511 Nuclear Physics Lab

Last week, I discussed ROOT and LabView and demonstrated some features of ROOT. This week, you will work with C++ and ROOT to produce histograms and fit functions, simulate statistical experiments that you previously performed by hand, and read files containing data from the previous statistical experiments.

Objectives

1. Use C/C++ syntax within the ROOT environment.
2. Use the ROOT command line, macros, and functions to execute procedures.
3. Use some common ROOT classes, including TF1, TH1, random number generators, etc.
4. Simulate coin tossing, dice rolling, and decay statistics, and generate corresponding histograms with appropriate functional fits.
5. Read data files containing results from previous lab sessions.
6. Overlay simulated results with previous lab results, fit each, and compare.

C++ Refresher for ROOT Users

If this were a computer science course, we would delve into the dirty details of compilers, linkers, memory management, object-oriented programming concepts, development life-cycles, etc., but we're physicists, so I'll only introduce details as needed. Also, use the C++ cheat sheet that I provided last week as a reference. Your programs will be executed within the root environment, so start ROOT, and we'll get started.

The first exercise will be taken from an online tutorial (http://www-root.fnal.gov/root/CPlusPlus/index.html) at Fermilab. Do only the section called "ROOT as pocket calculator." After finishing, you should understand all of the following:

- C++ statements end with semi-colons; get used to it.
- Omitting the semi-colon tells CINT to output the result (does not work in macros or functions).
- Variables must be declared before usage.
- cout can be used to print to screen.
- Int_t and Double_t are ROOT types for integer and floating point values, respectively.
- Converting from Double_t to Int_t causes truncation.
- Arithmetic and assignment operators.
- **MOST IMPORTANT – ROOT provides many functions, and you should get used to referencing the ROOT website for class and function specifications, examples, tutorials, code snippets, and the manual.**

What about strings of characters?

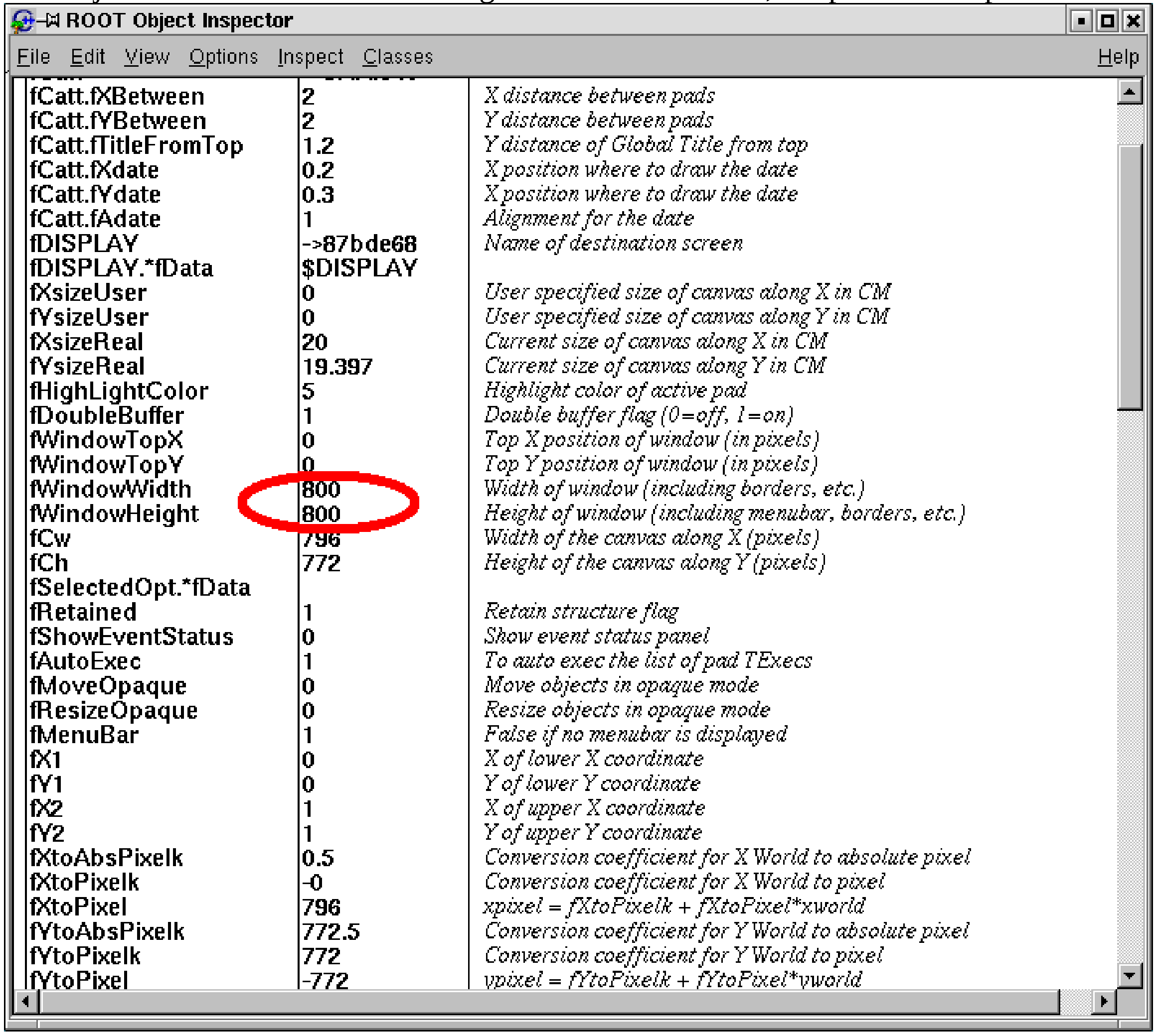
char* myName = "J\366rn Langheinrich is not my name."
Pointer to several char
.p myName
Pointing to a memory address like 0x87054b0. The name is stored at this place.

ROOT Basics

The following exercise will expose you to some ROOT elements that you will need in order to generate your own simulations, histograms, and functions. Most of the exercise was copied or adapted from Jörn Langheinrich's ROOT tutorial. Bold lines indicate ROOT input.

TCanvas* C = new TCanvas("C", "I created this beautiful window", 800, 800);
Internal name "C", visible title "I created...", x-size 800 pixel, y-size 800 pixel
Pointer to new canvas
.p C;
Shows only the pointer

C->Inspect();
Introducing the -> symbol:
The object TCanvas comes with a large number of methods, the pointer C-> points to the methods();



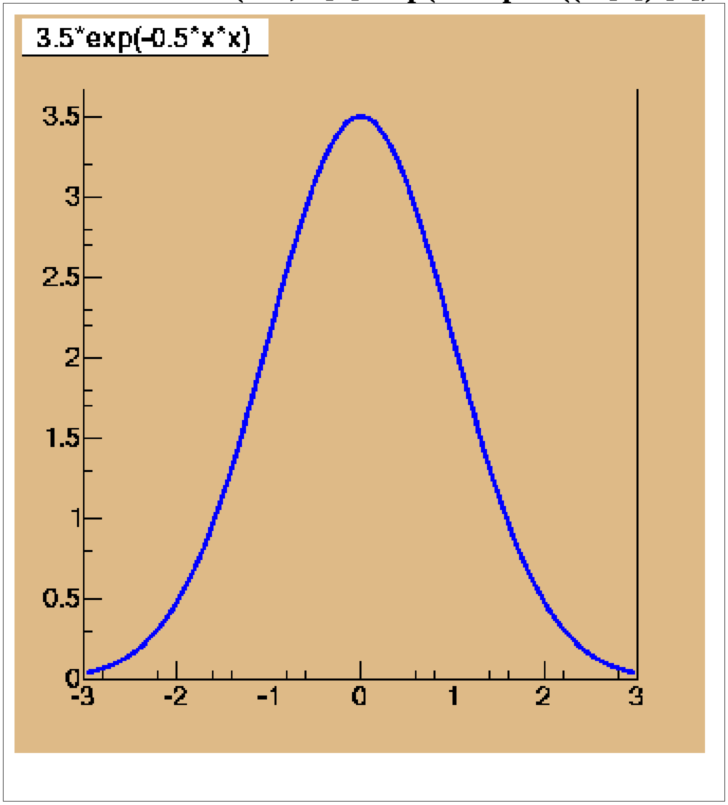
fCatt.fXBetween	2	*X distance between pads*
fCatt.fYBetween	2	*Y distance between pads*
fCatt.fTitleFromTop	1.2	*Y distance of Global Title from top*
fCatt.fXdate	0.2	*X position where to draw the date*
fCatt.fYdate	0.3	*X position where to draw the date*
fCatt.fAdate	1	*Alignment for the date*
fDISPLAY	->87bde68	*Name of destination screen*
fDISPLAY.*fData	$DISPLAY	
fXsizeUser	0	*User specified size of canvas along X in CM*
fYsizeUser	0	*User specified size of canvas along Y in CM*
fXsizeReal	20	*Current size of canvas along X in CM*
fYsizeReal	19.397	*Current size of canvas along Y in CM*
fHighLightColor	5	*Highlight color of active pad*
fDoubleBuffer	1	*Double buffer flag (0=off, 1=on)*
fWindowTopX	0	*Top X position of window (in pixels)*
fWindowTopY	0	*Top Y position of window (in pixels)*
fWindowWidth	800	*Width of window (including borders, etc.)*
fWindowHeight	800	*Height of window (including menubar, borders, etc.)*
fCw	796	*Width of the canvas along X (pixels)*
fCh	772	*Height of the canvas along Y (pixels)*
fSelectedOpt.*fData		
fRetained	1	*Retain structure flag*
fShowEventStatus	0	*Show event status panel*
fAutoExec	1	*To auto exec the list of pad TExecs*
fMoveOpaque	0	*Move objects in opaque mode*
fResizeOpaque	0	*Resize objects in opaque mode*
fMenuBar	1	*False if no menubar is displayed*
fX1	0	*X of lower X coordinate*
fY1	0	*Y of lower Y coordinate*
fX2	1	*X of upper X coordinate*
fY2	1	*Y of upper Y coordinate*
fXtoAbsPixelk	0.5	*Conversion coefficient for X World to absolute pixel*
fXtoPixelk	-0	*Conversion coefficient for X World to pixel*
fXtoPixel	796	*xpixel = fXtoPixelk + fXtoPixel*xworld*
fYtoAbsPixelk	772.5	*Conversion coefficient for Y World to absolute pixel*
fYtoPixelk	772	*Conversion coefficient for Y World to pixel*
fYtoPixel	-772	*ypixel = fYtoPixelk + fYtoPixel*yworld*

test it: make it larger using the pointer drag and drop option.
C->SetFillColor(42);
Disappointed? Pointer drag and drop, what happens?
C->Modified();
C->Update();

TF1* f=new TF1("f", "3.5*exp(-0.5*x*x)", -3., 3.);

Internal name "f", mathematical function, range: left edge, right edge
Creating a new function, which is defined in the range from -3 to 3. How would you improve this example? It looks uglier, so why is it better?

TF1* f=new TF1("f", "[0]*exp(-0.5*pow((x-[1])/[2],2.))", -3., 3.);



f->SetParameters(3.,-2.,0.2);

C++ functions in math

exp(x) e^x

pow(x,y) x^y

sqrt(x) $\sqrt{x}$

log(x) ln x

f->Draw();
f->SetLineColor(4);
f->SetLineWidth(5);

Basic colors
1 = black
2 = red
3 = green
4 = blue
5 = yellow
6 = magenta
7 = cyan

Measured values are usually visualized using histograms.

Example:

Tossing Coins

7 Pennies at the same time and count the number of tails. What are the possible results? We repeat this experiment at least 7 times. If you did this in class already, use that data.

TH1F* h=new TH1F("h", "number of tails", 8, -0.5, 7.5);
Internal name "h", visible title ".. tails", 8 channels, left edge -0.5, right edge 7.5
h->Draw();
h->SetFillColor(48);
Just an example for 7 tosses which yield 3, 5, 3, 2, 4, 6, 3 tails, the numbers can be different in your case...
h->Fill(3);
h->Fill(5);
h->Fill(3);
h->Fill(2);
h->Fill(4);
h->Fill(6);
h->Fill(3);
and so on...

Simulating "tossing coins" by using the random number generator gRandom rather than wasting coins.
gRandom->Rndm() generates a random number between 0 and 1.
.p gRandom->Rndm() < 0.5
Create the "tail" condition: Is the random number (range 0 to 1) less than 0.5?
In C++ : yes = 1, no = 0 , so we can use the condition in an aritmethic expression.

int tails = 0;
double p = 0.5;
tails += (gRandom->Rndm() < p);
tails += (gRandom->Rndm() < p);
remember, that you can repeat a ROOT command you typed before by using the "arrow up" key
...
tails += (gRandom->Rndm() < p);
.p tails

bored by writing lines? So let's create a loop:
```
{
int tails = 0;
double p = 0.5;
for (int j=0; j<7; j++) {
tails += ( gRandom->Rndm() < p );
}
}
.p tails
```

What is the advantage of using a variable p rather than the number 0.5?

{The brackets include one or several program lines, to be repeated – 7 times in our case }

We put everything together to the first real program. The program has to be written using an editor of your choice. If you don't have an integrated C++ system on your Windows box, try to use something simple like notepad (don't use word). On a Linux system, most users use emacs.

1. Open emacs by clicking on the emacs icon (if you don't have it, type **emacs**)
2. Once in emacs click on File -> Open File
3. Type the file name **coins.C** (capital C as extension)
4. Enter the program below
5. Click on File -> Save (Current Buffer)
6. Keep the window open - you might want to modify the program later on.

Hints for typing the program

1. Program starts and ends with curly brackets
2. Use comments starting with //, they make it easier to understand for human
3. Use some indent space for commands in brackets. If you use emacs, it will be taken care of automatically.

To execute the program once it is ready, you have to use the special command .x in the ROOT window. (ROOT command, not existing in C++)

```
        .x coins.C
```

```
{
int n = 7; // number of coins
double p = 0.5; // probability for tail in a single cast
int toss = 7; // number of tosses

TCanvas* C = new TCanvas("C", "I created this new window", 800, 800);
C->SetFillColor(42); // creating the canvas

TH1F* h=new TH1F("h", "number of tails", n+1, -0.5, n+0.5);
h->Draw();
h->SetFillColor(48); // creating the empty histogram

for (int i = 0; i < toss; i++) {
int tails = 0;
for (int j = 0; j < n; j++) { // toss n coins (repetitions)
tails += ( gRandom->Rndm() < p ); // a single coin (1 = tail, 0=head as above)
}
h-> Fill(tails);
C->Modified();
C->Update(); // make the change in the histogram visible on screen
}
h->Fit("gaus"); // try to describe histogram with a Gauß- function
h->GetFunction("gaus")->SetLineColor(4); // blue color
}
```

The **h->Fit** line creating the Gauss function (blue line in the picture below) and trying to get the best possible fit with the histogram need some explanations (later).

What is the magnitude of the difference between the Gauss function and the histogram?

How would you change the program to roll dice (count dice showing a "six") rather than tossing coins?

What happens, if you increase the number n while decreasing p?

Play with the values n, p and toss !

Additional exercises

By this time, I'm sure the groups are staggered in progress, so continue with the following exercises, asking for help as necessary.

- Look up the available random number generators on the ROOT website. In particular, how do you generate random numbers over Gauss and Poisson distributions?
- Copy coins.C to dice.C. In the new file, turn your “script” into a function that takes number of dice, probability, and number of casts as parameters. Use your function to generate histograms similar to those produced by hand in previous labs. Fit the data with appropriate distribution functions.
- Instead of writing the distribution values directly to histograms, write the values to a file. (Reference the C++ cheat sheet provided or any other resource, e.g., http://www.cplusplus.com/doc/tutorial/files/).
- Read the data from the file and build the histograms again.
- Create a new file containing data from previous labs, and use ROOT to histogram and fit the data. On the same canvas, draw a histogram from comparable simulated data.