Second Round in Root :/

METU HEP Study Group

How to use Reference Guide

TROOT



ROOT » CORE » BASE » TROOT

The TROOT object is the entry point to the ROOT system.
The single instance of TROOT is accessible via the global gROOT.
Using the gROOT pointer one has access to basically every object created in a ROOT based program. The TROOT object is essentially a container of several lists pointing to the main ROOT objects.

The following lists are accessible from gROOT object:

- gROOT->GetListOfClasses
- gROOT->GetListOfColors
- gROOT->GetListOfTypes
- gROOT->GetListOfGlobals
- gROOT->GetListOfGlobalFunctions
- gROOT->GetListOfFiles
- gROOT->GetListOfMappedFiles
- gROOT->GetListOfSockets
- gROOT->GetListOfSecContexts
- gROOT->GetListOfCanvases
- gROOT->GetListOfStyles
- gROOT->GetListOfFunctions
- gROOT->GetListOfSpecials (for example graphical cuts)
- gROOT->GetListOfGeometries
- gROOT->GetListOfBrowsers
- gROOT->GetListOfCleanups
- gROOT->GetListOfMessageHandlers

gROOT->Reset();

void **Reset** (Option_t* option = "")

Delete all global interpreter objects created since the last call to Reset

If option="a" is set reset to startup context (i.e. unload also all loaded files, classes, structs, typedefs, etc.).

This function is typically used at the beginning (or end) of an unnamed macro to clean the environment.

IMPORTANT WARNING:
Do not use this call from within any function (neither compiled nor interpreted. This should only be used from a unnamed macro (which starts with a { (curly braces)). For example, using TROOT::Reset from within an interpreted function will lead to the unloading of the dictionary and source file, including the one defining the function being executed.

TDirectory



ROOT » CORE » BASE » TDirectory

gDirectory->GetListOfKeys()
gDirectory->Get("a_hist")

TStyle



ROOT » CORE » BASE » TStyle

TStyle objects may be created to define special styles.
By default ROOT creates a default style that can be accessed via
the gStyle pointer.

This class includes functions to set some of the following object attributes.

- Canvas
- Pad
- Histogram axis
- Lines
- Fill areas
- Text
- Markers
- Functions
- Histogram Statistics and Titles

```
gStyle->SetTitleFillColor(kWhite)
gStyle->Reset()
gStyle->SetOptStat(0)
gStyle->SetHistLineWidth(2)
```

TPad



ROOT » GRAF2D » GPAD » TPad

The Pad class is the most important graphics class in the ROOT system.

A Pad is contained in a Canvas.

virtual void **SetGrid** (Int_t valuex = 1, Int_t valuey = 1)
virtual void **SetGridx** (Int_t value = 1) TOGGLE
virtual void **SetGridy** (Int_t value = 1) TOGGLE
virtual void **SetLogx** (Int_t value = 1) TOGGLE
virtual void **SetLogy** (Int_t value = 1) TOGGLE
virtual void **SetLogz** (Int_t value = 1) TOGGLE

```
gPad->SetLogy()
gPad->SetLogy(0)
gPad->SetLogy(1)
gPad->SetGridx()
gPad->BuildLegend(...)
```

TLegend * **BuildLegend** (Double_t x1 = 0.5, Double_t y1 = 0.67, Double_t x2 = 0.88, Double_t y2 = 0.88, const char* title = "")

Build a legend from the graphical objects in the pad

A simple method to to build automatically a TLegend from the primitives in a TPad. Only those deriving from TAttLine, TAttMarker and TAttFill are added, excluding TPave and TFrame derived classes.
x1, y1, x2, y2 are the Tlegend coordinates.
title is the legend title. By default it is " ".

If the pad contains some TMultiGraph or THStack the individual graphs or histograms in them are added to the TLegend.

2D & 3D Histograms




```
TH2F *h_2D = new TH2F("h_2D", "h_2D Title", xbin, xmin, xmax, ybin, ymin, ymax);

h_2D->Fill(x,y)
h_2D->Fill(x,y,w) //fill with weight


TH3F *h_3D = new TH3F("h_3D", "h_3D Title", xbin, xmin, xmax, ybin, ymin, ymax, zbin, zmin, zmax);

h_3D->Fill(x,y,z)
h_3D->Fill(x,y,z,w) //fill with weight
```

2D & 3D Histograms




```
{
gROOT->Reset();

TCanvas *cc1 = new TCanvas("cc1","cc1", 800, 600);
TCanvas *cc2 = new TCanvas("cc2","cc2", 800, 600);
gRandom->SetSeed();
   Float_t px, py, x, y, z;
TH2F *h_2D = new TH2F("h_2D", "h_2D Title", 40,-4,4,40,-20,20);
for (int i=0; i<10000;i++){
      gRandom->Rannor(px,py);
      h_2D->Fill(px-1,5*py);
      h_2D->Fill(2+0.5*px,2*py-10.,0.1);
}

TH3F *h_3D = new TH3F("h_3D", "h_3D Title", 15,-2,2,15,-2,2,15,0,4);
for (int i=0; i<10000;i++) {
      gRandom->Rannor(x, y);
      z = x*x + y*y;
      h_3D->Fill(x,y,z);
}
```

2D & 3D Histograms




```
gStyle->SetOptStat(0); //to remove statictics box
gStyle->SetPalette(1); //to define color palette

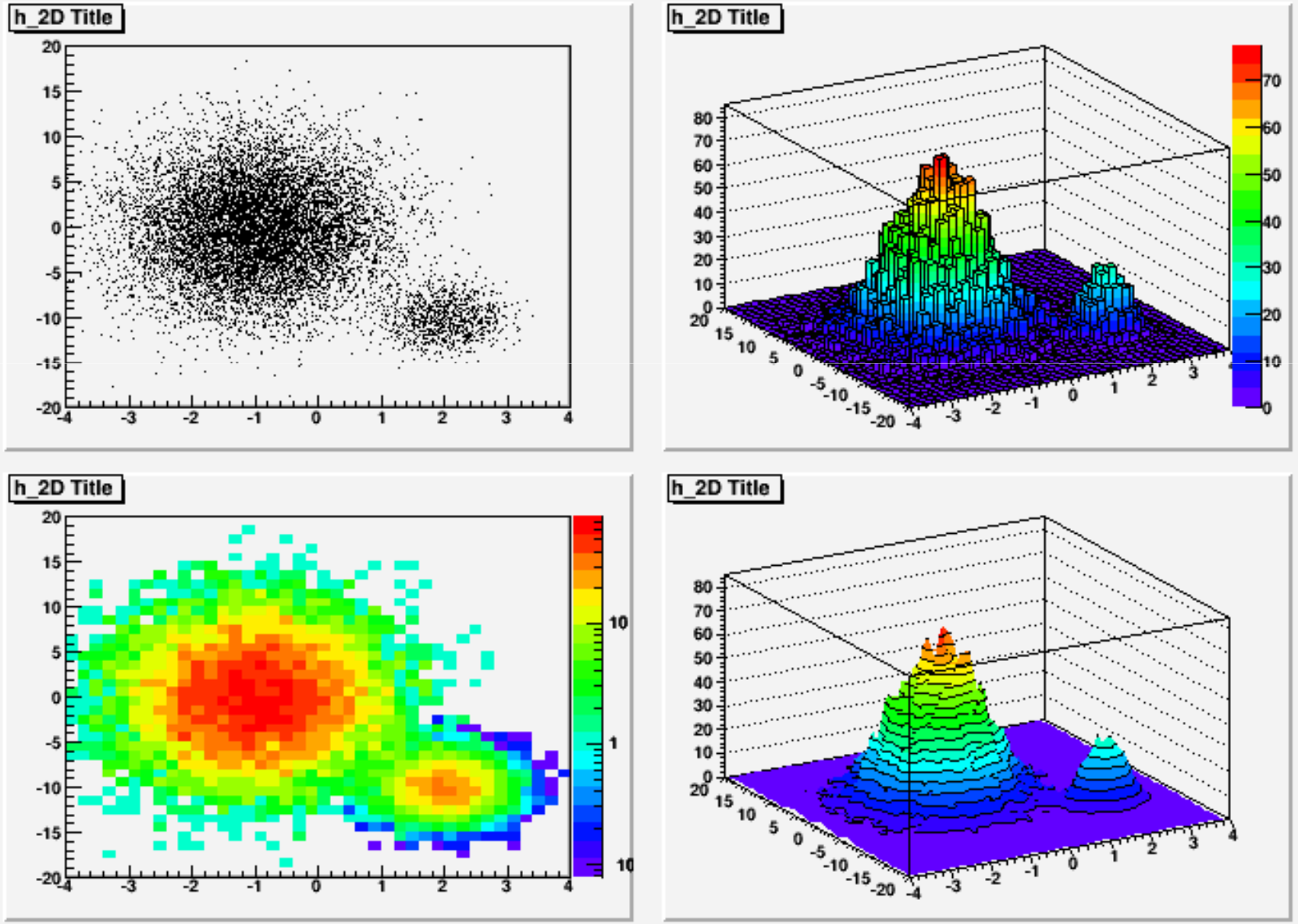
cc1->cd();
cc1->Divide(2,2);
cc1->cd(1);
h_2D->Draw();
cc1->cd(2);
h_2D->Draw("LEGO2Z");
cc1->cd(3);
gPad->SetLogz();
h_2D->Draw("COLZ");
cc1->cd(4);
h_2D->Draw("SURF2");
h_2D->Draw("CONT1 SAME");

cc2->cd();
cc2->Divide(2,2);
cc2->cd(1);
h_3D->Draw();
cc2->cd(2);
h_3D->Draw("BOX");
cc2->cd(3);
h_3D->SetFillColor(kRed);
h_3D->Draw("ISO");
cc2->cd(4);
TH2F *h_3D_pro = new TH2F("h_3D_pro", "h_3D_pro Title", 15,-2,2,15,0,4);
h_3D_pro =(TH2F*)h_3D->Project3D("xz");
h_3D_pro->Draw("CONT4Z");

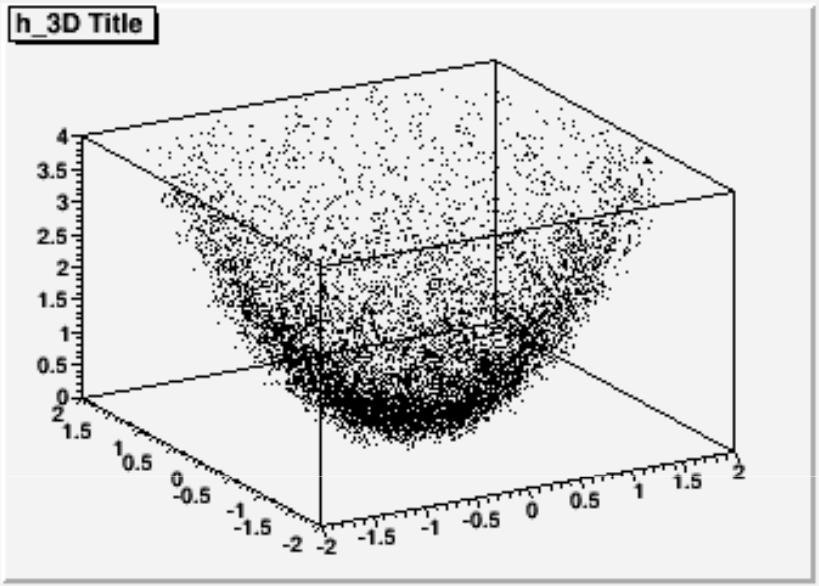
}
```

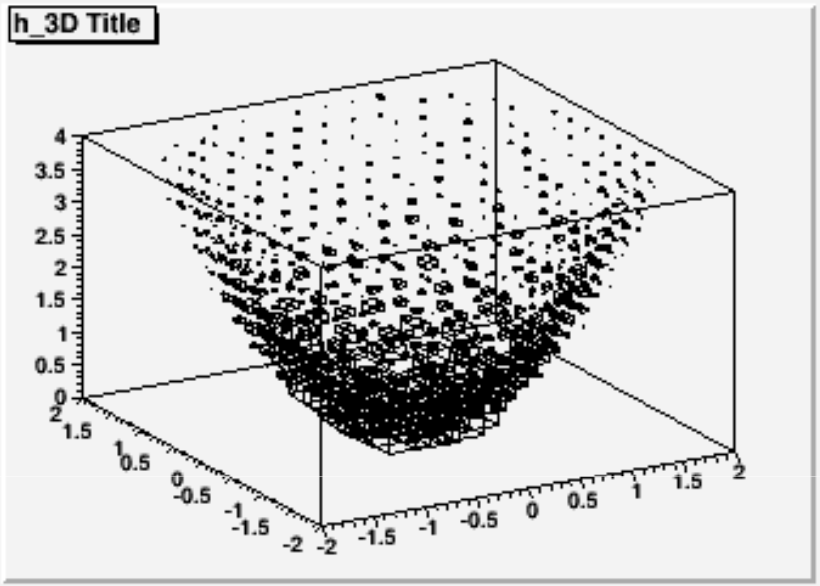
ROOT » HIST » HISTPAINTER » THistPainter

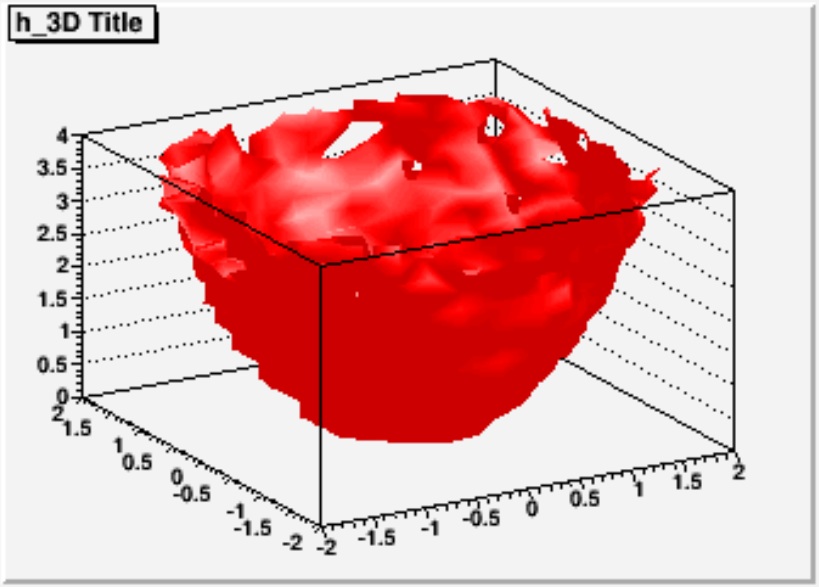
2D Drawing Options

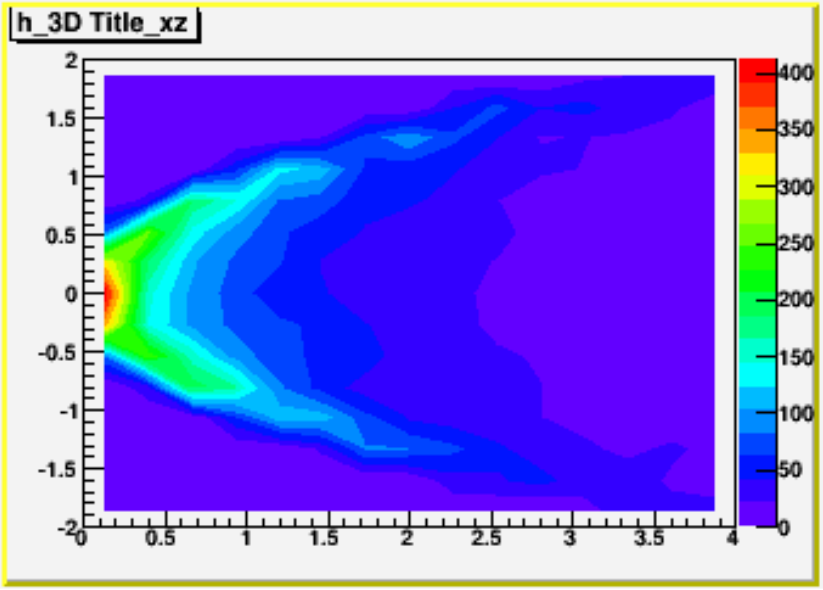


3D Drawing Options









Fitting a Histogram

```
TF1 *fit_nick = new TF1("fit_nick",func_name, xmin, xmax, Nparameters);
```

- `gaus`: Gaussian function with 3 parameters:
  `f(x) = p0*exp(-0.5*((x-p1)/p2)^2))`
- `expo`: an Exponential with 2 parameters: `f(x) = exp(p0+p1*x)`
- `polN`: a polynomial of degree N: `f(x) = p0 + p1*x + p2*x^2 +...`
- `landau`: Landau function with mean and sigma. This function has been adapted from the `CERNLIB` routine `G110 denlan`.

Fit Statistics

You can change the statistics box to display the fit parameters with the **`TStyle::SetOptFit(mode)`** method.
This parameter has four digits: `mode = pcev` (`default = 0111`)

- `p` = 1 print probability
- `c` = 1 print Chi-square/number of degrees of freedom
- `e` = 1 print errors (if `e=1`, `v` must be 1)
- `v` = 1 print name/values of parameters

For example, to print the fit probability, parameter names/values, and errors, use:

```
gStyle->SetOptFit(1011);
```

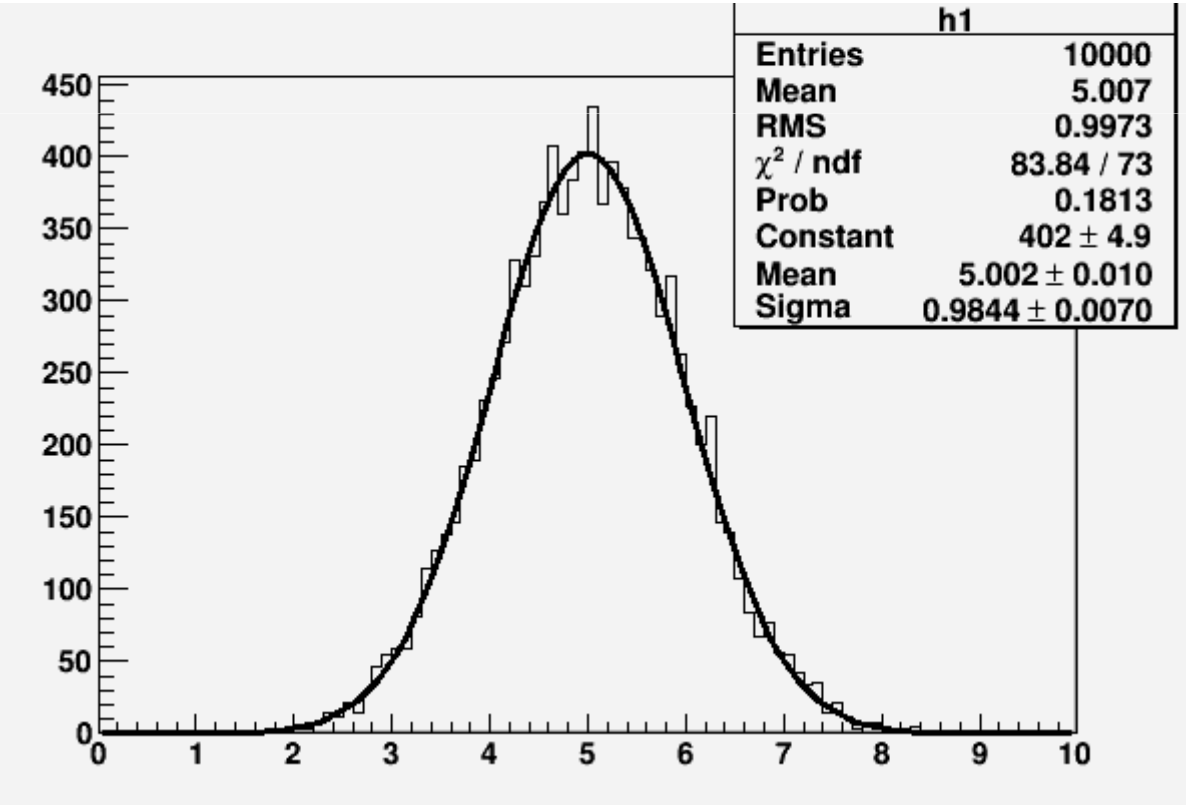
Fitting a Histogram

```
{
TH1F *h1= new TH1F("h1","",100, 0,10 );

for(int i =0; i<10000; i++) h1-> Fill(gRandom->Gaus(5,1));

TCanvas *c1 = new TCanvas();
c1->cd();

gStyle->SetOptFit(1111);
h1->Fit("gaus");
h1->Draw();
}
```



Fitting a Histogram

```
Double_t fitf(Double_t *v, Double_t *par){
      Double_t arg = 0;
      if (par[2] != 0) arg = (v[0] - par[1])/par[2];

      Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
      return fitval;
   }



void fit2(){

TH1F *h1= new TH1F("h1","",100, 0,10);
//TF1 *fit_nick = new TF1("fit_nick",func_name, xmin, xmax, Nparameters);
TF1 *func = new TF1("fit",fitf,-3,3,3);

for(int i =0; i<10000; i++) h1-> Fill(gRandom->Gaus(5,1));

func->SetParameters(500,h1->GetMean(),h1->GetRMS());
h1->Fit("fit");
h1->Draw();

cout<<"Amplitude = "<<func->GetParameter(0)<<endl;
cout<<"Mean = "<<func->GetParameter(1)<<endl;
cout<<"RMS = "<<func->GetParameter(2)<<endl;
}
```

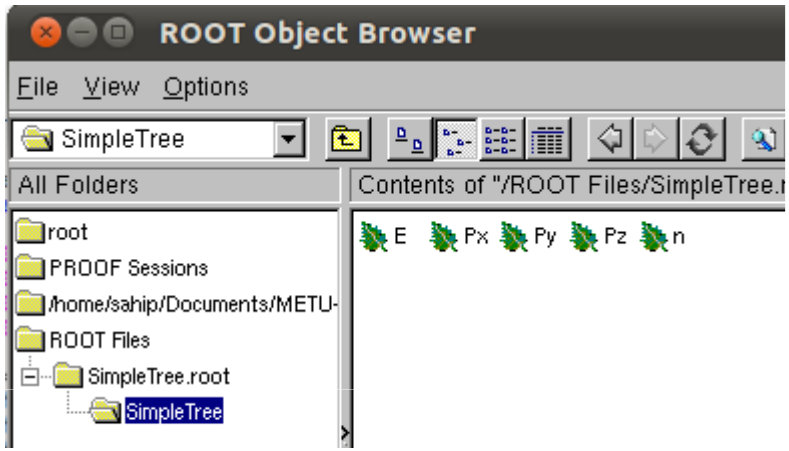
Creating a TTree

```
{
TFile *f = new TFile("SimpleTree.root","recreate");
gRandom->SetSeed(0);
// for each event, you want up to 50000 candidates (or whatever)
Int_t n;
Double_t E[5000];
Float_t Px[5000];
Float_t Py[5000];
Float_t Pz[5000];

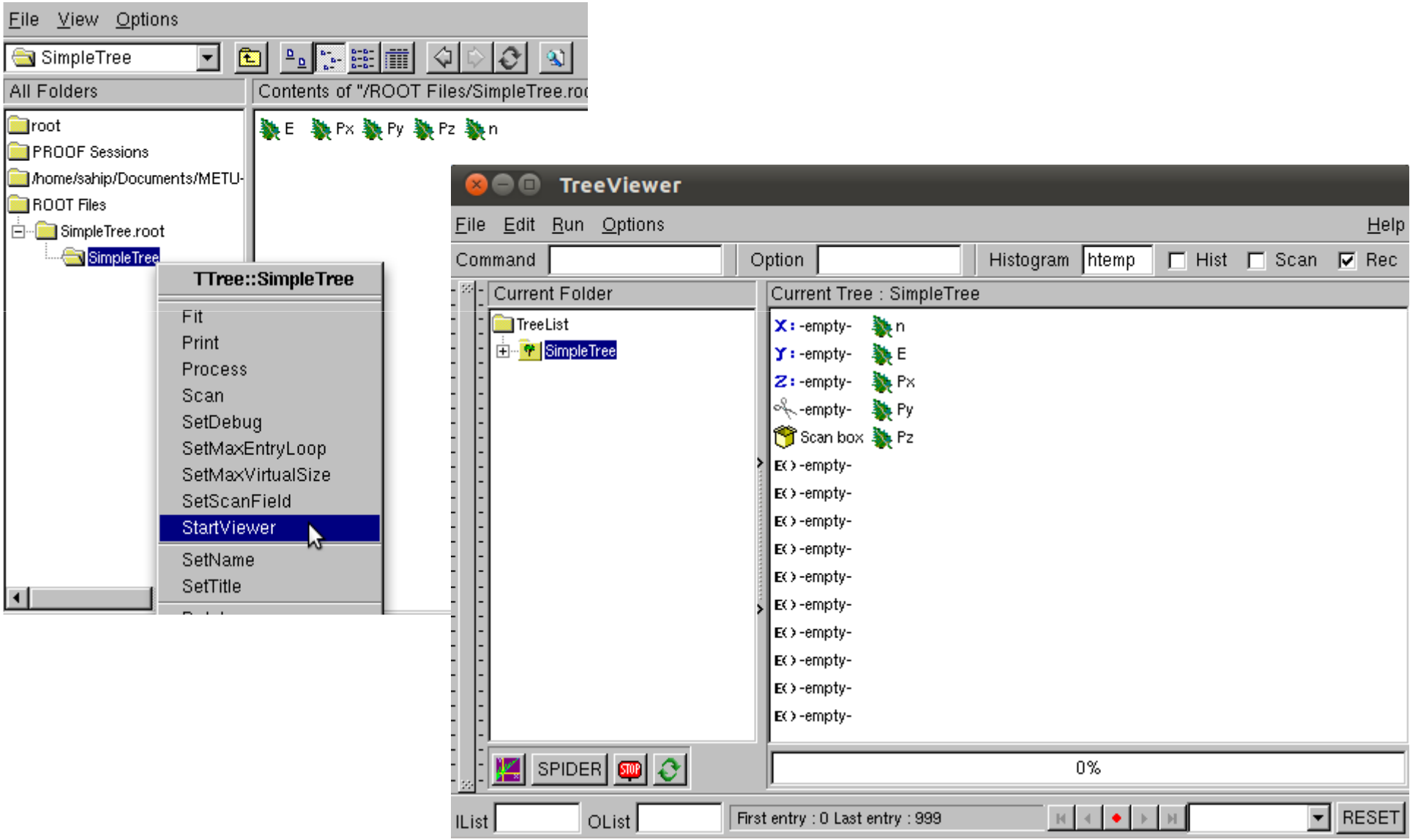
TTree* pMyTree= new TTree("SimpleTree","An example of a tree");
pMyTree->Branch("n",&n,"n/I");
pMyTree->Branch("E",&E,"E[n]/D");
pMyTree->Branch("Px",&Px,"Px[n]/F");
pMyTree->Branch("Py",&Py,"Py[n]/F");
pMyTree->Branch("Pz",&Pz,"Pz[n]/F");

for (Int_t i=0;i<1000;i++){ // fill in 1000 events
n=5000*gRandom->Rndm();   // the number of entries per events is not fixed

for (Int_t j=0;j<n;j++){
E[j]=gRandom->Gaus(10.,2.);
Px[j]=gRandom->Gaus(5.,1.);
Py[j]=gRandom->Gaus(5.,1.);
Pz[j]=gRandom->Gaus(7.,0.6);
}
pMyTree->Fill();
}
pMyTree->Print();
pMyTree->Write();
f->Close();
delete f;
}
```



Tree Viewer


File View Options
SimpleTree
All Folders
Contents of "/ROOT Files/SimpleTree.roo
E Px Py Pz n
root
PROOF Sessions
/home/sahip/Documents/METU-
ROOT Files
SimpleTree.root
SimpleTree
TTree::SimpleTree
Fit
Print
Process
Scan
SetDebug
SetMaxEntryLoop
SetMaxVirtualSize
SetScanField
StartViewer
SetName
SetTitle
TreeViewer
File Edit Run Options Help
Command
Option
Histogram htemp
Hist
Scan
Rec
Current Folder
Current Tree : SimpleTree
TreeList
SimpleTree
X : -empty-
Y : -empty-
Z : -empty-
-empty-
Scan box
n
E
Px
Py
Pz
SPIDER
STOP
0%
IList
OList
First entry : 0 Last entry : 999
RESET

View a TTree

```
root [0] TFile *f =TFile::Open("SimpleTree.root")
root [1] f.ls()
TFile**         SimpleTree.root
 TFile*         SimpleTree.root
  KEY: TTree    SimpleTree;1    An example of a tree
root [2] TTree *t = (TTree*)f->Get("SimpleTree");
root [3] t.ls()
OBJ: TTree      SimpleTree      An example of a tree : 0 at: 0x8bf6828
root [4] t->Print()
******************************************************************************
*Tree    :SimpleTree: An example of a tree                                   *
*Entries :     1000 : Total =        49814305 bytes  File  Size =   43166148 *
*        :          : Tree compression factor =   1.15                       *
******************************************************************************
*Br    0 :n          : n/I                                                    *
*Entries :     1000 : Total  Size=       4614 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    1 :E          : E[n]/D                                                 *
*Entries :     1000 : Total  Size=   19913367 bytes  File Size  =   17569761 *
*Baskets :      647 : Basket Size=      32000 bytes  Compression=   1.13     *
*.............................................................................*
*Br    2 :Px         : Px[n]/F                                                *
*Entries :     1000 : Total  Size=    9965418 bytes  File Size  =    8515664 *
*Baskets :      381 : Basket Size=      32000 bytes  Compression=   1.17     *
*.............................................................................*
*Br    3 :Py         : Py[n]/F                                                *
*Entries :     1000 : Total  Size=    9965418 bytes  File Size  =    8515565 *
*Baskets :      381 : Basket Size=      32000 bytes  Compression=   1.17     *
*.............................................................................*
*Br    4 :Pz         : Pz[n]/F                                                *
*Entries :     1000 : Total  Size=    9965418 bytes  File Size  =    8490206 *
*Baskets :      381 : Basket Size=      32000 bytes  Compression=   1.17     *
*.............................................................................*
```

View an Entry

```
root [5] t->Show(511)
======> EVENT:511
 n               = 83
 E               = 11.8219,
                   10.3529, 9.48515, 10.8071, 7.7706, 11.5606,
                   8.90923, 9.7761, 11.2901, 11.2006, 9.31022,
                   10.763, 11.1418, 6.37039, 11.5307, 8.01946,
                   11.6578, 8.22838, 7.69951, 14.0844
 Px              = 5.16611,
                   5.9918, 1.7149, 4.51329, 5.08942, 5.15023,
                   4.81965, 6.16279, 5.96522, 6.24578, 5.44932,
                   6.92044, 3.91019, 4.27655, 5.21861, 4.65351,
                   4.41642, 4.7973, 4.90052, 4.08844
 Py              = 4.23325,
                   4.84083, 5.66527, 5.01014, 5.79457, 2.85575,
                   3.78535, 6.67048, 6.93939, 3.51474, 5.32159,
                   5.19602, 6.36595, 5.52877, 5.42982, 6.52629,
                   6.9459, 6.20705, 5.57639, 3.78599
 Pz              = 6.08486,
                   7.82969, 7.02946, 7.31801, 8.05795, 6.67152,
                   7.2133, 6.35405, 7.46586, 6.59077, 7.44913,
                   7.26589, 7.31643, 6.78815, 7.76111, 6.49753,
                   7.70481, 7.66222, 7.42085, 7.40755
```

```
root [6] t->Scan("Px:Py:Pz","E>13")
************************************************************
*    Row   * Instance *        Px *        Py *        Pz *
************************************************************
*        0 *       20 *  2.415447 * 3.0202052 * 6.2658257 *
*        0 *       27 * 4.5105466 * 5.1591963 * 7.4365420 *
*        0 *       37 * 4.2716021 * 6.1484923 * 6.9797821 *
*        0 *       38 * 6.3484106 * 4.6090154 * 6.0907464 *
*        0 *       62 * 4.1705055 * 5.6820163 * 6.2514858 *
*        0 *       75 * 5.9908289 * 3.9871811 * 7.3174977 *
*        0 *      120 * 4.9582252 * 5.8506879 * 5.8685717 *
*        0 *      134 * 5.8118195 * 5.6076307 * 7.6930637 *
*        0 *      155 * 5.0427470 * 4.4596400 * 7.5213441 *
*        0 *      204 * 5.7708339 * 5.6943559 * 7.5327315 *
*        0 *      214 * 5.1713051 * 4.7301683 * 7.0819129 *
*        0 *      216 * 6.2895302 * 4.9783506 * 7.2643475 *
*        0 *      226 * 5.4870266 * 4.3375368 * 6.0564212 *
*        0 *      229 * 4.1922369 * 5.8159542 * 5.6723938 *
*        0 *      235 * 3.9523932 * 3.1352086 * 6.9435815 *
*        0 *      237 * 4.1801471 * 3.6749422 * 7.1382741 *
*        0 *      239 * 5.3821206 * 4.4594950 * 6.2897706 *
*        0 *      256 * 5.1144018 * 3.6491513 * 7.3667731 *
*        0 *      298 * 4.8394932 * 4.8084921 * 7.4467949 *
*        0 *      302 * 5.7643408 * 5.1806216 * 7.7340064 *
*        0 *      318 * 5.6942563 * 4.7240548 * 6.7441024 *
*        0 *      360 * 4.8304753 * 4.2699794 * 6.6585359 *
*        0 *      364 * 2.8974437 * 5.9857754 * 6.9879112 *
*        0 *      366 * 4.8883719 * 5.8918213 * 7.5586490 *
*        0 *      369 * 6.3947782 * 5.5404529 * 6.7790231 *
Type <CR> to continue or q to quit ==>
```

Scan over Branch

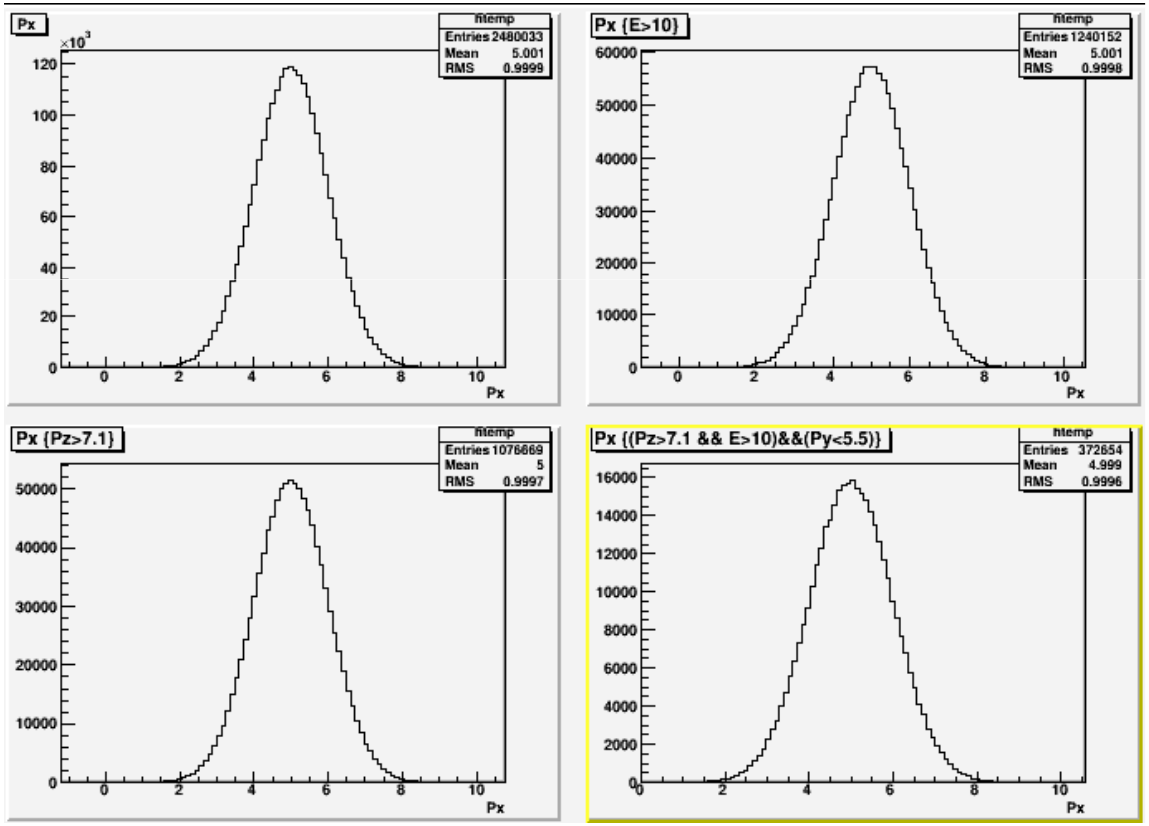
Reading a TTree

```
{
TFile *f = TFile::Open("SimpleTree.root");
TTree *t = (TTree*)f->Get("SimpleTree");
TCanvas *c1 = new TCanvas("c1", "c1",800,600);

c1->Divide(2,2);
c1->cd(1);
t->Draw("Px");
c1->cd(2);
t->Draw("Px","E>10");

TCut cut_Pz = "Pz>7.1";
c1->cd(3);
t->Draw("Px", cut_Pz);

c1->cd(4);
TCut cut_both = "Pz>7.1 && E>10";
t->Draw("Px", cut_both && "Py<5.5");
}
```



Plot and Save the histograms

```
{
TFile *f = TFile::Open("SimpleTree.root");
TTree *t = (TTree*)f->Get("SimpleTree");

TFile *f_out = new TFile("HistFromTree.root", "RECREATE");

TCut cut_Pz = "Pz>7.1";
TCut cut_both = "Pz>7.1 && E>10";

TH1F *h1;
TH1F *h2 = new TH1F("h2","h2", 100, 0, 20);

TH2F *h3;
TH2F *h4 = new TH2F("h4","h4", 100, 0, 20, 100, 0, 20);

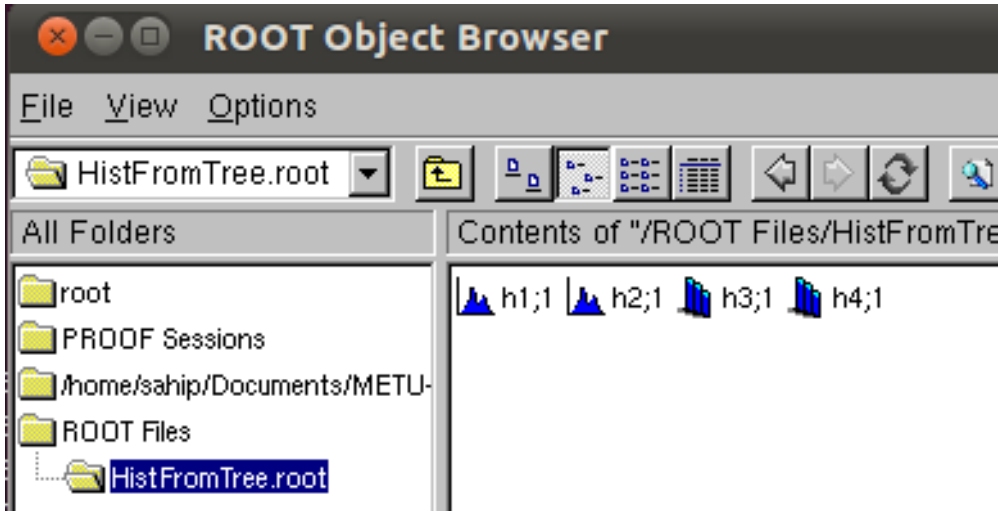
t->Draw("Px>>h1(10,0,20)");
t->Draw("sqrt(Px)>>h2","E>10");

t->Draw("Px:Pz>>h3(10,0,20,10,0,20)", cut_Pz);
t->Draw("Px:(E*E)", cut_both && "Py<5.5");

h1 = (TH1F*)gDirectory->Get("h1");
h2 = (TH1F*)gDirectory->Get("h2");
h3 = (TH2F*)gDirectory->Get("h3");
h4 = (TH2F*)gDirectory->Get("h4");

h1->SetTitle("any thing");

h1->Write();
h2->Write();
h3->Write();
h4->Write();
f_out->Close();
}
```



Scanning a TTree

```
{
TFile *f = TFile::Open("SimpleTree.root");
TTree *t = (TTree*)f->Get("SimpleTree");

TCut cut_E = "13.001<E && E<14.000";
TCut cut_more = "(Px + Py)>sqrt(Pz)";
Int_t n_cand;

t->SetScanField(0);
n_cand = t->Scan("n:E:Px:Py:Pz", cut_E && cut_more);

cout<<"Number of Candidates = "<<n_cand<<endl;
}
```

```
*        25 *      1144 *      1269 * 13.202915 * 3.5355043 * 5.0025601 * 7.2678828 *
*        25 *      1172 *      1269 * 13.951913 * 3.4824037 * 5.4915385 * 7.0050497 *
*        25 *      1176 *      1269 * 13.129513 * 7.4804701 * 4.3536672 * 6.0447416 *
*        25 *      1187 *      1269 * 13.697414 * 6.4904317 * 5.5387902 * 6.3781523 *
*        25 *      1225 *      1269 * 13.144189 * 4.4679236 * 3.4619309 * 7.2369699 *
*        25 *      1235 *      1269 * 13.416295 * 6.3650646 * 4.4754891 * 8.0118513 *
*        26 *        25 *       111 * 13.307999 * 5.7381243 * 4.5471320 * 7.2405266 *
*        26 *        61 *       111 * 13.080853 * 5.1640324 * 5.4534726 * 7.4173822 *
*        26 *        62 *       111 * 13.737961 * 5.7007074 * 5.1810188 * 6.0090780 *
*        27 *        32 *      3336 * 13.992061 * 4.5376348 * 8.3902425 * 6.7258725 *
*        27 *        39 *      3336 * 13.014567 * 3.6548173 * 5.1706137 * 6.4480633 *
*        27 *        68 *      3336 * 13.142185 * 5.2713441 * 5.2035727 * 6.5590457 *
*        27 *        91 *      3336 * 13.934973 * 4.3383884 * 6.0009412 * 7.9147329 *
*        27 *        99 *      3336 * 13.574074 * 4.2782797 * 4.4822106 * 7.0927839 *
*        27 *       123 *      3336 * 13.247356 * 5.5579948 * 6.8066926 * 5.9230794 *
*        27 *       152 *      3336 * 13.061938 * 5.6932210 * 4.1634464 * 7.2977752 *
*        27 *       172 *      3336 *  13.84182 * 5.3630709 * 4.4689698 * 7.3606171 *
```

Get scan results to a file

```
root -b -q scan_SimpleTree.C >scan.log
```

```
*******************************************
*                                         *
*       W E L C O M E  to  R O O T        *
*                                         *
*   Version  5.18/00b     10 March 2008   *
*                                         *
*  You are welcome to visit our Web site  *
*          http://root.cern.ch            *
*                                         *
*******************************************

ROOT 5.18/00b (branches/v5-18-00-patches@22563, Apr 06 2010, 01:56:00 on linux)

CINT/ROOT C/C++ Interpreter version 5.16.29, Jan 08, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.

Processing scan_SimpleTree.C...
*****************************************************************************************
*    Row   * Instance *         n *         E *        Px *        Py *        Pz *
*****************************************************************************************
*        0 *       27 *      1359 * 13.705671 * 4.5105466 * 5.1591963 * 7.4365420 *
*        0 *       38 *      1359 * 13.802341 * 6.3484106 * 4.6090154 * 6.0907464 *
*        0 *      134 *      1359 * 13.349669 * 5.8118195 * 5.6076307 * 7.6930637 *
*        0 *      155 *      1359 * 13.816945 * 5.0427470 * 4.4596400 * 7.5213441 *
*        0 *      204 *      1359 * 13.419538 * 5.7708339 * 5.6943559 * 7.5327315 *
*        0 *      214 *      1359 * 13.664789 * 5.1713051 * 4.7301683 * 7.0819129 *
*        0 *      237 *      1359 * 13.951655 * 4.1801471 * 3.6749422 * 7.1382741 *
*        0 *      256 *      1359 * 13.092885 * 5.1144018 * 3.6491513 * 7.3667731 *
*        0 *      302 *      1359 * 13.728945 * 5.7643408 * 5.1806216 * 7.7340064 *
*        0 *      371 *      1359 * 13.469422 * 3.6909067 * 4.7993240 * 5.3941192 *
*        0 *      385 *      1359 * 13.348177 * 5.8166604 * 3.4019532 * 7.6830515 *
*        0 *      406 *      1359 * 13.560100 * 7.2534594 * 5.7838411 * 6.7845282 *
*        0 *      431 *      1359 * 13.883697 * 4.8645153 * 3.0847601 * 7.4035944 *
*        0 *      437 *      1359 * 13.295088 * 5.3227801 * 6.0427827 * 7.3000707 *
```

Next:
DATA ANALYSIS