ROOT analysis, simple simulation and curve fitting write-up for the course "FUF065/FIM465 Advanced Subatomic Detection and Analysis Methods" Version 0.92

Thomas Nilsson
Fundamental Physics, Chalmers University of Technology
thomas.nilsson@chalmers.se

March 30, 2011

1 Introduction

ROOT is a C++-based data analysis package tailored at handling large data amounts typical for nuclear and particle physics experiments. ROOT can be used as a C++ class library, but in addition in a command-line mode using the CINT interpreter. ROOT is the system of choice for the majority of modern experiments, including the LHC experiments and has superseded and extended the earlier FORTRAN-based CERNLIB and PAW. ROOT can be used on a number of platforms, please visit http://root.cern.ch for more information.

On a correctly installed system, just open a command-line window and type:

```
thomas-nilssons-macbook-air:~ tnilsson$ root
  ****************************************
  *                                      *
  *       W E L C O M E  to  R O O T      *
  *                                      *
  *   Version   5.28/00  14 December 2010   *
  *                                      *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch           *
  *                                      *
  ****************************************

ROOT 5.28/00 (trunk@37585, Dec 14 2010, 15:20:27 on macosx64)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

If this does not work, e.g. on one of the Linux machines of the Subatomic Physics group, add the following to your **.bashrc** file:

```
export ROOTSYS=/usr/local/bin/root
export PATH=$ROOTSYS/bin:$PATH
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
```

2 Basic plotting

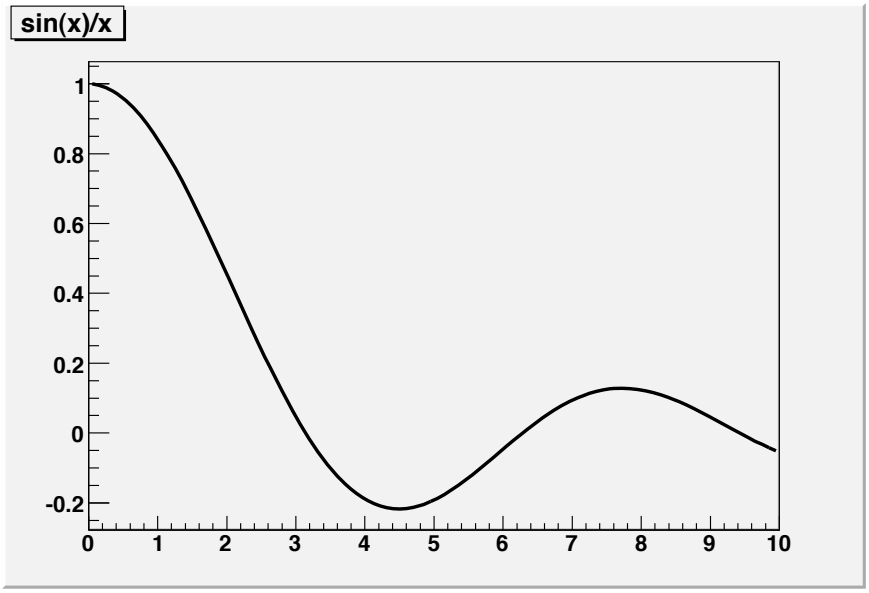
Starting out with some simple plots, we have to remember that everything in ROOT is based on C++ classes, most actions are methods used upon these instances of the classes.

When e.g. creating a function, which is an instance of the class TF1, one can press TAB to get the possible arguments:

```
root [2] TF1 f1( <TAB>
TF1 TF1()
TF1 TF1(const char* name, const char* formula, Double_t xmin = 0, Double_t xmax = 1)
TF1 TF1(const char* name, Double_t xmin, Double_t xmax, Int_t npar)
TF1 TF1(const char* name, void* fcn, Double_t xmin, Double_t xmax, Int_t npar)
TF1 TF1(const char* name, ROOT::Math::ParamFunctor f, Double_t xmin = 0, Double_t xmax = 1, Int_t npar =
TF1 TF1(const char* name, void* ptr, Double_t xmin, Double_t xmax, Int_t npar, char* className)
TF1 TF1(const char* name, void* ptr, void*, Double_t xmin, Double_t xmax, Int_t npar, char* className, c
TF1 TF1(const TF1& f1)
```

Now take the version in the second row, but change the default values, then draw the function:

```
root [2] TF1 f1("f1","sin(x)/x",0,10)
root [3] f1.Draw()
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



Check the properties of **f1** by dumping the object - most things here are utterly uninteresting...

```
root [5] f1.Dump()
==> Dumping object at: 0x087fbb60, name=f1, class=TF1
```

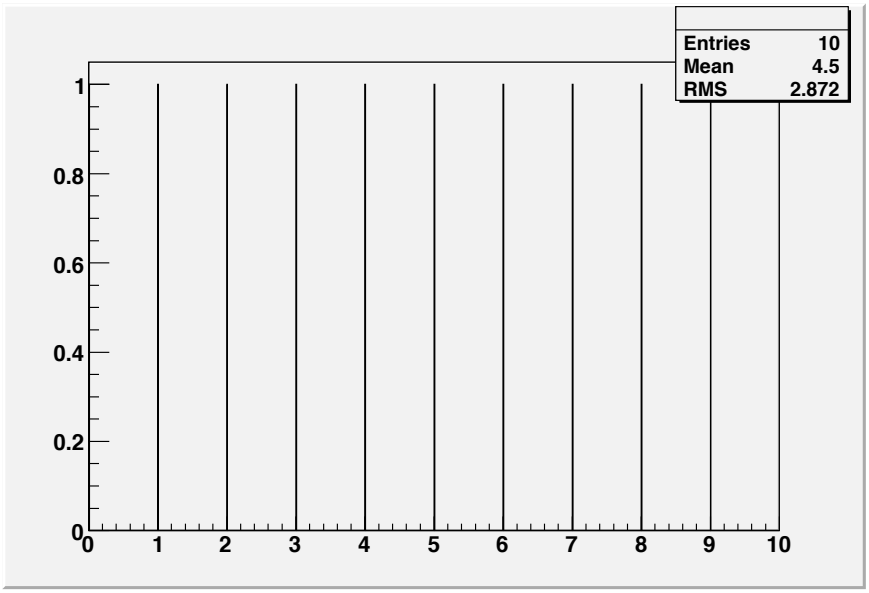
```
fXmin                       0           Lower bounds for the range
fXmax                       10          Upper bounds for the range
fNpx                        100         Number of points used for the graphical representation
fType                       0           (=0 for standard functions, 1 if pointer to function)
fNpfits                     0           Number of points used in the fit
fNDF                        0           Number of degrees of freedom in the fit
fNsave                      0           Number of points used to fill array fSave
fChisquare                  0           Function fit chisquare
*fIntegral                  ->0         ![fNpx] Integral of function binned on fNpx bins
*fParErrors                 ->0         [fNpar] Array of errors of the fNpar parameters
*fParMin                    ->0         [fNpar] Array of lower limits of the fNpar parameters
*fParMax                    ->0         [fNpar] Array of upper limits of the fNpar parameters
*fSave                      ->0         [fNsave] Array of fNsave function values
*fAlpha                     ->0         !Array alpha. for each bin in x the deconvolution r of fInteg
*fBeta                      ->0         !Array beta.  is approximated by x = alpha +beta*r *gamma*r**
*fGamma                     ->0         !Array gamma.
*fParent                    ->0         !Parent object hooking this function (if one)
*fHistogram                 ->8630e28   !Pointer to histogram used for visualisation

...

fOptimal                    ->87fbc28   !pointer to optimal function
fName                       ->87fbb6c   object identifier
fName.*fData                f1
fTitle                      ->87fbb74   object title
fTitle.*fData               sin(x)/x
fUniqueID                   0           object unique identifier
fBits                       0x03000008  bit field status word
fLineColor                  1           line color
fLineStyle                  1           line style
fLineWidth                  3           line width
fFillColor                  19          fill area color
fFillStyle                  0           fill area style
fMarkerColor                1           Marker color index
fMarkerStyle                1           Marker style
fMarkerSize                 1           Marker size
```

Now, create a 1-D histogram with 1000 bins that holds floats between 0 and 10:

```
root [4] TH1F hist1( <TAB>
TH1F TH1F()
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, Double_t xlow, Double_t xup)
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, const Float_t* xbins)
TH1F TH1F(const char* name, const char* title, Int_t nbinsx, const Double_t* xbins)
TH1F TH1F(const TVectorF& v)
TH1F TH1F(const TH1F& h1f)
root [4] TH1F hist1("hist1","Test histogram",1000,0,10)
```

Make a small "one-liner" (C++-style loop) to fill the histogram with some data:

```
root [13] for (Int_t i=0;i<10;i++){hist1.Fill(float(i));}
root [14] hist1.Draw()
```



Now clear the histogram and fill with something useful like a Gaussian random number with $\sigma = 1$ and $\mu = 5$:

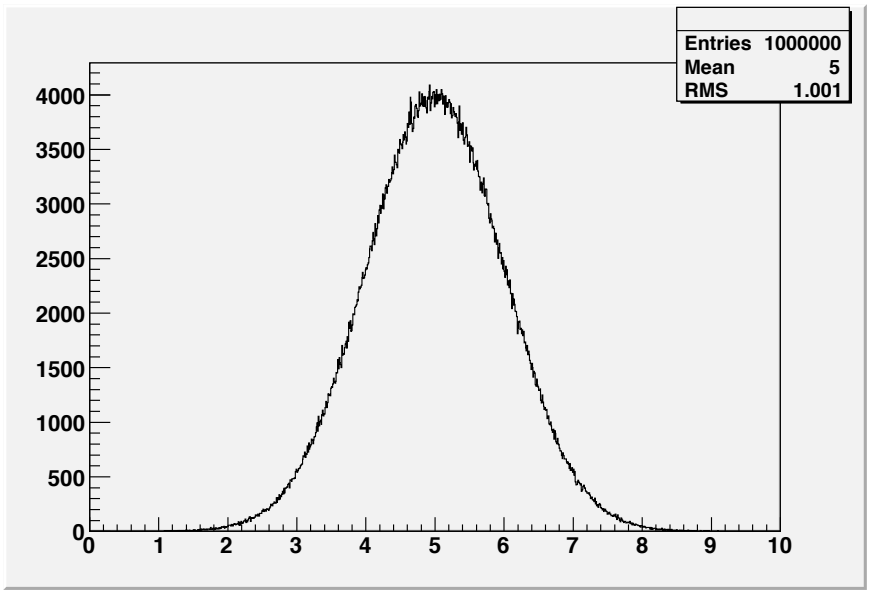
```
root [12] hist1.Reset()
root [26] for (Int_t i=0;i<1000000;i++){hist1.Fill(gRandom->Gaus(5,1));}
root [27] hist1.Draw()
```

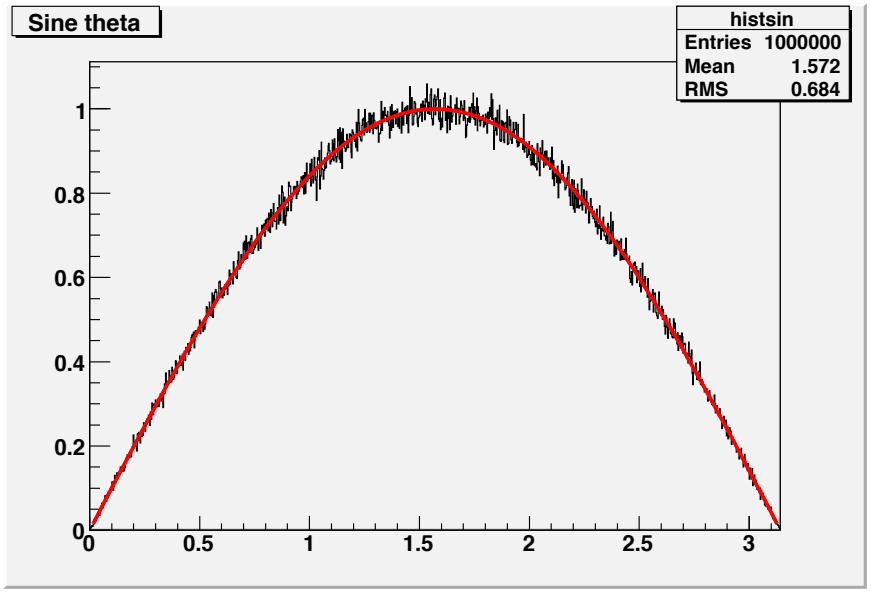
We now create a new function, $sin(x), x \subset [0, \pi]$, and a new histogram. We can then integrate the function directly, later used for normalisation. Note that $sin(x)$ and π are taken from from the `TMath` library which provides mathematical constants and functions in ROOT that should be consequent regardless of operating systems and architectures. Thus, this is preferred to using standard C++ functions, although these do work.

```
root [38] TF1 sintheta("sin_theta","TMath::Sin(x)",0,TMath::Pi())
Root [39] TH1F histsin("histsin","Sine theta",1000,0,TMath::Pi())
Root [56] sintheta.Integral(0,TMath::Pi())
(Double_t)2.00000000000000000e+00
```

Now use the function to create 10^6 random numbers and fill the histogram with the appropriate weighting factor (2nd argument) to get the right normalisation. Set the line colour to red and draw the function on top.

```
root [63] for (Int_t i=0;i<1000000;i++){histsin.Fill(sintheta.GetRandom(),2./TMath::Pi()/1000.);}
root [64] histsin->Draw()
root [54] sintheta.SetLineColor(2)
root [65] sintheta->Draw("same")
```


Entries 1000000
Mean 5
RMS 1.001
4000
3500
3000
2500
2000
1500
1000
500
0
0 1 2 3 4 5 6 7 8 9 10


Sine theta
histsin
Entries 1000000
Mean 1.572
RMS 0.684
1
0.8
0.6
0.4
0.2
0
0 0.5 1 1.5 2 2.5 3

This can now be used to generate random angles in a spherical coordinate system, taking the $d\Omega = sin(\theta)d\theta d\phi$ into account. We create a 2D- and a 3D-histogram and fill with random numbers, :

```
root [121] TH2F angles("angles","Theta and phi angles",50,0,TMath::Pi(),50,0,2*TMath::Pi())
for (Int_t i=0;i<1000000;i++){
end with '}', '@':abort > angles.Fill(sintheta.GetRandom(),gRandom->Uniform(0,2*TMath::Pi()));}
root [100] TH3F sphere("sphere","Spherical rep",100,-1,1,100,-1,1,100,-1,1)
root [115] for (Int_t i=0;i<1000000;i++){
end with '}', '@':abort > Float_t theta=sintheta.GetRandom();
end with '}', '@':abort > Float_t phi=gRandom->Uniform(0,2*TMath::Pi());
end with '}', '@':abort > Float_t x=TMath::Sin(theta)*TMath::Cos(phi);
end with '}', '@':abort > Float_t y=TMath::Sin(theta)*TMath::Sin(phi);
end with '}', '@':abort > Float_t z=TMath::Cos(theta);
end with '}', '@':abort > sphere.Fill(x,y,z);
end with '}', '@':abort > }
```

Note that one can split the command on several lines, however, then it is no longer available in the "arrow-up" history. If things get more complex, one rather puts the commands in an **unnamed macro** as will be addressed later. We now create a new canvas to plot the distributions, split it into two pads and use each for drawing.

```
root [123] TCanvas plots("plots","Test plot the spherical coordinates")

root [8] plots.Divide(2,1)
root [9] plots.cd(1)
(class TVirtualPad*)0x984c248
root [10] angles.Draw("surf1")

root [14] plots.cd(2)
(class TVirtualPad*)0x9879fc8
root [15] sphere.Draw()
```

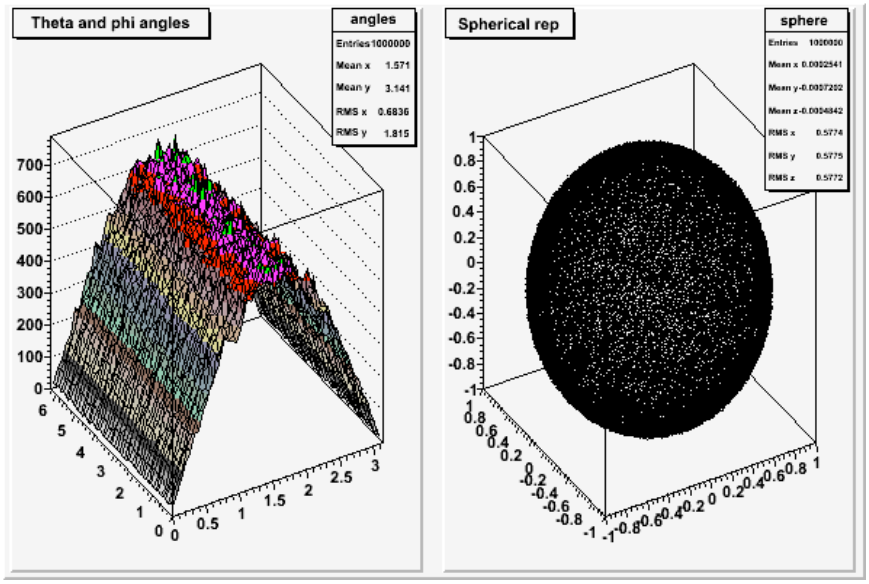
The standard colour coding is rather ugly, change the "palette" to get a temperature-like scale, and redraw this pad in the canvas:

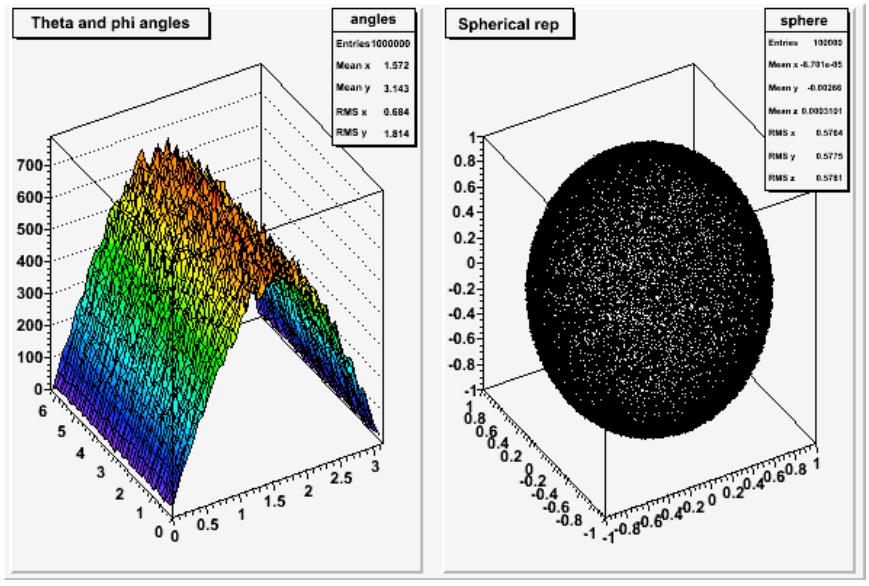
```
root [17] gStyle->SetPalette(1)
root [18] plots.cd(1)
(class TVirtualPad*)0x984c248
root [19] angles.Draw("surf1")
```

Now we can try to rotate the plots with the mouse to see all angles. This exercise will become useful when generating isotropic distributions.

3 Working with multi-parameter data

Let us start with a simulated example provided with the ROOT tutorials. We open an existing ROOT-file and check its contents by:


Theta and phi angles
angles
Entries 1000000
Mean x 1.571
Mean y 3.141
RMS x 0.6836
RMS y 1.815
Spherical rep
sphere
Entries 1000000
Mean x 0.0002541
Mean y -0.0007202
Mean z -0.0004842
RMS x 0.5774
RMS y 0.5775
RMS z 0.5772


Theta and phi angles
angles
Entries 1000000
Mean x 1.572
Mean y 3.143
RMS x 0.684
RMS y 1.814
Spherical rep
sphere
Entries 1000000
Mean x -8.701e-05
Mean y -0.00266
Mean z 0.0003101
RMS x 0.5764
RMS y 0.5775
RMS z 0.5781

```
root [0] TFile *g = new TFile("hsimple.root")
root [1] g->ls()
TFile**         hsimple.root    Demo ROOT file with histograms
 TFile*         hsimple.root    Demo ROOT file with histograms
  KEY: TH1F     hpx;1   This is the px distribution
  KEY: TH2F     hpxpy;1 py vs px
  KEY: TProfile hprof;1 Profile of pz versus px
  KEY: TNtuple  ntuple;1        Demo ntuple
```

The file contains a few histograms and a **TNtuple** (see e.g. http://en.wikipedia.org/wiki/Tuple) which is a simple special case of a **tree**. We can now look at the components of this tree:

```
root [2] ntuple->Print()
******************************************************************************
*Tree    :ntuple    : Demo ntuple                                            *
*Entries :    25000 : Total =          504510 bytes  File  Size =     400912 *
*        :          : Tree compression factor =   1.25                       *
******************************************************************************
*Br    0 :px        :                                                        *
*Entries :    25000 : Total  Size=     100835 bytes  File Size  =      89135 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.08     *
*.............................................................................*
*Br    1 :py        :                                                        *
*Entries :    25000 : Total  Size=     100835 bytes  File Size  =      89138 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.08     *
*.............................................................................*
*Br    2 :pz        :                                                        *
*Entries :    25000 : Total  Size=     100835 bytes  File Size  =      87573 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.10     *
*.............................................................................*
*Br    3 :random    :                                                        *
*Entries :    25000 : Total  Size=     100871 bytes  File Size  =      86569 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.11     *
*.............................................................................*
*Br    4 :i         :                                                        *
*Entries :    25000 : Total  Size=     100826 bytes  File Size  =      30145 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   3.18     *
*.............................................................................*
```

There are five parameters in the tree, that have a value for each row/instance/event. To look at the values, do:

```
root [3] ntuple->Scan()
************************************************************************
*    Row   *        px *        py *        pz *    random *         i *
************************************************************************
*        0 * 0.7333162 * -0.334522 * 0.6496582 * 0.7560786 *         0 *
*        1 * 1.1401898 * 1.0532406 * 2.4093489 * 0.6752759 *         1 *
*        2 * -1.891556 * 1.7905917 * 6.7842035 * 0.9607744 *         2 *
*        3 * -0.189015 * -0.452100 * 0.2401217 * 0.6592899 *         3 *
*        4 * -1.510613 * -1.944864 * 6.0644493 * 0.0584281 *         4 *
```

```
*        5 * 0.1561557 * 0.1052679 * 0.0354659 * 0.4782272 *         5 *
*        6 * -1.208183 * -0.583504 * 1.8001855 * 0.6153143 *         6 *
*        7 * 0.3560560 * -0.532592 * 0.4104308 * 0.3613421 *         7 *
*        8 * 0.6012339 * 0.7635735 * 0.9445269 * 0.7553101 *         8 *
*        9 * 1.0431286 * 0.0555231 * 1.0912001 * 0.5204855 *         9 *
*       10 * -0.289692 * -0.276306 * 0.1602667 * 0.5363826 *        10 *
*       11 * 0.3931210 * -2.166756 * 4.8493762 * 0.4711117 *        11 *
...
```

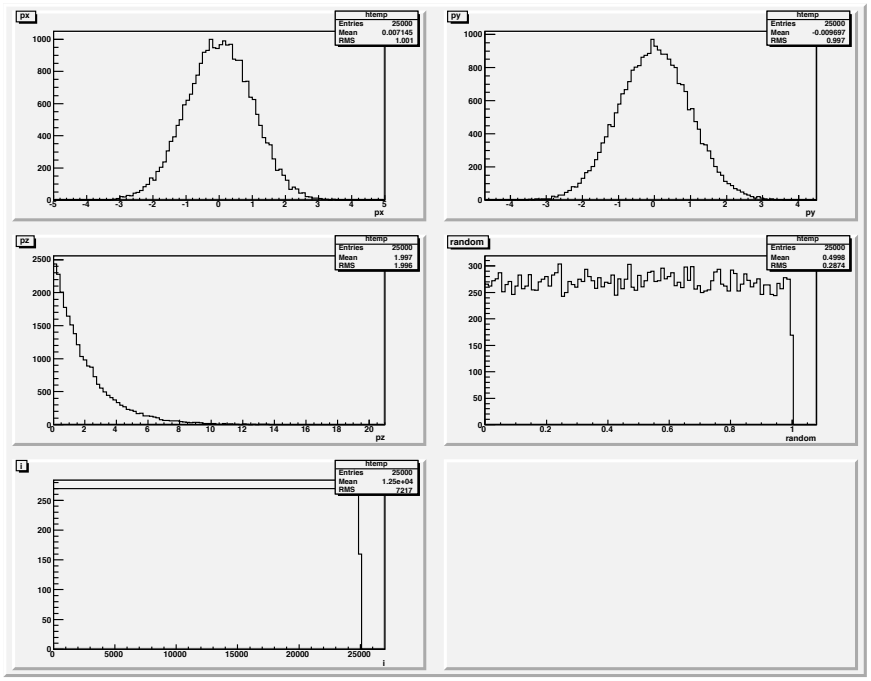
We can conclude that the *px,py,pz* denote some kind of three-dimensional momenta, and that the *random* and i simply denote random numbers and the event numbers, respectively. Plot the parameters to check this:

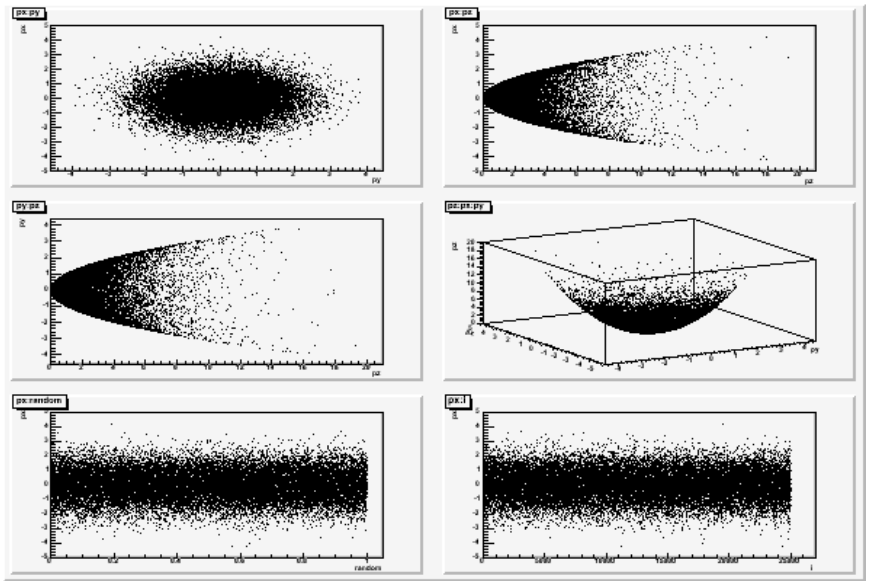
```
root [4] TCanvas plots("plotsim","Plot simulated data")
root [5] plots.Divide(2,3)
root [6] plots.cd(1)
(class TVirtualPad*)0x9047b18
root [7] ntuple->Draw("px")
root [8] plots.cd(2)
(class TVirtualPad*)0x905e740
root [9] ntuple->Draw("py")
root [10] plots.cd(3)
(class TVirtualPad*)0x905e950
root [11] ntuple->Draw("pz")
root [12] plots.cd(4)
(class TVirtualPad*)0x905eb60
root [13] ntuple->Draw("random")
root [14] plots.cd(5)
(class TVirtualPad*)0x92719e8
root [15] ntuple->Draw("i")
```

Now investigate how the parameters are related to each other by plotting combinations:

```
root [18] plots.cd(1)
(class TVirtualPad*)0x9047b18
root [19] ntuple->Draw("px:py")
root [20] plots.cd(2)
(class TVirtualPad*)0x905e740
root [21] ntuple->Draw("px:pz")
root [22] plots.cd(3)
(class TVirtualPad*)0x905e950
root [23] ntuple->Draw("py:pz")
root [24] plots.cd(4)
(class TVirtualPad*)0x905eb60
root [28] ntuple->Draw("pz:px:py")
root [29] plots.cd(5)
(class TVirtualPad*)0x92719e8
root [31] ntuple->Draw("px:random")
root [32] plots.cd(6)
(class TVirtualPad*)0x9271bf8
root [33] ntuple->Draw("px:i")
```

By looking at the code (from **hsimple.C**) producing the parameters, we can probably understand the patterns


px
htemp
Entries 25000
Mean 0.007145
RMS 1.001
py
htemp
Entries 25000
Mean -0.009697
RMS 0.997
pz
htemp
Entries 25000
Mean 1.997
RMS 1.996
random
htemp
Entries 25000
Mean 0.4998
RMS 0.2874
i
htemp
Entries 25000
Mean 1.25e+04
RMS 7217


px:py
px:pz
py:pz
pz:pt:py
px:random
px:i

```
  for (Int_t i = 0; i < 25000; i++) {
     gRandom->Rannor(px,py);
     pz = px*px + py*py;
     Float_t random = gRandom->Rndm(1);
     hpx->Fill(px);
     hpxpy->Fill(px,py);
     hprof->Fill(px,pz);
     ntuple->Fill(px,py,pz,random,i);
  ...
```

Let us continue with a non-physics example; some old personnel data from CERN that is provided with the ROOT tutorials. The raw data in **cernstaff.dat** looks like

```
  202    15    58    28     0    10    13     40      11975   PS  DE
  530    15    63    33     0     9    13     40      10228   EP  CH
  316    15    56    31     2     9    13     40      10730   PS  FR
  361    15    61    35     0     9     7     40       9311   PS  FR
  302    15    52    24     2     9     8     40       9966   PS  DE
  303    15    60    33     0     7    13     40       7599   PS  IT
  302    15    53    25     1     9     9     40       9868   ST  CH
  361    15    60    32     1     8     5     40       8012   PS  IT
  340    15    51    28     0     8    13     40       8813   PS  DE
  361    15    56    32     1     7    13     40       7850   PS  FR
  361    15    51    29     0     7    13     40       7599   PS  FR
  303    15    54    31     2     8    13     40       9315   PS  CH
  302    15    54    29     0     7    13     40       7599   PS  CH
...
```

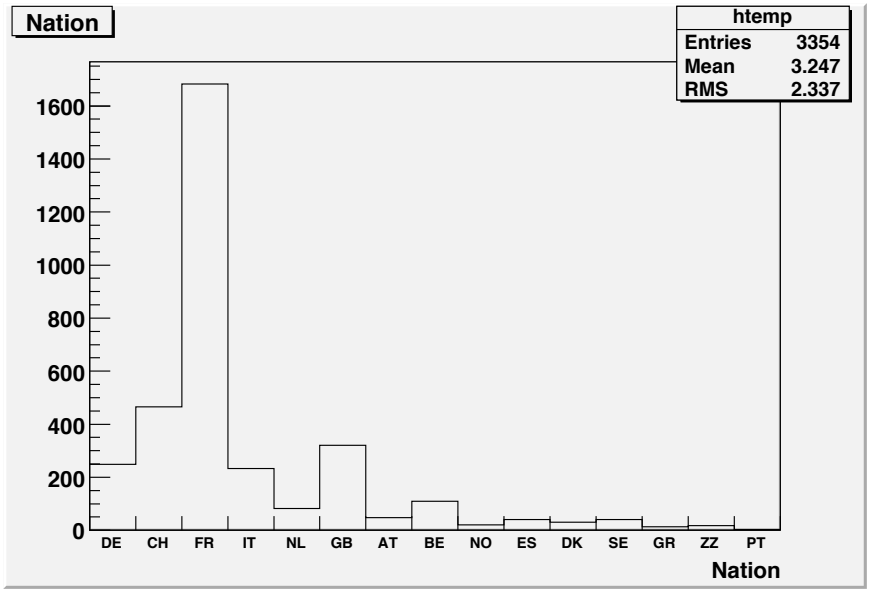
A small program **cernbuild.C** reads the column-wise data, creates a *tree* with *branches* for each parameter and finally saves the tree in **cernstaff.root**. We can then open this file, check what is inside and work with the tree **T**:

```
root [4] TFile f("cernstaff.root")
root [5] f.ls()
TFile**         cernstaff.root
 TFile*         cernstaff.root
  KEY: TTree    T;1     CERN 1988 staff data
root [6] T.Print()
******************************************************************************
*Tree    :T         : CERN 1988 staff data                                    *
*Entries :     3354 : Total =          176148 bytes  File  Size =      48066 *
*        :          : Tree compression factor =   2.74                       *
******************************************************************************
*Br    0 :Category  : Category/I                                              *
*Entries :     3354 : Total  Size=      14054 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    1 :Flag      : Flag/i                                                  *
*Entries :     3354 : Total  Size=      14030 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    2 :Age       : Age/I                                                   *
*Entries :     3354 : Total  Size=      14024 bytes  One basket in memory    *
```

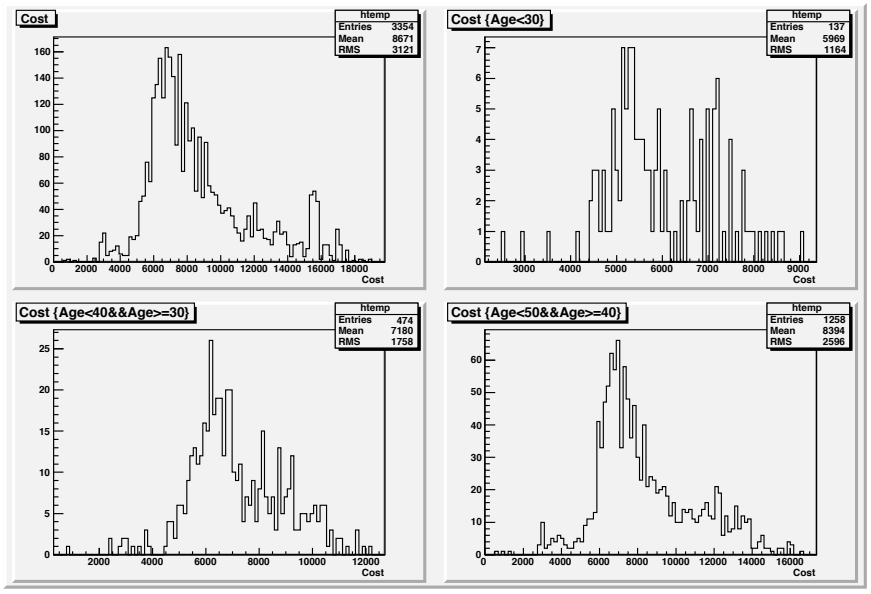
```
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.....................................................................*
*Br    3 :Service   : Service/I                                        *
*Entries :     3354 : Total  Size=      14048 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.....................................................................*
*Br    4 :Children  : Children/I                                       *
*Entries :     3354 : Total  Size=      14054 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.....................................................................*
*Br    5 :Grade     : Grade/I                                          *
*Entries :     3354 : Total  Size=      14036 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *

...

*.....................................................................*
*Br    9 :Division  : Division/C                                       *
*Entries :     3354 : Total  Size=      25307 bytes  File Size  =       8325 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   2.49     *
*.....................................................................*
*Br   10 :Nation    : Nation/C                                         *
*Entries :     3354 : Total  Size=      24190 bytes  File Size  =       6680 *
*Baskets :        1 : Basket Size=      32000 bytes  Compression=   3.05     *
*.....................................................................*
```

We can also plot non-numerical data:

```
root [33]  T.Draw("Nation","")
```



and put conditions (C++-style) on another parameter as the second argument, to see the distribution of cost/person in different age intervals:




```
root [16]  TCanvas plots("plotstaff","Plot some multipar data")
root [17] plots.Divide(2,2)
root [18] plots.cd(1)
(class TVirtualPad*)0xa7cce30
root [21]  T.Draw("Cost","","")
(Long64_t)3354
root [22] plots.cd(2)
(class TVirtualPad*)0xab76548
root [23]  T.Draw("Cost","Age<30","")
(Long64_t)137
root [24] plots.cd(3)
(class TVirtualPad*)0xab76758
root [25]  T.Draw("Cost","Age<40&&Age>=30","")
(Long64_t)474
root [26] plots.cd(4)
(class TVirtualPad*)0xab76968
root [27]  T.Draw("Cost","Age<50&&Age>=40","")
```

4 Some real physics data!

Now there will be a real challenge; to look at some actual data from the S245 experiment at GSI. The experimental set-up is depicted in figure 1. A relativistic beam of radioactive ions enters the set-up, where each ion is characterised separately with respect to velocity, charge and momentum vector. In the target, a reaction might take place, breaking up the incoming ion into lighter ions and nucleons, that can be detected in a range of various detectors. The ions continuing in the beam direction are bent by a dipole magnet to separate the reaction products from the non-reacting beam.

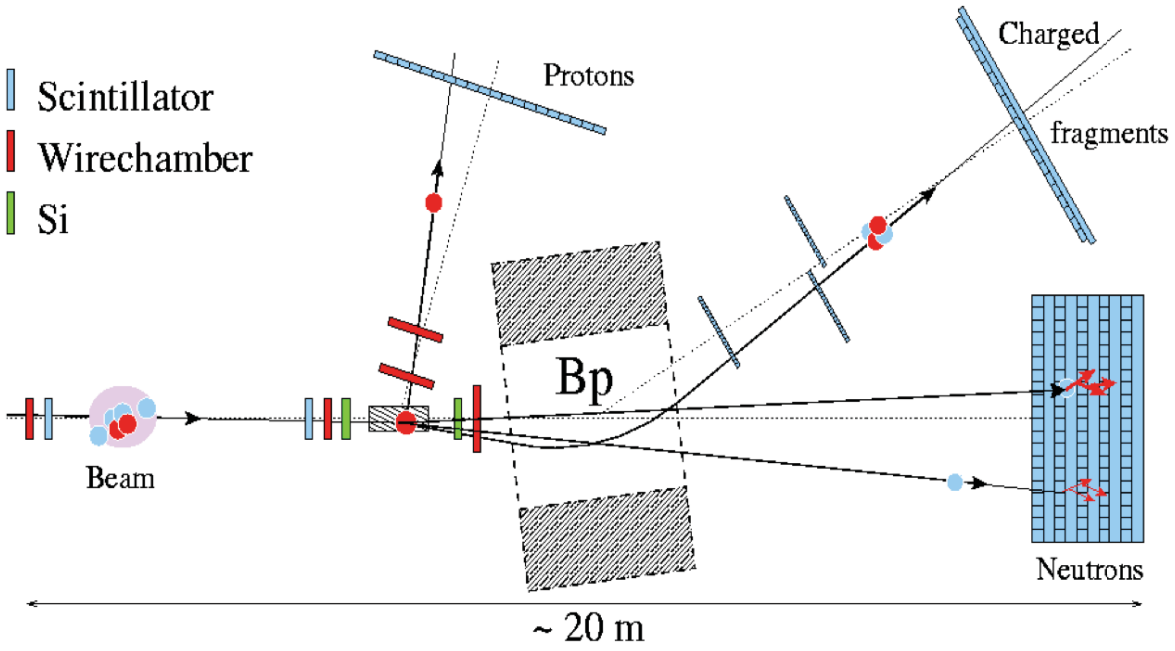


Figure 1: Schematic view of the S245 set-up at GSI. The detectors are labeled with their corresponding parameters.

There are several data files, with different kind of selections made on the relevant event. Start with data where the proton wall has detected something:

```
root [69] TFile *f = new TFile("r10_470-529_beam_proton_hit_track.root")
root [70] f.ls()
TFile**         r10_470-529_beam_proton_hit_track.root  HBOOK file: r10_470-529_beam_proton_hit_track.r
 TFile*         r10_470-529_beam_proton_hit_track.root  HBOOK file: r10_470-529_beam_proton_hit_track.r
  KEY: TTree    h509;1  LAND
root [71] h509->Print()
******************************************************************************
*Tree    :h509      : LAND                                                   *
*Entries :   120551 : Total =        16899058 bytes  File  Size =    7874836 *
*        :          : Tree compression factor =   2.15                       *
******************************************************************************
*Br    0 :Tpat      : Tpat/s                                                 *
*Entries :   120551 : Total  Size=     241943 bytes  File Size  =      70682 *
*Baskets :        3 : Basket Size=      64000 bytes  Compression=   2.72     *
*.............................................................................*
*Br    1 :Evnt      : Evnt/I                                                 *
*Entries :   120551 : Total  Size=     483349 bytes  File Size  =     237073 *
*Baskets :        7 : Basket Size=      64000 bytes  Compression=   1.89     *
*.............................................................................*

...

*.............................................................................*
*Br   30 :Tomul     : Tomul/I                                                *
```

```
*Entries :   120551 : Total  Size=     483362 bytes  File Size  =      29618 *
*Baskets :        7 : Basket Size=      64000 bytes  Compression=  15.12     *
*.............................................................................*
*Br   31 :Tot       : Tot[Tomul]/F                                            *
*Entries :   120551 : Total  Size=     539143 bytes  File Size  =      90738 *
*Baskets :       15 : Basket Size=      64000 bytes  Compression=   5.58     *
*.............................................................................*
*Br   32 :Toe       : Toe[Tomul]/F                                            *
*Entries :   120551 : Total  Size=     539143 bytes  File Size  =      92785 *
*Baskets :       15 : Basket Size=      64000 bytes  Compression=   5.46     *
*.............................................................................*
*Br   33 :Tox       : Tox[Tomul]/F                                            *
*Entries :   120551 : Total  Size=     539143 bytes  File Size  =      93562 *
*Baskets :       15 : Basket Size=      64000 bytes  Compression=   5.41     *
*.............................................................................*
*Br   34 :Toy       : Toy[Tomul]/F                                            *
*Entries :   120551 : Total  Size=     539143 bytes  File Size  =      93598 *
*Baskets :       15 : Basket Size=      64000 bytes  Compression=   5.41     *
*.............................................................................*
```

Now several parameters are indexed, like the Time-of-flight wall shown in the above listing. This is needed since these detectors are capable of detecting several ions or particles for each incoming ion, i.e. in the same event. The number of hits, the *multiplicity*, is given by e.g. the `Tomul` parameter, and the corresponding x-,y-,... coordinates can be addressed through e.g. `Tox[0..Tomul]`. If the index is omitted, the **Draw**-command will cycle through the indices.

Before continuing, it could be useful to scan through all the parameters in the data for checking. But going through all the single parameters (the **Leaves** of the data **tree** is tedious, so we would like to access them in a systematic way, plot a number of them on a page and continue with the next set. This becomes somewhat more entangled, so instead of the command line, the commands are put in an **unnamed script** named `drawLeaves.C`:

```
{
  // Macro to plot all parameters into a multi-page pdf-file of a
  // simple TTree h509 that is already opened within ROOT
  // Execute the macro by ".x drawLeaves.C"

//Create a canvas and divide it into 9 pads
  TCanvas *cc= new TCanvas("cc","Test multipage");
  cc->Divide(3,3);

  // Get an array of leaves in the TTree into an array
  TObjArray *listLeaves=h509->GetListOfLeaves();
  // Check how many pages are needed
  Int_t numberCanvas=listLeaves->LastIndex()/9;

  //Add one if the number is not divisible with 9
  if (listLeaves->LastIndex()%9>0) numberCanvas++;

  // Start looping the pages
```

```
  for (Int_t i=0;i<numberCanvas;i++){

    //... and the pads in each page
    for (Int_t j=0;j<9;j++){
      cc->cd(j+1);
      //gPad->SetLogy();
      gPad->Clear();
      printf("Plotting canvas %i, pad %i\n",i,j);
      // Check that we still are within the boundaries of the leaves,
      // pick out the parameter name from the array and draw it
      if (i*9+j<=listLeaves->LastIndex())
h509->Draw(listLeaves->At(i*9+j)->GetName());
    }
 //Open a pdf-file for writing, note the opening bracket after the file name
    if (i==0) cc->Print("Parscan.pdf(","pdf");
    // Close the file after last page
    else if (i==numberCanvas-1) cc->Print("Parscan.pdf)","pdf");
    // Pages in between
    else cc->Print("Parscan.pdf","pdf");
  }
}
```

This script can now be executed by the command

```
root [146] .x drawLeaves.C
```

and generates a multi-page pdf-file with all parameters plotted.

4.1 Incoming ions

First we look at the incoming ions by plotting the charge as a function of the relativistic $\beta = v/c$:

```
h509->Draw("inz:inbeta")
```

All ions have to approximately have the same *rigidity* expressed in $B\rho$, i.e. in a magnet field of strength B, the ions will follow a circular track with radius ρ, which is related to the charge and total momentum in MeV/c by:

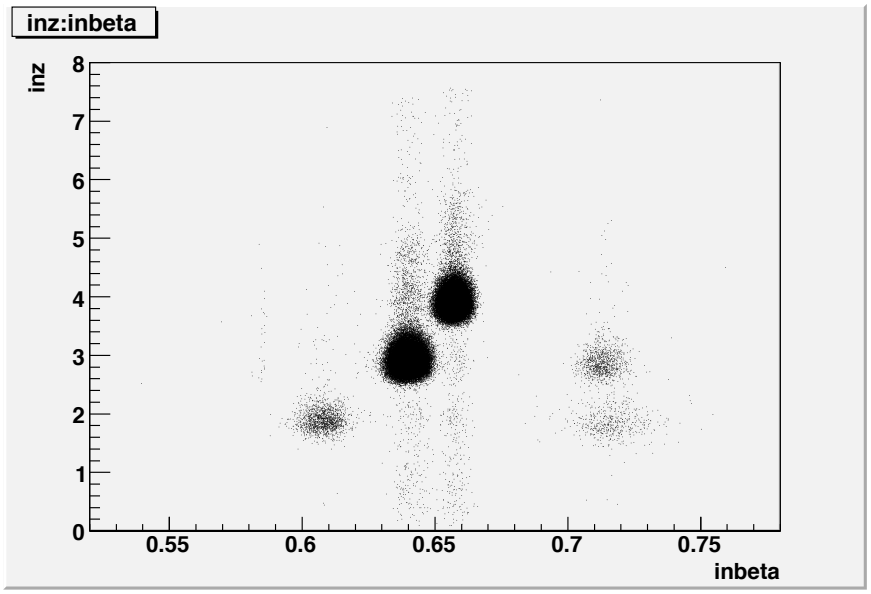
$$B \cdot \rho = \frac{p}{300Z}[T \cdot m] \tag{1}$$

These data were taken with a setting of $9.5T \cdot m$. Since $p = Am_n\gamma\beta$ and $\gamma = 1/\sqrt{1-\beta^2}$, we can construct and use A/Z instead and identify the isotope mixture of the beam (alternatively, create a string with the formula):

```
root [94] h509->Draw("inz:9.5*300./935.*sqrt(1/(inbeta*inbeta)-1)")
root [96] TString aoverz = "9.5*300./935.*sqrt(1/(inbeta*inbeta)-1)"
root [99] h509->Draw("inz:"+aoverz)
```

This can now be used to select the incoming ions by a graphical cut. However, this is quite messy in ROOT, following the steps below:

- In the graphics pad, choose *View* → *Toolbar* go get a toolbar above the plot.



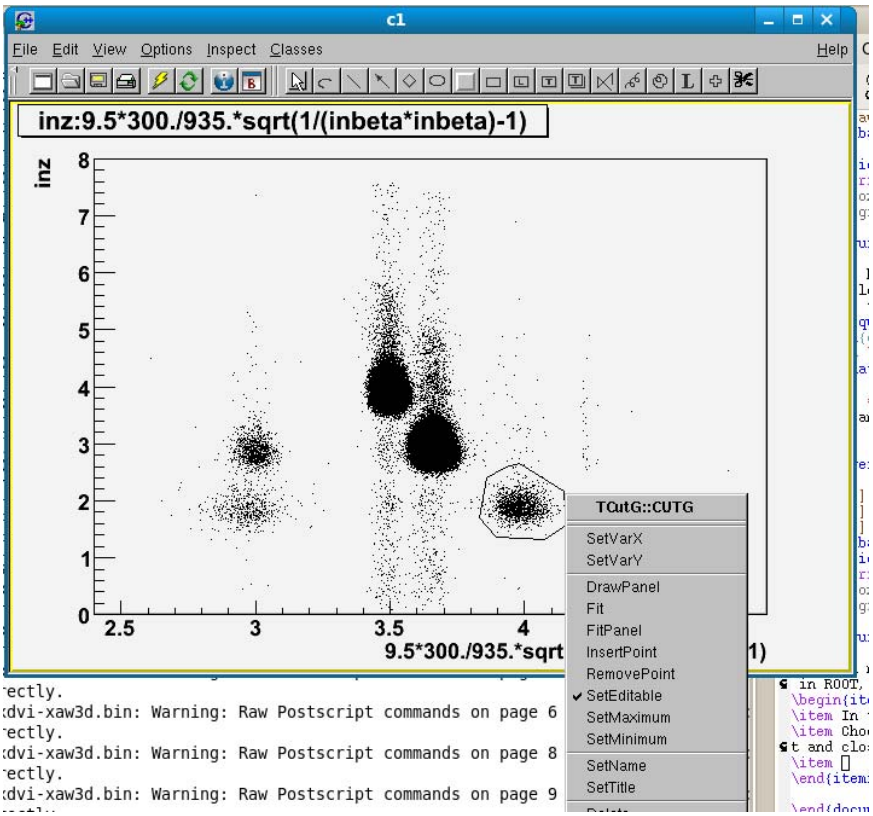


Figure 2: Cut selection

- Choose the scissors symbol, click around the spot corresponding to e.g. ^8He to enclose it and close the cut by double-clicking.
- Right-click on the cut line to see a context menu (as shown in fig. 4.1, and choose *SetName*. In the pop-up window, change **CUTG** to e.g. **He8**
- Now, to make the cut available to ROOT, use the command:

```
root [11] TCutG *He8 = (TCutG*) gPad->GetPrimitive("He8")
```

We can now check the properties of the cut to ensure that everything is OK.

```
root [12] He8->Print()
x[0]=3.9643, y[0]=2.53814
x[1]=4.11246, y[1]=2.19915
x[2]=4.14838, y[2]=1.54237
x[3]=4.05409, y[3]=1.33051
x[4]=3.93736, y[4]=1.35169
x[5]=3.85654, y[5]=1.58475
x[6]=3.85205, y[6]=2.11441
x[7]=3.91491, y[7]=2.36864
x[8]=3.9643, y[8]=2.53814
root [13] He8->SetLineColor(2)
root [14] He8->Draw()
```

Since the creation of these cuts was tedious, they can be saved to a file for later use:

```
root [11] TFile *cutfile = new TFile("cutfile.root","recreate")
root [12] Be14->Write()
(Int_t)(321)
root [13] Li11->Write()
(Int_t)(328)
root [14] He8->Write()
(Int_t)(331)
root [15] cutfile.ls()
TFile**         cutfile.root
 TFile*         cutfile.root
  KEY: TCutG    Be14;1  Graph
  KEY: TCutG    Li11;1  Graph
  KEY: TCutG    He8;1   Graph
root [20] cutfile->Close()
```

...and later read by:

```
root [21] TFile *cutfile = new TFile("cutfile.root")
root [22] cutfile.ls()
TFile**         cutfile.root
 TFile*         cutfile.root
  KEY: TCutG    Be14;1  Graph
  KEY: TCutG    Li11;1  Graph
  KEY: TCutG    He8;1   Graph
```

```
root [23] cutfile->ReadAll()
root [24] cutfile->Close()
root [25]
root [25] He8->Print()
x[0]=3.9643, y[0]=2.53814
x[1]=4.11246, y[1]=2.19915
x[2]=4.14838, y[2]=1.54237
x[3]=4.05409, y[3]=1.33051
x[4]=3.93736, y[4]=1.35169
x[5]=3.85654, y[5]=1.58475
x[6]=3.85205, y[6]=2.11441
x[7]=3.91491, y[7]=2.36864
x[8]=3.9643, y[8]=2.53814
```

After having manipulated files, we might end up in the wrong *directory* inside ROOT, yielding:

```
root [34] h509->Draw("inz:"+aoverz)
Error: Symbol h509 is not defined in current scope  (tmpfile):1:
Error: Failed to evaluate h509->Draw("inz:"+aoverz)
*** Interpreter error recovered ***
```

Simply go to the directory with the data tree by:

```
root [35] f.cd()
```

These cuts can now be used to select the different incoming ions. This is visualised by plotting the events on top with different colours:

```
root [30] h509->Draw("inz:"+aoverz)
root [31] h509->SetMarkerColor(2)
root [32] h509->Draw("inz:"+aoverz,"He8","same")
(Long64_t)1469
root [33] h509->SetMarkerColor(3)
root [34] h509->Draw("inz:"+aoverz,"Li11","same")
(Long64_t)60656
root [35] h509->SetMarkerColor(4)
root [36] h509->Draw("inz:"+aoverz,"Be14","same")
(Long64_t)53310
```

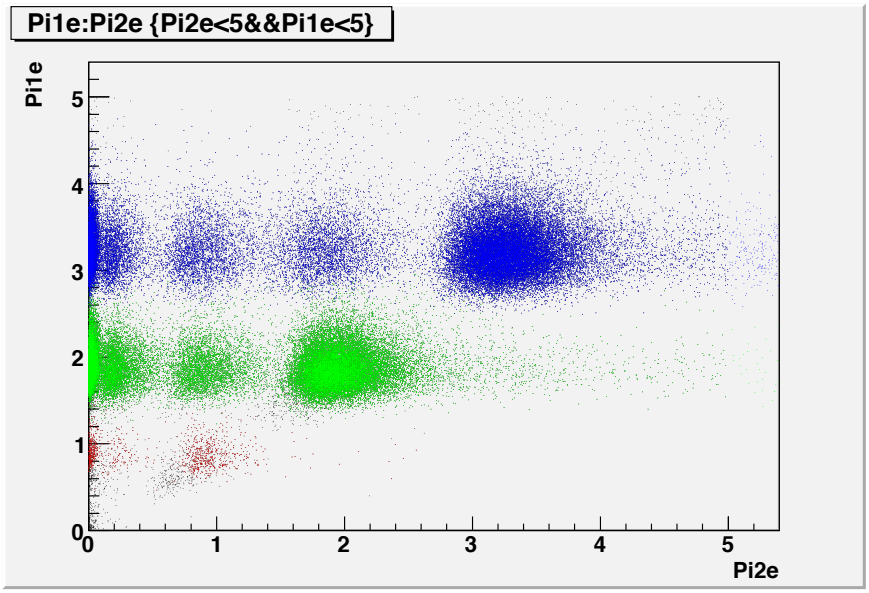
Now, we can start to look at the first detectors just up- and downstream from the reaction target. These *silicon pad* detectors register the energy loss, at these energies we can approximate the deposited energy by the main behaviour of the Bethe-Bloch formula to $\Delta E \propto Z^2/\beta^2$. Regretfully, these are in this data file not (yet) calibrated to give the actual Z of the ions, but we can infer this and still use the information. By plotting the energy loss before and after the target, corresponding to the parameters `Pi1e` and `Pi2e` respectively, against each other, the *charge-changing* reactions become visible.

```
root [66] h509->SetMarkerColor(1)
root [67] h509->Draw("Pi1e:Pi2e","Pi2e<5&&Pi1e<5")
```

By using our old cuts, the events related to the separate ions can be identified:

```
root [37] h509->SetMarkerColor(1)
root [38] h509->Draw("Pi1e:Pi2e","Pi2e<5&&Pi1e<5")
(Long64_t)91704
```

```
root [39] h509->SetMarkerColor(2)
root [40] h509->Draw("Pi1e:Pi2e","He8","same")
(Long64_t)1469
root [41] h509->SetMarkerColor(3)
root [42] h509->Draw("Pi1e:Pi2e","Li11","same")
(Long64_t)60656
root [43] h509->SetMarkerColor(4)
root [44] h509->Draw("Pi1e:Pi2e","Be14","same")
(Long64_t)53310
```



The large rightmost blue peak shows $14Be$-ions exiting the target with unchanged charge, i.e. still as a beryllium isotope. To the left, there are further peaks where the ions have $Z = 3, 2(, 1)$ (although the latter is partially below threshold) showing that lithium, helium and hydrogen ions have been created. There are corresponding peaks stemming from the ^{11}Li and ^{8}He beams.

Question: What are the black events at the edge of the ^{11}Li and ^{8}He peaks?

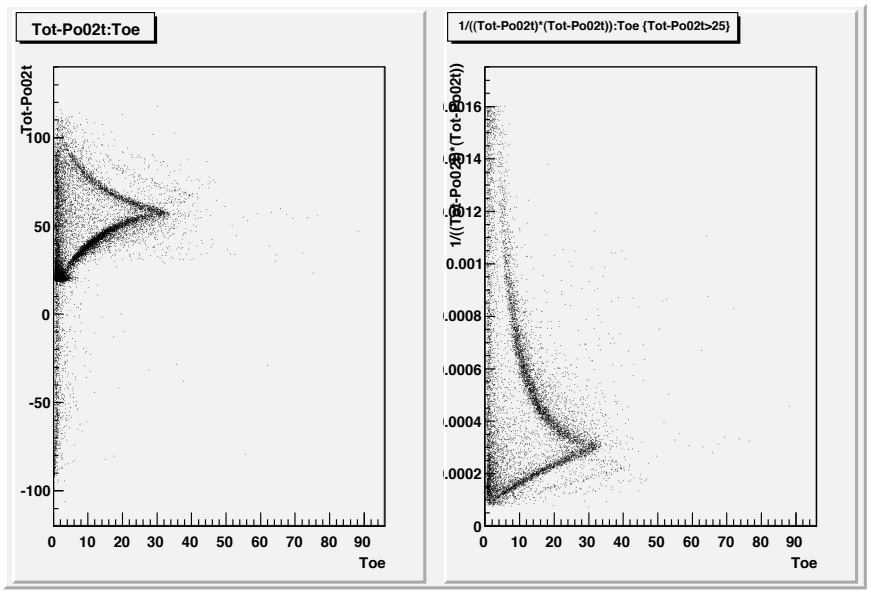
4.2 Scattered protons

Protons (and other charged particles) could be detected in the plastic scintillator hodoscope situated close to the reaction target. Start by looking at the *time-of-flight* information vs. the energy deposited in the scintillator:

```
root [50] h509->Draw("Tot:Toe")
```

However, the time has to be relative to something, here the time from the **Pos2**-detector is taken as a reference.

```
root [57] TCanvas *cc= new TCanvas("cc","Proton wall")
root [58] cc->Divide(2,1)
root [59] cc->cd(1)
(class TVirtualPad*)0x864f040
root [60] h509->Draw("Tot-Po02t:Toe")
```



But if the time-of-flight is know, this can also be used for determining the energy of the particle, without fixing the distance and in the non-relativistic case, $T = mv/2 \propto (\Delta t)^{-2}$ (finding reasonable limits requires some trial-and-error).

```
root [61] cc->cd(2)
root [73] h509->Draw("1/((Tot-Po02t)*(Tot-Po02t)):Toe","Tot-Po02t>25")
```

Now, there are still two branches in the plot, but we note that there is a linear dependency between the energy of the particle and the deposited energy in the detector when following the lower branch. This means that the particles are fully stopped in the scintillator, however, with increasing energy they will start to **punch through** the detector, and only lose a fraction of their energy. This fraction of the energy, in accordance with the Bethe-Bloch formula, decreases as the energy of the particle increases. The detector works as a ΔE-detector in this case.

Question: Assuming that the strongest distribution of events stems from protons, what other can be seen vaguely in the plots? Motivate!

Exercise: Check differences in the above plots distributions when demanding a specific incoming beam.

4.3 Outgoing ions

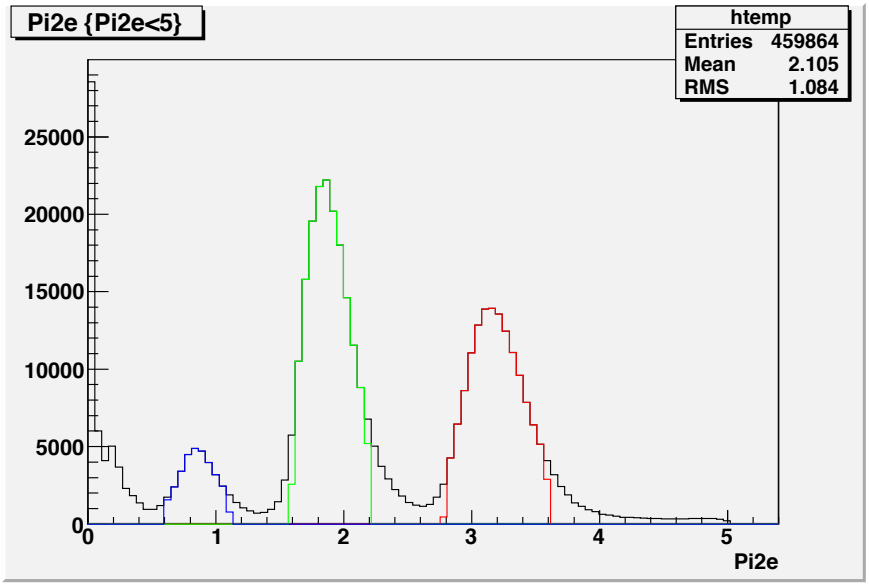
The outgoing ions are subsequently analysed in the ALADIN dipole magnet, meaning that they will follow different tracks depending ipon their rigidity. This can than be exploited both for ion identification and momentum measurements. A full-fledged **tracker** is the ultimate method, but we will see that also by simpler means, a good identification can be made. We now have to use another data file which has the information from the detectors downstreams of the magnet:

```
root [10] TFile *f = new TFile("r10_53x_beam_frag_hit_track.root")
```

First, start by making conditions for the charges of the ions following the reaction target:

```
root [12] h509->Draw("Pi2e","Pi2e<5")
(Long64_t)459864
root [13] TCut z4out = "Pi2e>2.8&&Pi2e<3.6"
```

```
root [14] TCut z3out = "Pi2e>1.6&&Pi2e<2.2"
root [15] TCut z2out = "Pi2e>0.6&&Pi2e<1.1"
root [16] h509->SetLineColor(2);h509->Draw("Pi2e",z4out,"same")
root [17] h509->SetLineColor(3);h509->Draw("Pi2e",z3out,"same")
root [18] h509->SetLineColor(4);h509->Draw("Pi2e",z2out,"same")
```



We can make use of the Zs2 (wire chamber) directly after the target, and the Tf (time-of-flight hodoscope) behind the dipole magnet to look at positions, all calibrated in centimeters. However, looking directly reveals only broad distributions, with pronounced maxima in the middle and to the left respectively. In the latter case, this corresponds to the non-reacted beam which was intentionally put in this position (close to the beam direction), since the reaction products normally for very neutron-rich nuclei have less rigidity.

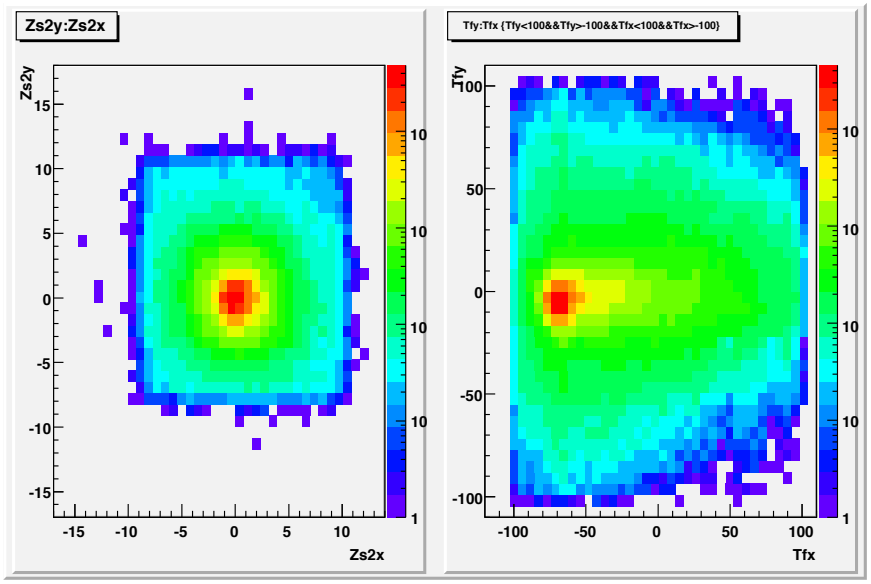
```
root [9] TCanvas *beam = new TCanvas("beam","Beam detectors post-target")
root [10] beam->Divide(2,1)
root [11] beam->cd(1)
(class TVirtualPad*)0x202c600
root [12] h509->Draw("Zs2y:Zs2x","","colz")
(Long64_t)515924
root [13] beam->cd(2)
(class TVirtualPad*)0x202ca00
root [14] h509->Draw("Tfy:Tfx","Tfy<100&&Tfy>-100&&Tfx<100&&Tfx>-100","colz")
```

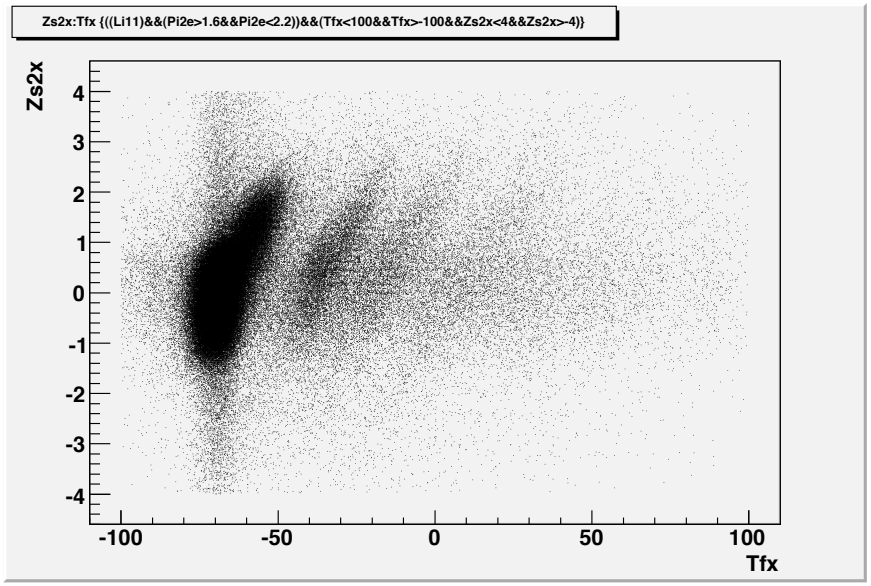
However, by using the combined position information of the detectors in the bending plane (x-) direction and select e.g. incoming ^{11}Li ions and demanding $Z = 3$ of the outgoing ion:

```
root [42] h509->Draw("Zs2x:Tfx","Li11"&&z3out&&"Tfx<100&&Tfx>-100&&Zs2x<4&&Zs2x>-4")
```

Thus, by separating the offset in the Zs2x from the position in the Tf, four elliptic distributions can be identified that correspond to $^{11,9,8,7}Li$ from left to right. With this information, the full reaction can now be identified.

Hand-in task 1: Use the above information and that the flight path is 7.54 m to construct a $p_\perp$ distribution for a two-neutron removal reaction. (Similar information was published in a Phys. Lett. B 1995...)


Zs2y:Zs2x
Zs2y
Zs2x
Tfy:Tfx {Tfy<100&&Tfy>-100&&Tfx<100&&Tfx>-100}
Tfy
Tfx


Zs2x:Tfx {((Li11)&&(Pi2e>1.6&&Pi2e<2.2))&&(Tfx<100&&Tfx>-100&&Zs2x<4&&Zs2x>-4)}
Zs2x
Tfx

5 Simulations

Simulating a physical process and the response in a given detector set-up is an integrated part of any contemporary subatomic physics experiment. There are highly specialised and complicated simulation packages like GEANT4 and FLUKA, but the threshold effort needed to use these is far beyond the scope of this course. However, some general concepts can be explored.

5.1 Relativistic kinematics

There are several classes in ROOT related to kinematics that are very helpful and makes it a "4-vector calculator". The class TLorentzVector contains all relevant parameters for a four vector with (-,-,-,+) metric:

```
root [87] TLorentzVector *vect = new TLorentzVector()
root [88] vect
(class TLorentzVector*)0x914af18
root [89] vect->Dump()
==> Dumping object at: 0x0914af18, name=TLorentzVector, class=TLorentzVector

fP                          ->914af24   3 vector component
fP.fX                       0
fP.fY                       0
fP.fZ                       0
fP.fUniqueID                0           object unique identifier
fP.fBits                    0x03000000  bit field status word
fE                          0           time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                   0           object unique identifier
fBits                       0x03000000  bit field status word

root [90] vect->SetPxPyPzE(0,0,0,939) //A proton at rest
```

Assume that we would like to simulate $\overline{\Lambda}\Lambda$ production in $\overline{p}p$ collisions. The recipe would be to transform the antiproton beam and the proton target to the CM system, determine the total energy and from that subtract the rest mass of the created particles. The excess mass is then shared by the product as kinetic energy, with momentum vectors in opposite, random directions. Finally, the lambdas have to be transformed back to the lab system. These steps are all straightforward, but involves quite some algebra. In the following, we show how to perform this using the TLorentzVector class. However, since there is the need to define both momentum and energy several times for particles on the mass shell, a function can be defined and put into a file **p.C** (ROOT cannot handle function definitions in the command line):

```
double p (double T, double M)
{
  return sqrt ((T + M) * (T + M) - M * M);
}
```

Load the file to get access to the functions contained and test with:

```
root [99] .L p.C
root [108] p(1,939) // T = 1 MeV << Mp
(double)4.33474336033864844e+01
root [109] p(1e6,939) // T = 1 TeV >> Mp
(double)1.00093855955298280e+06
```

First, start with creating 4-vectors for the incoming beam and the target, and then use the involved masses and energies to set them:

```
root [110] TLorentzVector beam,target
root [111] Double_t mp = 939
root [112] Double_t Tp = 1600
root [113] beam.SetPxPyPzE(0,0,p(Tp,mp),Tp+mp)
root [114] target.SetPxPyPzE(0,0,0,mp)
```

The CM system will then just be the sum of the 4-vectors:

```
root [115] TLorentzVector W=beam+target
root [116] W.Dump()
==> Dumping object at: 0x08cd9bd0, name=TLorentzVector, class=TLorentzVector

fP                          ->8cd9bdc   3 vector component
fP.fX                       0
fP.fY                       0
fP.fZ                       2358.98
fP.fUniqueID                0           object unique identifier
fP.fBits                    0x03000000  bit field status word
fE                          3478        time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                   0           object unique identifier
fBits                       0x03000000  bit field status word
```

The easiest way of changing the inertial system is to use the **boost** method, giving the beta vector as an argument. Since W is our CM system, we can simply use its velocity, boost with the negative value which should transfer to the CM system where the sum of the momenta is zero:

```
root [117] Double_t betaCM=W.Beta()
root [118] W.Boost(0,0,-betaCM)
root [119] W.Dump()
==> Dumping object at: 0x08cd9bd0, name=TLorentzVector, class=TLorentzVector

fP                          ->8cd9bdc   3 vector component
fP.fX                       0
fP.fY                       0
fP.fZ                       -7.32747e-15
fP.fUniqueID                0           object unique identifier
fP.fBits                    0x03000000  bit field status word
fE                          2555.72     time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                   0           object unique identifier
fBits                       0x03000000  bit field status word
```

Now create the outgoing particles, starting with a Λ that obtains half of the excess energy and is emitted along the z-axis.

```
root [120] TLorentzVector lambda,lambda_bar
root [94] Double_t ml=1116
root [95] Double_t Tl = W.E()/2-ml
root [96] Double_t pl = p(Tl,ml) // CM excess energy
root [97] lambda.SetPxPyPzE(0,0,pl,Tl+ml)
root [128] lambda.Dump()
==> Dumping object at: 0x08ed64c0, name=TLorentzVector, class=TLorentzVector

fP                          ->8ed64cc   3 vector component
```

```
fP.fX                       0
fP.fY                       0
fP.fZ                       622.467
fP.fUniqueID                0           object unique identifier
fP.fBits                    0x03000000  bit field status word
fE                          161.858     time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                   0           object unique identifier
fBits                       0x03000000  bit field status word
```

To randomise the direction of the Λ, we use the tricks from section 2 to create spherical random polar angles:

```
root [139]  TF1 sintheta("sin_theta","sin(x)",0,TMath::Pi())
root [141] lambda.SetTheta(sintheta.GetRandom())
root [143] lambda.SetPhi(gRandom->Uniform(0,2*TMath::Pi()))
```

To now create the $\overline{\Lambda}$, use the remaining components of the C.M. vector and compare the results:

```
root [147] lambda_bar=W-lambda
root [151] lambda.Dump()
==> Dumping object at: 0x08ed65e8, name=TLorentzVector, class=TLorentzVector

fP                          ->8ed65f4   3 vector component
fP.fX                       -243.292
fP.fY                       -62.538
fP.fZ                       -569.529
fP.fUniqueID                0           object unique identifier
fP.fBits                    0x03000000  bit field status word
fE                          1277.86     time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                   0           object unique identifier
fBits                       0x03000000  bit field status word
root [152] lambda_bar.Dump()
==> Dumping object at: 0x08ed0bf8, name=TLorentzVector, class=TLorentzVector

fP                          ->8ed0c04   3 vector component
fP.fX                       243.292
fP.fY                       62.538
fP.fZ                       569.529
fP.fUniqueID                0           object unique identifier
fP.fBits                    0x03000000  bit field status word
fE                          1277.86     time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                   0           object unique identifier
fBits                       0x03000000  bit field status word
```

As expected, the particle-antiparticle pair is emitted with equal momentum back-to-back in the CM. Now, transform back to the lab:

```
root [153] lambda.Boost(0,0,betaCM)
root [154] lambda_bar.Boost(0,0,betaCM)
root [155] lambda.Dump()
==> Dumping object at: 0x08ed65e8, name=TLorentzVector, class=TLorentzVector

fP                          ->8ed65f4   3 vector component
```

```
fP.fX                         -243.292
fP.fY                         -62.538
fP.fZ                         404.436
fP.fUniqueID                  0          object unique identifier
fP.fBits                      0x03000000  bit field status word
fE                            1213.31     time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                     0          object unique identifier
fBits                         0x03000000  bit field status word
root [156] lambda_bar.Dump()
==> Dumping object at: 0x08ed0bf8, name=TLorentzVector, class=TLorentzVector

fP                            ->8ed0c04   3 vector component
fP.fX                         243.292
fP.fY                         62.538
fP.fZ                         1954.55
fP.fUniqueID                  0          object unique identifier
fP.fBits                      0x03000000  bit field status word
fE                            2264.69     time or energy of (x,y,z,t) or (px,py,pz,e)
fUniqueID                     0          object unique identifier
fBits                         0x03000000  bit field status word

root [157] lambda.Theta()
(const Double_t)5.5580074010266007e-01
root [158] lambda_bar.Theta()
(const Double_t)1.2782096373521811e-01

root [159] lambda.E()-lambda.M()
(const double)9.73124353956791310e+01
root [160] lambda_bar.E()-lambda_bar.M()
(const double)1.14868756460432155e+03
```

Now, the angles and the kinetic energies (extracted as total energy minus rest mass) do of course no longer correspond. We can put the Monte-Carlo part in a loop and execute it many times (in an unsigned macro called **sim_lambda.C**), then plot the angles to see which are needed to cover in an experiment:

```
{
  TF1 sintheta("sin_theta","sin(x)",0,TMath::Pi());
  TH2D lambda_ang("lambda_ang","Angles for lambda",100,0,TMath::Pi(),100,-TMath::Pi(),TMath::Pi());
  TH2D lambda_ang_bar("lambda_ang_bar","Angles for lambda_bar",100,0,TMath::Pi(),100,-TMath::Pi(),TMath
  TH2D lambda_theta_CM("lambda_theta_CM","CM theta angles for lambdas",100,0,TMath::Pi(),100,0,TMath::P
  TH2D lambda_theta_lab("lambda_theta_lab","Lab theta angles for lambdas",100,0,.7,100,0,.7);
  TLorentzVector beam,target;
  Double_t mp = 939;
  Double_t Tp = 1600;
  beam.SetPxPyPzE(0,0,p(Tp,mp),Tp+mp);
  target.SetPxPyPzE(0,0,0,mp);
  TLorentzVector W=beam+target;
  Double_t betaCM=W.Beta();
  W.Boost(0,0,-betaCM);
  Double_t ml=1116;
  Double_t Tl = W.E()/2-ml;
  Double_t pl = p(Tl,ml); // CM excess energy
```

```
  lambda.SetPxPyPzE(0,0,pl,Tl+ml);

  for (Int_t i=0;i<100000;i++){
    TLorentzVector lambda,lambda_bar;
    lambda.SetPxPyPzE(0,0,pl,Tl+ml);
    lambda.SetTheta(sintheta.GetRandom());
    lambda.SetPhi(gRandom->Uniform(0,2*TMath::Pi()));
    lambda_bar=W-lambda;
    lambda_theta_CM->Fill(lambda.Theta(),lambda_bar.Theta());

    // Go back to lab system
    lambda.Boost(0,0,betaCM);
    lambda_bar.Boost(0,0,betaCM);
    lambda_theta_lab->Fill(lambda.Theta(),lambda_bar.Theta());
    lambda_ang.Fill(lambda.Theta(),lambda.Phi());
    lambda_ang_bar.Fill(lambda_bar.Theta(),lambda_bar.Phi());
  }

  TCanvas c2;
  c2.Divide(2,2);
  c2.cd(1);
  lambda_ang.Draw("colz");
  c2.cd(2);
  lambda_ang_bar.Draw("colz");
  c2.cd(3);
  lambda_theta_CM.Draw("colz");
  c2.cd(4);
  lambda_theta_lab.Draw("colz");
}
```

It is obvious that the lambdas get emitted at forward angles only, which is an effect of *kinematical focusing* at high energies. However, in this case, the life-time of the Λ is only 0.2631 ns so most particles will not reach the detectors (except possibly an inner tracker) but will decay beforehand. The most probable (64%) decay mode is $\Lambda \to p + \pi^-$ (change to antiparticles for $\overline{\Lambda}$ decay) so these products will have to be detected. We can of course continue the successive transformations lab-CM-lab-... with randomised angles etc. to follow a decay chain to be able to simulate the full process, but instead the built-in **TGenPhaseSpace** class can do the job. This is a procedure for *n-body phase space* that generates combinations of decay into n channels while preserving four-momenta and calculating a probability for the generated combination. A standard example is given in **PhaseSpace.C** where the reaction $\gamma + p \to p + \pi^- + \pi^+$ is simulated. This is then presented in a *Dalitz plot*.

```
{
// example of use of TGenPhaseSpace by Valerio Filippini
   gROOT.Reset();
   gSystem.Load("libPhysics");

   TLorentzVector target(0.0, 0.0, 0.0, 0.938);
   TLorentzVector beam(0.0, 0.0, .65, .65);
   TLorentzVector W = beam + target;

   Double_t masses[3] = { 0.938, 0.139, 0.139} ;
```

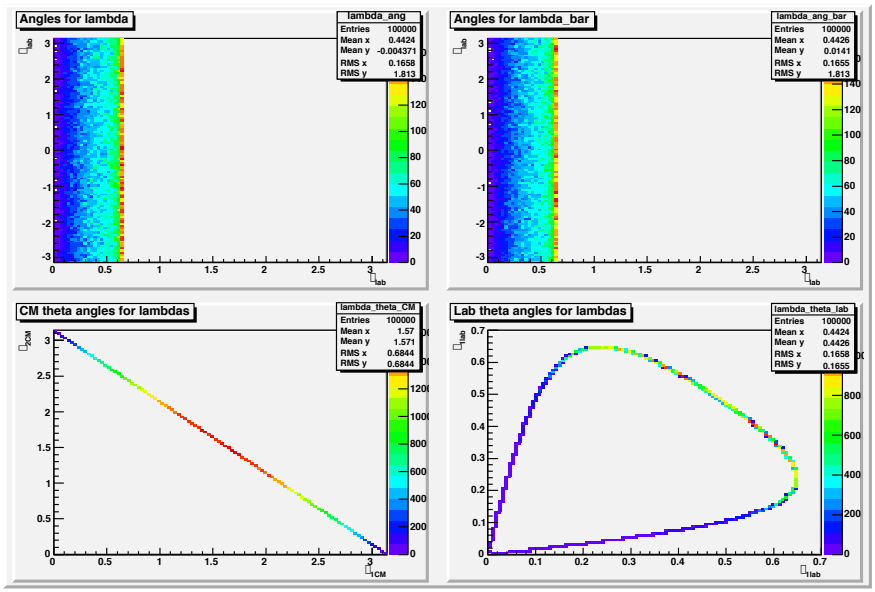
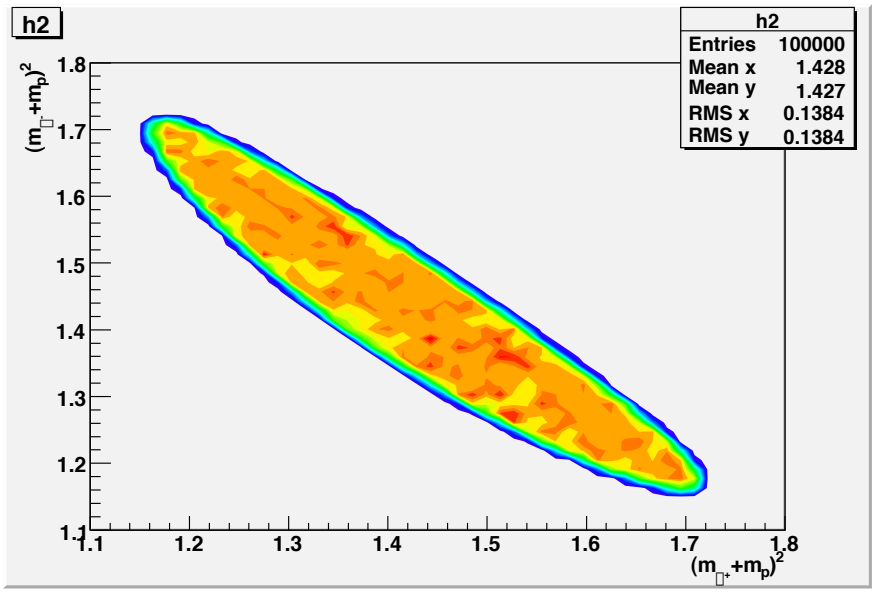


Figure 3: Polar angles for Λ and $\overline{\Lambda}$(upper panels), C.M. θ-angles (lower left) and laboratory θ-angles (lower right).

```
   TGenPhaseSpace event;
   event.SetDecay(W, 3, masses);

   TH2F h2("h2","h2", 50,1.1,1.8, 50,1.1,1.8);

   for (Int_t n=0;n<100000;n++) {
      Double_t weight = event.Generate();

      TLorentzVector *pProton = event.GetDecay(0);

      TLorentzVector *pPip    = event.GetDecay(1);
      TLorentzVector *pPim    = event.GetDecay(2);

      TLorentzVector pPPip = *pProton + *pPip;
      TLorentzVector pPPim = *pProton + *pPim;

      h2.Fill(pPPip.M2() ,pPPim.M2() ,weight);
   }
   h2.SetXTitle("(m_{#\pi^{+}}+m_{p})^{2}");
   h2.SetYTitle("(m_{#\pi^{-}}+m_{p})^{2}");
   h2.Draw("cont");
}
```

Some comments to the code above; the energies and masses are here given in GeV units, the TGenPhaseSpace class delivers a weight(probability) for each event, and the decay products are given as pointers (hence the *) to four-vectors. The Dalitz plot is generated by adding the vectors for two combinations of two products and using the squared mass.

Hand-in task 2: The Hoyle state in ^{12}C with an excitation energy of 7.65 MeV can decay directly into three α-particles as well as through intermediate states in ^{8}Be.



1. Make a Dalitz plot with all such possible decay channels.
2. Make the same plot but adding the effects of limited angular and energy resolution by assuming that a cubic box of DSSSDs (double-sided silicon strip detectors), each with 32 strips in both directions, is used to study the decay products.

6 Curve fitting and calibrations

The resulting experimental output will normally be compared to some theoretical distribution, or we might simply want to find the position of a peak to be used for calibration purposes. The data from S245 for the incoming beam can be used as an example for this, start by creating a histogram with the $\sqrt{E} \propto Z$ for the silicon detector before the target:

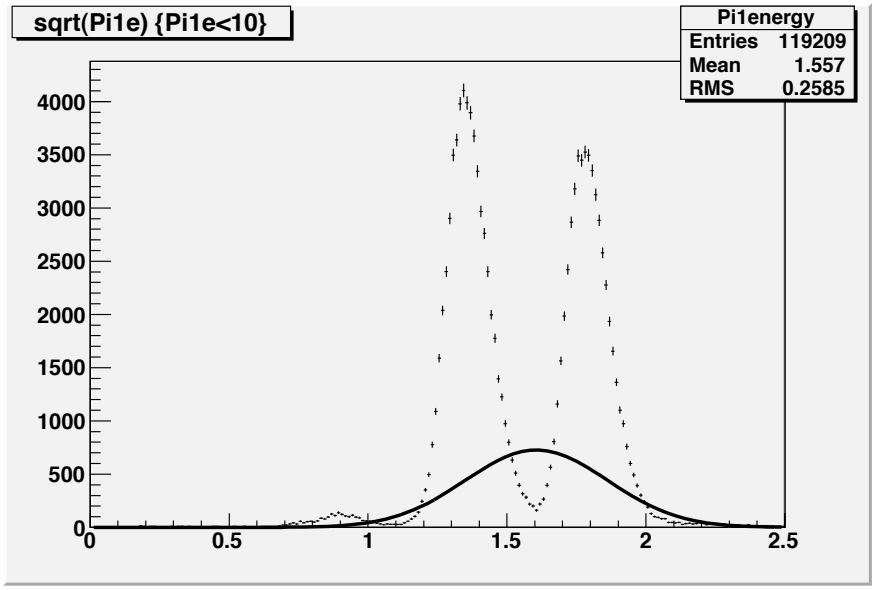
```
root [6] TFile *f = new TFile("r10_470-529_beam_proton_hit_track.root")
root [7] h509->Draw("sqrt(Pi1e)>>Pi1energy(200,0,2.5)","Pi1e<10","e")
```

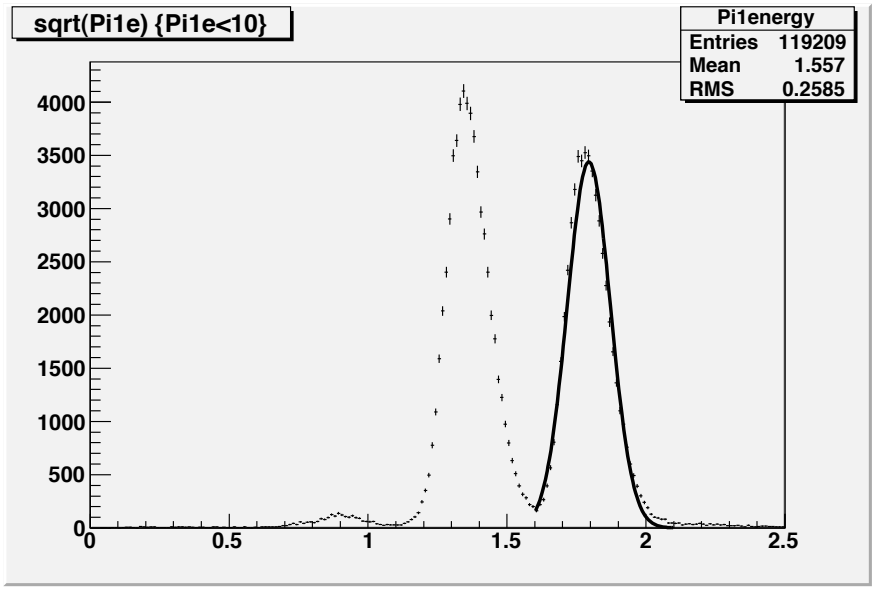
The latter command using the >> symbols projects the data into a histogram that is simultaneously created, furthermore, the "e" option plots the error bars for each bin. The peaks now correspond to $Z = 2, 3, 4$ respectively, but are located at the wrong positions. We can find the position of these peaks and use this for calibrating the scale. A simple Gaussian fit to the histogram will fail since there are several peaks:

```
root [15] Pi1energy->Fit("gaus")
```

Instead, we have to give the fit procedure some upper and lower limits, we can do this by putting in as arguments 4 and 5:

```
root [19] Pi1energy->Fit("gaus","","",1.6,2.1)
 FCN=1108.65 FROM MIGRAD    STATUS=CONVERGED      71 CALLS          72 TOTAL
                     EDM=1.1429e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     3.43774e+03   1.97575e+01   2.42347e-01  -6.27615e-06
   2  Mean         1.79443e+00   3.56753e-04   5.60349e-06  -1.31587e+00
   3  Sigma        7.77020e-02   3.05112e-04   1.55144e-05  -6.62654e-02
```

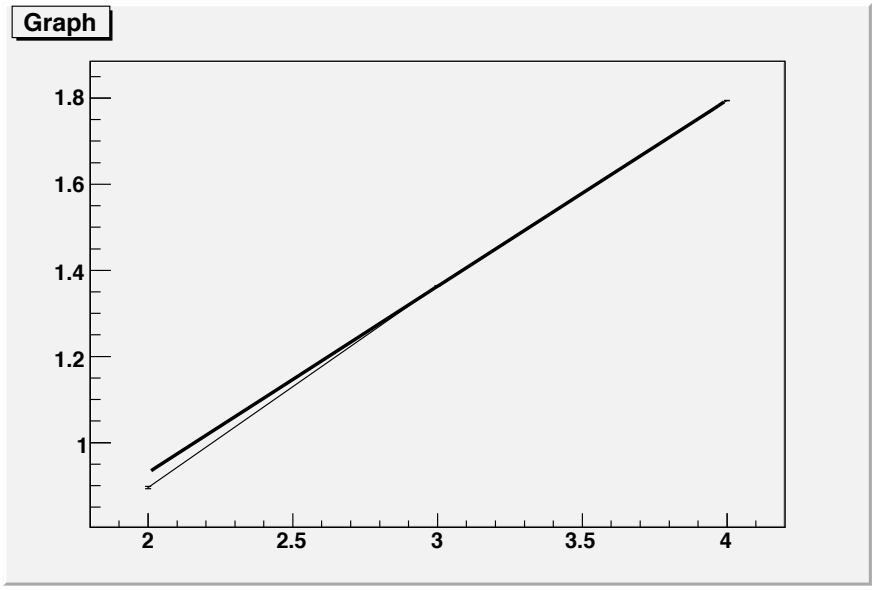

sqrt(Pi1e) {Pi1e<10}
Pi1energy
Entries 119209
Mean 1.557
RMS 0.2585
4000
3500
3000
2500
2000
1500
1000
500
0
0
0.5
1
1.5
2
2.5


sqrt(Pi1e) {Pi1e<10}
Pi1energy
Entries 119209
Mean 1.557
RMS 0.2585
4000
3500
3000
2500
2000
1500
1000
500
0
0
0.5
1
1.5
2
2.5

In principle, we could now select limits suitable for all three peaks and manually do some linear regression, but we can let ROOT do it for us by putting the values in a simple graph with three points with error bars and fill it with the fit result (the mean) and the corresponding error:

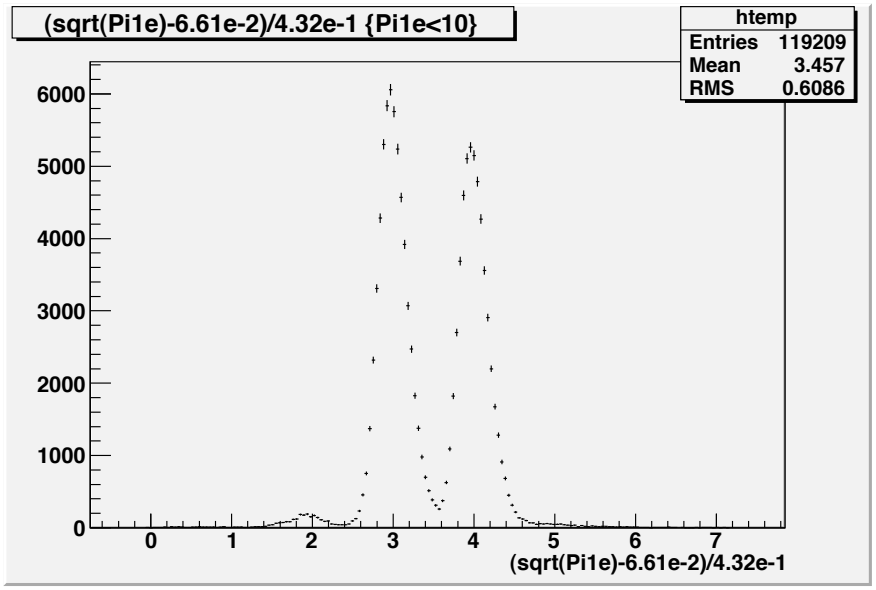
```
root [16] Pi1energy->Fit("gaus","","",1.6,2.1)
 FCN=1108.65 FROM MIGRAD    STATUS=CONVERGED      71 CALLS          72 TOTAL
                     EDM=1.1429e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                          STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     3.43774e+03   1.97575e+01   2.42347e-01  -6.27615e-06
   2  Mean         1.79443e+00   3.56753e-04   5.60349e-06  -1.31587e+00
   3  Sigma        7.77020e-02   3.05112e-04   1.55144e-05  -6.62654e-02
(Int_t)(0)
root [17] Pi1energy->Fit("gaus","","",1.6,2.1)
 FCN=1108.65 FROM MIGRAD    STATUS=CONVERGED      71 CALLS          72 TOTAL
                     EDM=1.1429e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                          STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     3.43774e+03   1.97575e+01   2.42347e-01  -6.27615e-06
   2  Mean         1.79443e+00   3.56753e-04   5.60349e-06  -1.31587e+00
   3  Sigma        7.77020e-02   3.05112e-04   1.55144e-05  -6.62654e-02
(Int_t)(0)
root [18] TGraphErrors cal(3)
root [19] gaus->GetParameter(1)
(const Double_t)1.79443127632170207e+00
root [20] cal.SetPoint(0,4,gaus->GetParameter(1))
root [21] cal.SetPointError(0,0,gaus->GetParError(1))
root [22] Pi1energy->Fit("gaus","","",1.1,1.6)
 FCN=1239.19 FROM MIGRAD    STATUS=CONVERGED      65 CALLS          66 TOTAL
                     EDM=4.30957e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                          STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     3.92304e+03   2.06008e+01   2.75160e-01  -1.63579e-04
   2  Mean         1.36400e+00   3.37079e-04   5.40413e-06  -2.67196e-01
   3  Sigma        7.64973e-02   2.56824e-04   1.40259e-05  -3.11371e+00
(Int_t)(0)
root [23] cal.SetPoint(1,3,gaus->GetParameter(1))
root [24] cal.SetPointError(1,0,gaus->GetParError(1))
root [25] Pi1energy->Fit("gaus","","",0.5,1.1)
 FCN=110.43 FROM MIGRAD    STATUS=CONVERGED      71 CALLS          72 TOTAL
                     EDM=3.39144e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                          STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     1.01637e+02   2.98616e+00   1.12071e-02  -5.53115e-05
   2  Mean         8.96017e-01   2.60183e-03   1.32005e-05  -7.58969e-02
   3  Sigma        1.10897e-01   2.62924e-03   2.91138e-05   3.18090e-03
(Int_t)(0)
root [26] cal.SetPoint(2,2,gaus->GetParameter(1))
root [27] cal.SetPointError(2,0,gaus->GetParError(1))
root [28] cal.Draw("alp")
root [29] cal.Fit("pol1")
Fitting results:
```

```
Parameters:
NO. VALUE ERROR
0 6.610129e-02 1.654316e-03
1 4.322450e-01 4.730732e-04
```



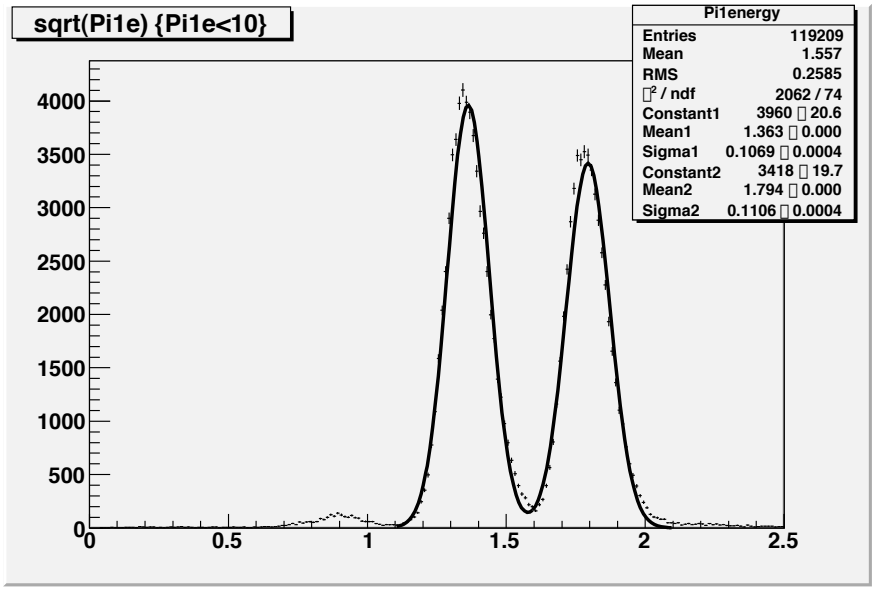
However, this gives $(Channelnumber) = kZ + m$ so we have to turn around the factors to get the peaks in the right position. We can define a TString with the expression to minimise typing:

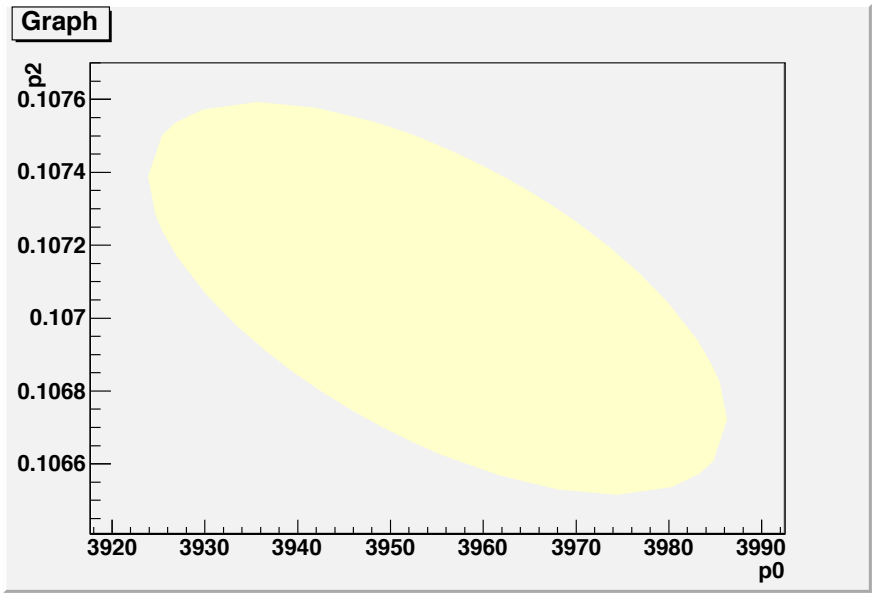
```
root [33] TString Pi1e_cal="(sqrt(Pi1e)-6.61e-2)/4.32e-1"
root [35] h509->Draw(Pi1e_cal,"Pi1e<10","e")
(Long64_t)119209
```



In this fitting example, the curves were separated well enough to use different intervals for the fits. However, a more elegant solution is to use a defined function with parameters, here exemplified by a sum of two gaussian distributions. The numbers in brackets correspond to the parameters in an expression like $Ae^{-(\frac{(x-\mu_1)}{\sigma_1})^2} + Be^{-(\frac{(x-\mu_2)}{\sigma_2})^2}$ However, we will have to give **start values** to the function that are realistic (and optionally relevant names).

```
root [104] TF1 *twogaus = new TF1("twogaus","[0]*TMath::Exp(-pow((x-[1])/[2],2))
end with ';', '@':abort > +[3]*TMath::Exp(-pow((x-[4])/[5],2))",0,5)
root [156] twogaus->SetParName(0,"Constant1")  //Optional
root [157] twogaus->SetParameter(0,4000)
root [158] twogaus->SetParName(1,"Mean1")       //Optional
root [159] twogaus->SetParameter(1,1.4)
root [160] twogaus->SetParName(2,"Sigma1")      //Optional
root [161] twogaus->SetParameter(2,0.2)
root [156] twogaus->SetParName(3,"Constant2")  //Optional
root [157] twogaus->SetParameter(3,4000)
root [158] twogaus->SetParName(4,"Mean2")       //Optional
root [159] twogaus->SetParameter(4,1.8)
root [160] twogaus->SetParName(5,"Sigma2")      //Optional
root [161] twogaus->SetParameter(5,0.2)
root [189] Pi1energy->Fit("twogaus","","",1.1,2.1)
 FCN=2061.64 FROM MIGRAD    STATUS=CONVERGED     225 CALLS         226 TOTAL
                     EDM=6.73494e-09    STRATEGY= 1  ERROR MATRIX UNCERTAINTY   2.1 per cent
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant1    3.96022e+03   2.05633e+01   3.10603e-02  -5.97611e-06
   2  Mean1        1.36309e+00   3.32723e-04  -7.75933e-08   1.61267e-01
   3  Sigma1       1.06888e-01   3.53162e-04  -5.03150e-08  -4.62162e-01
   4  Constant2    3.41768e+03   1.97101e+01  -7.24361e-02  -5.80780e-07
   5  Mean2        1.79436e+00   3.55445e-04  -8.30102e-07   1.69067e-01
   6  Sigma2       1.10643e-01   4.32385e-04   1.06575e-06  -2.25894e-01
```





This can of course be generalised to any function with n parameters, if even more complicated fits are needed, the function could be an external programme. The underlying minimisation engine is called **Minuit**, which has been used since early 1970-ties but in the ROOT is implemented i C++. Some of the special functions can be accessed through a GUI by:

```
root [87] Pi1energy->FitPanel()
```

which gives a window with a lot of possibilities - some of them rather frustrating... Try e.g. directly changing the fit parameters manually in the **Set Parameters...** window, or open the **Advanced...** window to check the correlation between fit parameters in a **Contour** plot.