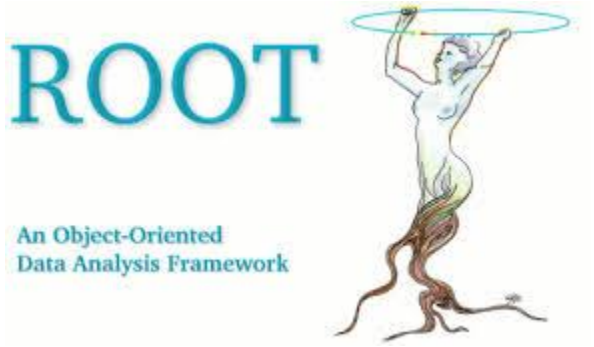


Tree: Chains & Analysis

Harinder Singh Bawa
California State University Fresno

Outline

ROOT Browser(GUI) 

ROOT Scripting 

Histograms 1-D/2-D/3-D 

Histograms Operations 

Graphs, Error-bars (Assymetric/symmetric) 

Functions & Fitting 

Trees 

Input/Output, chaining and plotting

Why Should You Use a Tree?

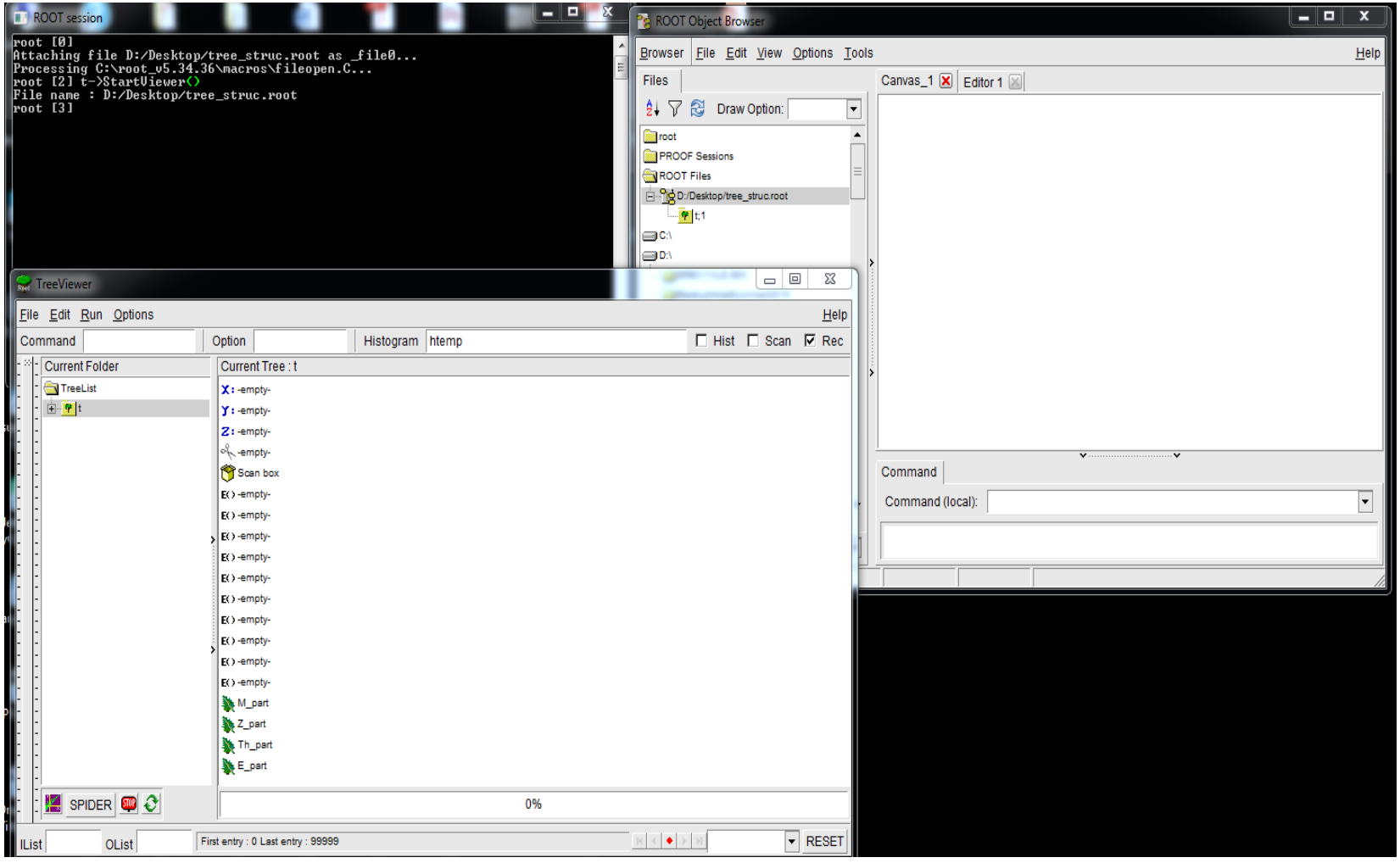
- We saw how objects can be saved in ROOT files. In case you want to store large quantities of same-class objects, ROOT has designed the TTree and TNtuple classes specifically for that purpose.
- The TTree class is optimized to reduce disk space and enhance access speed. A TNtuple is a TTree that is limited to only hold floating-point numbers; a TTree on the other hand can hold all kind of data, such as objects or arrays in addition to all the simple types.

Using a tree

Graphically (GUI) & Command Line

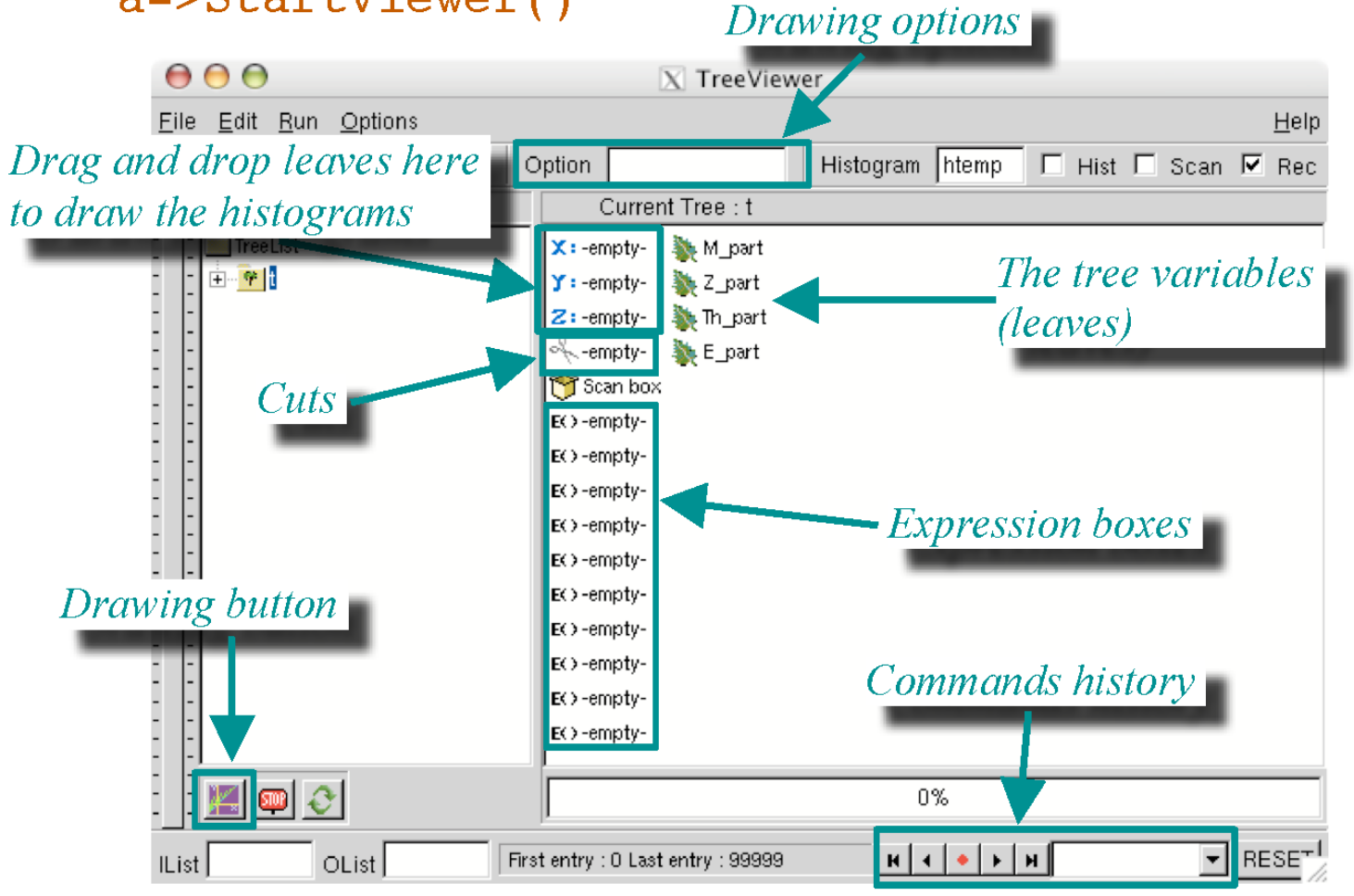
For Practice.

http://zimmer.csufresno.edu/~hbawa/tree_struc.root

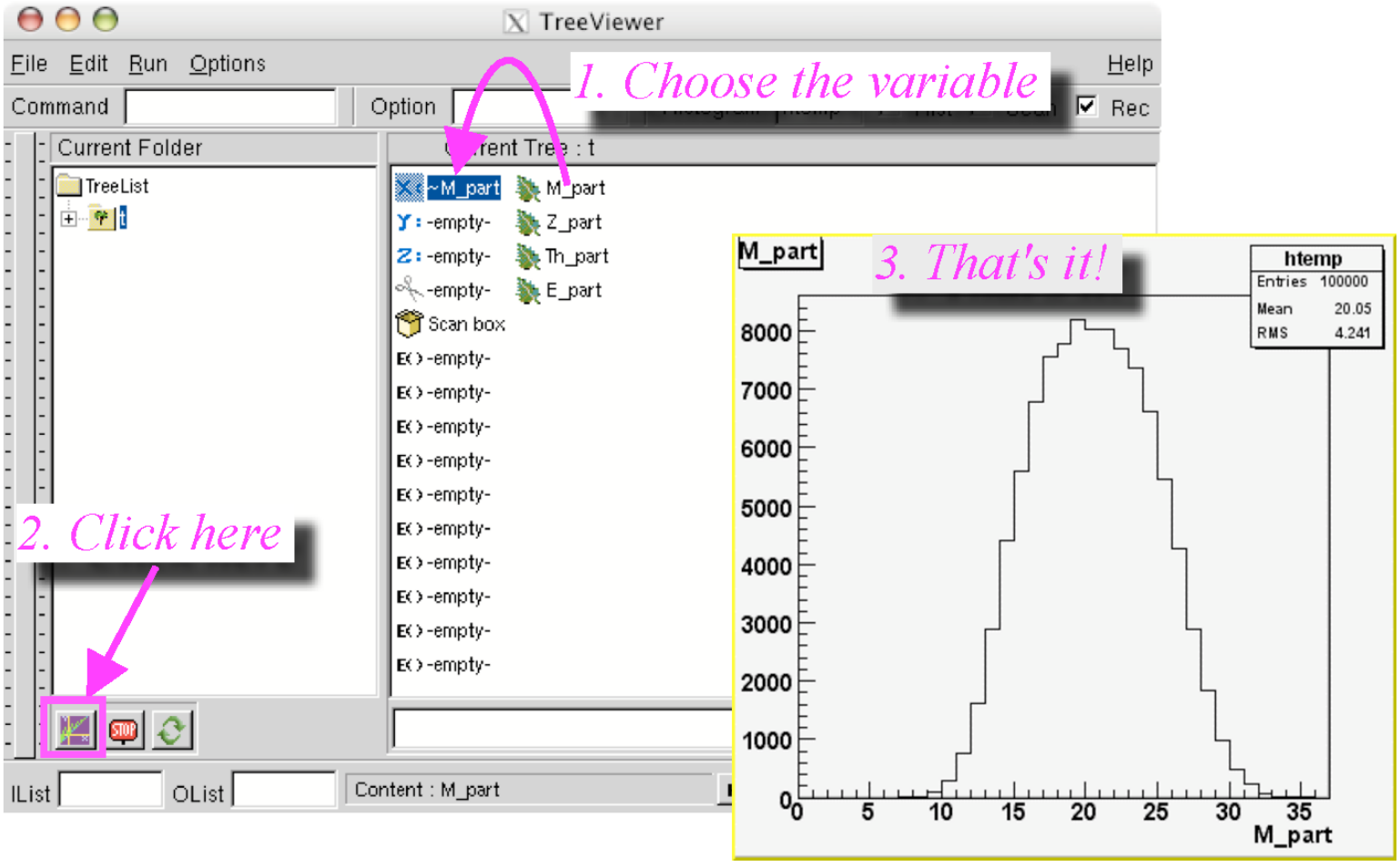


The graphical interface

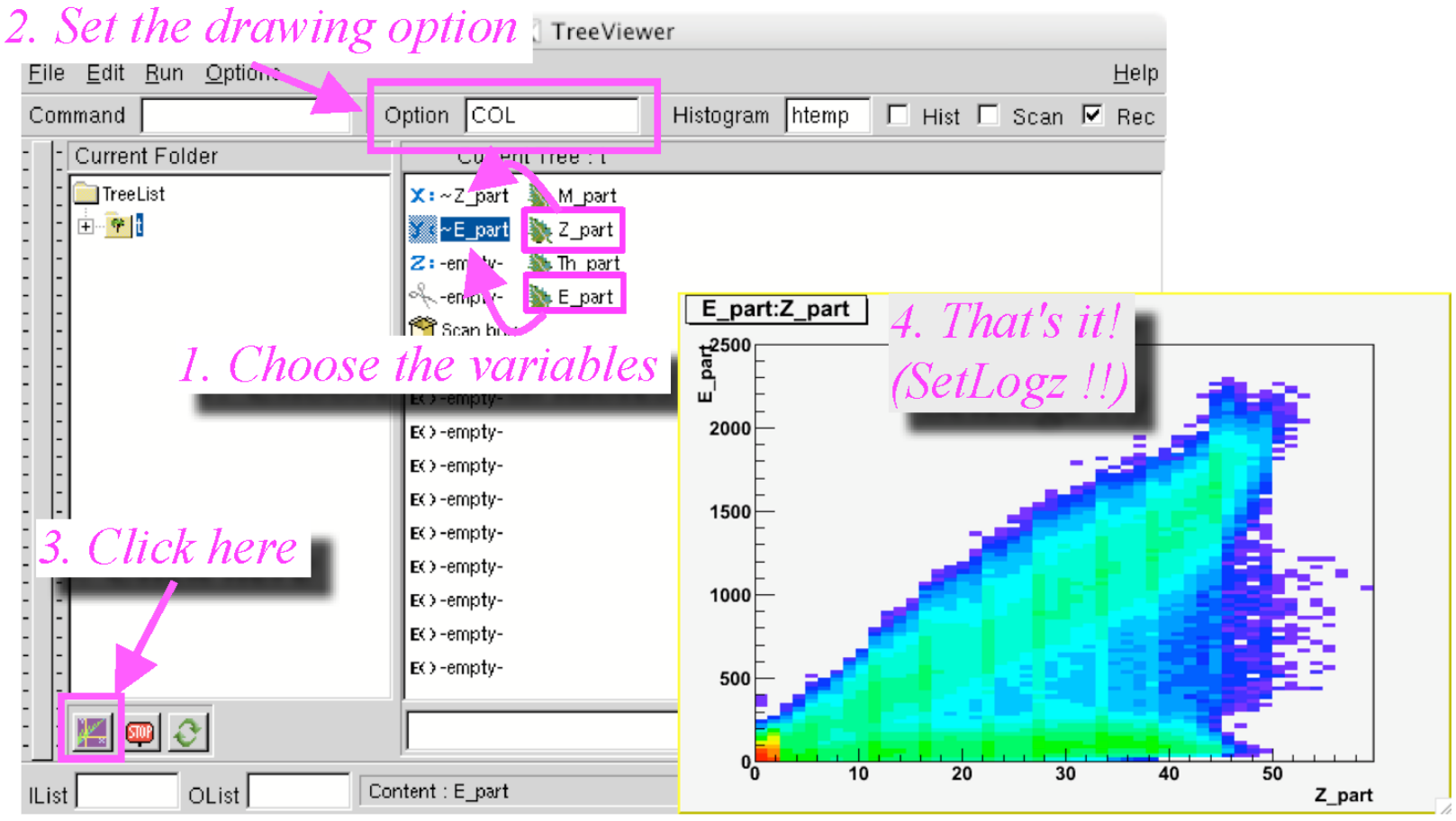
```
a->StartViewer()
```



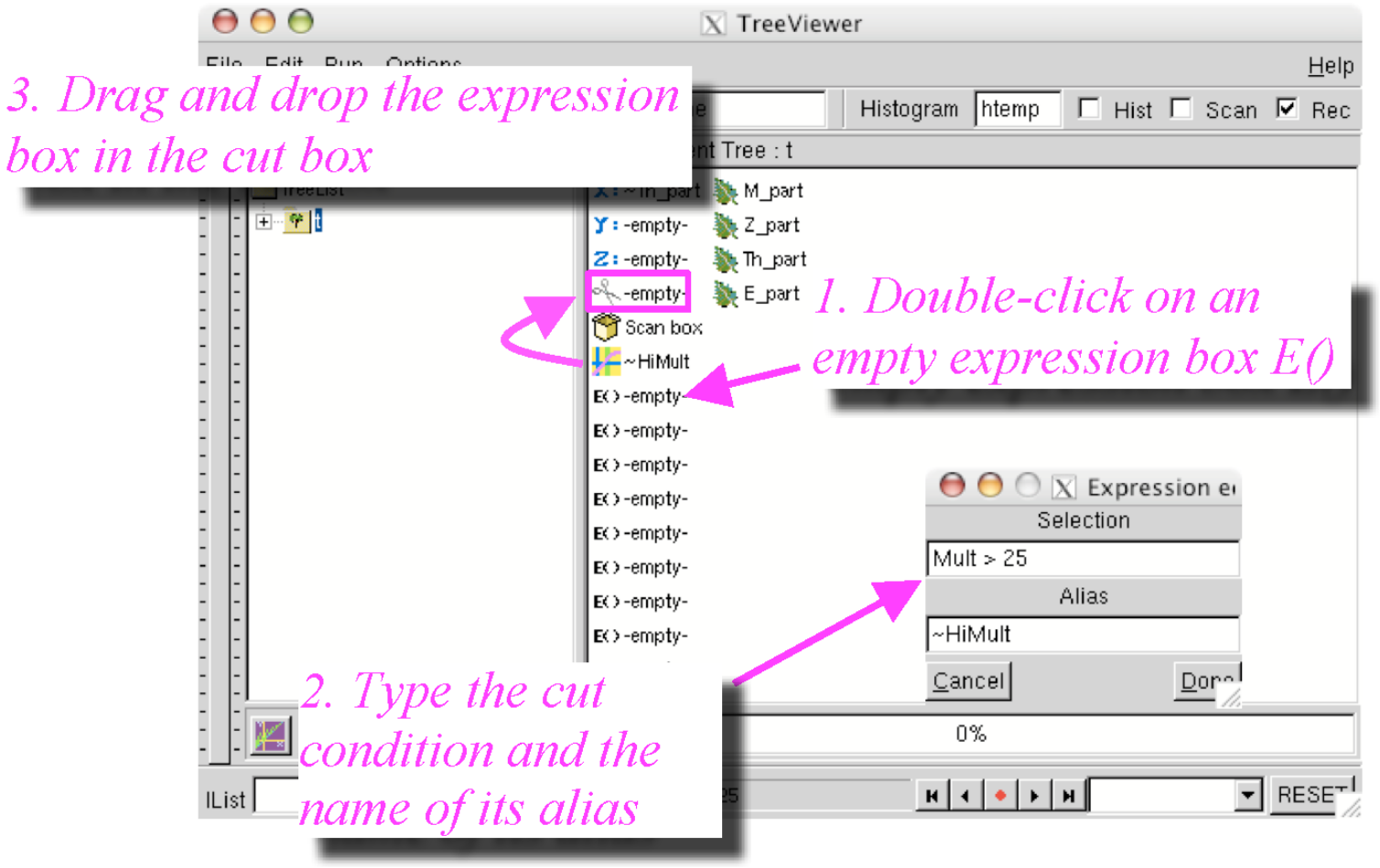
Plotting a 1D histogram



Plotting a 2D histogram

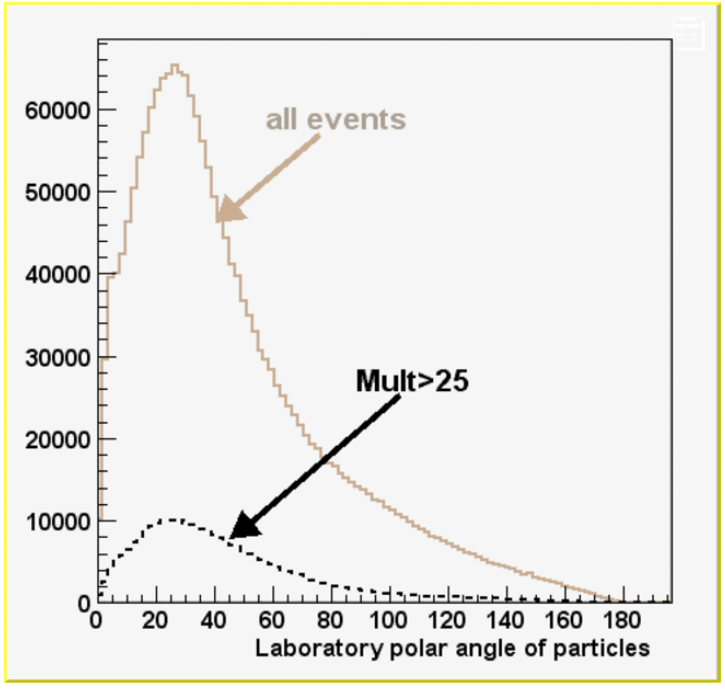


Using cuts

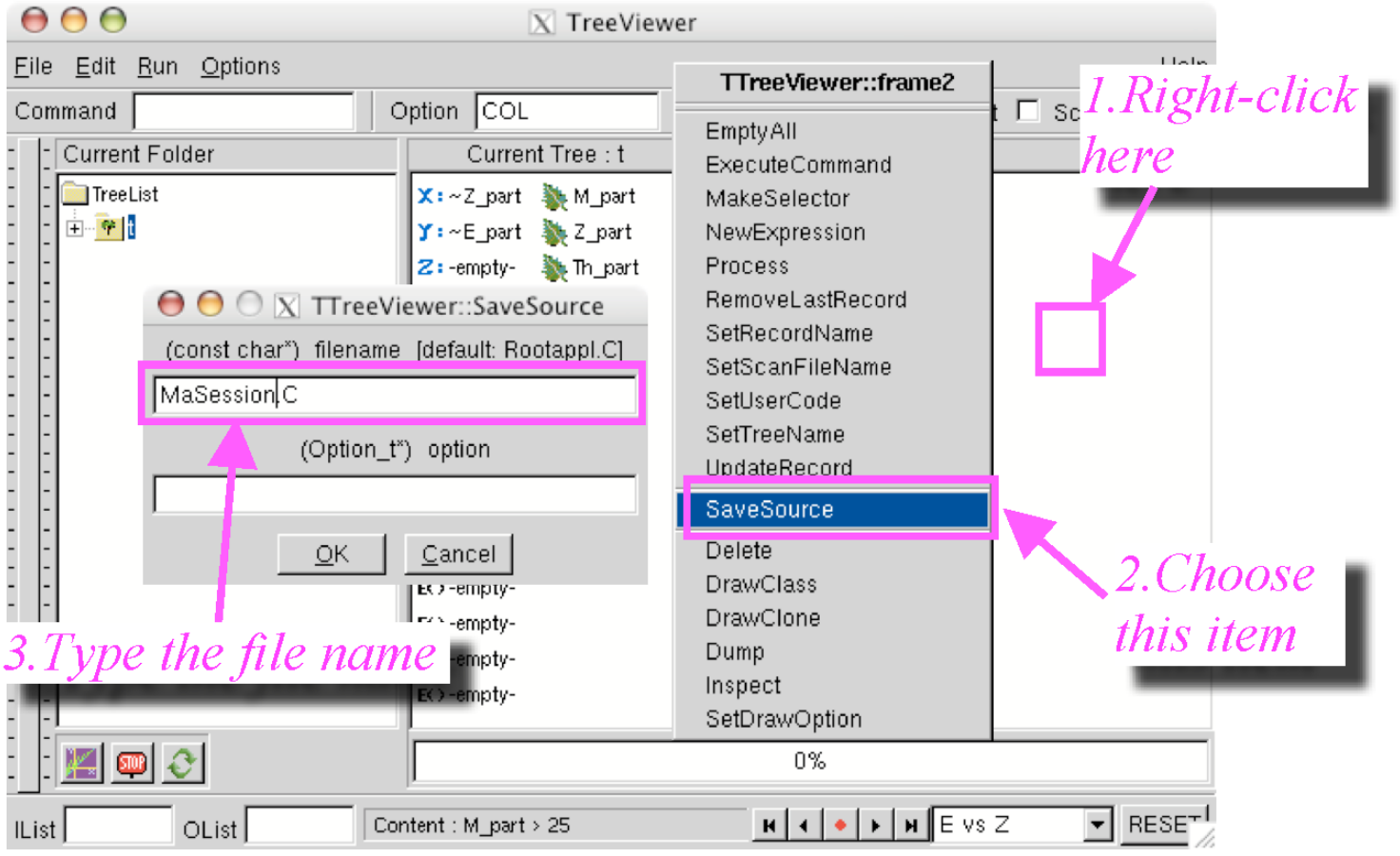

TreeViewer
Help
Histogram
htemp
Hist
Scan
Rec
Tree : t
M_part
Z_part
Th_part
E_part
-empty-
Scan box
~HiMult
E( ) -empty-
3. Drag and drop the expression box in the cut box
1. Double-click on an empty expression box E()
Expression e
Selection
Mult > 25
Alias
~HiMult
Cancel
Done
0%
RESET
2. Type the cut condition and the name of its alias

Using cuts (2)

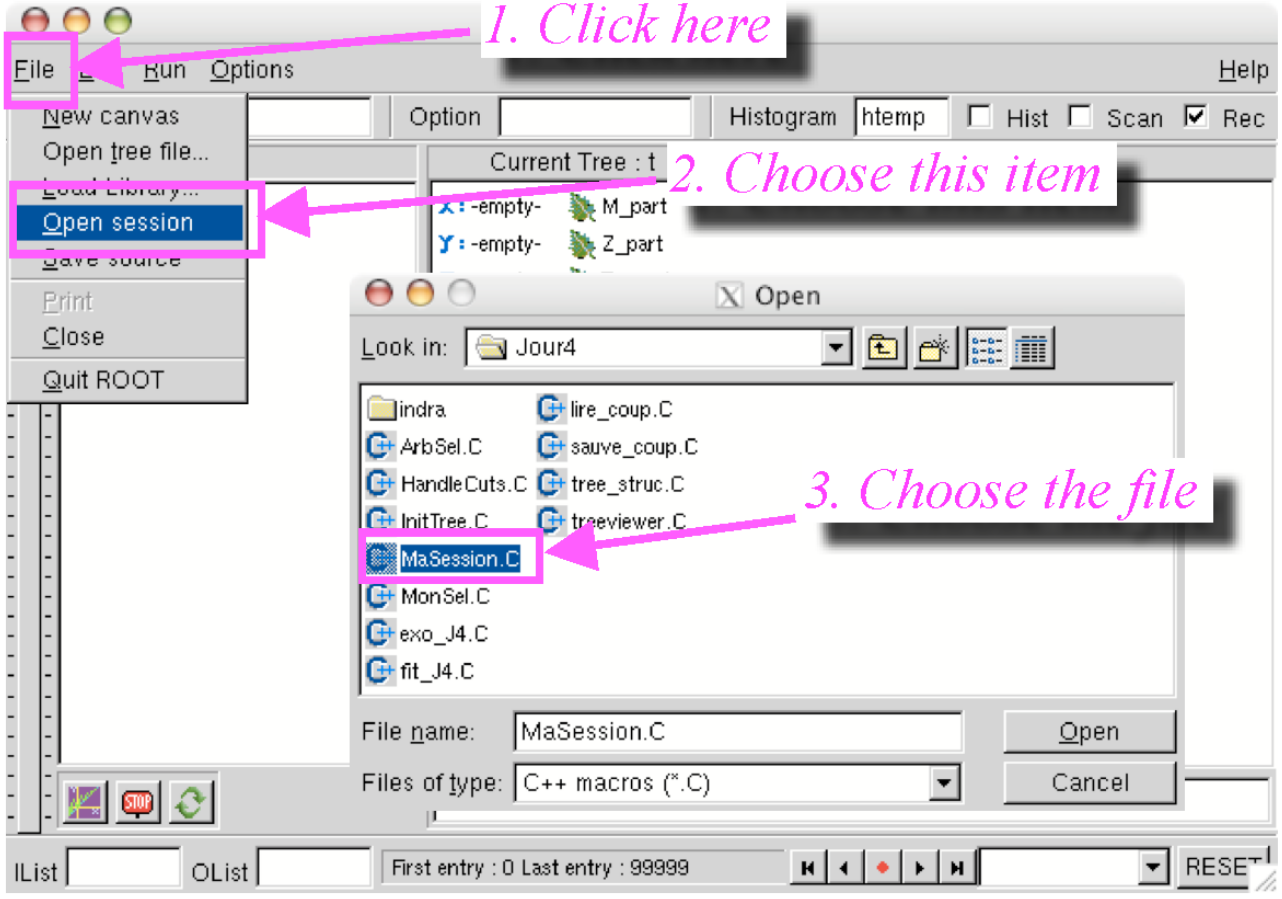
- Drag and drop the **`Th_part`** variable on the *x* axis
- Drag and drop the cut in the "scissors box"
- Double-click on the "scissors box" to disable the cut selection (red line)
- Draw the histogram **without the cut selection**
- Enable the cut selection
- Type `"same"` in the drawing option field
- Draw the histogram **with the cut selection**
- Record the display
- Perfect the presentation of the figure !



Save it...


TreeViewer
File Edit Run Options
Command
Option COL
Current Folder
TreeList
Current Tree : t
X : ~Z_part
Y : ~E_part
Z : -empty-
M_part
Z_part
Th_part
TTreeViewer::SaveSource
(const char*) filename [default: Rootappl.C]
MaSession.C
(Option_t*) option
OK
Cancel
TTreeViewer::frame2
EmptyAll
ExecuteCommand
MakeSelector
NewExpression
Process
RemoveLastRecord
SetRecordName
SetScanFileName
SetUserCode
SetTreeName
UpdateRecord
SaveSource
Delete
DrawClass
DrawClone
Dump
Inspect
SetDrawOption
0%
IList
OList
Content : M_part > 25
E vs Z
1.Right-click here
2.Choose this item
3.Type the file name

Everything is not lost…



Tree on the Command Line

- The contents of the tree can be drawn from the command-line:

Draw(const char *exp, const char *cut, Option_t *option, Long64_t nent, Long64_t first)

exp: expression describing what to draw e.g “y:x”, or “sqrt(x/y*z*z)”. For 2-D(or 3-D) plots, expressions are separated by a “:”. Convention: z:y:x. Statement “x>>histoname” will save to predefined histogram.

cut: expression describing some conditions, e.g “z>0”

option: drawing option(see histograms)

Example:

ROOT [0] TTree* mytree=new TTree(“mytree”, “Test Tree”);

ROOT[1] Double_t values[3];

ROOT{2]TBranch *b2=mytree->Branch(“b2”,values,”x/D:y:z”);

ROOT[3] macroToFillTree();

ROOT[4] mytree->Draw(“sqrt(z):x*y”, “z>0”,”SURF4”,1000,10);

Tree on the Command Line(Cont'd)

- The structure of the Tree can be printed:

```
mytree->Print();
```

```
******************************************************************************
*Tree    :mytree    : TestTree                                                 *
*Entries :        2 : Total =            1281 bytes  File  Size =          0 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :b2        : a/D:b:c:d:e
*Entries :        2 : Total  Size=       1001 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
```

Tree on the Command Line(Cont'd)

- You can get a list of (part of) the contents:

```
mytree->Scan("a:b");
```

```
*************************************
*     Row    *           a *           b *
*************************************
*           0 *           0 *           0 *
*           1 *           1 *           0 *
*************************************
```

Tree on the Command Line(Cont'd)

- You can also inspect the contents of a specific entry numerically:

```
mytree->Show(1);
```

```
======> EVENT:1
 a                    = 1
 b                    = 0
 c                    = 0
 d                    = -3.14
 e                    = 0
```

Accessing the tree data

- Selecting the events and print variables values:

```
a->Scan("Mult:Z[30]:Energie[30]","Mult>30","",1000,0)
```

Selection

Event number

```
*************************************************
*     Row   *       Mult *     Z[30] * Energie[3 *
*************************************************
*      46 *         32 *         2 * 47.778400 *
*      95 *         31 *         2 * 48.006801 *
*     399 *         31 *         1 * 28.520700 *
*     461 *         31 *         2 * 67.939399 *
*     628 *         32 *         2 * 69.046302 *
*************************************************
==> 5 selected entries
```

Various commands

Command	Action
T->Print();	Prints the content of the tree
T->Scan();	Scans the rows and columns
T->Draw("x");	Draw a branch of tree
How to apply cuts: T->Draw("x","x>0"); T->Draw("x","x>0 && y>0");	 Draw "x" when "x>0" Draw "x" when both x >0 and y >0
T->Draw("y"," ","same");	Superimpose "y" on "x"
T->Draw("y:x");	Make "y vs x" 2d scatter plot
T->Draw("z:y:x");	Make "z:y:x" 3d plot
T->Draw("sqrt(x*x+y*y)");	Plot calculated quantity
T->Draw("x>>h1");	Dump a root branch to a histogram

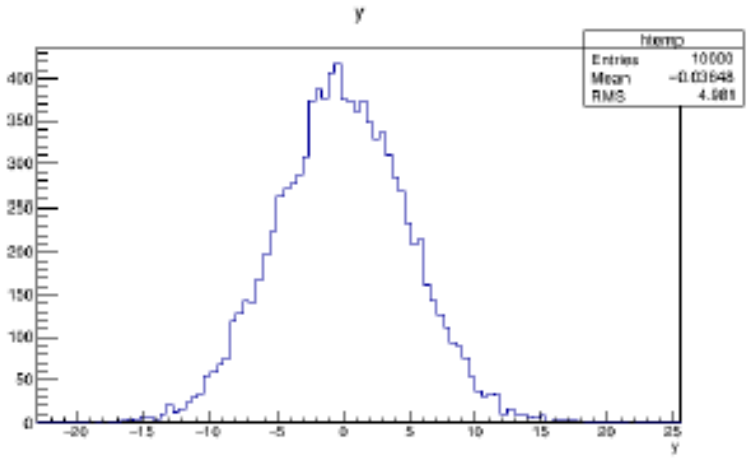
Example: Command Line Analyses

```
#include "TRandom.h"
#include "TFile.h"
#include "TTree.h"

void ExampleTree(const char * filename= "tree.root") {

   TTree data("tree","Example TTree");
   double x, y, z, t;
   data.Branch("x",&x,"x/D");
   data.Branch("y",&y,"y/D");
   data.Branch("z",&z,"z/D");
   data.Branch("t",&t,"t/D");

// fill it with random data

   for (int i = 0; i<10000; ++i) {
      x = gRandom->Uniform(-10,10);
      y = gRandom->Gaus(0,5);
      z = gRandom->Exp(10);
      t = gRandom->Landau(0,2);

      data.Fill();
   }
// write in a file

   TFile f(filename,"RECREATE");
   data.Write();
   f.Close();

}
```

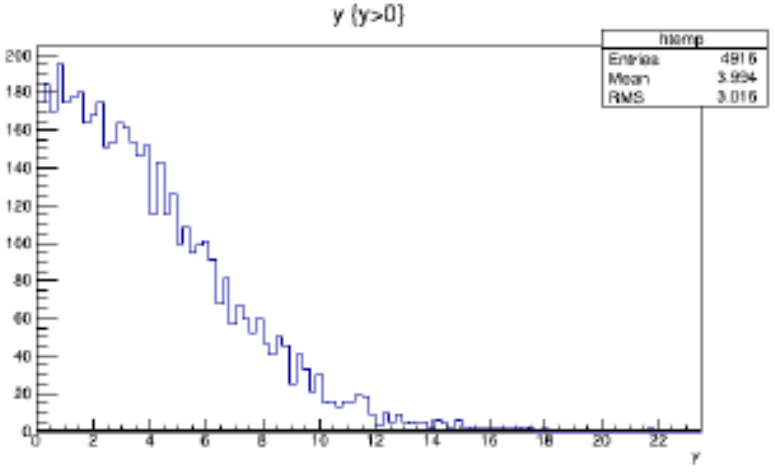
Scanning and Browsing a Tree

```
[harinder@harinder-OptiPlex-755 Exercises]$ root -l
root [0] TFile f("tree.root")
root [1] f.ls()
TFile**         tree.root
 TFile*         tree.root
  KEY: TTree    tree;1  Example TTree
root [2] tree->Scan()
************************************************************
*    Row   *         x *         y *         z *         t *
************************************************************
*        0 * 9.9948349 * -2.173821 * 0.5424387 * -0.612444 *
*        1 * -0.300527 * 4.1213184 * 2.9530392 * 3.3599092 *
*        2 * 4.7990596 * 0.0395610 * 4.1758331 * 0.3248392 *
*        3 * 6.0880602 * -4.925330 * 17.803898 * 2.3463682 *
*        4 * -2.153720 * -2.868095 * 34.954463 * 0.5302660 *
*        5 * -6.117022 * 3.8369828 * 5.4484498 * 22.924656 *
*        6 * 3.3112786 * 10.304510 * 12.156219 * -1.950201 *
*        7 * -8.741646 * -2.271012 * 3.2100651 * 5.3128856 *
*        8 * 4.2777012 * 9.2215651 * 3.5772235 * -2.077455 *
*        9 * -7.415144 * 11.302227 * 24.868109 * -1.812724 *
*       10 * 0.9474282 * 7.6234604 * 12.305172 * 21.557962 *
*       11 * -5.457643 * -6.407330 * 9.1531087 * 6.6468874 *
*       12 * -6.586581 * -5.453365 * 11.670097 * 2.8072593 *
*       13 * -6.351783 * -12.29070 * 18.275389 * 2.1263476 *
*       14 * 2.5567294 * -1.661886 * 3.5925466 * 2.3901384 *
*       15 * -8.197848 * -5.952137 * 8.2887077 * 2.3339561 *
*       16 * -1.615081 * 0.4030978 * 4.7062692 * 109.15314 *
*       17 * -2.591395 * 1.5039703 * 19.603690 * 8.6652320 *
*       18 * 6.5947936 * 1.2195639 * 10.592969 * -3.074309 *
*       19 * 2.2613264 * -3.939634 * 6.9536157 * 7.8710863 *
*       20 * -7.108989 * -6.082385 * 11.686474 * -1.895370 *
*       21 * -5.158606 * -0.598714 * 5.1116584 * 5.4938359 *
*       22 * -1.347475 * -4.175319 * 3.3290098 * 5.4394669 *
*       23 * 3.7155345 * 8.9050217 * 20.961120 * -2.862777 *
*       24 * 6.4014408 * -1.842549 * 17.116538 * -1.637794 *
Type <CR> to continue or q to quit ==> q
************************************************************
(Long64_t)25
root [3] tree->StartViewer()
```

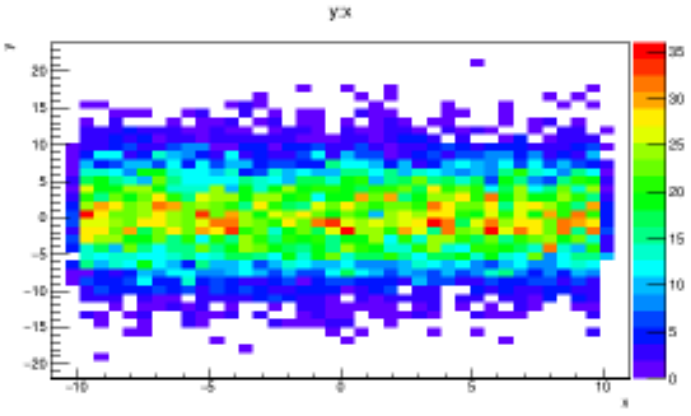
tv__tree->Draw("y","","", 10000, 0);



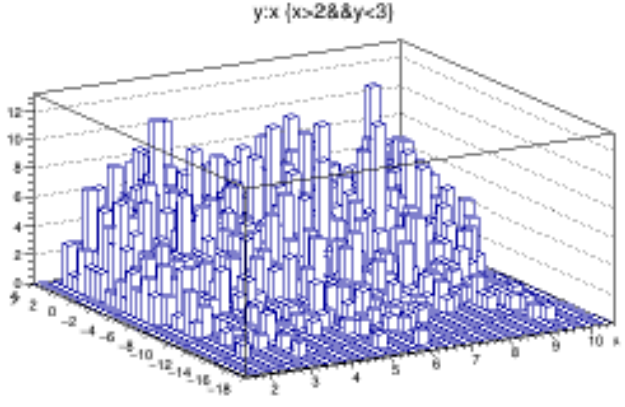
tv__tree->Draw("y","y>0","", 10000, 0);



tv__tree->Draw("y:x","","colz", 10000, 0);



tv__tree->Draw("y:x","x>2&&y<3","LEGO", 10000, 0);



Doing an Analysis

- The TBrowser is not of much use in a typical analysis
- Usually we want to

❑Throw away events we are not interested in

❑Select only certain objects within an event we are interested in

❑Calculate and plot quantaties of interest in an event (i.e. energy ratios)

❑Calculate and plot properties of objects in an event (electron pT)

-

Doing an Analysis

- To do this we need to Loop over each event (entry) in the TTree
- For each event we need to

❑Read all of the variables for the event and store them in memory

❑Perform our selection on these variables

❑Calculate and plot the quantities of interest

- This can be simple if you only care about a few variables
- But most analyses will need >100 of variables
- ROOT has a built in method for looping over a TTree or TCHain called a MakeClass/MakeSelector

MakeClass Option

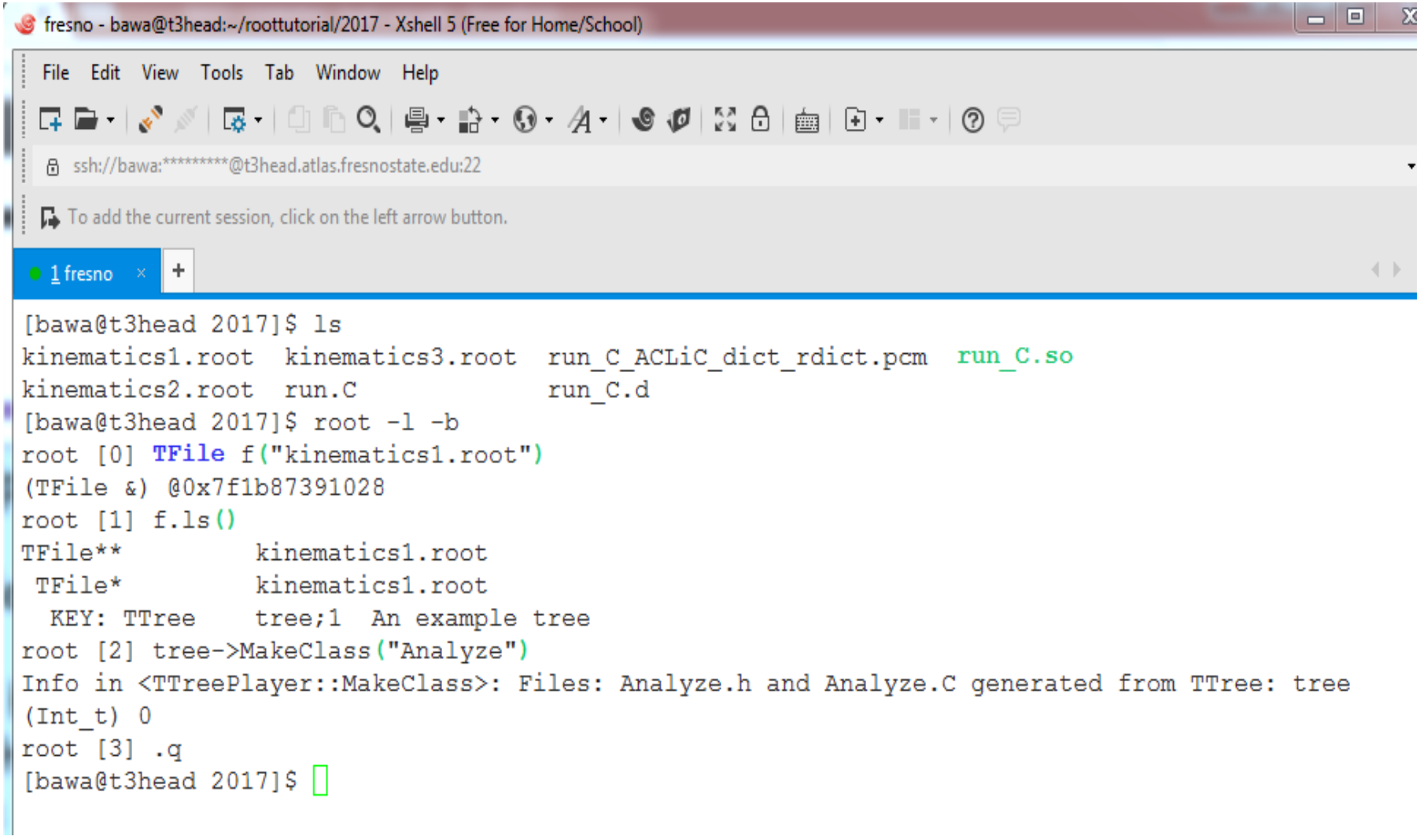
Example: Creating tree

```
#include<iostream>
#include "TTree.h"
#include "TMath.h"
#include "TFile.h"
#include "TRandom3.h"
int run(){
TRandom3 random;
        //First open a file
        TFile *f1=new TFile("kinematics3.root","RECREATE");
        //Create a TTree object
        TTree* tree=new TTree("tree","An example tree");
        float px,py,pz,E,m;
        //Create branches
        tree->Branch("pxval",&px,"px/F");
        tree->Branch("pyval",&py,"py/F");
        tree->Branch("pzval",&pz,"pz/F");
        tree->Branch("Eval",&E,"E/F");
        tree->Branch("mval",&m,"m/F");
        for(int i=0; i<10000; i++){
                        px=random.Gaus(5,1);
                        py=random.Poisson(6);
                        pz=random.Exp(3);
                        E=random.Exp(9);
                        m=random.Gaus(10,1);
                        tree->Fill();
}
        f1->Write();
        f1->Close();
        return 0;
}
```

Output files

- http://zimmer.csufresno.edu/~hbawa/kinematics1.root
- http://zimmer.csufresno.edu/~hbawa/kinematics2.root
- http://zimmer.csufresno.edu/~hbawa/kinematics3.root

Analysing tree: MakeClass



Analyze.C

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

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

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

   Long64_t nentries = fChain->GetEntries();

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

Setup code goes here

Loop code goes here

Analyze.h

```
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Wed Mar  8 16:50:55 2017 by ROOT version 6.04/14
// from TTree tree/An example tree
// found on file: kinematics1.root
//////////////////////////////////////////////////////////

#ifndef Analyze_h
#define Analyze_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>

// Header file for the classes stored in the TTree if any.

class Analyze {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TChain
   Int_t           fCurrent; //!current Tree number in a TChain

// Fixed size dimensions of array or collections stored in the TTree if any.

   // Declaration of leaf types
   Float_t         pxval;
   Float_t         pyval;
   Float_t         pzval;
   Float_t         Eval;
   Float_t         mval;

   // List of branches
   TBranch        *b_px;   //!
   TBranch        *b_py;   //!
   TBranch        *b_pz;   //!
   TBranch        *b_E;   //!
   TBranch        *b_m;   //!

   Analyze(TTree *tree=0);
   virtual ~Analyze();
   virtual Int_t    Cut(Long64_t entry);
   virtual Int_t    GetEntry(Long64_t entry);
   virtual Long64_t LoadTree(Long64_t entry);
   virtual void     Init(TTree *tree);
   virtual void     Loop();
   virtual Bool_t   Notify();
   virtual void     Show(Long64_t entry = -1);
};

#endif

#ifdef Analyze_cxx
Analyze::Analyze(TTree *tree) : fChain(0)
{
// if parameter tree is not specified (or zero), connect the file
// used to generate this class and read the Tree.
   if (tree == 0) {
      TFile *f = (TFile*)gROOT->GetListOfFiles()->FindObject("kinematics1.root");
      if (!f || !f->IsOpen()) {
         f = new TFile("kinematics1.root");
      }
```

Here is the input rootfile

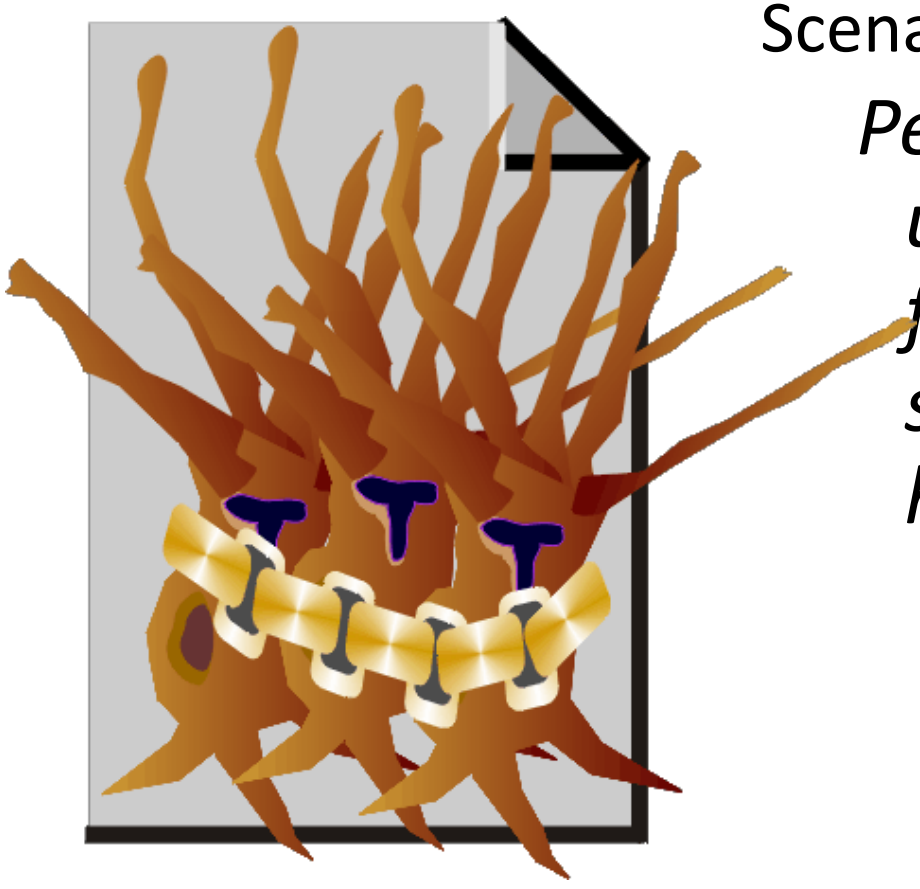
Analyzing one object(var)

```
#include <TH1.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <iostream>
void Analyze::Loop()
{
        //   In a ROOT session, you can do:
        //      root> .L Analyze.C
        //      root> Analyze t
        //      root> t.GetEntry(12); // Fill t data members with entry number 12
        //      root> t.Show();       // Show values of entry 12
        //      root> t.Show(16);     // Read and show values of entry 16
        //      root> t.Loop();       // Loop on all entries
        if (fChain == 0) return;
        Long64_t nentries = fChain->GetEntriesFast();
        cout<<"Number of Entries\t"<<nentries<<endl;
        TH1D *hist_px=new TH1D("hist_px","px distribution",100,0,100);
        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(jentry%1000==0)cout<<"Number of evts analysed\t"<<jentry<<endl;
                //calling variables from .h file(header)
                b_px->GetEntry(jentry);
                hist_px->Fill(pxval);
        }
        TFile *f1=TFile::Open("Output.root","RECREATE");
        hist_px->Write();
        f1->Close();
}
```

```
[bawa@t3head 2017]$ root -l -b
root [0] .L Analyze.C+
Info in <TUnixSystem::ACLiC>: creating shared library /nfs/t3nfs_common/home/bawa/roottutorial/2017/./Analyze_C.so
root [1] Analyze t
(Analyze &) @0x7f1325b6f028
root [2] t.Loop()
Number of Entries       10000
Number of evts analysed 0
Number of evts analysed 1000
Number of evts analysed 2000
Number of evts analysed 3000
Number of evts analysed 4000
Number of evts analysed 5000
Number of evts analysed 6000
Number of evts analysed 7000
Number of evts analysed 8000
Number of evts analysed 9000
root [3] .q
[bawa@t3head 2017]$ ls -ltr *.root
-rw-r--r-- 1 bawa bawa 159861 Mar  8 16:02 kinematics1.root
-rw-r--r-- 1 bawa bawa 158886 Mar  8 16:05 kinematics2.root
-rw-r--r-- 1 bawa bawa 158454 Mar  8 16:07 kinematics3.root
-rw-r--r-- 1 bawa bawa   3804 Mar  8 17:16 Output.root
[bawa@t3head 2017]$
```

run

Chains

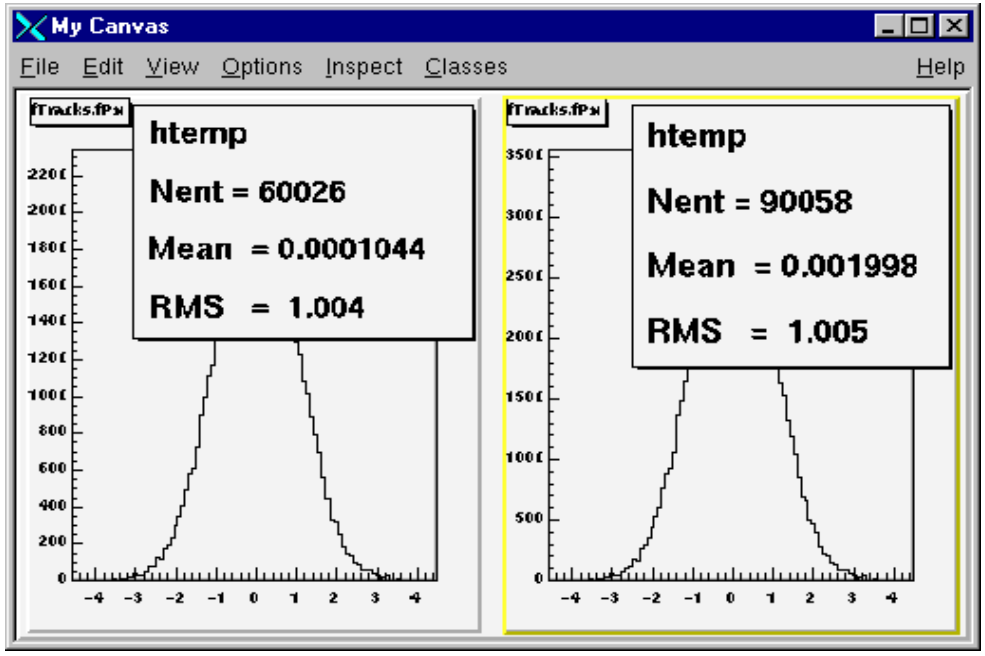


Scenario:

Perform an analysis using multiple ROOT files. All files are of the same structure and have the same tree.

Chains (cont.)

TChain::Add()



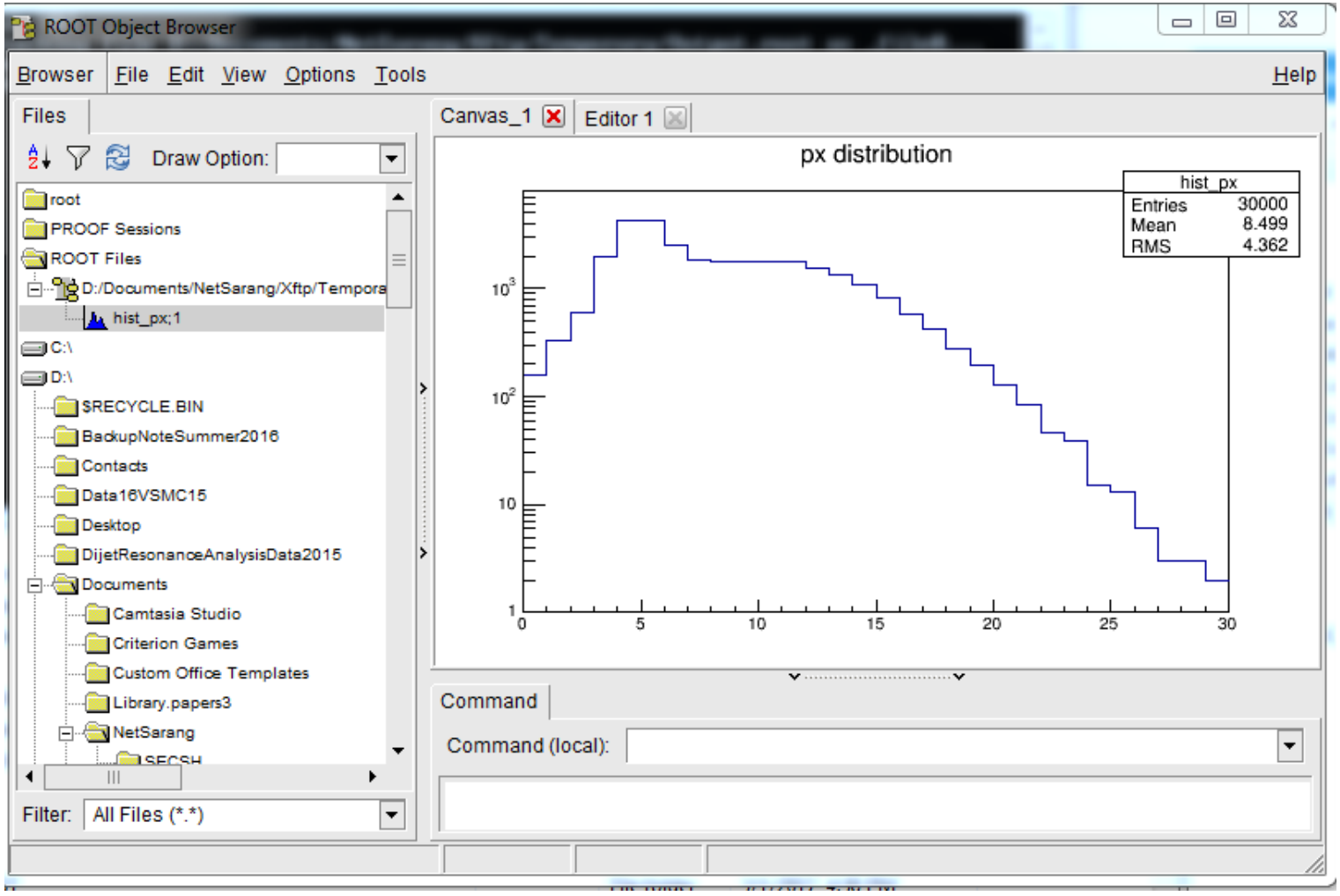

```
root [3] TChain chain("T");
root [4] chain.Add("Event.root")
root [5] chain.Draw("fTracks.fPx")
root [6] myCanvas->cd(2);
root [7] chain.Add("Event50.root")
root [8] chain.Draw("fTracks.fPx")
```

Demo: Changing "MyClass" to use a Chain

1. Just add few lines in Analyze.h

```
#ifdef Analyze_cxx
Analyze::Analyze(TTree *tree) : fChain(0)
{
// if parameter tree is not specified (or zero), connect the file
// used to generate this class and read the Tree.

        TChain * chain = new TChain("tree","");
        chain->Add("kinematics1.root");
        chain->Add("kinematics2.root");
        chain->Add("kinematics3.root");
        tree=chain;
//      if (tree == 0) {
//              TFile *f = (TFile*)gROOT->GetListOfFiles()->FindObject("kinematics1.root");
//              if (!f || !f->IsOpen()) {
//                      f = new TFile("kinematics1.root");
//              }
//              f->GetObject("tree",tree);

//      }
        Init(tree);
}
```

Demo: Changing "MyClass" to use a Chain



Global variable

ROOT has a set of global variables that apply to the session.
For example, ***gDirectory*** always holds the current directory, and ***gStyle*** holds the current style.
All global variables begin with "*g*" followed by a capital letter.

Global Variables

In addition to gROOT, there are some more global variables that apply to the session:

gROOT: gROOT holds information relative to the current session. By using the gROOT pointer you get the access to basically every object created in a ROOT program.

gFile: gFile is the pointer to the current opened file .

```
TFile *fin = new TFile(...);  // gFile => fin
TFile *fout = new TFile(...); // gFile => fout
```

gDirectory: gDirectory is a pointer to the current directory.

```
TFile f1("AFile1.root");
root[] gDirectory->pwd()
```

gPad: A graphic object is always drawn on the active pad. It is convenient to access the active pad, no matter what it is. For that we have gPad that is always pointing to the active pad.

gRandom: gRandom is a pointer to the current random number generator.

```
gRandom->Gaus()
```

gEnv: gEnv is the global variable with all the environment settings for the current session.

- ✓gFile: gFile is the pointer to the current opened file in the ROOT session.
- ✓gDirectory: gDirectory is a pointer to the current directory. The concept and role of a directory is explained in the chapter “Input/Output”.
- ✓gPad: A graphic object is always drawn on the active pad. We have gPad that is always pointing to the active pad. For example, if you want to change the fill color of the active pad to blue, but you do not know its name, you can use gPad.

  root> gPad->SetFillColor()
- ✓gRandom: gRandom is a pointer to the current random number generator. By default, it points to a TRandom3 object. Root> gRandom=new TRandom2(0)

gEnv

- gEnv is the global variable (of type TEnv) with all the environment settings for the current session. This variable is set by reading the contents of a .rootrc file (or $ROOTSYS/etc/system.rootrc) at the beginning of the root session.
- The behavior of a ROOT session can be tailored with the options in the .rootrc file. At start-up, ROOT looks for a .rootrc file in the following order:
- ./.rootrc //local directory
- $HOME/.rootrc //user directory
- $ROOTSYS/etc/system.rootrc //global ROOT directory

If more than one .rootrc: Env is merged !!

gROOT

- The single instance of TROOT is accessible via the global gROOT and holds information relative to the current session.
- By using the gROOT pointer, you get the access to every object created in a ROOT program.
- During a ROOT session, the gROOT keeps a series of collections to manage objects. They can be accessed via gROOT::GetListOf... methods.

```
gROOT->GetListOfClasses()
gROOT->GetListOfColors()
gROOT->GetListOfTypes()
gROOT->GetListOfGlobals()
gROOT->GetListOfGlobalFunctions()
gROOT->GetListOfFiles()
gROOT->GetListOfMappedFiles()
gROOT->GetListOfSockets()
gROOT->GetListOfCanvases()
gROOT->GetListOfStyles()
gROOT->GetListOfFunctions()
gROOT->GetListOfSpecials()
gROOT->GetListOfGeometries()
gROOT->GetListOfBrowsers()
gROOT->GetListOfMessageHandlers()
```

Logon and Logoff scripts

- The rootlogon.C and rootlogoff.C files are scripts loaded and executed at start-up and shutdown.
- The rootalias.C file is loaded but not executed. It typically contains small utility functions. For example, the rootalias.C script that comes with the ROOT distributions (located in $ROOTSYS/tutorials) defines the function edit(char *file). This allows the user to call the editor from the command line. This particular function will start the VI editor if the environment variable EDITOR is not set.
- root[0] edit("c1.C")
- For more details, see $ROOTSYS/tutorials/rootalias.C.

gROOT(Our example)

vi run.C

For lazy or clumsy persons only (i.e almost everybody..)!

```
{  gROOT->ProcessLine(".L Analyze.C+");
gROOT->ProcessLine("Analyze t");
gROOT->ProcessLine("t.Loop()");   }
```

bash> root –b –q –l run.C

In rootlogon.C (automatically executing macro)

```
{
gStyle->SetPalette(1);
gROOT->ProcessLine(".L $ROOTSYS/lib/libPhysics.so");
}
```

*also write: root setting commands

Plotting Script FYI

- Download following files

http://zimmer.csufresno.edu/~hbawa/GraphToolKit.h
http://zimmer.csufresno.edu/~hbawa/HistToolKit.h
http://zimmer.csufresno.edu/~hbawa/SetAttributes.h

- First two files have functions corresponding to Graphs plotting and Histogram plotting
- Third file is common Attribute file used in above GraphToolKit.h and HistToolKit.h

SetAttributes

```
int SetHistMarkerAttributes(TH1F* hist, int m
{
  hist->SetMarkerStyle(markerstyle) ;
  hist->SetMarkerColor(markercolor) ;
  hist->SetMarkerSize(markersize) ;
  return 0 ;
}
int SetHistAttributes(TH1F* hist, int linestyle=1, int linewidth=2, int linecolor=1,
                      int fillstyle=1001, int fillcolor=0,
                      int markerstyle=20, int markercolor=1, int markersize=1,
                      float xlabelsize=0.04, float xtitleoffset=1., float xtitlesize=0.04,
                      float ylabelsize=0.04, float ytitleoffset=1., float ytitlesize=0.04,
                      bool displaystats=false)
{
  SetHistFillAttributes(hist, fillstyle, fillcolor) ;
  SetHistAxisAttributes(hist,xlabelsize, xtitleoffset, xtitlesize, ylabelsize, ytitleoffset, ytitlesiz
  SetHistLineAttributes(hist, linestyle, linewidth, linecolor) ;
  SetHistMarkerAttributes(hist, markerstyle, markercolor, markersize) ;
  hist->SetStats(displaystats) ;

  return 0 ;
}

int SetPadFillAttributes(TPad* pad, int fillstyle=0, float fillcolor=0)
{
  pad->SetFillStyle(fillstyle) ;
  pad->SetFillColor(fillcolor) ;
  return 0 ;
}

int SetPadAttributes(TPad* pad, int fillstyle=0, int fillcolor=0,
 float leftmargin=0.1, float rightmargin=0.1, float bottommargin=0.1, float topmargin=0.1,
int bordermode=0, bool logx=false, bool logy=true, bool settickx=true, bool setticky=true)
{
  SetPadFillAttributes(pad, fillstyle, fillcolor) ;
  pad->SetMargin(leftmargin, rightmargin, bottommargin, topmargin) ;
  pad->SetBorderMode(bordermode) ;

  pad->SetLogx(logx) ;
```

SetHistAttributes Aa Abl .* 2 of 2

```
int DrawTwoHistsOnCanvas(string canvasname, TH1F* hist1, TH1F* hist2, string drawoption1="HIST", string drawoption2="HISTSame", bool logx=false, bool logy=true, bool isrectangle=true)
{
  gStyle->SetOptTitle(0); //this will disable the title for all coming histograms
  // define Canvas
  float height=600 ;
  float width ;
  if(isrectangle)
    width=800 ;
  else
    width=600 ;
  TCanvas *c = new TCanvas(canvasname.c_str(), "", width, height) ;

  // Define pads
  TPad *outpad = new TPad("extpad","extpad",0,0,1,1) ;// For marking outermost dimensions
  outpad->SetFillStyle(4000) ;//transparent
  TPad *interpad = new TPad("interpad","interpad",0,0,1,1) ;// For main histo
  SetPadAttributes(interpad, 0, 0, 0.1, 0.1, 0.1, 0.1, 0, logx, logy) ;

  interpad->Draw() ;
  outpad->Draw() ;

  outpad->cd() ;
  // Draw ATLAS Label
  //DrawATLASLabels(0.5, 0.8, 4) ;
  // draw center of mass energy and integrated luminosity
  //DrawCMEAndLumi(0.5, 0.7, "XX", "13 TeV") ;

  interpad->cd() ;
  // Set histogram attributes
  SetHistAttributes(hist1, 1, 2, 1, 0, 1, 20, 1) ;
  SetHistAttributes(hist2, 1, 2, 2, 0, 1, 22, 2) ;
  hist1->Draw(drawoption1.c_str()) ;
  hist2->Draw(drawoption2.c_str()) ;
  // define legend
  TLegend *leg=new TLegend(0.5,0.7,0.89,0.89) ;
  leg->SetFillStyle(0) ;
  leg->SetTextSize(0.04) ;
  leg->SetBorderSize(0) ;
  if(drawoption1.find("HIST")!=string::npos)
    leg->AddEntry(hist1, hist1->GetTitle(), "L") ;
  else
    leg->AddEntry(hist1, hist1->GetTitle(), "LPE") ;
  if(drawoption2.find("HIST")!=string::npos)
    leg->AddEntry(hist2, hist2->GetTitle(), "L") ;
  else
    leg->AddEntry(hist2, hist2->GetTitle(), "LPE") ;
  leg->Draw("same") ;
  c->Print((canvasname+".eps").c_str()) ;
  return 0 ;
}
```

2 histograms on canvas

Def TPad

Set histograms attributes

Def legend

Saving several pic in same Postscript

```
{
   TFile f("hsimple.root");
   TCanvas c1("c1","canvas",800,600);

   // select postscript output type
   // type = 111   portrait  ps
   // type = 112   landscape ps
   // type = 113   eps
   Int_t type = 111;

   // create a postscript file and set the paper size
   TPostScript ps("test.ps",type);
   ps.Range(16,24);  //set x,y of printed page

   // draw 3 histograms from file hsimple.root on separate pages
   hpx->Draw();
   c1.Update();      //force drawing in a macro
   hprof->Draw();
   c1.Update();
   hpx->Draw("lego1");
   c1.Update();
   ps.Close();
}
```

```
{
   TFile *f1 = new TFile("hsimple.root");
   TCanvas *c1 = new TCanvas("c1");
   TPostScript *ps = new TPostScript("file.ps",112);
   c1->Divide(2,1);
   // picture 1
   ps->NewPage();
   c1->cd(1);
   hpx->Draw();
   c1->cd(2);
   hprof->Draw();
   c1->Update();

   // picture 2
   ps->NewPage();
   c1->cd(1);
   hpxpy->Draw();
   c1->cd(2);
   ntuple->Draw("px");
   c1->Update();
   ps->Close();

   // invoke Postscript viewer
   gSystem->Exec("gs file.ps");
}
```

HW Exercise

- Create a simple ROOT tree containing 4 variables (for example x,y,z,t). Fill the tree with data (for example 10000 events) where x is generated according to a uniform distribution, y a gaussian and z an exponential and t a Landau distribution. Write also the tree in the file :"Outtree.root". (Marks 50)

- Download rootfiles: (Marks 50)

- http://zimmer.csufresno.edu/~hbawa/kinematics1.root
- http://zimmer.csufresno.edu/~hbawa/kinematics2.root
- http://zimmer.csufresno.edu/~hbawa/kinematics3.root

Make a Code and Chain the files together and save the following in "OutputKinematics.root" as well as in one postscript file "OutputKinematics.ps"

1. p_T =sqrt(px*px+py*py),
2. rapidity=1/2ln(E+pz/E-pz),
3. pseudorapidity=$\tanh^{-1}$ (pz/E)
4. p=sqrt(px*px+py*py+pz*pz)
5. Overlap plots in one canvas: pz and sqrt(p*p-(px*px+py*py))

EXTRA
Examples Ntuples:

What is TSelector

- Autogenerated code using TTree::MakeSelector()
  - Try it, load a ntuple into ROOT and use the MakeSelector function to create the code

```
root [4] physics->MakeSelector()
Info in <TTreePlayer::MakeClass>: Files: physics.h and physics.C generated from TTree: physics
(Int_t)0
```

- TTree::MakeSelector() creates two files in your current working directory
  - physics.C and physics.h
- By default, the name of the file is taken as the name of the TTree
  - You can specify your own name by sending a string of text to the MakeSelector
  - i.e. `physics->MakeSelector("myAnalysis")` will make myAnalysis.C and myAnalysis.h
  - myAnalysis.C(h) will have the exact same content as physics.C(h) except the name of the class, which this code defines, will be different
- These two pieces of code define a class that is used by TTree::Process() to loop over a ntuple
  - You can edit the .C file to include event and object selection as well as methods for filling and writing histograms
- But what is the benefit of using a TSelector? Open the .h file and I will show you.

TSelector

- TSelector::Begin
  - Called before looping over any entries in the TTree
  - Variables can be initialized here, such as histograms
  - However, variables declared here are local variables and can not be seen in the other functions
    - Therefore, your own variables need to be declared in an additional header file that you make yourself, and then initialized in Begin() (I will show you how I do this)

- TSelector::Process
  - This function is called for every entry in the TTree
  - It is the users responsibility to include a line of code that tells ROOT to load the entry
    - i.e. fChain->GetTree()->GetEntry(entry)
  - Event and object selections are applied here, histograms filled, and calculations made
  - All variable names in the TTree are defined and declared in the header file
    - To get the name of a variable you're interest in, look in the header file

- TSelector::Terminate
  - This is called after the last entry has been processed
  - Histograms can be written to an output file here

How do you use TSelector

- Look at the files in the ROOTTutorial2013 directory

  - Steering macro, `run.cxx`
    - This can be compiled, but it's not necessary
    - If its compiled, you must include all ROOT libraries in the `makefile`
    - It has 2 simple purposes
      1. Load the ntuple(s) into a TChain
      2. Process the TChain with your TSelector

  - TSelector (in our example, I called this `tutorial.C` and `tutorial.h`)
    - The ROOT ntuples you process should all have the same structure
      - Same TTree name and same variables
      - Generate the skeleton code for your TSelector by opening one of your ROOT ntuples interactively with ROOT and then use the MakeSelector() function of your TTree

  - myVariables.h
    - Variables declared in one function of the TSelector will not be seen inside any of the other functions
      - This means histograms declared in Begin() will no be seen in Process() where you would fill them, and also not seen in Terminate() where you would write them
    - Declare global scope variables in this header file to circumvent this problem

Steering Macro

This is the starting point for the analysis

To run the analysis from the shell command line, type

```
oot -l -q run.cxx
```

A very basic steering macro is shown below

- fChain is an instance of the class TChain
- "myTree" is the name of the TTree in the ROOT ntuples you want to analyse
- ntuples are added to the fChain using the Add() function
- Most people read an input text file containing a list of ntuples to add to the TChain
- mySelector.C is the TSelector code you created
- The ntuple is processed by mySelector.C by using the Process() function
- The name of the TSelector (including the full path if you like) is sent to the Process() function
- The "+" sign is added as a suffix if you want to compile the TSelector (always recommended to do this!!)

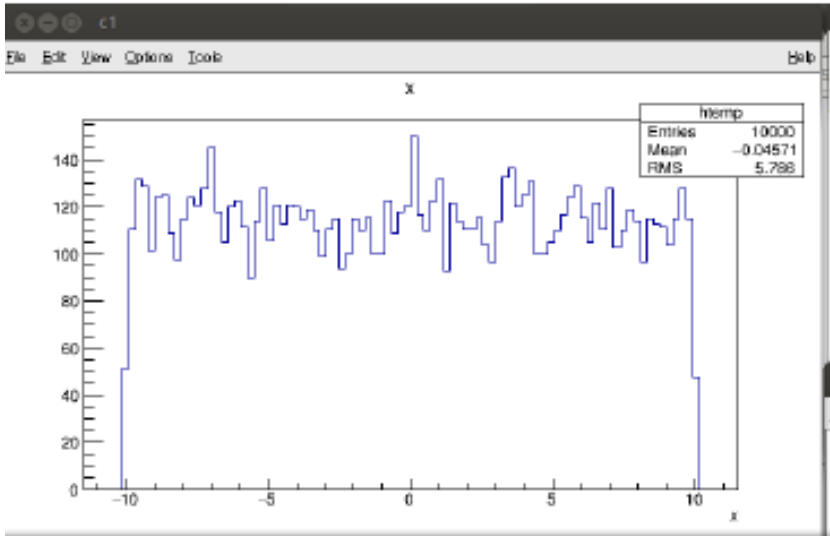
```
// A very basic steering macro

void run() {
 gRoot->Reset();

 TChain *fChain = new TChain("myTree")
 fChain->Add("./whatever1.root");
 fChain->Add("./whatever2.root");

 fChain->Process("./mySelector.C+");

 return;
}
```

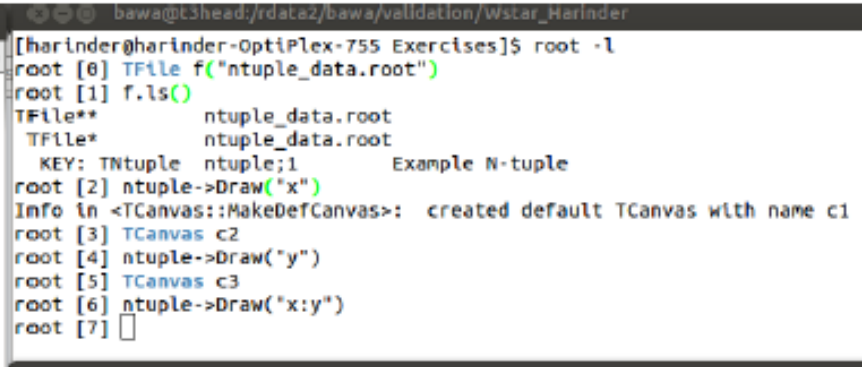
Ntuples Fill:

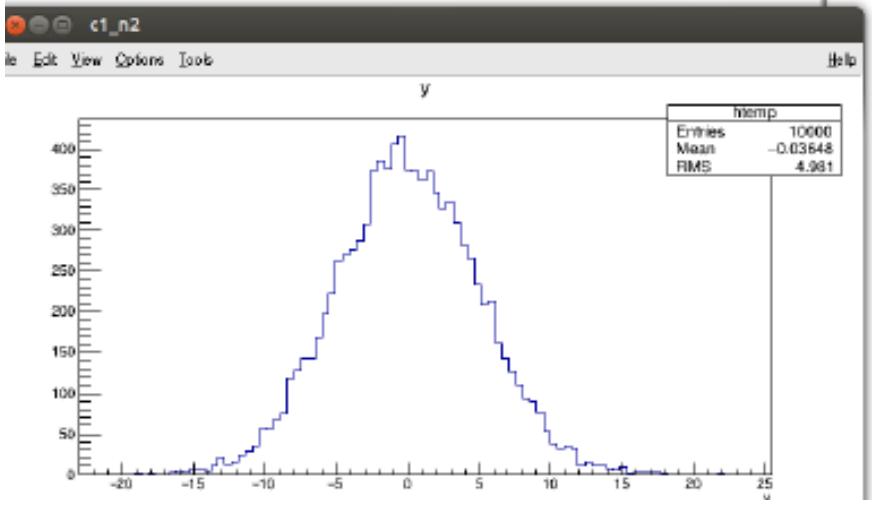
```
#include "TRandom.h"
#include "TFile.h"
#include "TNtuple.h"

void test() {

   TNtuple data("ntuple","Example N-tuple","x:y:z:t");

// fill it with random data
   for (int i = 0; i<10000; ++i) {
      float x = gRandom->Uniform(-10,10);
      float y = gRandom->Gaus(0,5);
      float z = gRandom->Exp(10);
      float t = gRandom->Landau(0,2);

      data.Fill(x,y,z,t);
   }
// write in a file
   TFile f("ntuple_data.root","RECREATE");
   data.Write();
   f.Close();

}
```

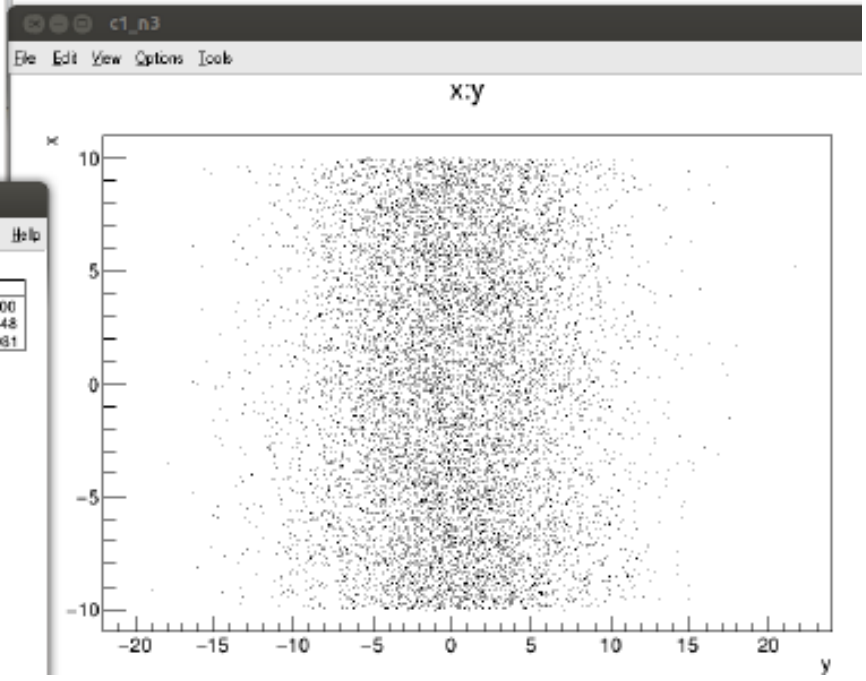
```
[harinder@harinder-OptiPlex-755 Exercises]$ root -l
root [0] TFile f("ntuple_data.root")
root [1] f.ls()
TFile**         ntuple_data.root
 TFile*         ntuple_data.root
  KEY: TNtuple  ntuple;1        Example N-tuple
root [2] ntuple->Scan()
************************************************************
*    Row   *         x *         y *         z *         t *
************************************************************
*        0 * 9.9948349 * -2.173821 * 0.5424387 * -0.612444 *
*        1 * -0.300527 * 4.1213183 * 2.9530391 * 3.3599093 *
*        2 * 4.7990598 * 0.0395610 * 4.1758332 * 0.3248392 *
*        3 * 6.0880603 * -4.925330 * 17.803897 * 2.3463683 *
*        4 * -2.153720 * -2.868095 * 34.954464 * 0.5302660 *
*        5 * -6.117022 * 3.8369829 * 5.4484496 * 22.924655 *
*        6 * 3.3112785 * 10.304511 * 12.156219 * -1.950201 *
*        7 * -8.741646 * -2.271012 * 3.2100651 * 5.3128857 *
*        8 * 4.2777013 * 9.2215652 * 3.5772235 * -2.077455 *
*        9 * -7.415144 * 11.302228 * 24.868108 * -1.812724 *
*       10 * 0.9474282 * 7.6234602 * 12.305172 * 21.557962 *
*       11 * -5.457643 * -6.407330 * 9.1531086 * 6.6468873 *
*       12 * -6.586581 * -5.453365 * 11.670097 * 2.8072593 *
*       13 * -6.351783 * -12.29070 * 18.275390 * 2.1263477 *
*       14 * 2.5567293 * -1.661886 * 3.5925467 * 2.3901383 *
*       15 * -8.197848 * -5.952137 * 8.2887077 *  2.333956 *
*       16 * -1.615081 * 0.4030978 * 4.7062692 * 109.15314 *
*       17 * -2.591395 * 1.5039703 * 19.603691 * 8.6652317 *
*       18 * 6.5947938 * 1.2195639 * 10.592968 * -3.074309 *
*       19 * 2.2613263 * -3.939634 * 6.9536156 * 7.8710861 *
*       20 * -7.108989 * -6.082385 * 11.686473 * -1.895370 *
*       21 * -5.158606 * -0.598714 * 5.1116585 * 5.4938359 *
*       22 * -1.347475 * -4.175319 * 3.3290097 * 5.4394669 *
*       23 * 3.7155344 * 8.9050216 * 20.961120 * -2.862777 *
*       24 * 6.4014411 * -1.842549 * 17.116537 * -1.637794 *
```

Ntuple:Draw()


c1
File Edit View Options Tools Help
x
htemp
Entries 10000
Mean -0.04571
RMS 5.786


bawa@t3head:/rdata2/bawa/validation/Wstar_Harinder
[harinder@harinder-OptiPlex-755 Exercises]$ root -l
root [0] TFile f("ntuple_data.root")
root [1] f.ls()
TFile** ntuple_data.root
TFile* ntuple_data.root
KEY: TNtuple ntuple;1 Example N-tuple
root [2] ntuple->Draw("x")
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [3] TCanvas c2
root [4] ntuple->Draw("y")
root [5] TCanvas c3
root [6] ntuple->Draw("x:y")
root [7]


c1_n2
File Edit View Options Tools Help
y
htemp
Entries 10000
Mean -0.03648
RMS 4.981


c1_n3
File Edit View Options Tools
x:y
x
y

Read ntuple & Writing on Screen

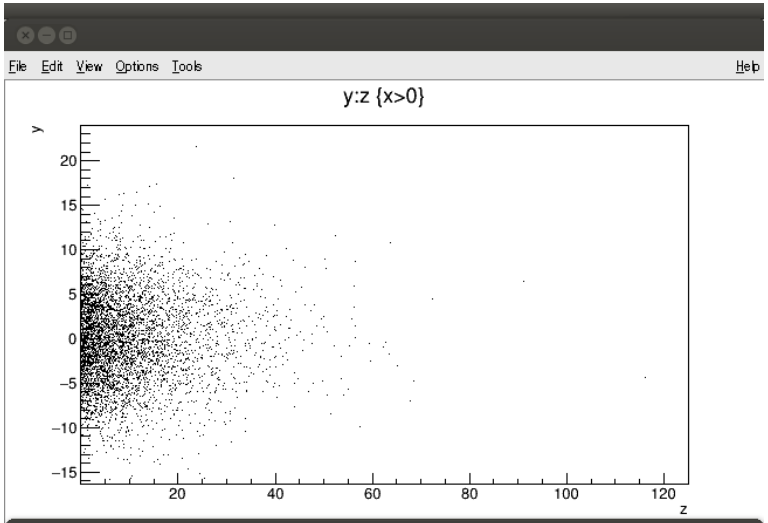
bawa@t3head:/rdata2/bawa/validation/Wstar_Harinder

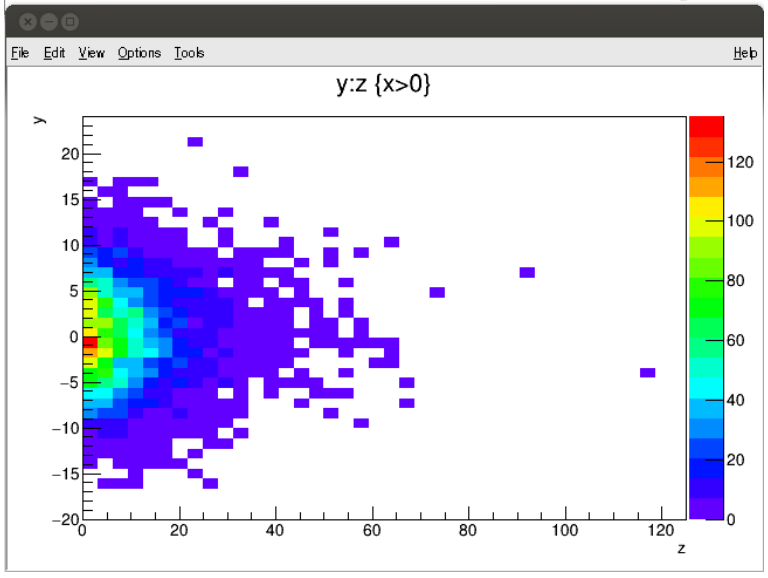
```
void exampleNtupleRead() {
   TFile f("ntuple_data.root");
   TNtuple *ntuple=0;
   f.GetObject("ntuple",ntuple);
   // loop on the ntuple entries

   for (int i = 0; i < ntuple->GetEntries(); ++i) {
      ntuple->GetEntry(i);
      float * raw_content = ntuple->GetArgs();
      float x = raw_content[0];
      float y = raw_content[1];
      float z = raw_content[2];
      float t = raw_content[3];
      if (i%1000==0) std::cout << x << "  " << y << "    " << z << "  " << t  << std::endl;
   }
// write in a file

   f.Close();
}
```

```
[harinder@harinder-OptiPlex-755 Exercises]$ root -l exampleNtupleRead.C
root [0]
Processing exampleNtupleRead.C...
9.99483  -2.17382   0.542439  -0.612445
5.26687  -0.567785   11.5874  -0.610093
0.521705  -1.77127   14.9077  0.442004
-9.28038  3.90928   0.152558  0.665459
-4.33669  -1.49484   2.11666  -0.778403
5.24322  -2.51007   10.8218  4.71695
5.43867  -4.75767   1.01158  -2.5583
-9.6626  -2.28142   0.0127684  8.1902
-4.96916  -7.83322   14.7144  -1.456
6.17546  -3.47435   6.39128  28.7686
root [1]
```

Ntuple:Draw()

```
void exampleNtupleDraw() {
        TFile f("ntuple_data.root");
        TNtuple *ntuple=0;
        f.GetObject("ntuple",ntuple);
        TCanvas *c=new TCanvas("c");
        ntuple->Draw("t");
        // to add a selection cut and a graphic option
        TCanvas *c1=new TCanvas("c1");
        ntuple->Draw("y:z","x>0","colz");
        TCanvas *c2=new TCanvas("c2");
        ntuple->Draw("y:z","x>0");
}
```





Ntuple:Draw()

```
void exampleNtupleDraw() {
        TFile f("ntuple_data.root");
        TNtuple *ntuple=0;
        f.GetObject("ntuple",ntuple);
        TCanvas *c=new TCanvas("c");
        ntuple->Draw("t");
        // to add a selection cut and a graphic option
        TCanvas *c1=new TCanvas("c1");
        ntuple->Draw("y:z","x>0","colz");
        TCanvas *c2=new TCanvas("c2");
        ntuple->Draw("y:z","x>0");
}
```