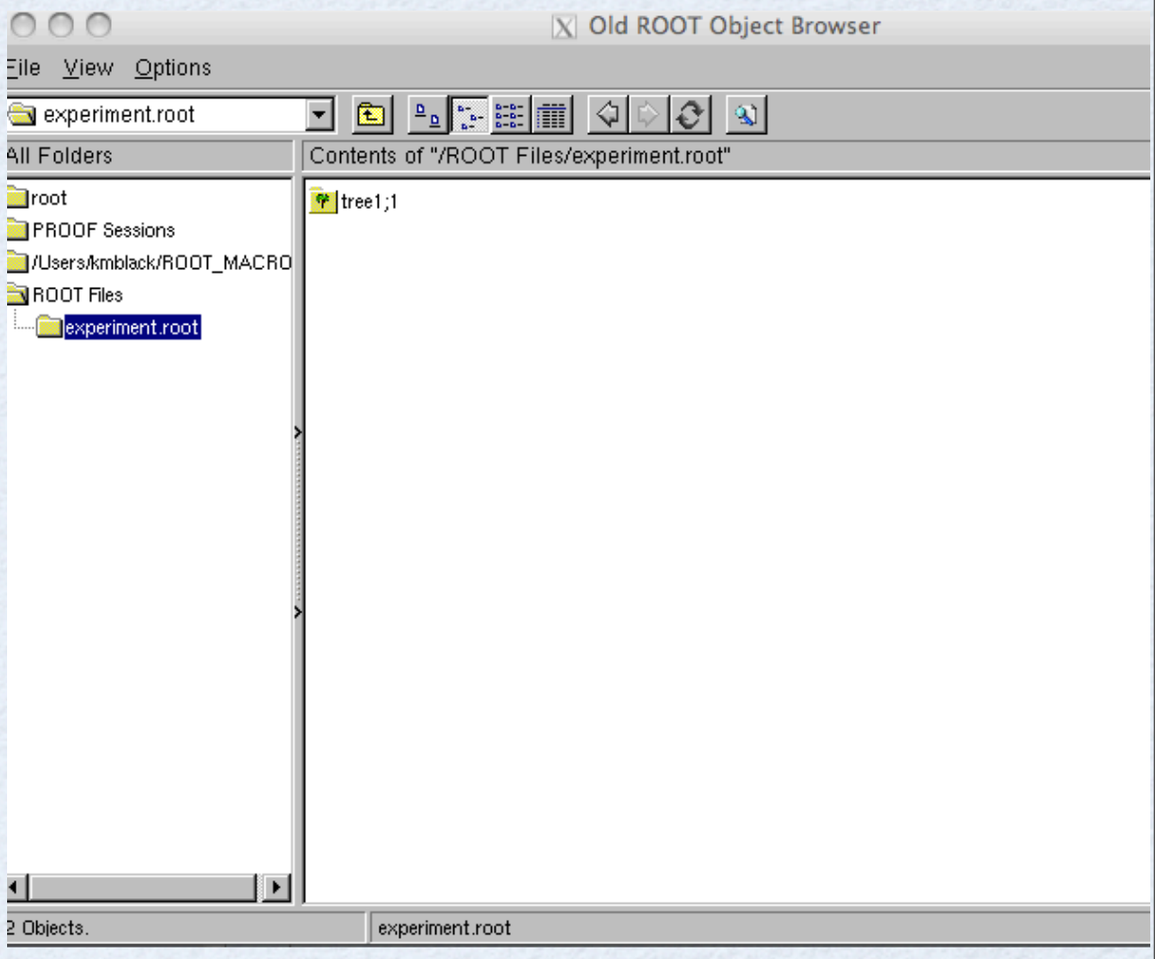
WORKING WITH DATA

WORKING WITH FILES

- ROOT stores information into files called “root tuples”.
- You can store basically any root quantity (histograms, graphs, floats, ints)
- Today we will discuss how to access that information and how to analyze it both on the command line and from C++
- Today you will be asked to write a bit of code in class accessing a root file i have prepared.

TBROWSER

- I have placed a file called experiment.root on my public area on lxplus and on the web
- you can retrieve it by doing
  - scp USERNAME@lxplus.cern.ch:~kblack/public/experiment.root .
    - or get it from blackboard site
- To open the file with root you can do from the command line
  - root experiment.root
- Open the TBrowser by doing
- TBrowser kb



TBROWSER

- Double click on the file experiment.root
  - there is just one object inside that called tree1
  - This is a TTree which contains 100,000 'simulated' physics events
  - Actually almost no real physics here - just a dummy file for you to get some practice with
  - In each event an imaginary particle moving in the z direction with energy 'ebeam' hits a target at z=0 and travels some distance zv before it gets deflected
  - We then measure the momentum of the outgoing deflected particle with some detector px, py, pz with some degree of confidence measured by the chi2 variable

- Right click on the tree1 and select scan
- that will bring up a window - just hit OK
- on your terminal you should see

```
root [2] ************************************************************************************************
*    Row   *     event *     ebeam *        px *        py *        pz *        zv *      chi2 *
************************************************************************************************
*        0 *         0 * 150.14041 * 14.333598 * -4.021043 * 143.54425 * 22.264110 * 0.9405828 *
*        1 *         1 * 149.78578 * 0.0509290 * -1.373310 * 148.60041 * 0.6140761 * 1.0205643 *
*        2 *         2 * 150.16188 * 4.0079331 * 3.8898270 * 145.68779 * 16.568965 * 0.8934459 *
*        3 *         3 * 150.14196 * 1.4559249 * 4.6634402 * 146.70643 * 11.466330 * 1.0217185 *
*        4 *         4 * 149.94201 * -10.34201 * 11.068926 * 148.32540 * 0.3665101 * 0.8540952 *
*        5 *         5 * 150.18460 * 17.084215 * -12.14251 * 143.10090 * 22.088724 * 0.9029421 *
*        6 *         6 * 150.01785 * 5.1921901 * 7.7853212 * 148.59400 * 2.2835574 * 1.0643705 *
*        7 *         7 * 150.05159 * 7.5478696 * -7.433323 * 144.44578 * 21.399364 * 0.9727888 *
*        8 *         8 * 150.07125 * 0.2315468 * -0.021122 * 147.78424 * 6.9644231 * 0.9281990 *
*        9 *         9 * 149.95890 * 1.2057533 * 7.2738246 * 146.99125 * 7.1694283 * 1.0203726 *
*       10 *        10 * 149.91664 * 5.3508482 * 3.9811716 * 140.69627 * 38.811821 * 1.0784103 *
*       11 *        11 * 149.87626 * -4.625746 * -0.079142 * 147.91272 * 4.0146312 * 0.8633663 *
*       12 *        12 * 150.10922 * -1.962243 * 11.455042 * 147.41201 * 6.7586140 * 1.0821087 *
*       13 *        13 * 150.02220 * -4.966853 * 4.2942524 * 145.05976 * 17.792921 * 0.9202339 *
*       14 *        14 * 149.85690 * 0.2636620 * 0.0957022 * 144.69007 * 22.262750 * 0.9285840 *
*       15 *        15 * 150.28263 * -7.495016 * 1.4888154 * 147.64340 * 7.0665574 * 1.0168865 *
*       16 *        16 * 149.93669 * 2.1853997 * -3.671361 * 148.74932 * 1.2530027 * 1.0101522 *
*       17 *        17 * 150.11419 * 10.453489 * 6.6687169 * 147.82991 * 4.4257226 * 1.0224413 *
*       18 *        18 * 150.09060 * -0.560505 * -0.909502 * 145.94816 * 14.877190 * 1.0072044 *
*       19 *        19 * 149.71482 * 8.2002382 * 3.5894899 * 147.13110 * 6.3213443 * 1.1409138 *
*       20 *        20 * 150.12364 * -7.086627 * -5.520020 * 145.06636 * 19.501110 * 1.1055847 *
*       21 *        21 * 150.19241 * -3.996811 * -8.783142 * 148.29959 * 3.2856993 * 1.0566752 *
*       22 *        22 * 149.74935 * 8.6041336 * 6.2061400 * 147.64433 * 2.6813583 * 1.0001720 *
*       23 *        23 * 150.14714 * 2.3585295 * -3.343346 * 148.47381 * 1.9962592 * 0.8632693 *
*       24 *        24 * 149.78625 * -0.198832 * -0.183111 * 147.52179 * 7.0266542 * 0.9161183 *
Type <CR> to continue or q to quit ==> ***************************************************************
```

TTREE ORGANIZATION

root [2] **
*   Row   *   event *   ebeam *   px *   py *   pz *   zv *   chi2 *
****
*   0 *   0 * 150.14041 * 14.333598 * -4.021043 * 143.54425 * 22.264110 * 0.9405828 *
*   1 *   1 * 149.78578 * 0.0509290 * -1.373310 * 148.60041 * 0.6140761 * 1.0205643 *
*   2 *   2 * 150.16188 * 4.0079331 * 3.8898270 * 145.68779 * 16.568965 * 0.8934459 *
*   3 *   3 * 150.14196 * 1.4559249 * 4.6634402 * 146.70643 * 11.466330 * 1.0217185 *
*   4 *   4 * 149.94201 * -10.34201 * 11.068926 * 148.32540 * 0.3665101 * 0.8540952 *
*   5 *   5 * 150.18460 * 17.084215 * -12.14251 * 143.10090 * 22.088724 * 0.9029421 *
*   6 *   6 * 150.01785 * 5.1921901 * 7.7853212 * 148.59400 * 2.2835574 * 1.0643705 *
*   7 *   7 * 150.05159 * 7.5478696 * -7.433323 * 144.44578 * 21.399364 * 0.9727888 *
*   8 *   8 * 150.07125 * 0.2315468 * -0.021122 * 147.78424 * 6.9644231 * 0.9281990 *
*   9 *   9 * 149.95890 * 1.2057533 * 7.2738246 * 146.99125 * 7.1694283 * 1.0203726 *
*   10 *   10 * 149.91664 * 5.3508482 * 3.9811716 * 140.69627 * 38.811821 * 1.0784103 *
*   11 *   11 * 149.87626 * -4.625746 * -0.079142 * 147.91272 * 4.0146312 * 0.8633663 *
*   12 *   12 * 150.10922 * -1.962243 * 11.455042 * 147.41201 * 6.7586140 * 1.0821087 *
*   13 *   13 * 150.02220 * -4.966853 * 4.2942524 * 145.05976 * 17.792921 * 0.9202339 *
*   14 *   14 * 149.85690 * 0.2636620 * 0.0957022 * 144.69007 * 22.262750 * 0.9285840 *
*   15 *   15 * 150.28263 * -7.495016 * 1.4888154 * 147.64340 * 7.0665574 * 1.0168865 *
*   16 *   16 * 149.93669 * 2.1853997 * -3.671361 * 148.74932 * 1.2530027 * 1.0101522 *
*   17 *   17 * 150.11419 * 10.453489 * 6.6687169 * 147.82991 * 4.4257226 * 1.0224413 *
*   18 *   18 * 150.09060 * -0.560505 * -0.909502 * 145.94816 * 14.877190 * 1.0072044 *
*   19 *   19 * 149.71482 * 8.2002382 * 3.5894899 * 147.13110 * 6.3213443 * 1.1409138 *
*   20 *   20 * 150.12364 * -7.086627 * -5.520020 * 145.06636 * 19.501110 * 1.1055847 *
*   21 *   21 * 150.19241 * -3.996811 * -8.783142 * 148.29959 * 3.2856993 * 1.0566752 *
*   22 *   22 * 149.74935 * 8.6041336 * 6.2061400 * 147.64433 * 2.6813583 * 1.0001720 *
*   23 *   23 * 150.14714 * 2.3585295 * -3.343346 * 148.47381 * 1.9962592 * 0.8632693 *
*   24 *   24 * 149.78625 * -0.198832 * -0.183111 * 147.52179 * 7.0266542 * 0.9161183 *
Type <CR> to continue or q to quit ==> **

Each entry has an event number, ebeam, px,py,pz,zv,chi2

Something missing?

- There is something missing from my description of the file which is very important on interpreting what this all means... What is it?

Units!

- I didn't tell you if zv was in mm, cm, miles or what the units of the energy/momentum were!
- In general these aren't stored in the ntuple!!
- Take zv in cm and energy and momentum in GeV

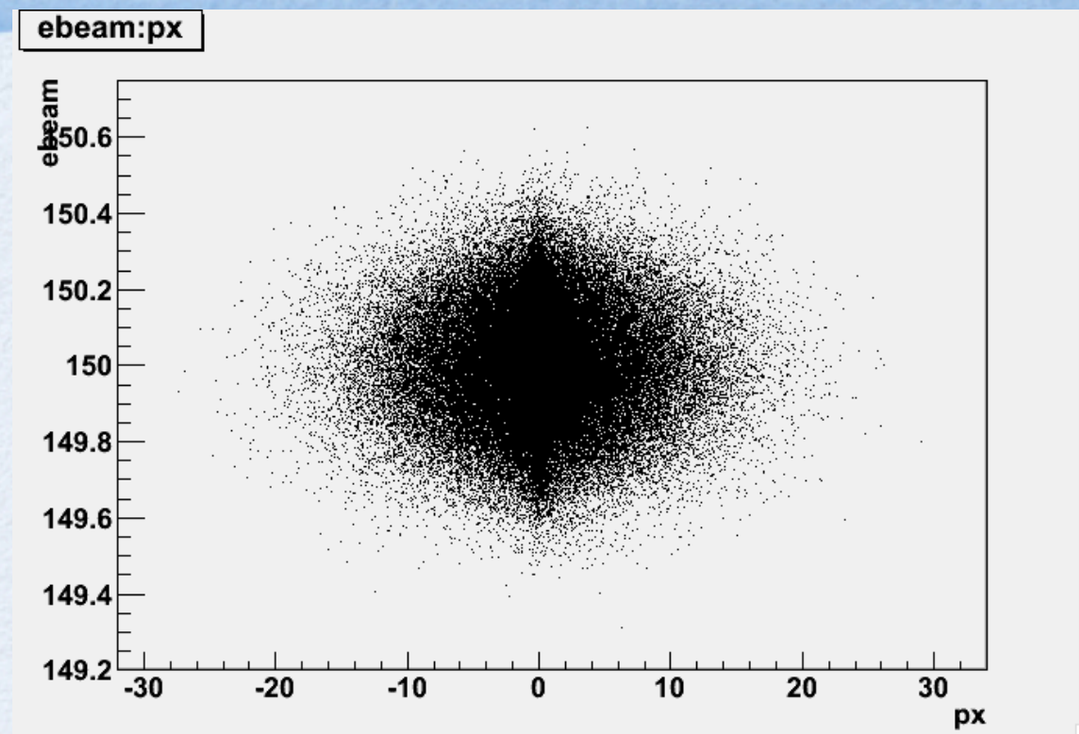
COMMAND LINE

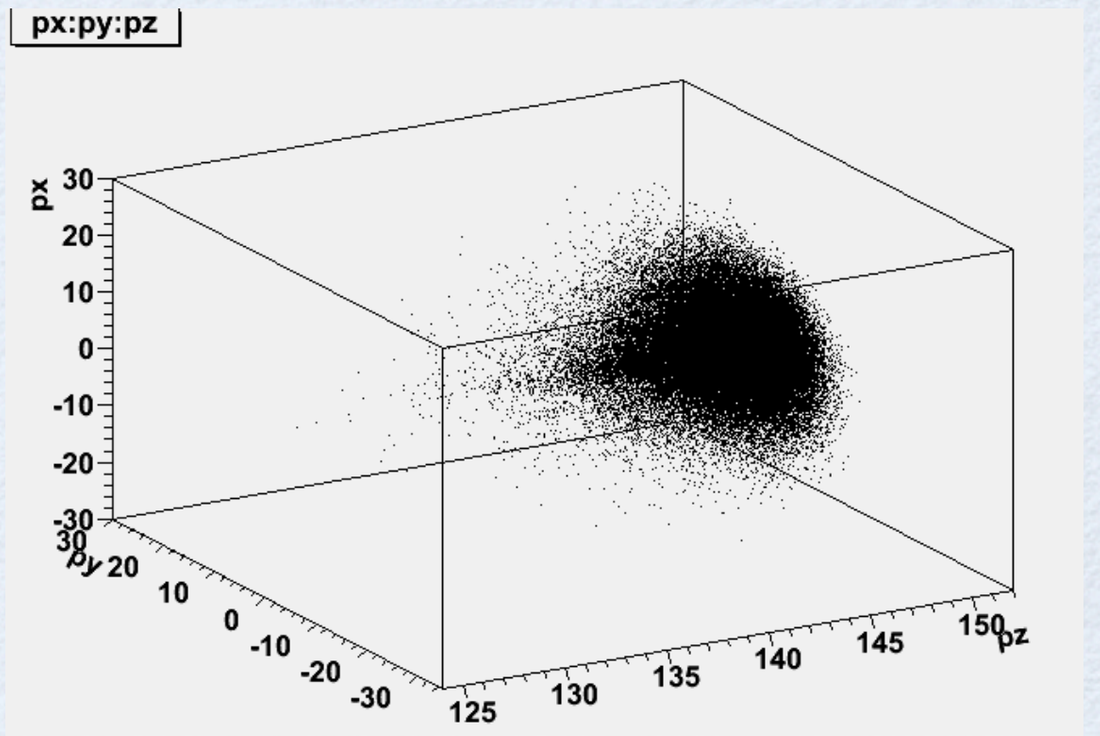
- You can scan the tree from the command line using (same as before
  - tree1->Scan()
- You can display the variables without the data with
  - tree1->Print()

```
root [7] tree1->Print()
******************************************************************************
*Tree    :tree1     : Reconstruction ntuple                                  *
*Entries :   100000 : Total =         3035119 bytes  File  Size =    2171135 *
*        :          : Tree compression factor =   1.30                       *
******************************************************************************
*Br    0 :event     : event/I                                                *
*Entries :   100000 : Total  Size=     453314 bytes  File Size  =     134514 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   2.85     *
*.............................................................................*
*Br    1 :ebeam     : ebeam/F                                                *
*Entries :   100000 : Total  Size=     453314 bytes  File Size  =     260330 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.47     *
*.............................................................................*
*Br    2 :px        : px/F                                                   *
*Entries :   100000 : Total  Size=     453258 bytes  File Size  =     359238 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.07     *
*.............................................................................*
*Br    3 :py        : py/F                                                   *
*Entries :   100000 : Total  Size=     453258 bytes  File Size  =     359138 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.07     *
*.............................................................................*
*Br    4 :pz        : pz/F                                                   *
*Entries :   100000 : Total  Size=     453258 bytes  File Size  =     292046 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.31     *
*.............................................................................*
*Br    5 :zv        : zv/F                                                   *
*Entries :   100000 : Total  Size=     453258 bytes  File Size  =     349087 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.10     *
*.............................................................................*
*Br    6 :chi2      : chi2/F                                                 *
*Entries :   100000 : Total  Size=     453294 bytes  File Size  =     321049 *
*Baskets :       12 : Basket Size=      32000 bytes  Compression=   1.20     *
*.............................................................................*
```

SCATTER PLOTS

- Instead of one variable you can do scatter plots of two variables,
  - ttree1->Draw("ebeam:px")
  - ttree1->Draw("px:py:pz")



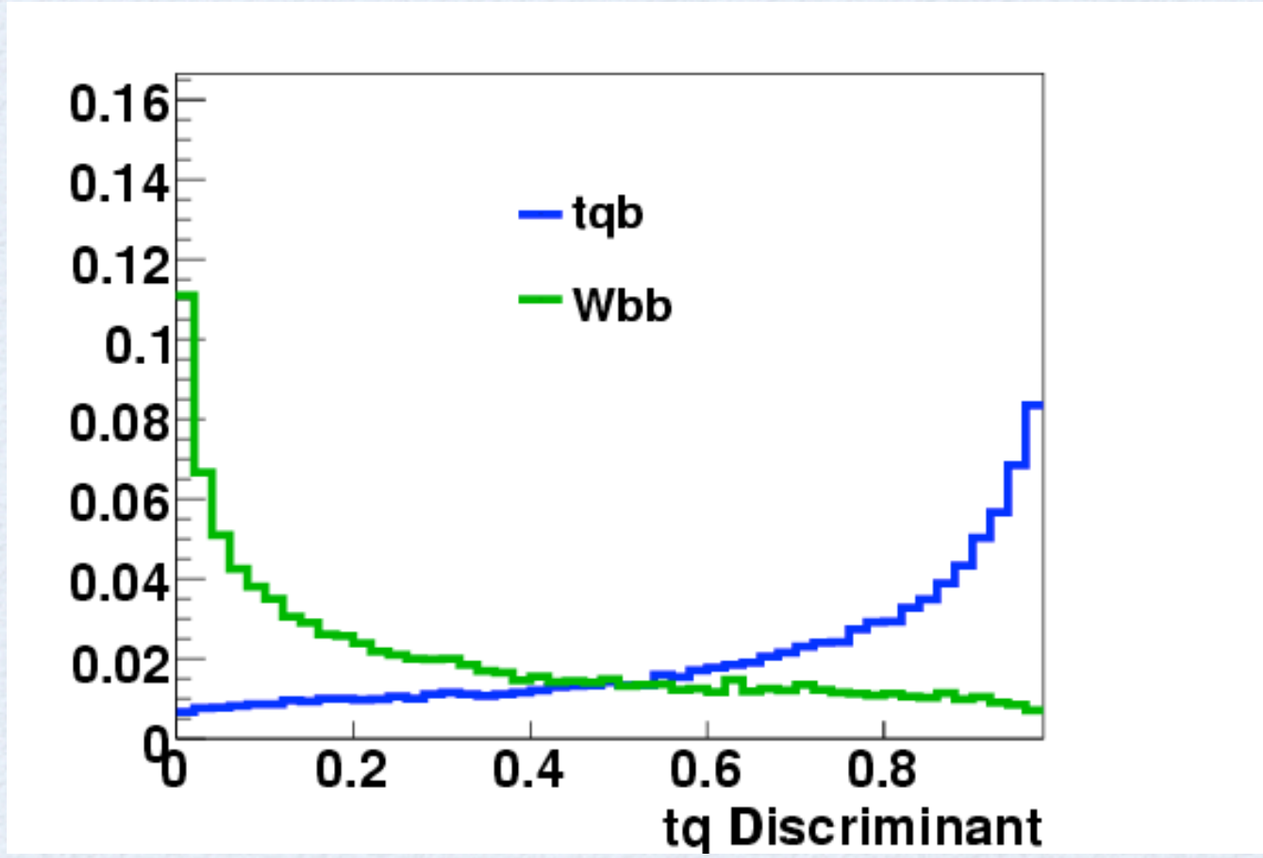


WE CAN ALSO MAKE 'CUTS'

- What is a cut?
- This is a term we often use in HEP analysis
- It just means you make a selection or series of selection on the data

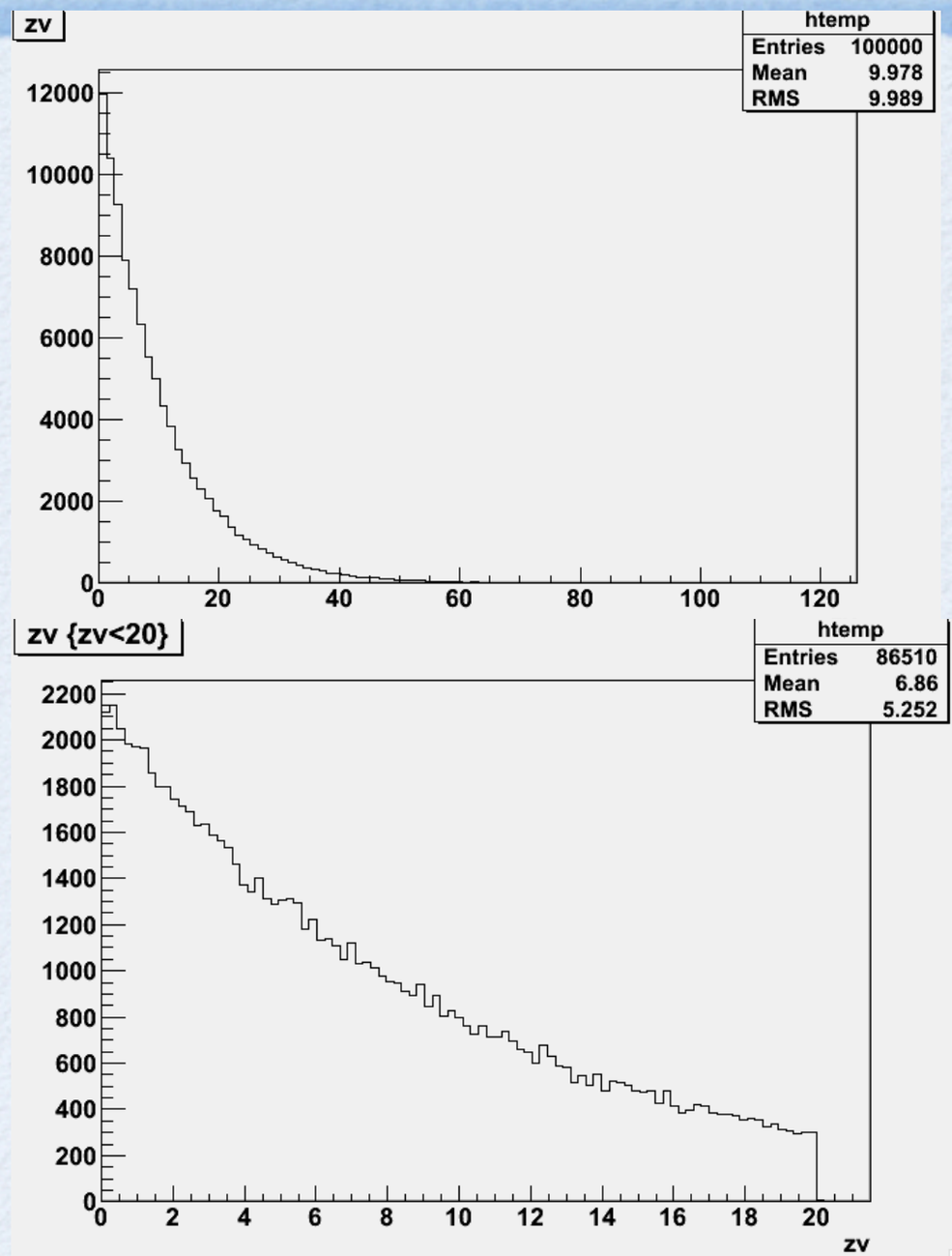
SELECTION

- For example if we were trying to find the top events
- One might want to 'cut' or select events with a discriminant greater than 0.5
- This enhances the fraction of events which are of the type we are interested in (of course at the cost of losing some!)



CUTTING ON THE COMMAND LINE

- tree1->Draw("zv")



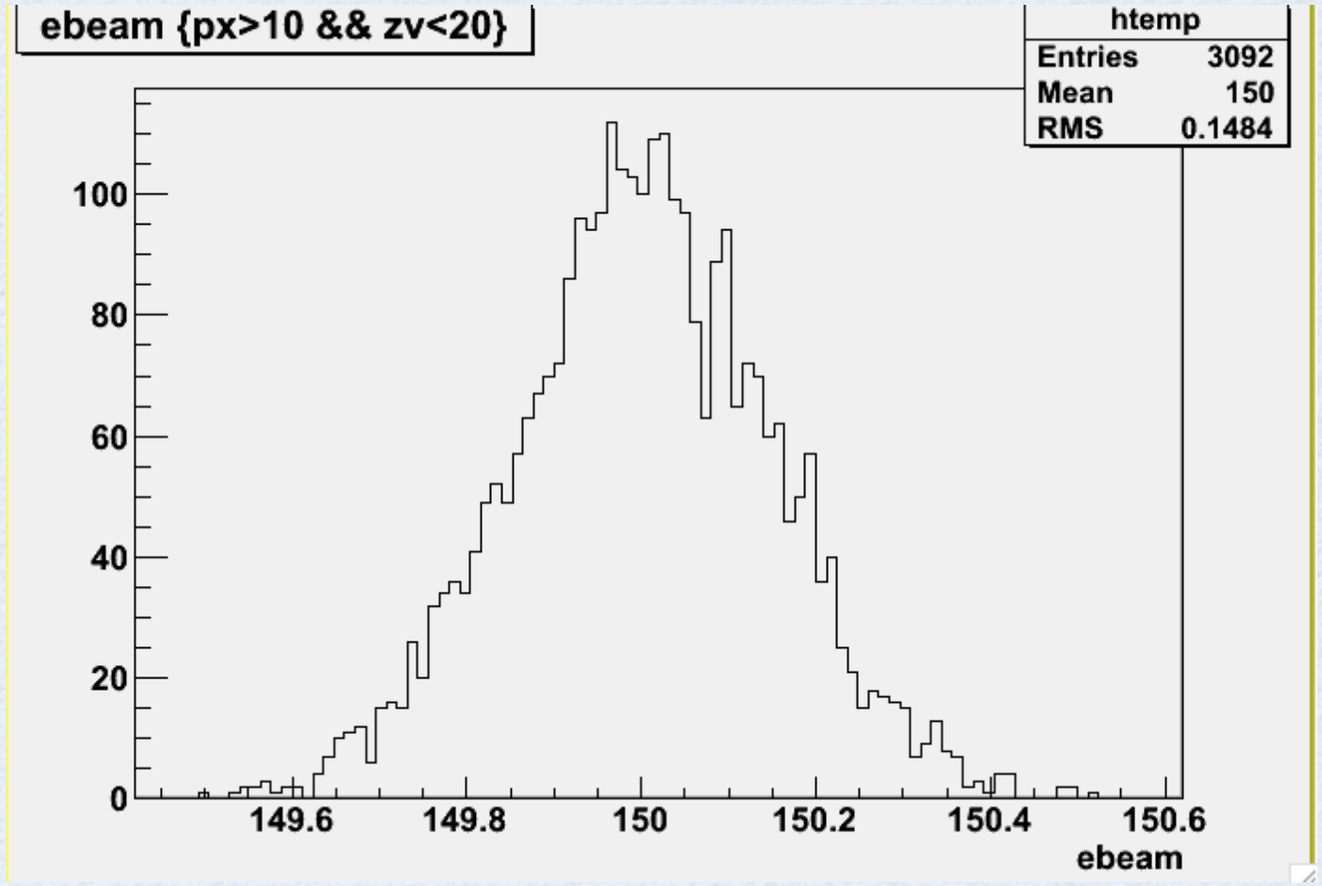
- tree1->Draw ("zv","zv<20")

MORE INTERESTINGLY...

- One can plot one variable while making a selection on another.

- tree1->Draw("ebeam","zv<20")

- tree1->Draw("ebeam","px>10 && zv<20")



ANALYZING DATA WITH C++

- The following slides show you a great way of doing some fast basic analysis
- However, once you close the root session and terminal you have to type it all over again!
- More complicated analysis is painful to do on the command line
- Better to keep track of it a macro

USING A MACRO

```
root [0] TFile myFile("experiment.root")
root [1] tree1->MakeClass("Analyze")
Info in <TTreePlayer::MakeClass>: Files: Analyze.h and Analyze.C generated from TTree: tree1
(Int_t)0
root [2] .q
```

- Simple way to get a running start
- TFile myFile("experiment.root")
- tree1->MakeClass(“Analyze”)
- now exit root with .q
- makes two files Analyze.C and Analyze.h

What are these files

- In C++ the convention is to organize code
  - **header files** .h (here we put declarations - classes, global variables, functions)
  - **source files** .C or .cpp (here we put the implementation of the code)
- Basic idea - give people a heads up on what they should be expecting. What functions exist ? What do they return and what variables do they take as input
- This way a user can look through without sorting through a mess of code

LET'S LOOK

Loop function

loops over each entry in root tuple and executes for each one

```
#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 where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch
   if (fChain == 0) return;
        // The Set-up code goes here.
   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
      // The Loop code goes here.
 }
      // The Wrap-up code goes here.
}
```

CLOSER LOOK

- fChain a pointer to the list of data files the macro will run over
  - in this case just one
- GetEntriesFast() just returns the number of events stored in the data
- nbytes, nb just counting amount of data read (not essential but comes with default root reading class)

```
if (fChain == 0) return;
    // The Set-up code goes here.
Long64_t nentries = fChain->GetEntriesFast();

Long64_t nbytes = 0, nb = 0;
```

READING EVENT

- Loop over each ‘entry’ or event
- Load the event into memory
- count the number of bytes that have been read
- LoadTree and GetEntry load the event from disk into memory to be manipulated

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

That is it!

- The MakeClass function is one way of getting root to generate a set of classes which allow you to access data
- Forms a skeleton for you to fill in to make calculations
- Doesn't actually do anything more than that ...yet

How do you run it

- We can run it on the command line
- Type
  - .L Analyze.C
    - Analyze a
    - a.Loop()
- Try it now - don't worry nothing really is suppose to happen yet

CONFUSING?

- What do these commands have to do with the root file “experiments.root”
- They don’t seem to reference the file, or anything it it. How does it know what to do?
- To understand that we have to notice that the first thing that is done we have to look at Analyze.h - which is the first thing which the source code .C

ANALYZE.H

- The code actually defines a class!
  - pointer to the "TTree" (this is just what root calls the organization of its data)
  - On the tree are leaves and branches
    - recall these are the same variables we saw by looking directly at the ttree
  - What does it mean that we see the branch and b_event and then the variable event

```
#ifndef Analyze_h
#define Analyze_h

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

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

   // Declaration of leaf types
   Int_t           event;
   Float_t         ebeam;
   Float_t         px;
   Float_t         py;
   Float_t         pz;
   Float_t         zv;
   Float_t         chi2;

// List of branches
TBranch        *b_event;   //!
TBranch        *b_ebeam;   //!
TBranch        *b_px;   //!
TBranch        *b_py;   //!
TBranch        *b_pz;   //!
TBranch        *b_zv;   //!
TBranch        *b_chi2;   //!
```

LOOK DOWN FURTHER

- This tells root to look in the file for the variables that are part of the file on disk
- Read them into memory
- And assign them to the addresses of the variables we defined above

```
fChain->SetBranchAddress("event", &event, &b_event);
fChain->SetBranchAddress("ebeam", &ebeam, &b_ebeam);
fChain->SetBranchAddress("px", &px, &b_px);
fChain->SetBranchAddress("py", &py, &b_py);
fChain->SetBranchAddress("pz", &pz, &b_pz);
fChain->SetBranchAddress("zv", &zv, &b_zv);
fChain->SetBranchAddress("chi2", &chi2, &b_chi2);
```

LOADING THE FILE

- Let's look at the code - it takes an argument of a TTree
- If there is no tree passed to it - it defaults to the file it was generated with
- When we told root to "MakeClass" it read in the data structure which is how it knows what variables to read in

```
Analyze::Analyze(TTree *tree)
{
// 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("experiment.root");
      if (!f) {
         f = new TFile("experiment.root");
      }
      tree = (TTree*)gDirectory->Get("tree1");

   }
   Init(tree);
}
```

Let's start editing

- Say we want to histogram of the chi2 variable, how can we do that in C++ code
- We do this using code that is vary similar to what we learned last time but we will write it in a slightly different way

- We would like the histogram to be available anywhere during the event processing
- We will add it as part of the class

First the header file

- We would like to add a 1-D histogram to our class
- First we have to tell root where it has to look to find out what a TH1F is
- Declare a pointer to a TH1F in the class definition
  - there now exists a member variable which is a pointer to h_chi2

```
#ifndef Analyze_h
#define Analyze_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TH1F.h>

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

// Declaration of leaf types
Int_t           event;
Float_t         ebeam;
Float_t         px;
Float_t         py;
Float_t         pz;
Float_t         zv;
Float_t         chi2;

TH1F*           h_chi2;
```

NOW THE SOURCE

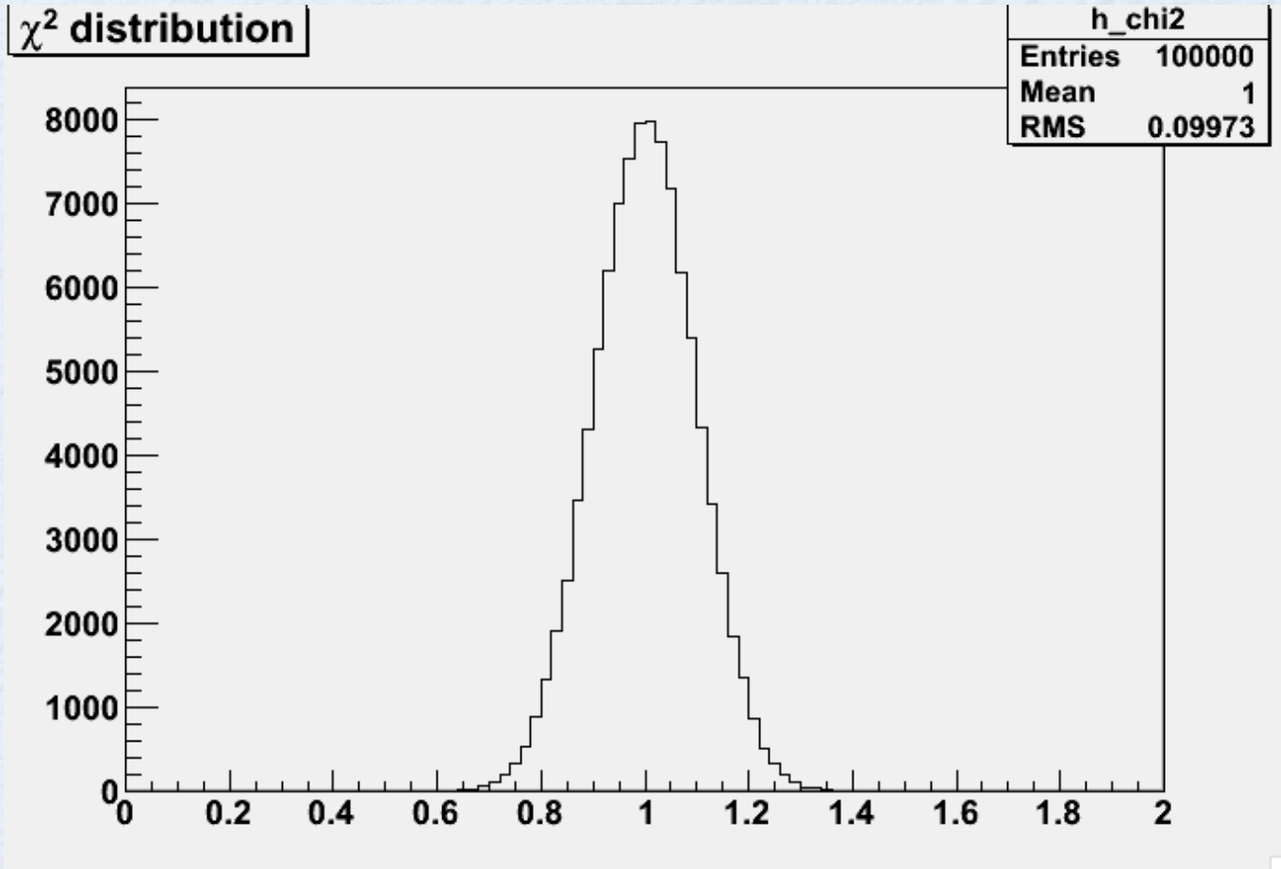
- We first `book' the histogram - which means that we declare it and its properties:
  - name, title, number of bins, lower and upper range
- Then for each entry we fill the histogram

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

void Analyze::Loop()
{
  h_chi2 = new TH1F("h_chi2", "#chi^{2} distribution", 100, 0,2);
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      h_chi2->Fill(chi2);
      // if (Cut(ientry) < 0) continue;
   }
}
```

THEN WE EXECUTE IT

```
root [0] .L Analyze.C
root [1] Analyze t
root [2] t.Loop()
root [3] h_chi2->Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```

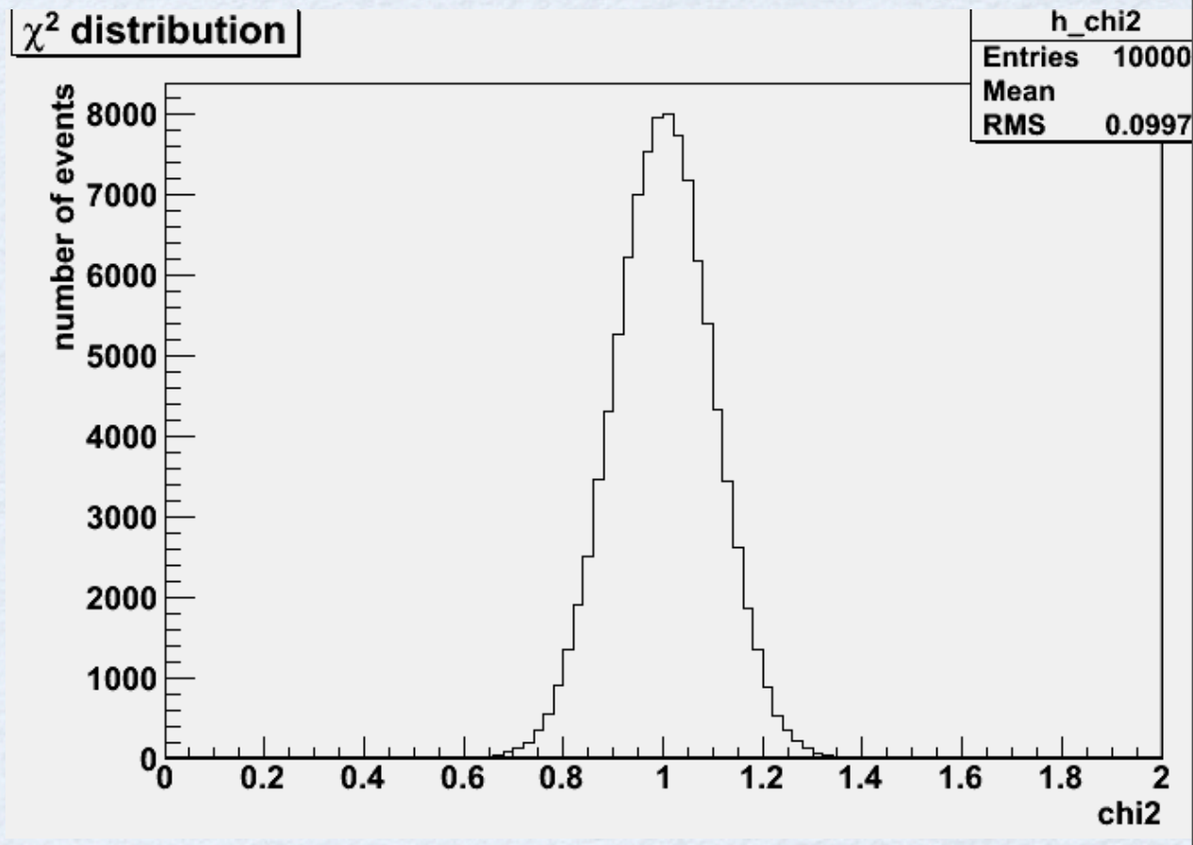
Stop to think about this:
[0] loads the macro
[1] makes an objet t of type Analyze
[2] calls the Loop function
[3] Draws the resulting histogram



Note - if you know anything about chi2s this looks rather fishy! In fact it is just a gaussian for matter of simplicity...

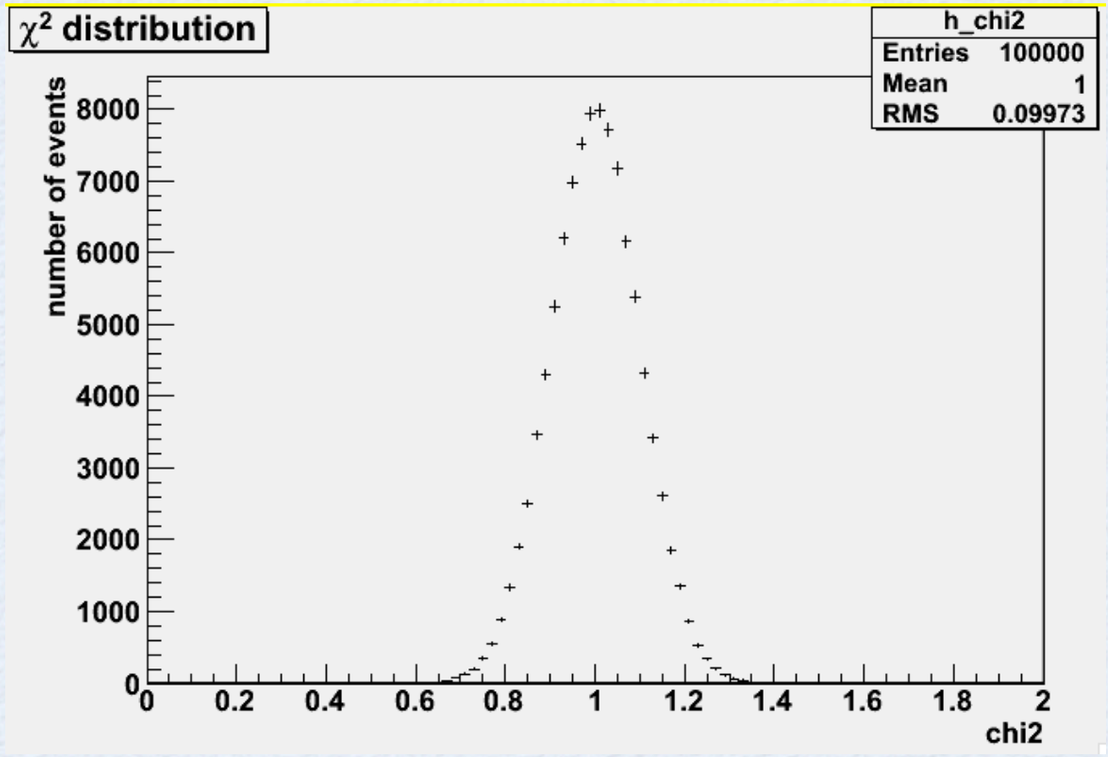
NOW LETS LABEL ...

- h_chi2->GetXaxis()->SetTitle("chi2");
- h_chi2->GetYaxis()->SetTitle("number of events");
- h_chi2->GetYaxis()->SetTitleOffset(1.3)
- h_chi2->Draw()



NOW ADD ERROR BARS

- h_chi2->Draw("e")



Now you try

- Add histograms to show:
  - px
  - py
  - pz
  - ebeam
- With axis labels an error bars

2-D HISTOGRAMS

- General syntax:
- TH2F hist("hist","Time vs. Energy",50,0,1000,50,0,5.e-9);
  - similar syntax but note
    - TH2F rather than 1F
    - Now two ranges x and y and number of bins in each

2-D PLOTS

- Make 2 dimensional histograms of
  - chi2 versus ebeam
  - pz versus ebeam
  - px versus py

ADDING VARIABLES

- If we want to add a new variable we can calculate derived variables inside the loop function
- For example , a derived quantity from the input variables
  - eg momentum after scattering transverse to original beam direction or angles in transverse and longitudinal direciton

EXERCISES

- Make histograms of the quantities
  - Transverse momentum after scattering
  - Angle in azimuthal direction
  - Angle in longitudinal direction
- Note : be careful with arctan!

$$p_T = \sqrt{p_x^2 + p_y^2}$$

$$\theta = \tan^{-1}(\frac{p_T}{p_z})$$

$$\phi = \tan^{-1}(\frac{p_y}{p_x})$$

APPLYING A CUT

- Write C++ code which counts the number of events with pz LESS than 145 GeV
- Have it output the answer to the terminal

ENERGY LOSS

- For a highly relativistic particle we can ignore the mass compared to momentum

$$E^2_{measured} = p^2_x + p^2_y + p^2_z$$

- We imagine that in interacting with the target the particle looses some energy

$$E_{loss} = E_{beam} - E_{measured}$$

- We will do a small study on this

TASKS

- Calculate E measured and plot it
- Calculate E loss and plot it
- Make a 2D scatter plot between E loss and the zv.
  - Is there are relationship between the two?