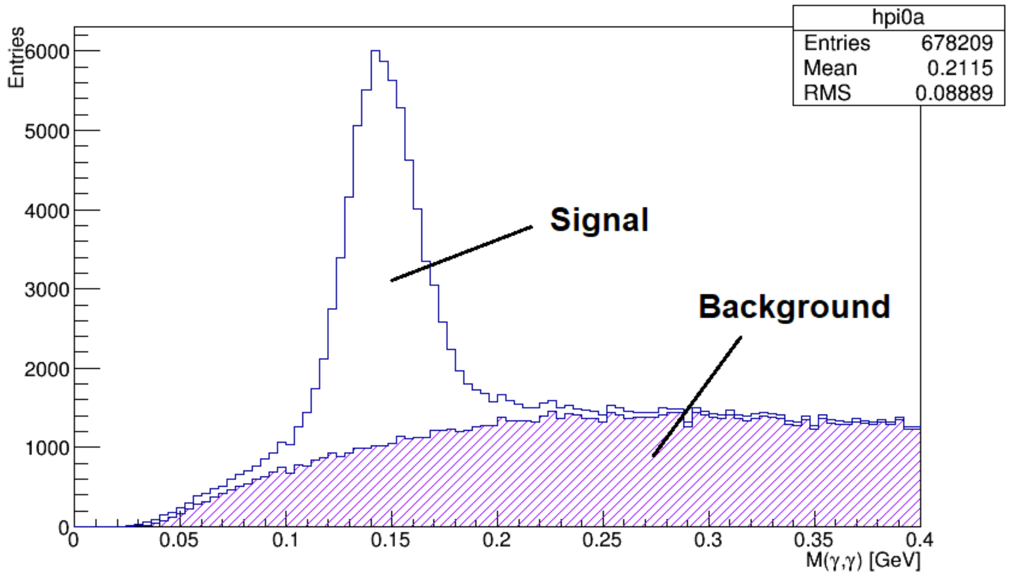
Particle Data Analysis in High Energy Physics

Lecture 8

Basics of ROOT



Ahmet Bingül

METU, Physics

Mar 2026

ROOT (Developed by HEP Community)

- ROOT
  is an “object oriented framework for data analysis”
  – read data from some source
  – write data (persistent objects)
  – selected data with some criteria
  – produce results as plots, numbers, fits, ...
- Integrates several tools like random number generations, fit methods (Minuit), Neural Network framework
- Supports “interactive” C++ and “compiled” C++ usage
- PyROOT is a Python extension module that allows the user to interact with any ROOT class from the Python interpreter.

Getting Started

ROOT Console

- Launch ROOT interactive console (CINT interpreter)

```
$ root
```

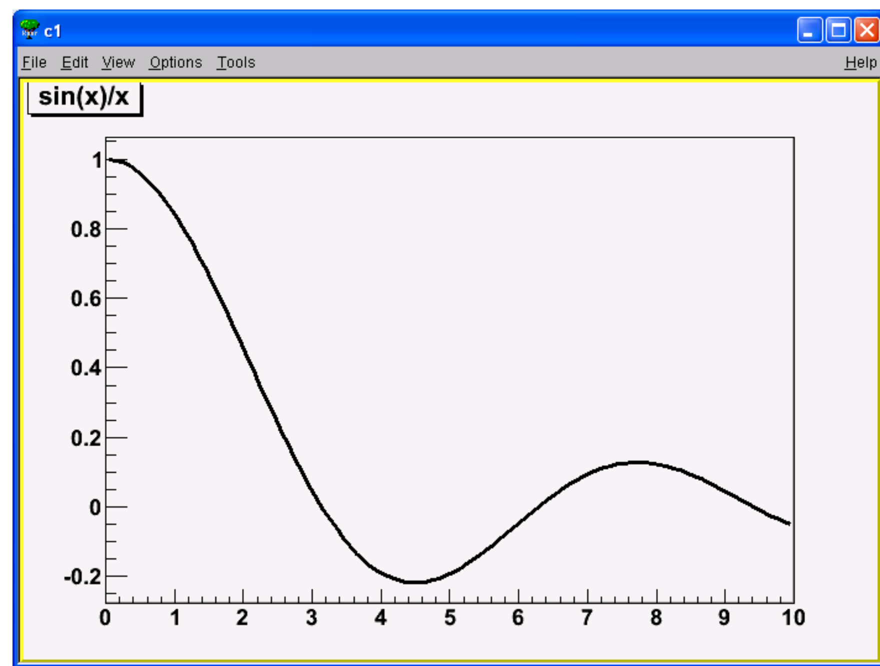
```
ROOT session
  ******************************************
  *                                        *
  *        W E L C O M E  to  R O O T      *
  *                                        *
  *   Version   5.25/02 29 September 2009  *
  *                                        *
  *  You are welcome to visit our Web site *
  *          http://root.cern.ch           *
  *                                        *
  ******************************************

ROOT 5.25/02 (trunk@30530, Sep 29 2009, 15:28:19 on win32)

CINT/ROOT C/C++ Interpreter version 5.17.00, Dec 21, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

ROOT Console

- Drawing sin(x)/x function between x = 0 and x = 10

ROOT session

```
  *******************************************
  *                                         *
  *        W E L C O M E  to  R O O T       *
  *                                         *
  *   Version   5.25/02 29 September 2009   *
  *                                         *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch            *
  *                                         *
  *******************************************

ROOT 5.25/02 (trunk@30530, Sep 29 2009, 15:28:19 on win32)

CINT/ROOT C/C++ Interpreter version 5.17.00, Dec 21, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] TF1 f("function","sin(x)/x",0.0,10.0)
root [1] f.Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [2] _
```



ROOT Macro Files

```
$ edit myfile.C
```

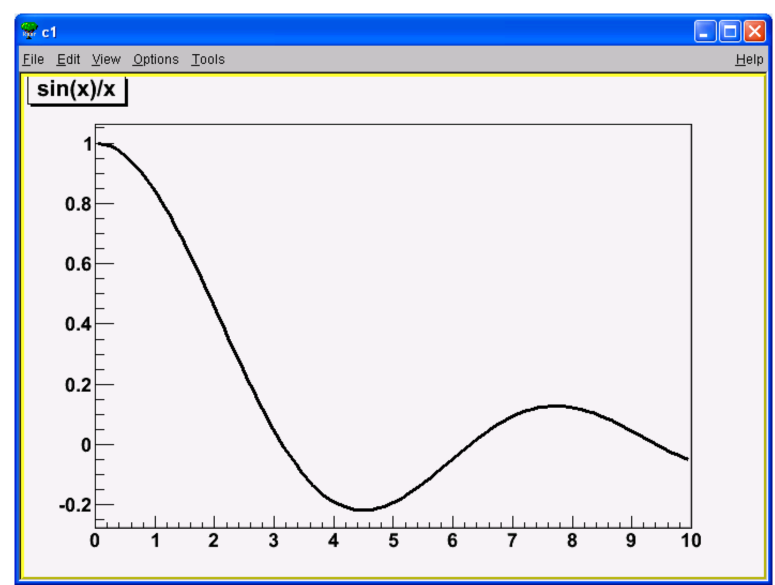
```
void myfile()
{
   auto *f = new TF1("fun","sin(x)/x",0,10);
   f->Draw();
}
```

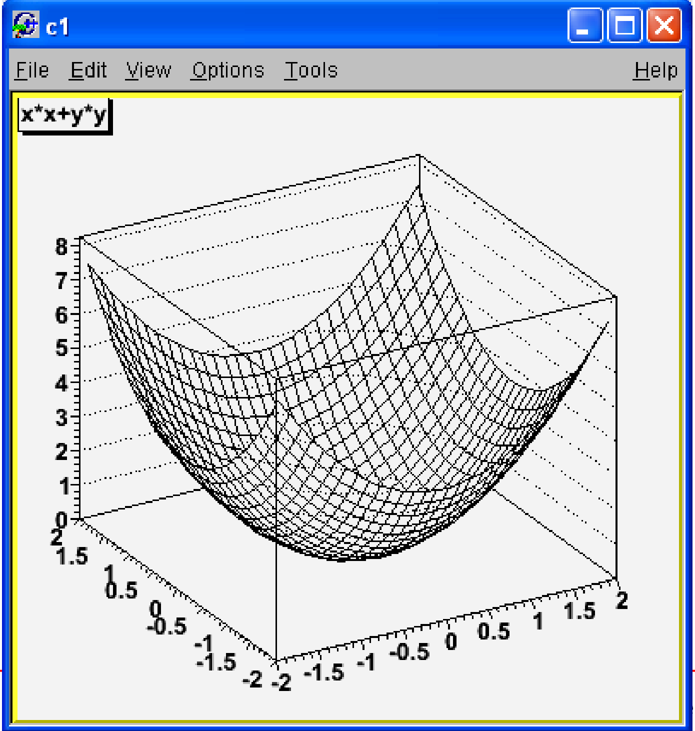
```
$ root myfile.C
```

```
$ edit fun2d.C
```

```
void fun2d()
{
   auto *f = new TF2("fun", "x*x+y*y",-2,2, -2,2);
   f->Draw();
}
```

```
$ root fun2d.C
```





Graphs

Graphs

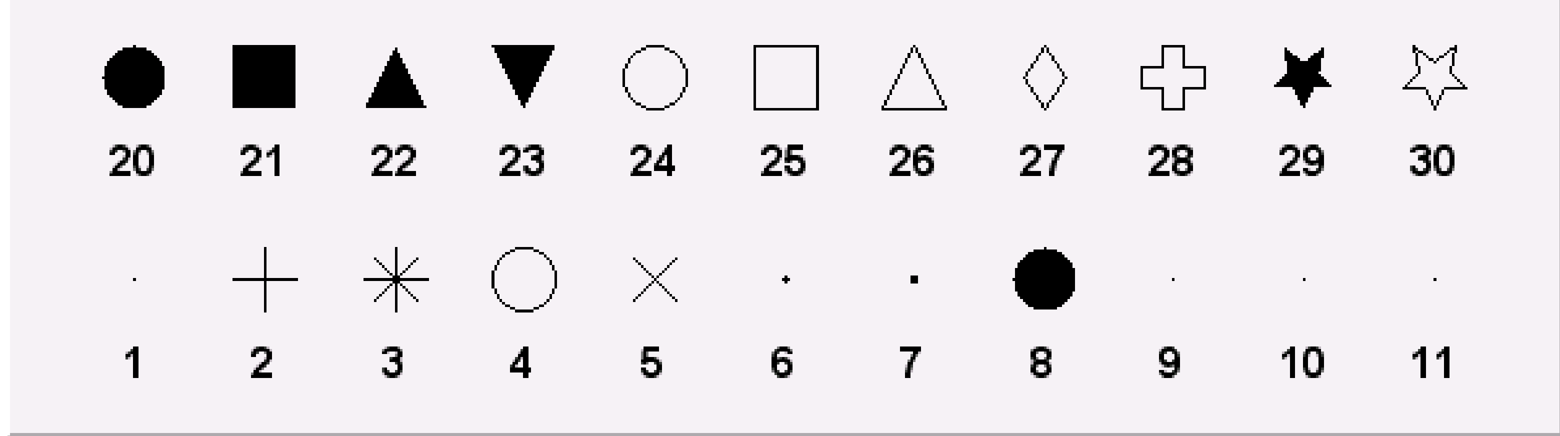
```cpp
void graph()
{
  const int n = 20;
  double x[n], y[n];

  for(int i=0; i<20; i++){
    x[i] = i*0.1;
    y[i] = x[i]*x[i] + 2.0;
  }

  TGraph *gr = new TGraph(n, x, y);
  gr->SetMarkerStyle(21);
  gr->SetMarkerColor(2);
  gr->SetMarkerSize(1.3);
  gr->Draw("APL");
}
```

```python
import ROOT
import numpy as np

n = 20
x = np.arange(0,2,0.1)
y = x**2+2

gr = ROOT.TGraph(n,x,y)

gr.SetMarkerStyle(21);
gr.SetMarkerColor(2);
gr.SetMarkerSize(1.3);
gr.Draw("APL");

input("Press Enter to continue...")
```

Some Graph Draw options

- `"L"` A simple poly-line between every points is drawn
- `"A"` Axis are drawn around the graph
- `"C"` A smooth curve is drawn
- `"*"` A star is plotted at each point
- `"P"` The current marker of the graph is plotted at each point
- `"B"` A bar chart is drawn at each point

Marker Styles



Colors

Histograms

Histograms

- Contain binned data
  probably the most important class in ROOT for the physicist
- Create a 1-dim Histo (float precision)

```
TH1F h("histo", "my histo; xtitle; ytitle", nbins, xmin, xmax);
```

  - `"histo"` is the name of the histogram
  - `"my histo; xtitle; ytitle"` are the title and x and y labels

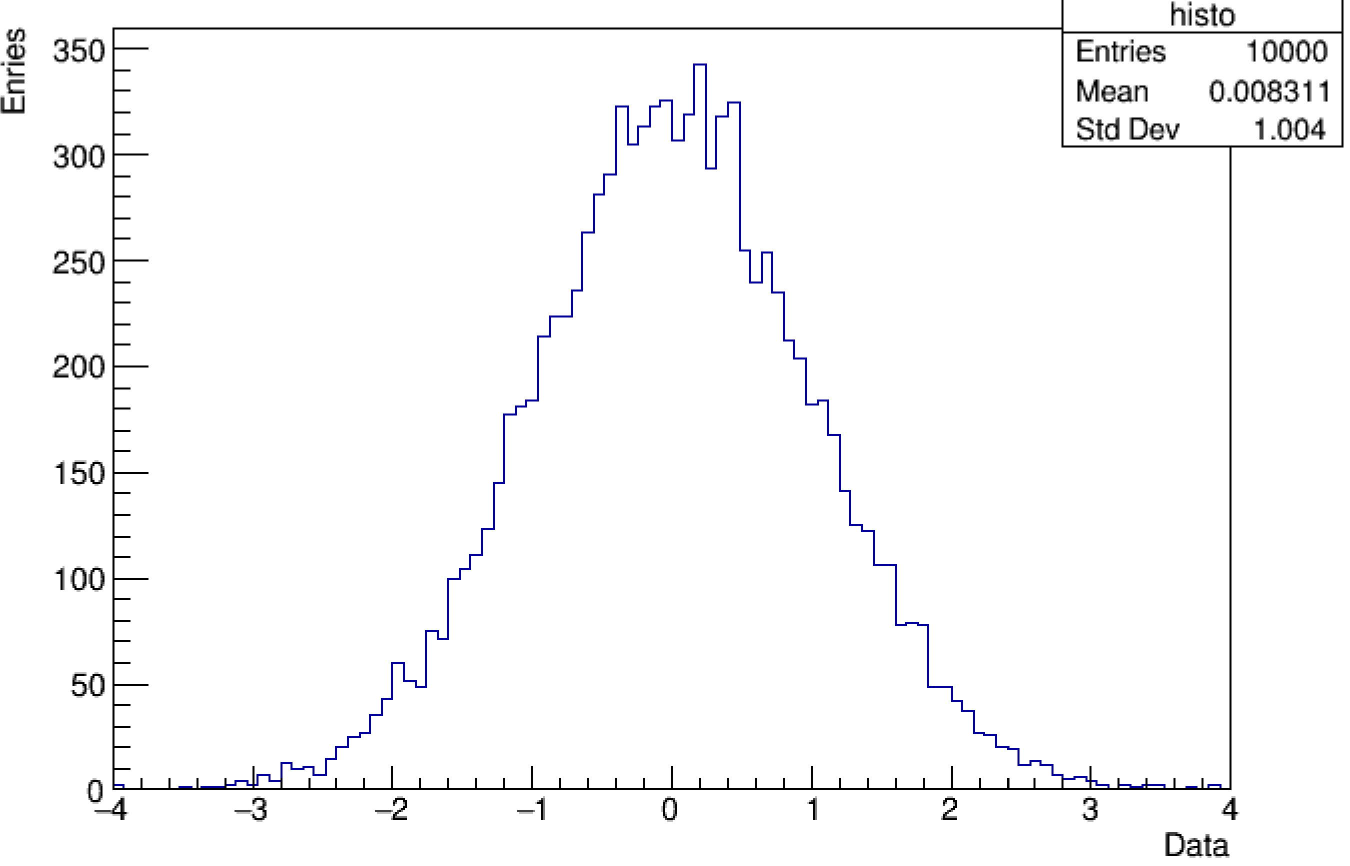
- Create a 1-dim Histo (double precision)

```
TH1D h("histo", "my histo; xtitle; ytitle", 20, 0 ,10);
```

Gaussian Distribution

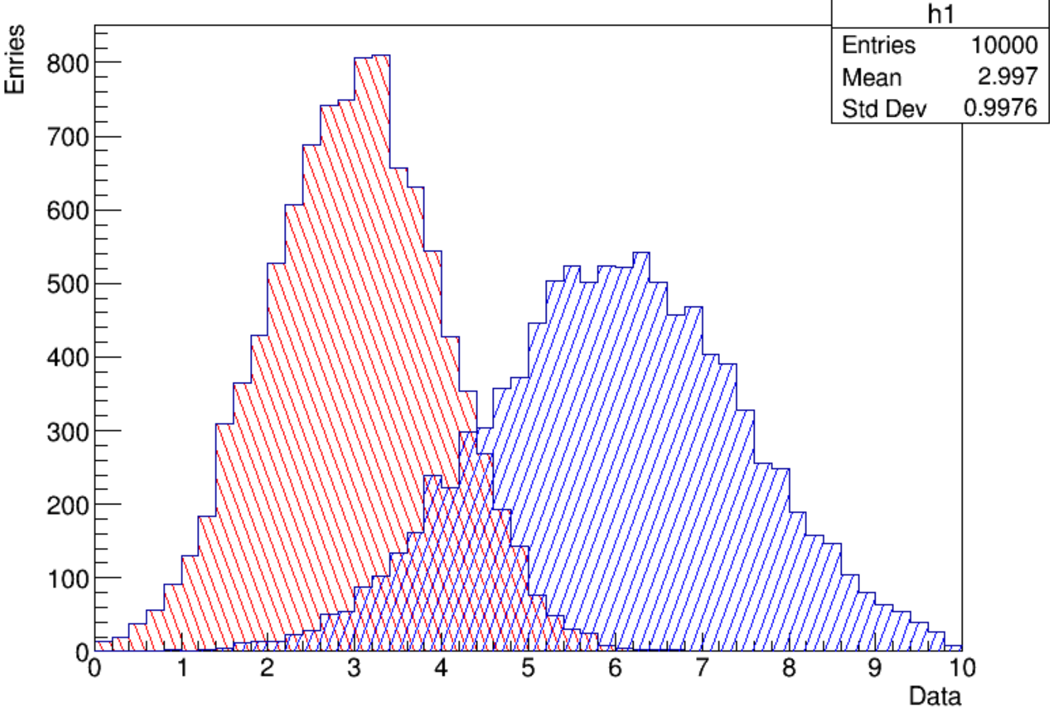
```
void hist(){
  TH1F *h = new TH1F("histo",
           ";Data;Enries",100,-4,4);

  h->FillRandom("gaus",10000);

  h->Draw();
  //h->Draw("error");
}
```

```
from ROOT import TH1F

h = TH1F("histo", \
         ";Data;Enries",100,-4,4)

h.FillRandom("gaus",10000)

h.Draw()
#h.Draw("error")

input("Press Enter to continue...")
```



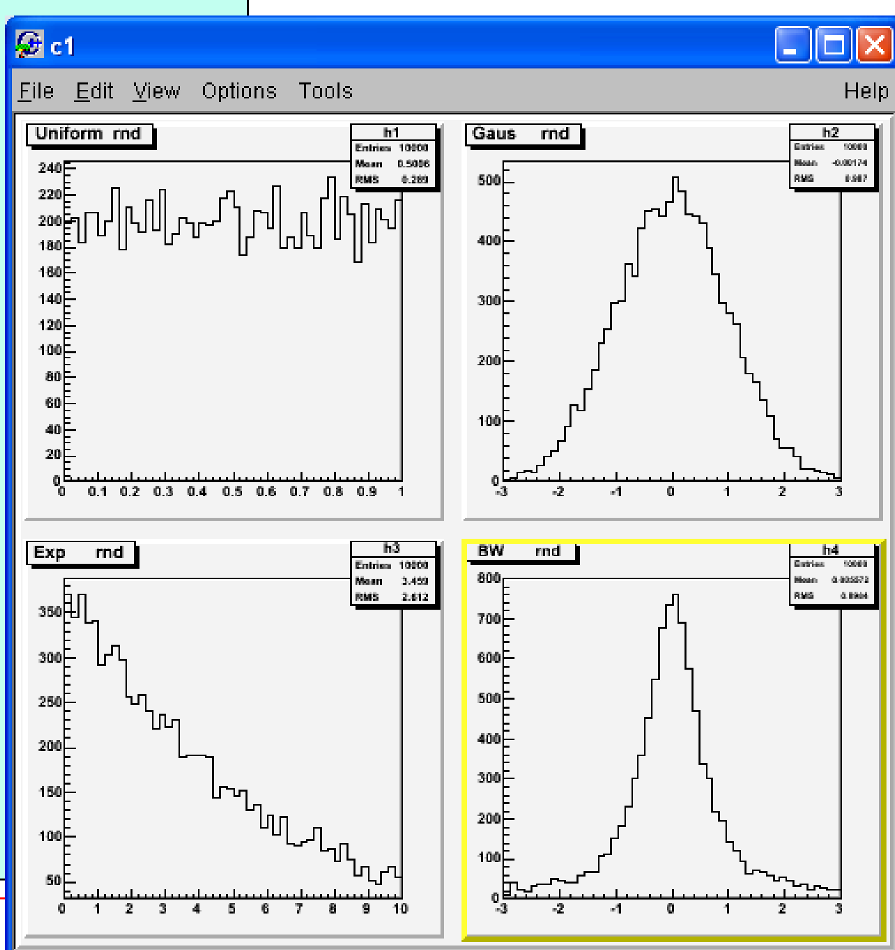
Superimposing Histograms

```
void test(){
  auto rnd = new Trandom();

  auto h1 = new TH1F("h1",";Data;Enries",50,0,10);
  auto h2 = new TH1F("h2",";Data;Enries",50,0,10);

  h1->SetFillStyle(3375);
  h1->SetFillColor(2);

  h2->SetFillStyle(3357);
  h2->SetFillColor(4);

  for(int i=0;i<10000;i++){
    double r1 = rnd->Gaus(3,1.0);
    double r2 = rnd->Gaus(6,1.5);
    h1->Fill(r1);
    h2->Fill(r2);
  }

  auto c1 = new TCanvas("c1","c1",800,600);

  h1->Draw();
  h2->Draw("same");
  c1->Print(«output.png")
}
```



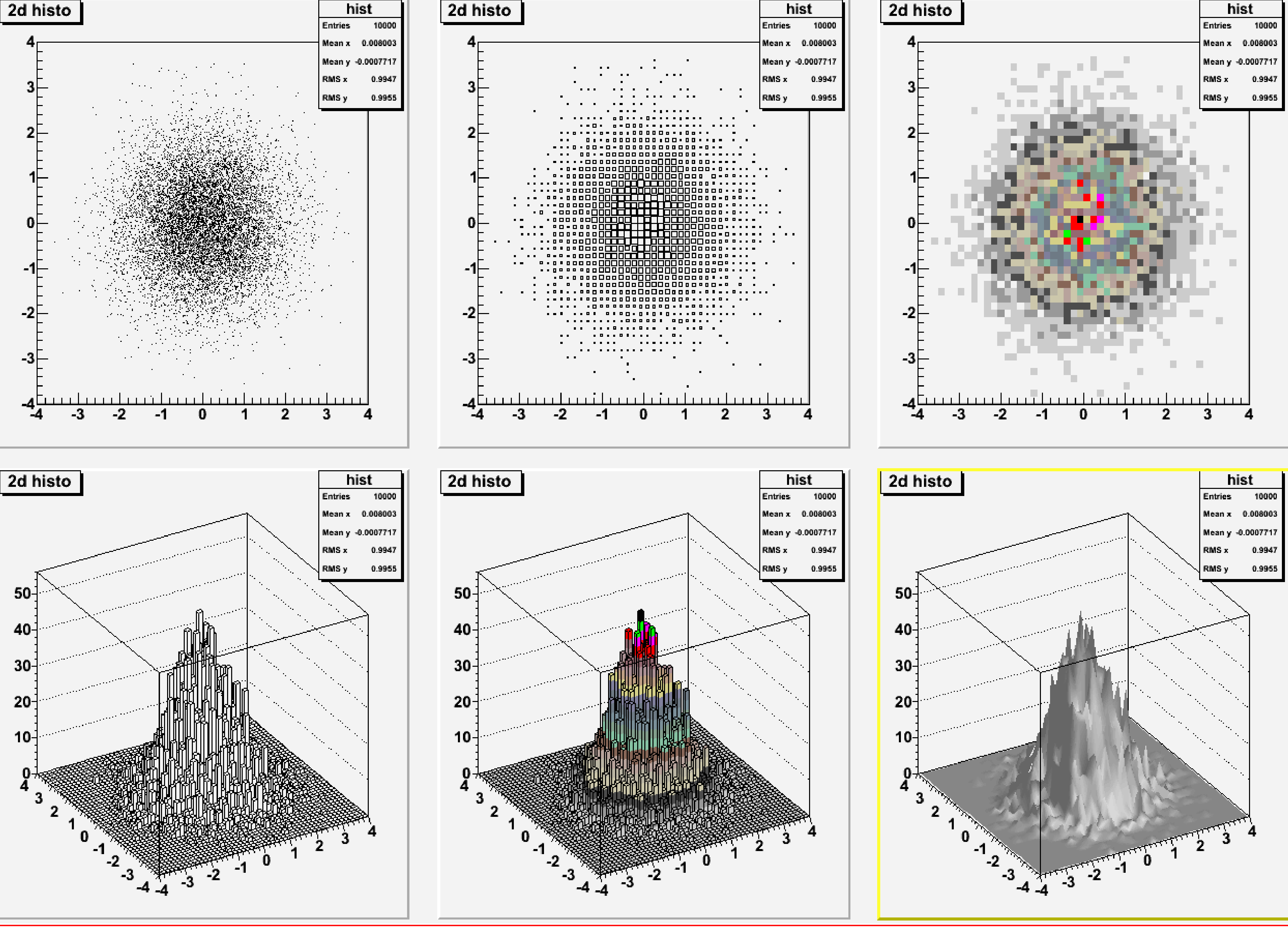
Dividing Canvas

```
void histo()
{
  auto     rnd = new TRandom3();
  TCanvas *c1 = new TCanvas("c1","c1",10,10,500,500);
  TH1F     *h1 = new TH1F("h1","Uniform  rnd",50, 0.0, 1.0);
  TH1F     *h2 = new TH1F("h2","Gaus     rnd",50,-3.0, 3.0);
  TH1F     *h3 = new TH1F("h3","Exp      rnd",50, 0.0,10.0);
  TH1F     *h4 = new TH1F("h4","BW       rnd",50,-3.0, 3.0);
  h1->SetMinimum(0);

  for(int i=0; i<10000; i++){
    h1->Fill( rnd->Uniform() );
    h2->Fill( rnd->Gaus(0,1) );
    h3->Fill( rnd->Exp(5) );
    h4->Fill( rnd->BreitWigner(0,1) );
  }

  c1->Divide(2,2);
  c1->cd(1); h1->Draw();
  c1->cd(2); h2->Draw();
  c1->cd(3); h3->Draw();
  c1->cd(4); h4->Draw();
}
```



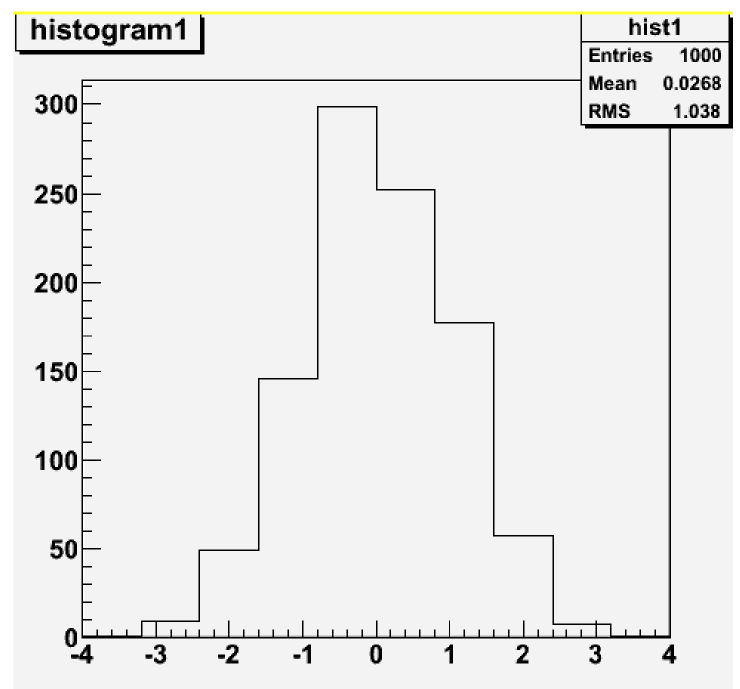
2D Histograms on Subpads

```
void hist()
{
  auto *rnd= new TRandom3();
  auto *c1 = new TCanvas("c1","c1",10,10,500,500);
  auto *h  = new TH2F("hist","2d histo", 50,-4,4, 50,-4,4);

  for(int i=0; i<10000; i++){
    double r1 = rnd->Gaus();
    double r2 = rnd->Gaus();
    h->Fill(r1, r2);
  }

  c1->Divide(3,2);

  c1->cd(1); h->Draw();          // default: scatter plot
  c1->cd(2); h->Draw("box");     // box plot
  c1->cd(3); h->Draw("col");     // colored

  c1->cd(4); h->Draw("lego");    // lego plot
  c1->cd(5); h->Draw("lego2");   // colored lego plot
  c1->cd(6); h->Draw("surf4");   // surface plot
}
```

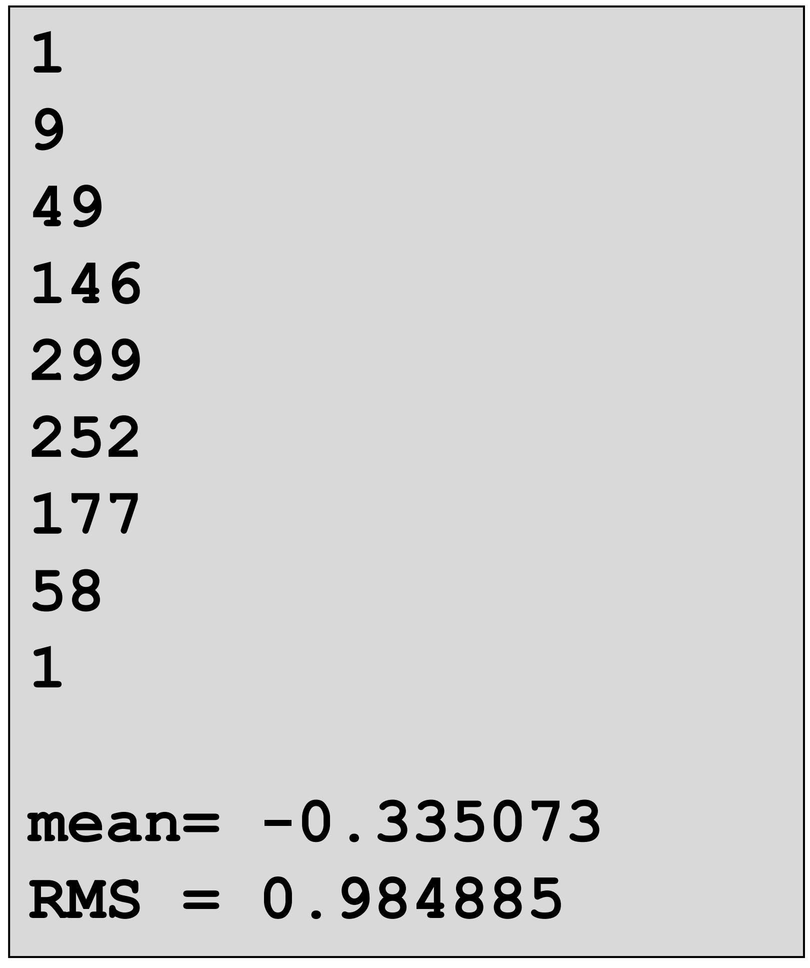

2d histo
hist
Entries 10000
Mean x 0.008003
Mean y -0.0007717
RMS x 0.9947
RMS y 0.9955

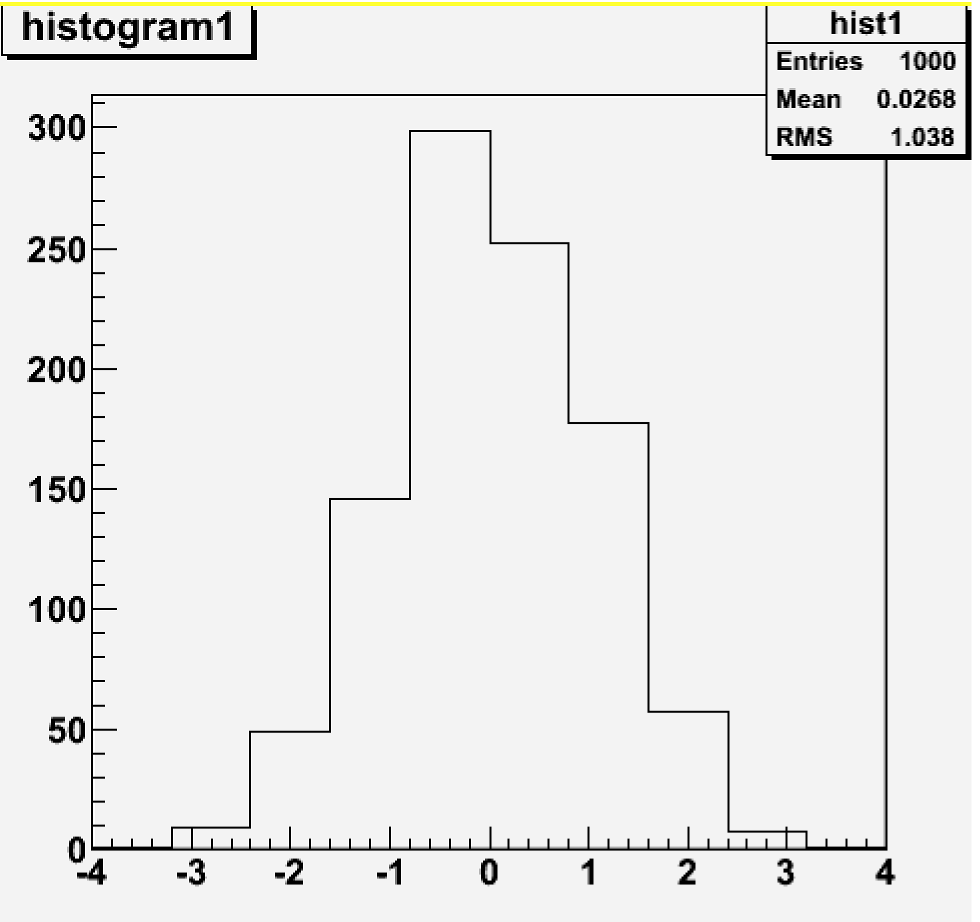
Getting Histogram Bin Content

```
void histo ()
{
   auto *rnd= new TRandom3();
   auto *c1 = new TCanvas("c1","c1",10,10,500,500);
   auto *h1 = new TH1F("hist1","histogram1",10,-4,4);

   for(int i=0; i<1000; i++){
      double r = rnd->Gaus(); // mean=0, sigma=1
      h1->Fill(r);
   }

   h1->Draw();
   for(int i=1; i<=10; i++){
      cout << h1->GetBinContent(i) << endl;
   }

   cout << "mean= " << h1->GetMean() << endl;
   cout << "RMS = " << h1->GetRMS() << endl;
}
```



Output



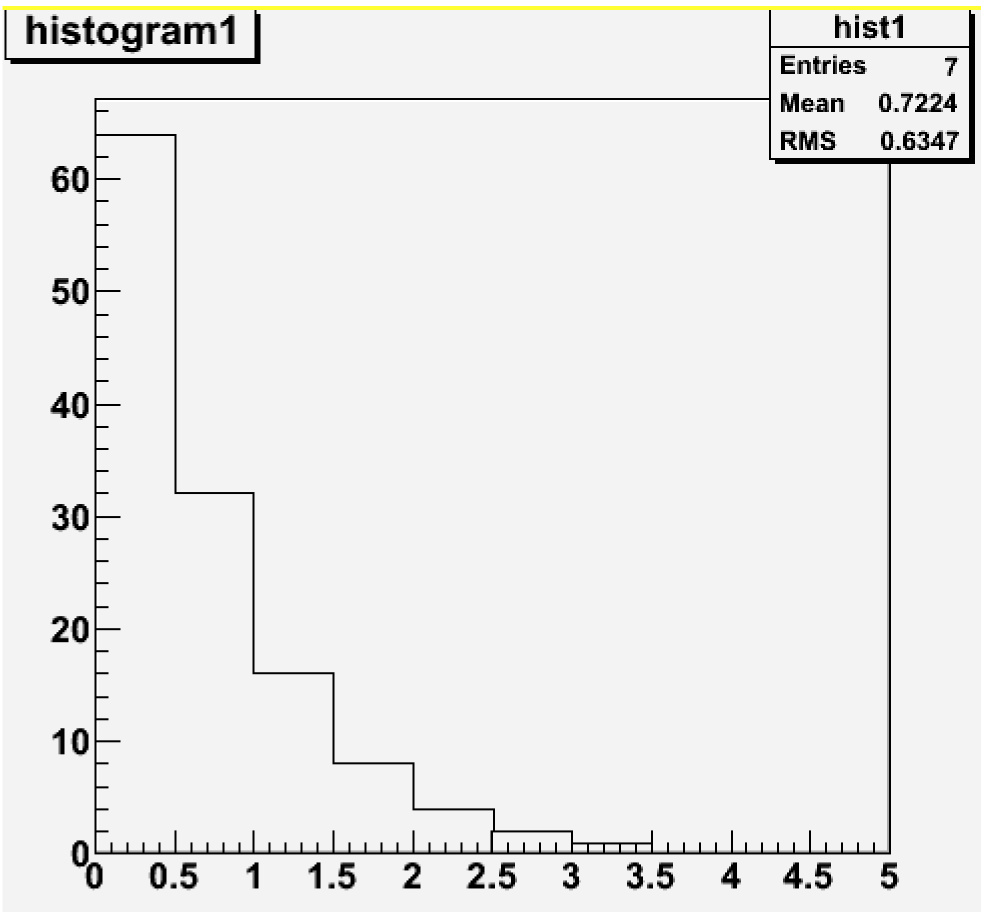


Setting Histogram Bin Content

```
void histo10()
{
  TCanvas *c1 = new TCanvas("c1","c1",10,10,500,500);
  TH1F    *h1 = new TH1F("hist1","histogram1",10,0,5);

  h1->SetBinContent(1,64);
  h1->SetBinContent(2,32);
  h1->SetBinContent(3,16);
  h1->SetBinContent(4,8);
  h1->SetBinContent(5,4);
  h1->SetBinContent(6,2);
  h1->SetBinContent(7,1);

  h1->Draw();

  cout << "mean= " << h1->GetMean() << endl;
  cout << "RMS = " << h1->GetRMS() << endl;
}
```

Output

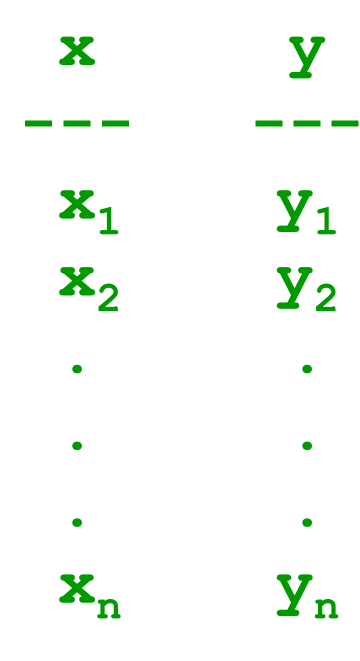
```
mean= 0.702381
RMS = 0.595714
```



Fitting

Curve Fitting

- Data is often given for discrete values along a continuum.
- You may require estimates at points between discrete values.
- In this section we will consider how to obtain values between the given experimental points using **curve fitting method**.



TGraph Fitting

1. Define **`TGraph`** pointer:

```
TGraph *gr = new TGraph(n, x, y);
```

x	y
x_1	y_1
x_2	y_2
.	.
.	.
.	.
x_n	y_n

2. Fit using predefined function:

```
gr->Fit("pol1");
gr->Fit("gaus");
```

or using user defined function

```
TF1 *f1 = new TF1("fun", "[0]*x+[1]", 0, 80);
f1->SetParameter(0, 1);
f1->SetParameter(1, 1);
gr->Fit(f1);
```

Linear function: ax+b

Example : *Linear Fit*

The distance required to stop an automobile is a function of its speed. The following data is collected to get this relationship:

v(km/h)	d(m)
24	4.8
32	6.0
40	10.2
48	12.0
64	18.0
80	27.0

Fit the data to a linear function and compute the goodness of the fit.

```
void gfit1()
{
  // values
  const int n = 6;
  double x[n] = {24, 32, 40, 48, 64, 80};
  double y[n] = {4.8, 6.0, 10.2, 12.0, 18.0, 27.0};

  TGraph *gr = new TGraph(n, x, y);
  gr->SetMarkerStyle(20);
  gr->SetMarkerSize(1);
  gr->Draw("AP");

  // fit with polynomials
  gr->Fit("pol1");
  //gr->Fit("pol2");
}
```

```
void gfit2(){
{
  // values
  const int n = 6;
  double x[n] = {24, 32, 40, 48, 64, 80};
  double y[n] = {4.8, 6.0, 10.2, 12.0, 18.0, 27.0};

  TGraph *gr = new TGraph(n, x, y);
  gr->SetMarkerStyle(20);
  gr->SetMarkerSize(1);
  gr->Draw("AP");

  // user defined fit function
  TF1 *f1 = new TF1("fitfun", "[0]*x+[1]", 0, 80);
  f1->SetParameter(0, 1);
  f1->SetParameter(1, 1);

  gr->Fit(f1);
}
```

Example : *Weighted Linear Fit*

The distance required to stop an automobile is a function of its speed. The following data is collected to get this relationship:

v(km/h)	d(m)
24	4.8 +- 0.3
32	6.0 +- 0.4
40	10.2 +- 1.0
48	12.0 +- 1.1
64	18.0 +- 1.4
80	27.0 +- 1.5

The **+- value** represents the measurement error (one standard deviation).
Fit the data to a linear function and compute the goodness of the fit.

```
void gfit3(){
  // values
  const int n  = 6;
  double  x[n] = {24, 32, 40, 48, 64, 80};
  double  y[n] = {4.8, 6.0, 10.2, 12.0, 18.0, 27.0};
  double ex[n] = {0.0};
  double ey[n] = {0.3, 0.4, 1.0, 1.1, 1.4, 1.5};

  TGraphErrors *gr = new TGraphErrors(n, x, y, ex, ey);
  gr->SetMarkerStyle(20);
  gr->SetMarkerSize(1);
  gr->Draw("AP");

  // user defined fit function
  TF1 *f1 = new TF1("fitfun", "[0]*x+[1]", 0, 80);
  f1->SetParameter(0, 1);
  f1->SetParameter(1, 1);

  gr->Fit(f1);
}
```

Example : *Non-Linear Fit*

Consider a charging RC circuit containing a resistance (R) and an initially uncharged capacitor (C). The switch is closed at $t = 0$ and using a voltmeter the following experimental data is obtained. Assume that each voltage measurement has %5 error.

t(sec)	Vc(Volts)
0.5	2.7
1.0	4.8
1.5	6.2
2.0	7.6
2.5	8.5
3.0	9.3

where t is time and Vc is potential difference the across the capacitor. Using least square fitting method, determine the time constant and the emf (V_0) of the circuit.

```
void gfit4(){
  // values
  const int n = 6;
  double  x[n]= {0.5, 1.0, 1.5, 2.0, 2.5, 3.0};
  double  y[n]= {2.7, 4.8, 6.2, 7.6, 8.5, 9.3};
  double ex[n]= {0.0};
  double ey[n]= {0.0};

  for(int i=0; i<n; i++) ey[i] = 0.05*y[i];

  TGraphErrors *gr = new TGraphErrors(n, x, y, ex, ey);
  gr->SetMarkerStyle(20);
  gr->SetMarkerSize(1.5);
  gr->Draw("AP");

  // user defined fit function
  TF1 *f1 = new TF1("ff", "[0]*(1-exp(-x/[1]))",0.5,3);
  f1->SetParameter(0, 6.0);
  f1->SetParameter(1, 0.8);

  gr->Fit(f1);
}
```



```
 EXT  PARAMETER                                   STEP         FIRST
 NO.   NAME       VALUE          ERROR            SIZE         DERIVATIVE
  1   p0         1.18583e+001   1.01870e+000   1.24531e-004   2.72520e-005
  2   p1         1.95738e+000   2.57916e-001   3.15288e-005  -1.05960e-004
root [6]
```

Histogram Fitting

1. Define **`TH1F`** pointer:

```
TH1F *h = new TH1F("histo","title", n,min, max);
```

2. Fit using predefined function:

```
h->Fit("pol1");
h->Fit("gaus");
```

or using user defined function:

```
TF1 *f1 = new TF1("fun", "[0]*x+[1]", min, max);
f1->SetParameter(0, 10);
f1->SetParameter(1, 100);
h->Fit(f1);
```

Linear function: ax+b

```
void hfit1(){
  TH1F *h = new TH1F("h","gauss dist",20,-3,3);

  for(int i=0;i<1000;i++){
    h->Fill(gRandom->Gaus());
  }
  // the Gaussian function
  char *fitfun="[0]*exp(-0.5*(x-[1])/[2]*(x-[1])/[2])";

  // user defined fit function
  TF1 *f1 = new TF1("f", fitfun);
  f1->SetParameter(0, 100.0);
  f1->SetParameter(1, 0.1);
  f1->SetParameter(2, 0.5);

  h->Fit(f1);
}
```

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$

Example:

Consider that a file contains the data given right.

These numbers are content of a 40-bin histogram.

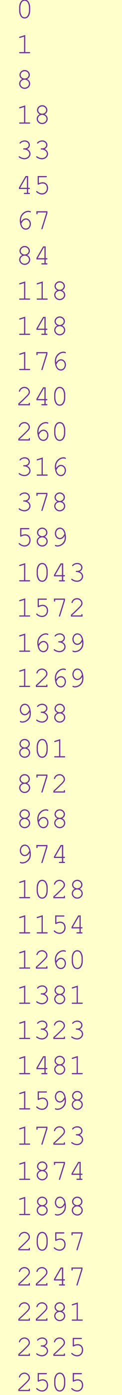
Assume that histogram range is [0, 300].

a) Read the file and draw the histogram.

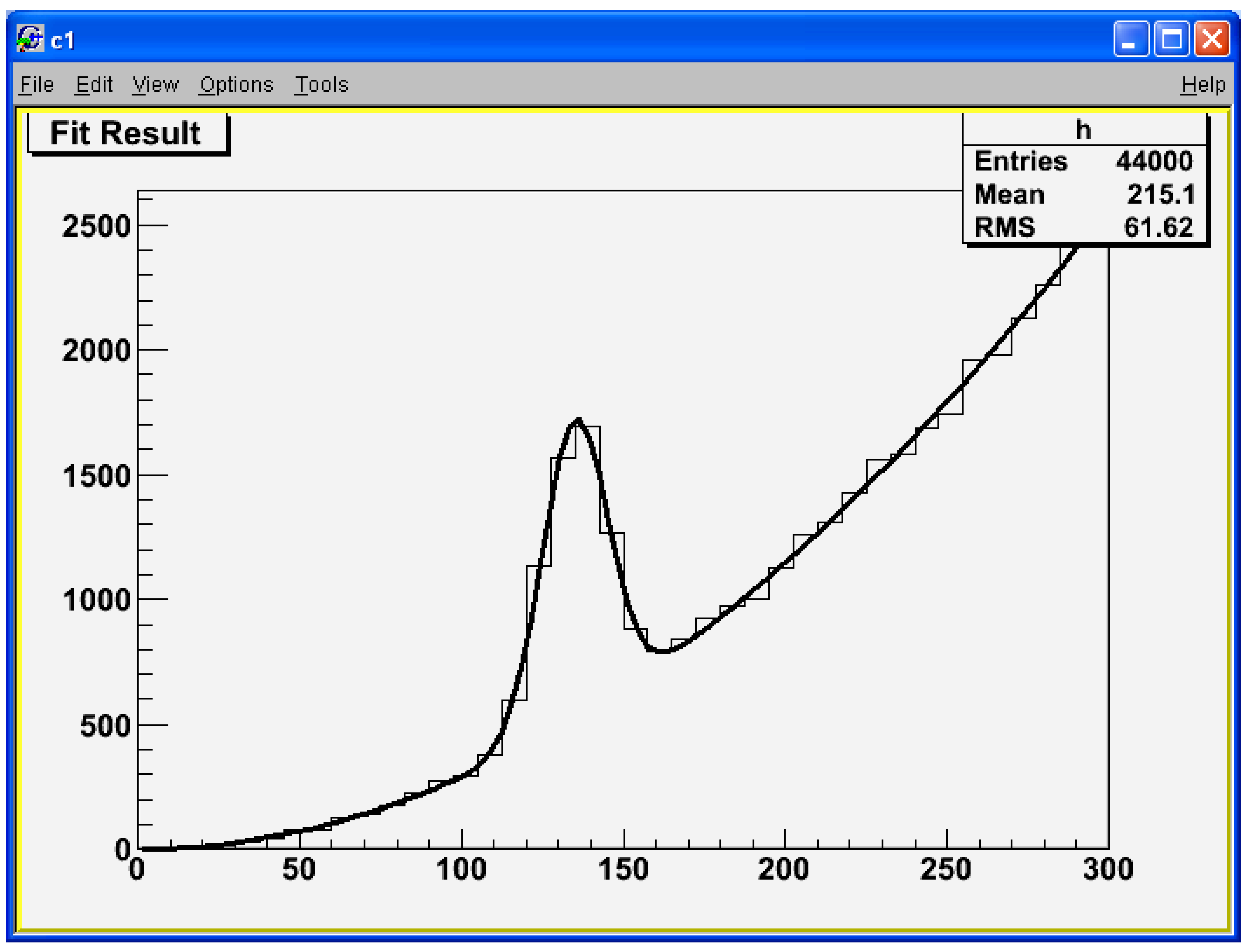
b) Fit the data to the function:

$$f(x) = a_0 \exp\left[-\frac{(x-a_1)^2}{2a_2^2}\right] + a_3 + a_4 x + a_5 x^2$$

where a_i are free parameters.



```
0
1
8
18
33
45
67
84
118
148
176
240
260
316
378
589
1043
1572
1639
1269
938
801
872
868
974
1028
1154
1260
1381
1323
1481
1598
1723
1874
1898
2057
2247
2281
2325
2505
```


c1
File Edit View Options Tools Help
Fit Result
h
Entries 44000
Mean 215.1
RMS 61.62
2500
2000
1500
1000
500
0
0 50 100 150 200 250 300

Root Files and Trees

The ROOT File

- In ROOT, objects are written in files*, represented by **`TFile`** instances
- TFiles are *binary* and can be compressed (transparently for the user)
- **TFiles are self-descriptive**:
  - The information how to retrieve objects from a file is stored with the objects

TFile in Action:

```
TFile f("myfile.root", "RECREATE");
```

Option	Description
NEW or CREATE	Create a new file and open it for writing, if the file already exists the file is not opened.
RECREATE	Create a new file, if the file already exists it will be overwritten.
UPDATE	Open an existing file for writing. If no file exists, it is created.
READ	Open an existing file for reading (default).

Writing Histogram in TFile

```cpp
TFile f("file.root", "RECREATE");
TH1F h("h", "h", 64, 0, 8);
h.Fill(5);
h.Fill(3);
h.Fill(2);
h.Write("h");
f.Close();
```

C++

Write to a file

Close the file and make sure the operation succeeded

Reading Histogram in TFile

```cpp
TFile f("file.root");
TH1F* h = f.Get<TH1F>("h");
h->Draw();
```

```python
import ROOT
f = ROOT.TFile("file.root")
f.h.Draw()
```

Python

Get the histogram by name! Possible only in Python

- In case you want to store large quantities of same-class objects, ROOT has designed the `TTree` classes specifically to save root files called `NTuples`.

- The `TTree` class is optimized to reduce disk space and enhance access speed.

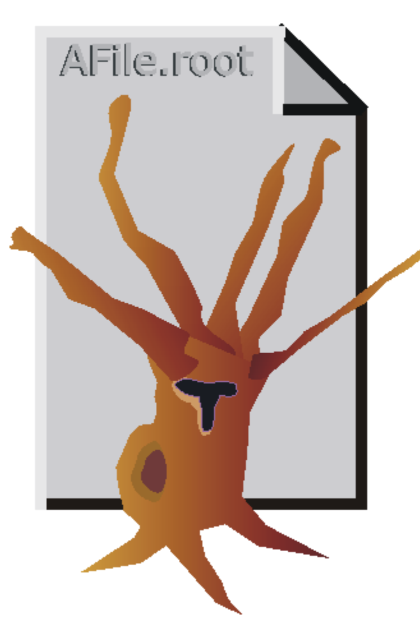
- A `TTree` can hold all kind of data, such as objects or arrays in addition to all the simple types.

Trees

- When using a **`TTree`**, we fill its branch buffers with leaf data and the buffers are written to disk when it is full.

- Using **`TTree`**, the following items can be written to a ROOT file

  - variables of ordinary data types (**`int`**, **`double`**, etc),
  - objects from derived data types (class).
  - histogram objects
  - vectors
  - any object derived from **`TObject`**
  - . . .



```cpp
// *** WRITING TREE ***
void treeWrite(){

   TRandom3 *rnd = new TRandom3();

// create a tree file mytree.root
   TFile *f  = new TFile("mytree.root","recreate");
   TTree *t1 = new TTree("Values","a simple tree variables");

// Define branch variables as numpy arrays
   Float_t px, py, pz;
   Int_t ev;


// set branch addresses
   t1->Branch("px",&px,"px/F");
   t1->Branch("py",&py,"py/F");
   t1->Branch("pz",&pz,"pz/F");
   t1->Branch("ev",&ev,"ev/I");

// fill the tree
   cout << "Filling tree..." << endl;
   for (Int_t i=0; i<10000; i++) {
     px = rnd->Gaus(0,2);
     py = rnd->Gaus(0,3);
     pz = rnd->Gaus(0,1);
     ev = i;
     t1->Fill();
   }

// save the Tree header
   t1->Write();
   cout << "done." << endl;
}
```

```python
import ROOT
import numpy as np

rnd = ROOT.TRandom3()

# create a tree file mytree.root
f  = ROOT.TFile("mytree.root","recreate")
t1 = ROOT.TTree("Values","a simple tree variables")

# Define branch variables as numpy arrays
px = np.zeros(1, dtype=np.float32)
py = np.zeros(1, dtype=np.float32)
pz = np.zeros(1, dtype=np.float32)
ev = np.zeros(1, dtype=np.int32)

# set branch addresses
t1.Branch("px",px,"px/F")
t1.Branch("py",py,"py/F")
t1.Branch("pz",pz,"pz/F")
t1.Branch("ev",ev,"ev/I")

# fill the tree
print("Filling tree...")
for i in range(10000):
     px[0] = rnd.Gaus(0,2)
     py[0] = rnd.Gaus(0,3)
     pz[0] = rnd.Gaus(0,1)
     ev[0] = i
     t1.Fill()

# save the Tree header
t1.Write();
print("done.")
```

```cpp
// *** READING TREE ***
void treeRead(){

// read the Tree generated by treeWrite (previously)
   TFile *f = new TFile("mytree.root");
   TTree *t1 = (TTree*) f->Get("Values");

// Define branch variables
   Float_t px, py, pz;
   Int_t ev;



// Set branch addresses
   t1->SetBranchAddress("px",&px);
   t1->SetBranchAddress("py",&py);
   t1->SetBranchAddress("pz",&pz);
   t1->SetBranchAddress("ev",&ev);

// book histogram(s)
   TH1F *hpz   = new TH1F("hpz","pz",100,-3,3);

# Read all entries and fill histogram
   cout << "Reading data..." << endl;
   Int_t nentries = t1->GetEntries();
   for (Int_t i=0; i < nentries; i++) {
      t1->GetEntry(i);
      hpz->Fill(pz);
   }

// Draw
   hpz->Draw();
}
```

```python
import ROOT
import numpy as np

# Read the tree generated by treeWrite
f  = ROOT.TFile("mytree.root")
t1 = f.Get("Values")

# Define branch variables as numpy arrays
px = np.zeros(1, dtype=np.float32)
py = np.zeros(1, dtype=np.float32)
pz = np.zeros(1, dtype=np.float32)
ev = np.zeros(1, dtype=np.int32)

# Set branch addresses
t1.SetBranchAddress("px", px)
t1.SetBranchAddress("py", py)
t1.SetBranchAddress("pz", pz)
t1.SetBranchAddress("ev", ev)

# Create histogram(s)
hpz = ROOT.TH1F("hpz", "pz distribution", 100, -3, 3)

# Read all entries and fill histogram
print("Reading data...")
nentries = t1.GetEntries()
for i in range(nentries):
    t1.GetEntry(i)
    hpz.Fill(pz[0])

# Draw
hpz.Draw()
input("Press Enter to exit...")  # keep canvas open
```

Trees in Analysis

- You can analyse the root file in ROOT prompt

Terminal:

```
$ root mytree.root
```

ROOT console:

```
root [0] new TBrowser
```

ROOT console:

```
root [0] Values->Print()

root [1] Values->Scan("px")
root [2] Values->Scan("px:py")
root [3] Values->Scan("px:py", "px>0 & py>0")

root [4] Values->Draw("pz")
root [5] Values->Draw("px:py")
root [6] Values->Draw("px:py", "px>0 & py>0")
```

RDateFrame

- **Build** a data-frame object by specifying your data-set
- Apply a series of **transformations** to your data
  - filter (e.g. apply some cuts) or
  - define new columns
- Apply **actions** to the transformed data to produce results
  (e.g. fill a histogram)

```python
import ROOT

df = ROOT.RDataFrame("Values", "mytree.root")




h1 = df.Filter("px > 0").Histo1D(("h1", ";py;Entries", 100, -9, 9), "py")
h2 = df.Filter("px > 0").Histo2D(("h2", ";px;py", 100, -6, 6, 100, -9, 9), "px", "py")

c1 = ROOT.TCanvas("c1", "c1", 800, 600)
h1.Draw()

c2 = ROOT.TCanvas("c2", "c2", 800, 600)
h2.Draw("COLZ")

input("Press Enter to exit...")
```

```python
import ROOT

df = ROOT.RDataFrame("Values", "mytree.root")

# Define new column pt
df = df.Define("pt", "sqrt(px*px + py*py)")

h1 = df.Filter("px > 0").Histo1D(("h1", ";py;Entries", 100, -9, 9), "py")
h2 = df.Filter("px > 0").Histo2D(("h2", ";px;py", 100, -6, 6, 100, -9, 9), "px", "py")
h3 = df.Histo1D(("h3", "pt distribution;pt;Entries", 100, 0, 10), "pt")

c1 = ROOT.TCanvas("c1", "c1", 800, 600)
h1.Draw()

c2 = ROOT.TCanvas("c2", "c2", 800, 600)
h2.Draw("COLZ")

c3 = ROOT.TCanvas("c3", "c3", 800, 600)
h3.Draw()

input("Press Enter to exit...")
```

Case Study: Photons in ALEPH Data

The csv file **AlephPhotonMC.tgz** contains event information, momentum components, and mother id and barcodes for photons generated in the process $e^+e^- \rightarrow Z \rightarrow \overline{qq} \rightarrow \text{hadrons}$, as recorded in the ALEPH full simulation. (motherid = 7 for π^0 and motherid = 17 for η meson)

Download file:

```
$ wget http://www1.gantep.edu.tr/~bingul/ep228/AlephPhotonMC.tgz
```

Extract file:

```
$ tar -xzf AlephPhotonMC.tgz
```

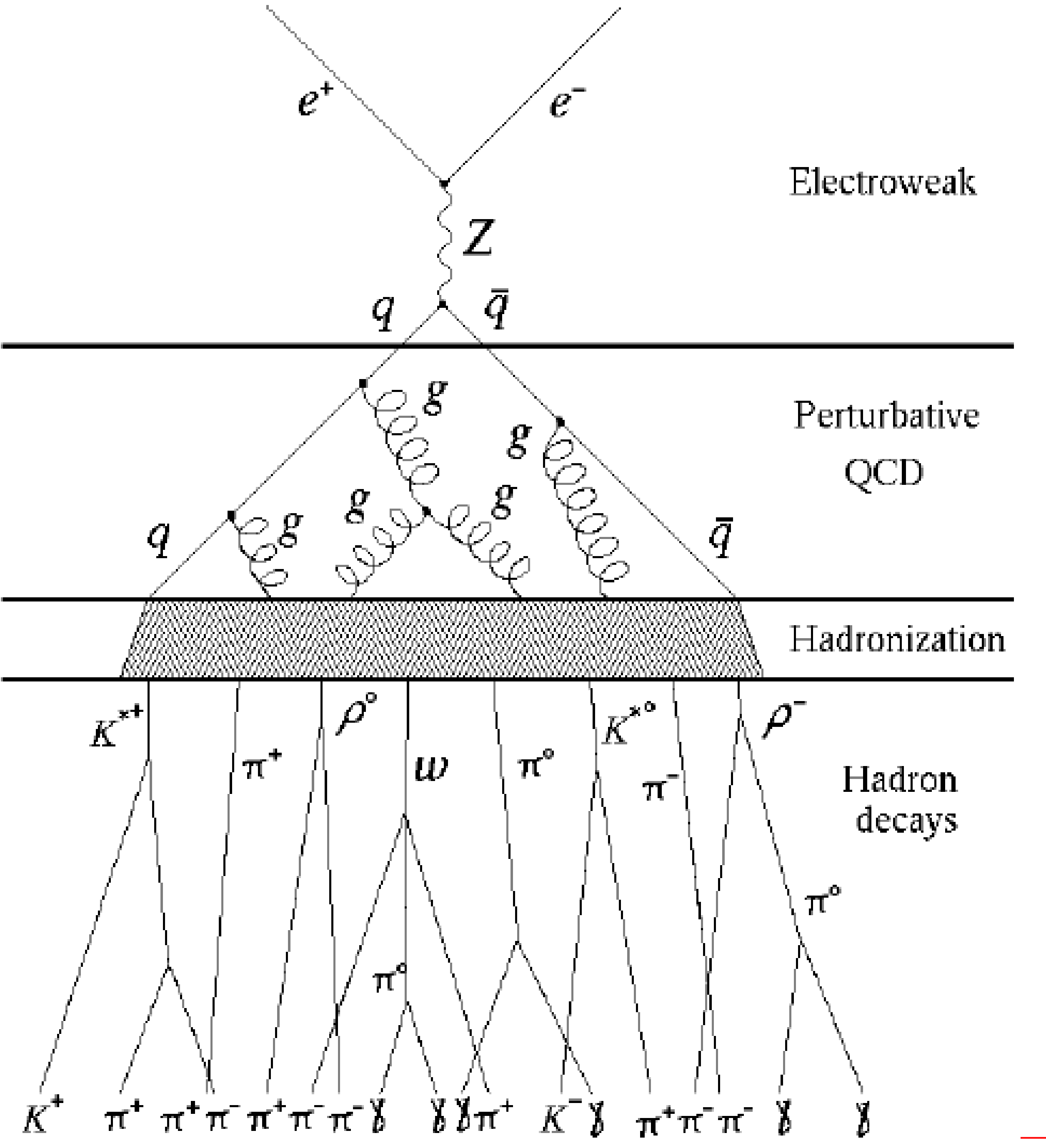
File content (first two events)

```
Event,px,py,pz,motherid,motherbrc
     0,0.77746,0.44144,0.68944,7,220
     0,-0.72786,0.82886,0.35566,7,235
     0,-2.3299,2.1592,0.71033,7,234
     0,-0.93539,0.88459,0.32983,7,236
     0,-0.84821,0.98503,0.29068,7,236
     0,-1.3061,1.3367,0.3014,7,234
     0,-1.8535,1.3631,0.13959,7,192
     0,-0.8809,0.78515,0.1822,7,234
     0,-0.80186,0.71226,0.056462,7,192
     0,0.84303,-0.824,-0.019339,7,242
     0,0.81343,-0.92873,-0.10445,7,241
     1,1.8145,2.4897,1.224,7,187
     1,-1.7508,-1.9248,-0.83291,7,197
     1,-2.4766,-2.6794,-1.6023,7,194
     1,-1.1369,-1.1833,-0.58768,7,193
     1,-0.86119,-1.4685,-0.59991,7,190. . .
```

- High energy collisions of sub-atomic particles can result in events containing a high multiplicity of hadronic particles.

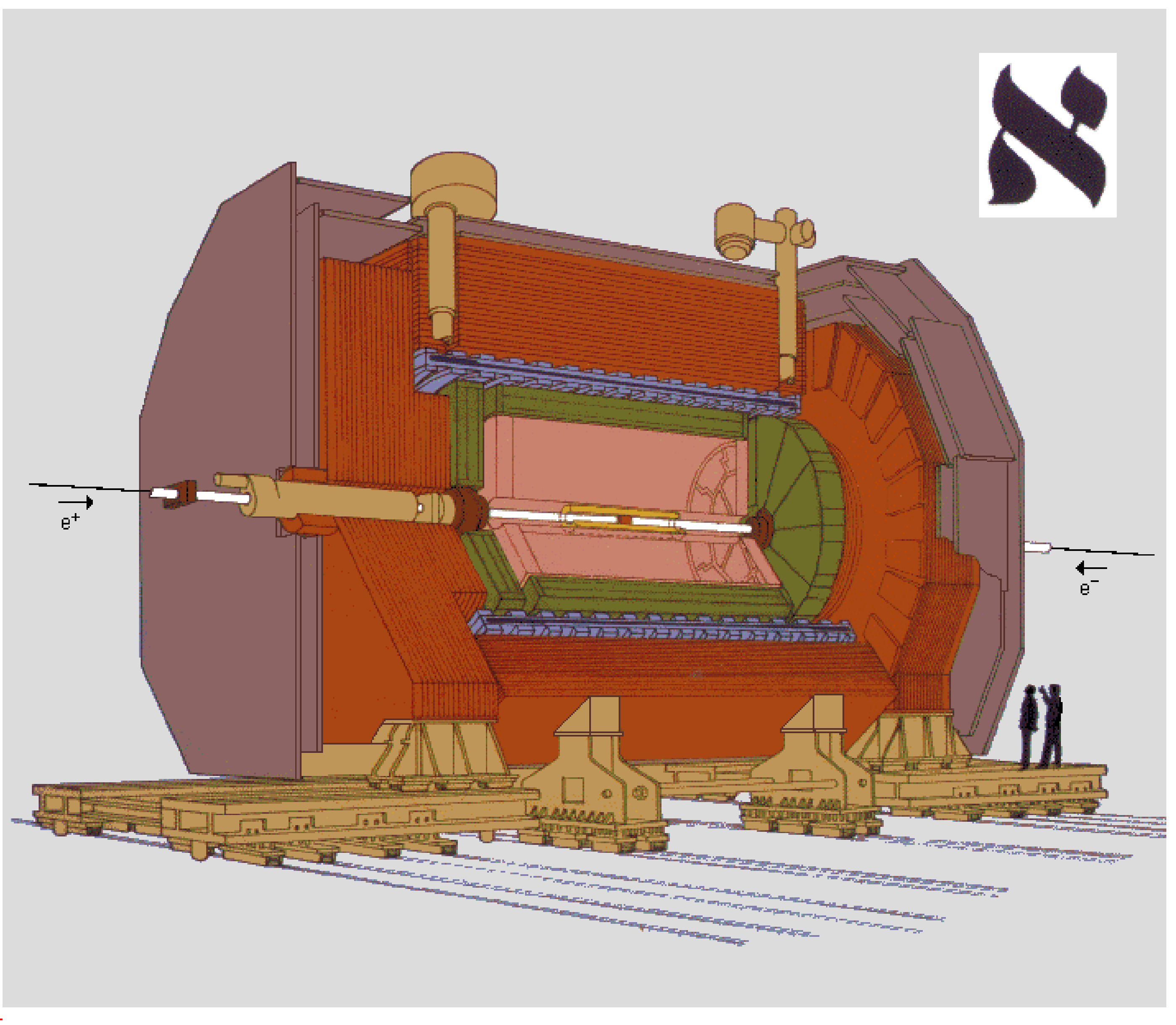
- An example was the production of Z^0 Bosons at LEP an e^+e^- collider with E_{CM} = 91.2 GeV (≈ 1.5 x 10^{-8} Joules)

$$e^+e^- \rightarrow Z \rightarrow q\bar{q} \rightarrow hadrons$$



e+
e-
Electroweak
Z
q
q̄
Perturbative
QCD
g
g
g
g
g
q
q̄
Hadronization
K*+
ρ°
K*°
ρ-
π+
ω
π°
π-
Hadron
decays
π°
π°
K+ π+ π+ π- π+ π- π- γ γ γ π+ K- γ π+ π- π- γ γ

ALEPH Detector



Vertex Detector

Inner Tracking Chamber

Time Projection Chamber

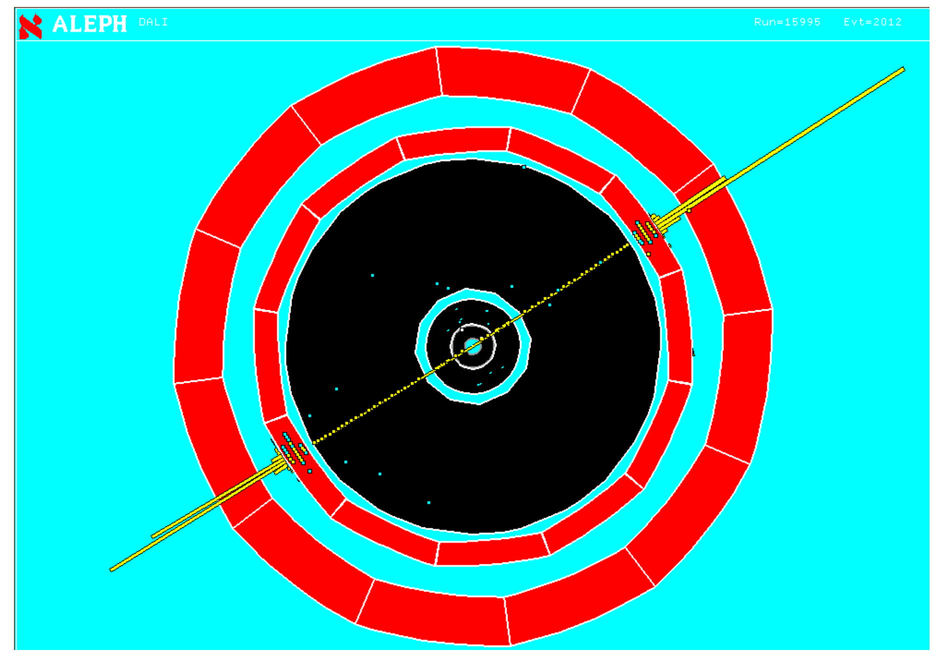
Electromagnetic Calorimeter

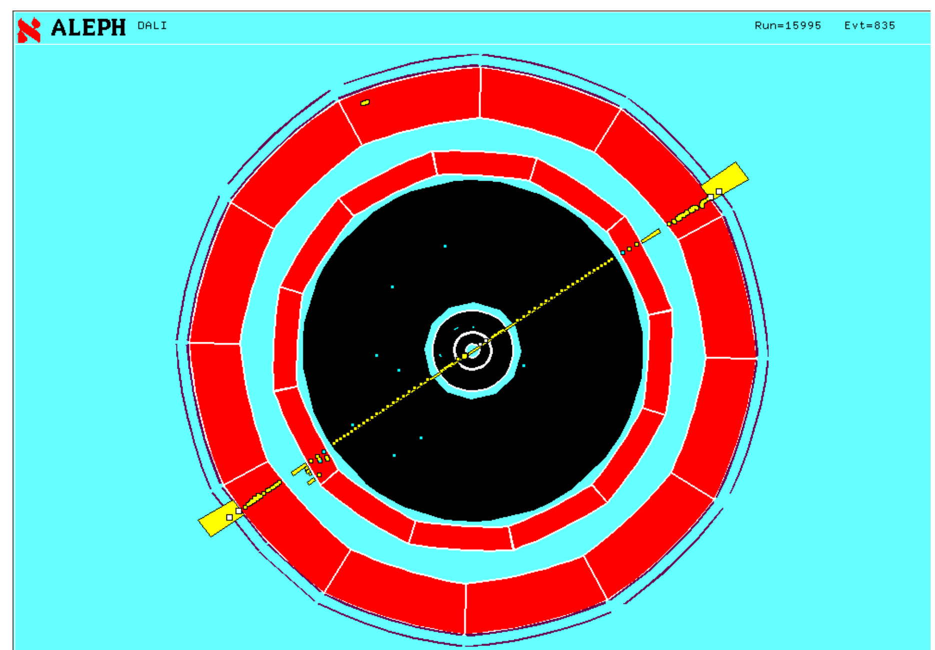
Superconducting Magnet Coil

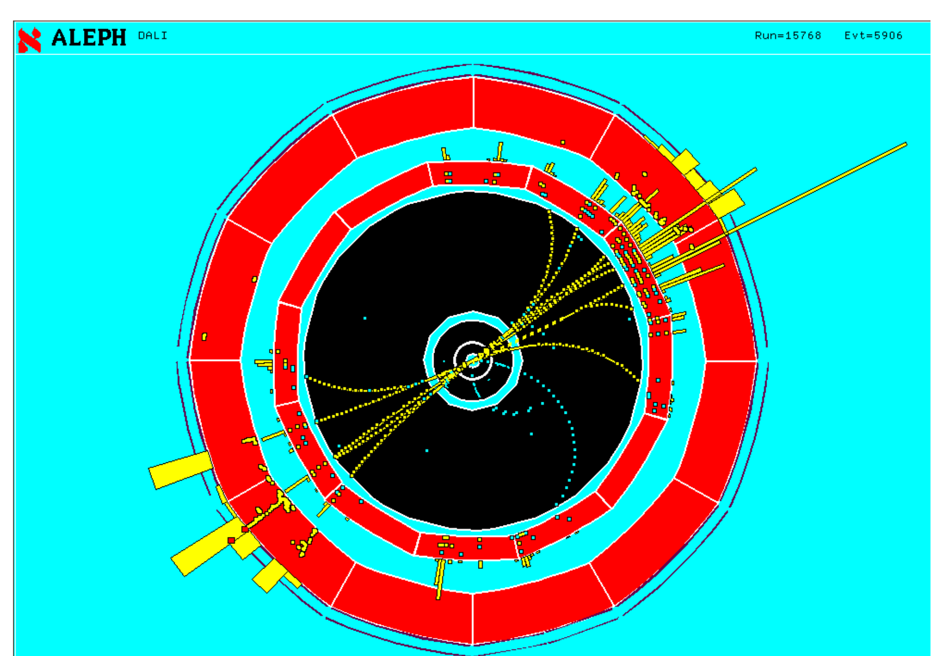
Hadron Calorimeter

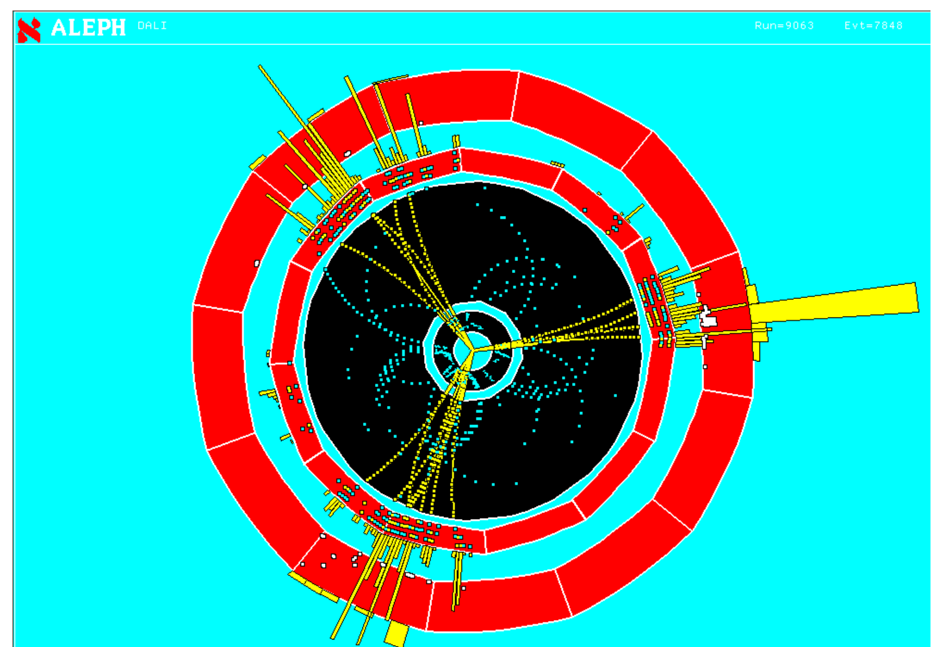
Muon Chambers

Luminosity Monitors

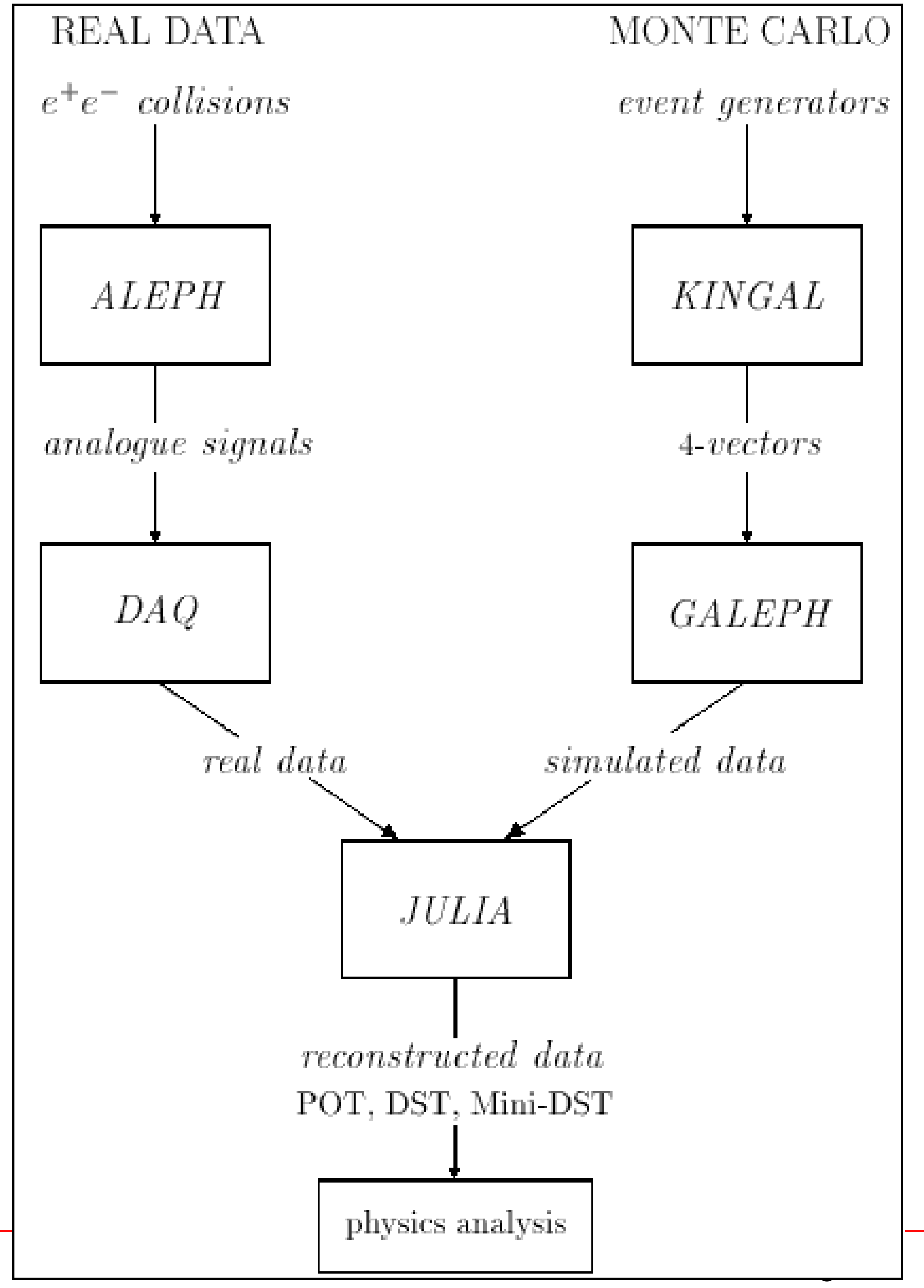

ALEPH
DALI
Run=15995 Evt=2012


ALEPH
DALI
Run=15995 Evt=835


ALEPH
DALI
Run=15768 Evt=5906


ALEPH
DALI
Run=9063 Evt=7848

Event Reconstruction and Simulation

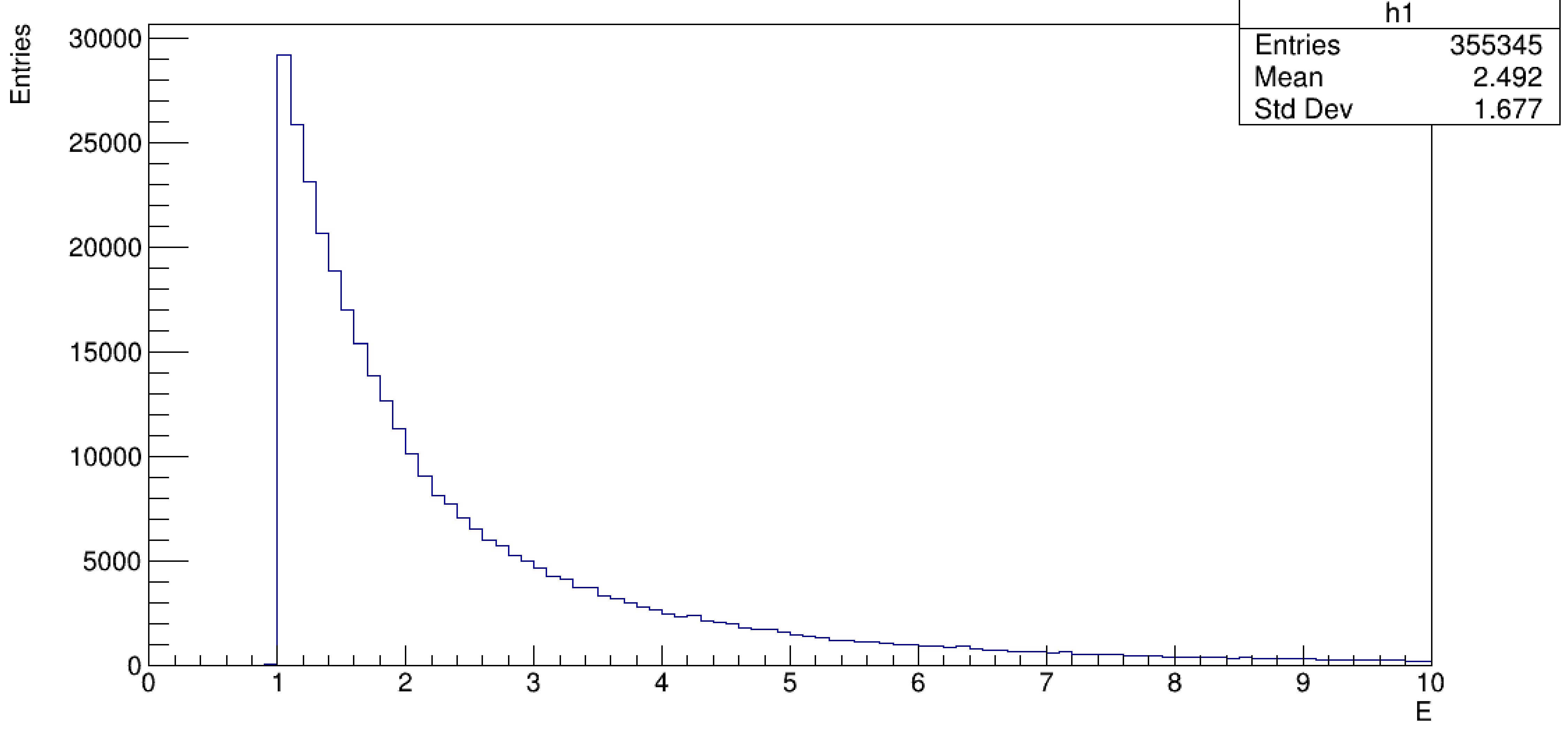


Following program reads the csv file and plots energy distribution of photons:

```
import ROOT

# Read CSV file
df = ROOT.RDF.FromCSV("AlephPhotonMC.csv")
print(df.GetColumnNames())

# Define new column Energy
df = df.Define("Energy", "sqrt(px*px + py*py + pz*pz)")

# Define histogram
h1 = df.Histo1D(("h1", "Energy distribution;E;Entries", 100, 0, 10), "Energy")

# draw
c1 = ROOT.TCanvas("c1", "c1", 800, 600)
h1.Draw()

input("Press Enter to exit...")
```

Following program converts a csv file to a root file.

```
import ROOT

# Read CSV file
df = ROOT.RDF.FromCSV("AlephPhotonMC.csv")

# Define new column Energy
df = df.Define("Energy", "sqrt(px*px + py*py + pz*pz)")

# Define new column Energy
df2 = df.Snapshot("GammaParticles", "AlephPhotonMC.root")

input("Press Enter to exit...")
```



Energy distribution
h1
Entries 355345
Mean 2.492
Std Dev 1.677
Entries
30000
25000
20000
15000
10000
5000
0
0 1 2 3 4 5 6 7 8 9 10
E

C++ Vectors and Trees

Putting Vectors in Trees

This program creates a root file named **`Tracks.root`** which contains 3 vector branches.

At each event vectors are resized.

Hence

vector size = event size

```
void treeVectors()
{
   auto rnd = new TRandom();
// vector pointers
    std::vector<float> *px = new vector<float>;
    std::vector<float> *py = new vector<float>;
    std::vector<float> *pz = new vector<float>;
// data files and trees
    TFile *file = new TFile("Tracks.root","recreate");
    TTree *t1   = new TTree("TrackContainer", "Ch. tracks");
// set branches
    t1->Branch("px", &px);
    t1->Branch("py", &py);
    t1->Branch("pz", &pz);
//  Fill vectors
    for(int event = 1; event <= 1000; i++){
       for(int j =1; j< rnd->Uniform(1,8); j++){
          px->push_back(rnd->Gaus(0,1));
          py->push_back(rnd->Gaus(0,2));
          pz->push_back(rnd->Gaus(0,3));
       }
       t1->Fill();
       px->clear();
       py->clear();
       pz->clear();
    }
// save data
    t1->Write();
    file->Close();
}
```

Reading Trees with Vectors

```
root [1] TrackContainer->Draw("px")

root [2] TrackContainer->MakeClass()
         Info in <TTreePlayer::MakeClass>: Files: TrackContainer.h and
         TrackContainer.C generated from TTree: TrackContainer
```

This program reads file **`Tracks.root`**. Then, prints the values of px data in each event.

```
#include "TrackContainer.C"

void TrackTest()
{
   // create a chain
   auto chain = new TChain("TrackContainer","TC");
   chain->Add("Tracks.root");

   // new object
   auto tc = new TrackContainer(chain);

   // number of data (events)
   int nEvent = chain->GetEntries();

   // loop over all data
   for(int ev = 0; ev < nEvent; ev++)
   {
     tc->GetEntry(ev);

     cout << "event #"<< ev
          << " event size = " << tc->px->size() << endl;

     for(int i=0; i<tc->px->size(); i++){
       cout << tc->px->at(i) << endl;
     }
     cout << endl;
   }
   cout << "ALL OK." << endl;
}
```

Root and Standalone C++

you can compile this file in command line:

```
$ g++ main.cc -o main `root-config --cflags --glibs`
```

and run

```
$ ./main
```

This will be faster in run time.

```
// compile:
//   $ g++ main.cc -o main `root-config --cflags --glibs`
// run:
//   $ ./main

#include <iostream>
#include <cmath>
#include <TTree.h>
#include <TChain.h>
using namespace std;
#include "TrackContainer.C"

void TrackTest(){
   auto chain = new   TChain("TrackContainer","TC");
   chain->Add("Tracks.root");

   auto tc = new TrackContainer(chain);
   int nEvent = chain->GetEntries();

   for(int ev = 0; ev < nEvent; ev++)
   {
     tc->GetEntry(ev);

     cout << "event #"<< ev
          << " event size = " << tc->px->size() << endl;

     for(int i=0; i<tc->px->size(); i++){
       cout << tc->px->at(i) << endl;
     }
     cout << endl;
   }
}

int main(){
  TrackTest();
}
```

Exercises

Using **`AlephPhotonMC.csv`** file

1. Plot energy distribution of photons originating from pions and eta mosons.

2. Plot *normalized* distributions in part (a) on the same canvas.

3. Plot polar angle (Θ) distribution of all photons where $\Theta = \mathrm{acos}(p_z/E)$

4. Plot distribution of number of reconstructed photons

5. Convert data in **`AlephPhotonMC.csv`** to vectorized ntuple file called **`AlephPhotonMCV.root`**

6. Using **AlephPhotonMCV.root**, plot two-photon invariant mass distribution using

$$M = \sqrt{2E_1E_2(1 - \cos\theta)}.$$

Hint: This is event by event process. Namely, you should check event numbers.
For example, lets consider event #2.

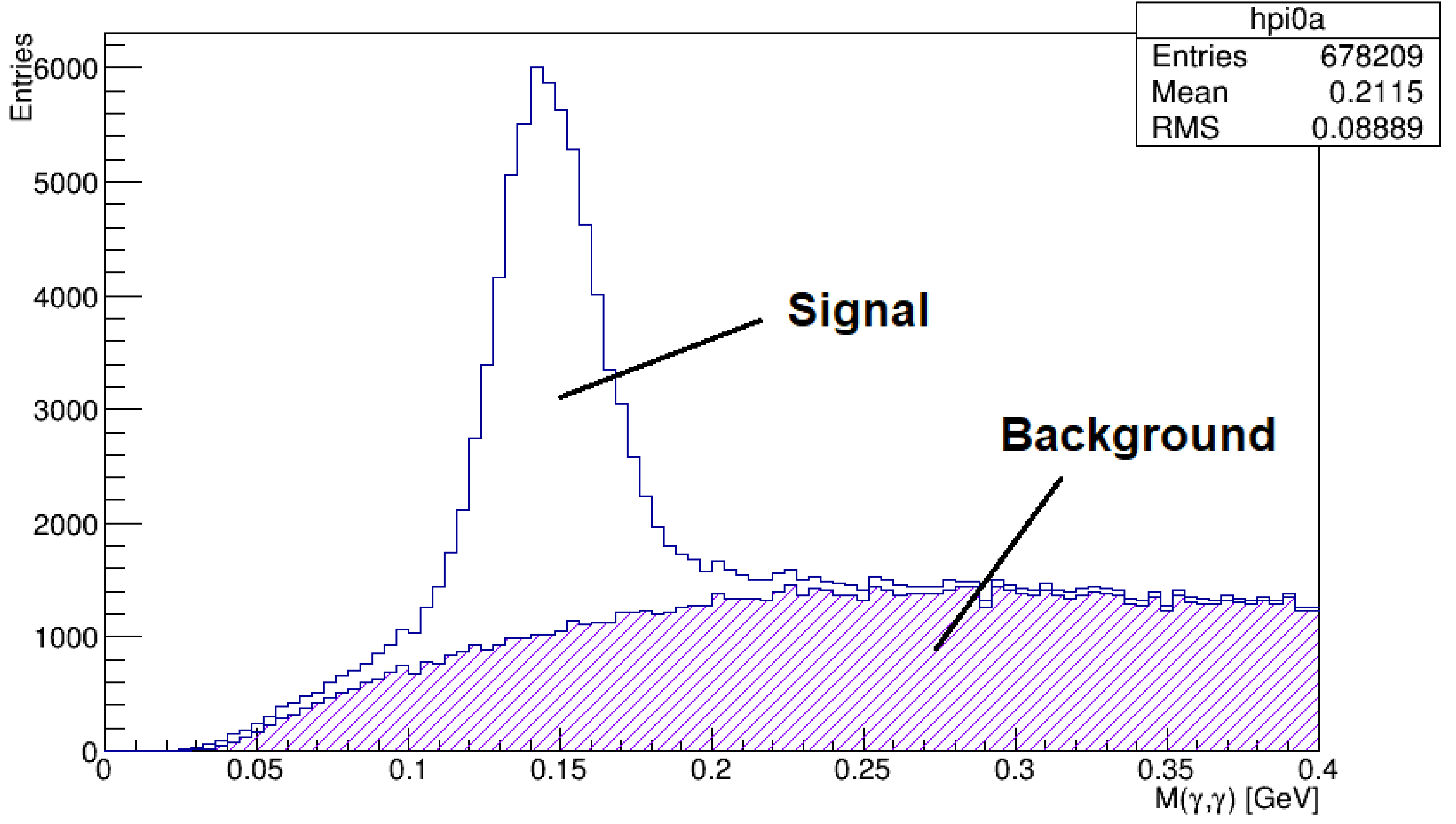
```
Event# Momentum Components           Mother     Photon#
2,    -0.43465,-0.75844, 3.1697, 7,176    <- 1
2,    -0.17662,-0.30763, 1.5961, 7,176    <- 2
2,    -0.29793,-0.19601, 1.4498, 7,174    <- 3
2,    -0.37523,-0.25699, 1.0823, 7,219    <- 4
2,     0.14454, 0.41163,-0.91978,7,197    <- 5
2,     0.80879, 0.48108,-2.5637, 7,204    <- 6
```

1. Combine photons in the event as follows:

```
11 12 13 14 15 16
23 24 25 26
34 35 36
45 46
56
```

2. Compute invariant mass (M) for each pair
3. Fill histogram
4. Draw histogram

For a photon pair, if motherid and motherbrc are the same, then this pair is a signal candidate. Otherwise, it is a combinatorial background. Here is an example two photon mass spectrum. Neutral pion signal ($\pi^0 \rightarrow \gamma\gamma$) and background are seen clearly!

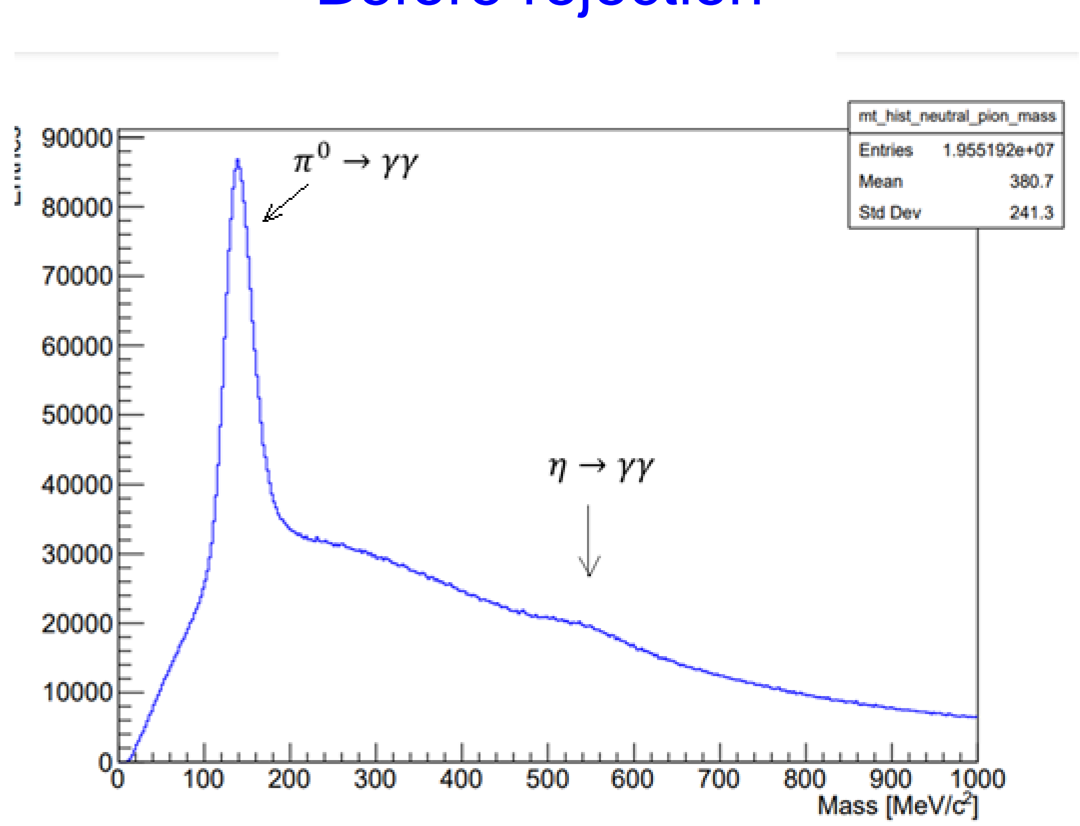


7. π^0 Rejection Process

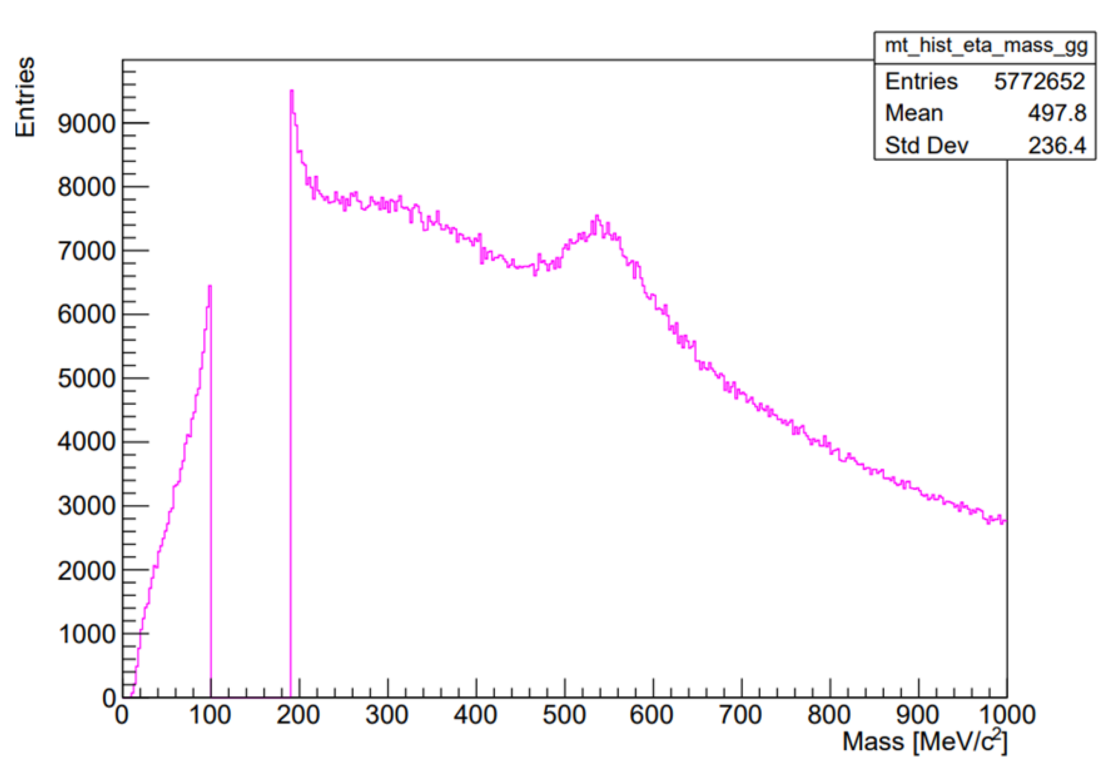
Photon pairs with an invariant mass in the range 100 MeV ≤ M ≤ 190 MeV are flagged.

Flagged photons are excluded from further analysis to reduce combinatoric background for eta mesons. Verify this method using **`AlephPhotonMCV.root`**. Can you opimize mass window cut above?

Before rejection



After rejection



8. π^0 Rejection Process for MC and Real Data

Download real data file:

```
$ wget http://www1.gantep.edu.tr/~bingul/ep228/AlephPhotonDA.tgz
```

Extract file:

```
$ tar -xzf AlephPhotonDA.tgz
```

This file has the same content as `AlephPhotonMC.tgz` except motherid = 0 and motherbrc = 0.

Repeat the exercises 5, 6 and 7 using Monte Carlo and real data by plotting **normalized** histograms of MC and real data on top of each other for each task.