     Show(Long64_t entry = -1);
47 };
48
49 #endif
```

In the hear file (test.h)

```
51 #ifdef test_cxx
52 test::test(TTree *tree) : fChain(0)
53 {
54 // if parameter tree is not specified (or zero), connect the file
55 // used to generate this class and read the Tree.
56    if (tree == 0) {
57       TFile *f = (TFile*)gROOT->GetListOfFiles()->FindObject("tree.root");
58       if (!f || !f->IsOpen()) {
59          f = new TFile("tree.root");
60       }
61       f->GetObject("t1",tree);
62
63    }
64    Init(tree);
65 }
66
67 test::~test()
68 {
69    if (!fChain) return;
70    delete fChain->GetCurrentFile();
71 }
72
73 Int_t test::GetEntry(Long64_t entry)
74 {
75 // Read contents of entry.
76    if (!fChain) return 0;
77    return fChain->GetEntry(entry);
78 }
79 Long64_t test::LoadTree(Long64_t entry)
80 {
81 // Set the environment to read one entry
82    if (!fChain) return -5;
83    Long64_t centry = fChain->LoadTree(entry);
84    if (centry < 0) return centry;
85    if (fChain->GetTreeNumber() != fCurrent) {
86       fCurrent = fChain->GetTreeNumber();
87       Notify();
88    }
89    return centry;
90 }
```

In the hear file (test.h)

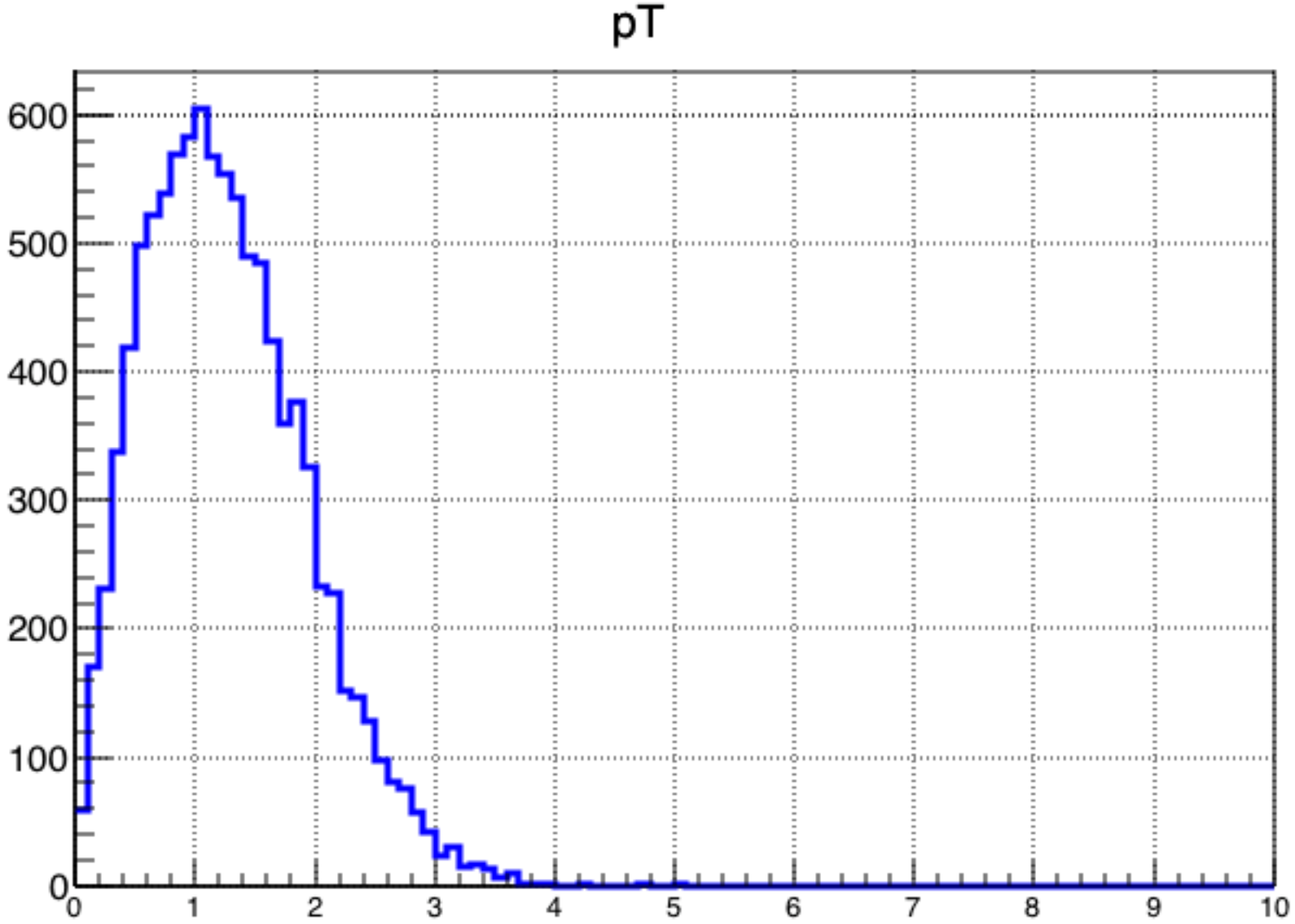
```
92 void test::Init(TTree *tree)
93 {
94    // The Init() function is called when the selector needs to initialize
95    // a new tree or chain. Typically here the branch addresses and branch
96    // pointers of the tree will be set.
97    // It is normally not necessary to make changes to the generated
98    // code, but the routine can be extended by the user if needed.
99    // Init() will be called many times when running on PROOF
100   // (once per file to be processed).
101
102   // Set branch addresses and branch pointers
103   if (!tree) return;
104   fChain = tree;
105   fCurrent = -1;
106   fChain->SetMakeClass(1);
107
108   fChain->SetBranchAddress("px", &px, &b_px);
109   fChain->SetBranchAddress("py", &py, &b_py);
110   fChain->SetBranchAddress("pz", &pz, &b_pz);
111   fChain->SetBranchAddress("random", &random, &b_random);
112   fChain->SetBranchAddress("ev", &ev, &b_ev);
113   Notify();
114 }
115
116 Bool_t test::Notify()
117 {
118   // The Notify() function is called when a new file is opened. This
119   // can be either for a new TTree in a TChain or when when a new TTree
120   // is started when using PROOF. It is normally not necessary to make changes
121   // to the generated code, but the routine can be extended by the
122   // user if needed. The return value is currently not used.
123
124   return kTRUE;
125 }
126
127 void test::Show(Long64_t entry)
128 {
129 // Print contents of entry.
130 // If entry is not specified, print current entry
131   if (!fChain) return;
132   fChain->Show(entry);
133 }
134 Int_t test::Cut(Long64_t entry)
135 {
136 // This function may be called from Loop.
137 // returns  1 if entry is accepted.
138 // returns -1 otherwise.
139   return 1;
140 }
141 #endif // #ifdef test_cxx
```

Exercise: How to run a macro

- Run ROOT
  - % root
- Compile the macro
  - root [0] .L test.C+
- Define the Class name
  - root [1] test aaa
- Looping over the events
  - root [0] aaa.Loop()

```
root [0] .L test.C+
Info in <TMacOSXSystem::ACLiC>: creating shared library ./test_C.so
root [1] test aaa
(test &) @0x10ff20b90
root [2] aaa.Loop()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

Exercise: Result



Easy way with a reserved macro

- inside x_test.C
  - .L test.C+
  - test aaa
  - aaa.Loop()
- You can run it using below simple command
  - % root < x_test.C

Exercise: Tree created

- Draw a plot for Transverse Momentum (pT) of the particles with px, py in the root file.

Resetting the Environment

- gROOT->Reset()
  - Calls destructors of all objects created on the stack
  - Objects on Heap are not deleted, but pointer variable is disassociated

Global Pointers

- `gSystem` : Interface to the operating system.
- `gStyle` : Interface to the current graphics style.
- `gPad` : Interface to the current graphics Pad.
- `gROOT` : Entry point to the ROOT system.
- `gRandom`: Interface to the current random number generator.

Global Variables

- gRandom

```
gRandom->Gaus(1,2)
```

  You can replace the random generator with your own:

```
delete gRandom;
gRandom = new TRandom2(0); //seed=0
```

- gFile

```
gFile->GetName()
```

- gDirectory

```
gDirectory->GetName()
```

- gSystem

```
gSystem->HostName()
```

gROOT

- global ROOT session object
- gROOT->GetListOf< list type >();
- gROOT->LoadMacro();
- gROOT->Time();
- gROOT->ProcessLine()

Resetting the Environment

- gROOT->Reset()
  - Calls destructors of all objects created on the stack
  - Objects on Heap are not deleted, but pointer variable is disassociated

Instructional Materials

- https://cernbox.cern.ch/index.php/s/pdmPP3u5lByEqrN

- To set environment for ROOT framework
  - source rootset.tcshrc (tcsh)
  - source rootset.bashrc (Bash)