

ROOT:

a brief starting tutorial

****preliminary version*** use at your own risk*

P. Oliva

July 13, 2009

Contents

1 Getting Started.	3
2 Writing a script.	9

1 Getting Started.

```
olivap@paogroupa:~$ root
*****
*
*           W E L C O M E  t o  R O O T           *
*
*   Version   5.20/00      24 June 2008   *
*
*   You are welcome to visit our Web site *
*           http://root.cern.ch           *
*
*****

ROOT 5.20/00 (trunk@24524, Aug 15 2008, 07:42:00 on linuxx8664gcc)

CINT/ROOT C/C++ Interpreter version 5.16.29, Jan 08, 2008
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0] .q
olivap@paogroupa:~$ root -l
root [0] .q
olivap@paogroupa:~$ █
```

Figure 1: How to start and quit ROOT.

ROOT is an Object Oriented Framework (OOF) dedicated to data analysis especially designed for high energy physics. In an OOF there are three fundamental concepts to understand: the Class, the Object and the Method. A Class is the description of a general “thing” in the system, an Object is simply an instance of a class and the Method is a function for a Class.

Being a framework, ROOT brings to you a lot of advantages like less code to write (existing one is enough for the major part of your needs and it’s well tested and integrated) and services you don’t need to worry about like graphics or networking. Being Object Orientated also brings advantages like the possibility to extend and modify a Class to include new things and to describe different parts of the real problem with the same modular structure.

Without going into details, we can tell that to define a Class we generally need to write down two files: a declaration file, called header file or include file (ClassSomething.h)¹, which contains the definition of the Class and the public and private methods (functions we can respectively invoke from outside or ones we can only call for internal use) and the implementation file with the same name of the header but with .cc extension², where all the operations characterizing all the methods are included (ClassSomething.cc).

For instance, we can have a Class called “Car” which describes a general

¹The very old header files mutated from C to C++ have no extension but a c before the name (i.e. cClassSomething)

²The extension could be also .c or .cpp

car, that is to say where it's stated that a car must have 4 wheels, up to 6 gears, up to 9 seats, a maximum speed, etc.; from the Car Class we can derive the Object, i.e. a Ferrari car, as instance of the Class. The Methods we can apply to the Ferrari Object are provided in the header file of the Class (let's say the file Car.h). Typically we'll have some "Get" methods and some "Set" methods, the first ones are useful to get a particular information (i.e. `GetMaxSpeed()` would be the method to get the maximum speed from the particular Object, and the syntax would be `Ferrari.GetMaxSpeed()`), the latter ones are used to set a value allowing the user to assign a certain value to a private variable contained in the header file. You may be aware that ROOT uses CINT command line, so that a command starts with a dot: `?.?` lists all the CINT commands, `.x [filename]` loads [filename] and execute function [filename] (i.e. `.x plot.C("function")`), `.L [filename]` loads [filename], `.! [shellcmd]` execute shell command (i.e. `.! ls -al`), etc.

A final remark is on the pointers: the memory of our computer can be seen as a succession of memory cells, each one of the minimal size that computers manage, the so called *one byte*. These cells are numbered in a consecutive way so that each cell has a unique address. The address that locates a variable within memory is what we call a reference to that variable. This reference to a variable can be obtained by preceding the identifier of a variable with an ampersand sign (&), known as reference operator, and which can be literally translated as "address of". For example `B=&A` means that variable B contains not the value of variable A but its address reference (a number that identify the memory cell). A variable that stores the reference to another variable is called a pointer, one of the most very powerful feature of the C++ language. In our example B points to the variable A so that in B is stored the address reference number of A. If we want to directly access the value of variable A then we have to use the star symbol: `C=*B`, in this way the new C variable store the value pointed by B, which is of course the content of A variable.

Considered that a pointer directly refers to the value that it points to, it becomes necessary to specify in its declaration which data type a pointer is going to point to, and this is done by declaring `type * name;` (i.e. `double * number;`). Therefore, when we have a method and we want to call the method on a pointer instead of on a normal variable, we must use the syntax `Ferrari->GetMaxSpeed()` instead of `Ferrari.GetMaxSpeed()` (thus, we have `Object.Method()` and `PointerToAnObject->Method()`).

Now that we have these generic notions we can start to use ROOT. First we need to install ROOT on our machine. Download³ the "pro" version and follow the install instructions⁴. We assume you have now installed ROOT

³<http://root.cern.ch/drupal/content/downloading-root>

⁴<http://root.cern.ch/root/Install.html>

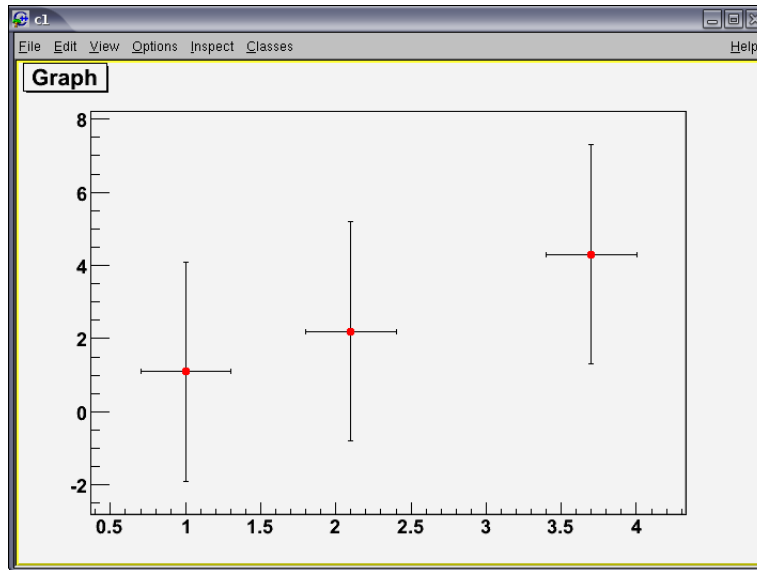


Figure 2: The default canvas c1.

on your machine. For further infos please refer to the official User's Guide⁵.

To start ROOT just type the command `root` in your shell. To quit ROOT type `.q` and... that's it! A way to avoid the annoying ROOT Splash Screen is to launch root with the argument `-l`, (`root -l`) as showed in Fig.1. You can easily use ROOT as a calculator but don't forget the semicolon at the end of each line. Remember to use up and down arrows to recall commands

```
user:~$ root -l
root [0] double a=(3*5+5)/2;
root [1] cout<<a<<endl;
10
root [2] double b=a+2;
root [3] cout<<b<<endl;
12
root [4] a=a*2;
root [5] cout<<a<<endl;
20
root [6] a=a+5;
root [7] double c=sqrt(a);
root [8] cout<<c<<endl;
5
```

if you want to reset ROOT simply type

```
root [9] gROOT->Reset();
root [10] cout<<c<<endl;
```

⁵<http://root.cern.ch/drupal/content/users-guide>



Figure 3: Drawing styles options.

```
Error: Symbol c is not defined in current scope (tmpfile):1:
*** Interpreter error recovered ***
```

and as you can see if you try to call back a previous defined variable you get an error (because the variable `c` is now not defined anymore). You can continue having fun using of course all the standard C++ commands: let's define two 3-D vectors and relative error vectors (from now on the progressive ROOT number line will be omitted for sake of simplicity). The first one is the vector `float x[3]` where `float` means 4 bytes variable type (while i.e. `double` is 8 bytes)

```
root [] float x[3] = {1.0, 2.1, 3.7};
root [] cout << x[0] <<"", "<< x[1] <<"", "<< x[2] <<endl;
1, 2.1, 3.7
```

is very important to note that the three components are `x[0]`, `x[1]`, `x[2]`, and not `x[1]`, `x[2]`, `x[3]`! If now we define the other vectors we have three points with their errors and we can plot them

```

root [] float y[3] = {1.1,2.2,4.3};
root [] float dx[3] = {0.3, 0.3, 0.3};
root [] float dy[3] = {3.0,3.0,3.0};
root [] TGraphErrors myfirstplot(3,x,y,dx,dy);
root [] myfirstplot.SetMarkerColor(2);
root [] myfirstplot.SetMarkerStyle(20);
root [] myfirstplot.SetMarkerSize(1.0);
root [] myfirstplot.Draw("ap");
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1

```

then a window will pop up: it's your first canvas⁶ that is automatically generated with the default name `c1` as showed in Fig.2. To save a your plot as a figure you need to write `c1->SaveAs("myimage.eps"); SaveAs("myimage.ps");, SaveAs("myimage.gif")` and so on.

You may have noticed that we gave some commands for the style output. You can use your own style using different values as shown in Fig.3. By right click on the `c1` canvas you have access to a menu and you can have fun playing around with all the options, changing x and y axis in logarithmic scale and so on. One can always label the axis directly from the prompt line typing `myfirstplot.GetAxis().SetTitle("X Axis");` since the Class `TGraphErrors` (and of course also the Class `TGraph`) has a Set method `SetTitle()` to be used once you got the axis with the method `GetXaxis()`. Needless to say, same with the y axis. Whenever you are in doubt on some method of any Class you can check out the Reference Guide⁷ of your ROOT version. For instance, if you want to fit your `myfirstplot` graph you just created, you can check if the Class `TGraphErrors` has some fit method. If you perform a search, let's say for ROOT version 5.22⁸, you'll find that a "*TGraphErrors is a TGraph with error bars*" so you inherits all the methods from `TGraph` and you can check them; between those methods you can easily read there is a method called `Fit()`⁹ which allow you to fit by default with an exponential (`Fit("expo")`), gaussian (`Fit("gaus")`) or polynomial of degree N (`Fit("polN")`), moreover you can define your own fitting function.

To draw any function you can use the Class `TF1` and `TF2` to plot namely one dimensional and two dimensional functions: let's plot the functions $func1 = \frac{\sin(x)}{x}$ and $func2 = \frac{\sin(x) \cdot \sin(y)}{(x \cdot y)}$. For the first one we can write

```

root [] TF1 f1("func1", "sin(x)/x", 0, 10);
root [] f1.Draw();

```

or we can use a pointer

⁶The option "ap" means "draw graph in current pad". For an extensive list of drawing options see <http://root.cern.ch/root/html/THistPainter.html>

⁷<http://root.cern.ch/drupal/content/reference-guide>

⁸<http://root.cern.ch/root/html522/ClassIndex.html>

⁹<http://root.cern.ch/root/html522/TGraph.html>

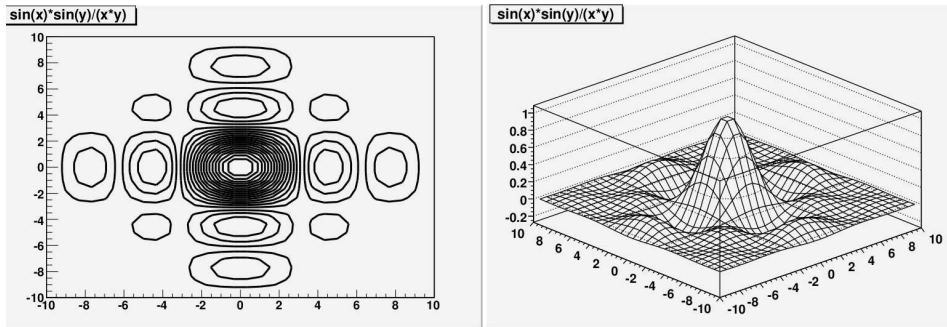


Figure 4: Left: Draw(); Right: Draw("surf");

```
root [] TF1 *f1 = new TF1("func1","sin(x)/x",0,10);
root [] f1->Draw();
```

the only difference is that the direct definition is valid only inside a certain scope (i.e. inside a for loop), while once outside the scope the variable `f1` is not defined anymore; but now let's not worry too much about it. Regarding the 2D function `func2` you have two ways to graph it: you can see the projection in the xy plane:

```
root [] TF2 *f2 = new TF2("func2","sin(x)*sin(y)/(x*y)",-10,10,-10,10);
root [] f2->Draw();
```

or you can ask for the surface plot with the `f2->Draw("surf")` option and many other options (`Draw("text")`, `Draw("col")`, `Draw("colz")`, `Draw("box")`, etc.¹⁰ see Fig.4).

To build up a one-dimensional histogram, you have the Class `TH1`¹¹. You have to give it a label, title, number of bins, and a range specifying `xmin` and `xMax`:

```
TH1F h1("h1","h1 title",100,0,10.);
```

or if you need you can use the pointer:

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

for instance we can draw a gaussian histogram

```
root [] TH1F h1("h1","h1 title",100,-4.,4.);
root [] h1.FillRandom("gaus",100000);
root [] h1.Draw();
```

Histograms differs pretty much from graphs: the graphs are a collection of points on the xy plane while the histogram are only bins to be filled with our events (on the y axis there are always simply the counts of how much often a certain event felt inside certain bins). Thus the histogram must be

¹⁰<http://root.cern.ch/root/html/THistPainter.html>

¹¹<http://root.cern.ch/root/html522/TH1.html>

filled after being declared. We already saw that we can fill an histogram with the Method `Fill()`; Let's define for example a vector `x[20]` with 20 components. We can easily insert its components in a histogram by using a for loop

```
root [] float x[20]={3.2, 1.2, 3.7, 2.3, 4.1, 1.9, 3.5, 1, 6.2,
                    4.4, 3.2, 4.4, 2.1, 1.1, 1.9, 6.8, 9.1, 6, 7.2, 1.9};
root [] TH1F h1("h1","h1 title",20,0.,10.);
root [] int i;
root [] for (i=0; i<=19; i++) {h1->Fill(x[i]);}
root [] h1->Draw();
```

note that we could have omitted the line `int i`; obtaining only a **Warning: Automatic variable i is allocated**. Note also that the for loop goes from 0 to 19 (20 entries): once again if we would have make it vary between 0 and 20 (21 entries) by mistake, we would have simply introduced an additional 21th entry with value 0.

2 Writing a script.

Mainly your task during this tutorial will be to learn how to write a script to solve problems. Before even to get in the specific of this we want to underline that ROOT can interpret your input even if they are not in perfect C++. the typical example is the semicolon at the end of the command. If you forget to write it ROOT will still understand that the line is over where you press enter. To avoid taking dangerous habits, we strongly encourage you to keep on writing in perfect C++ syntax even for the semicolon: this will avoid to forget them in the script when you will program on a simple text sheet with no syntax check.

To make ROOT loading a script you simply use the `.L` command. If you want (and we encourage this) to check wheather the syntax is C++ correct, you want to add the suffix `++` to your filename's extension. Let's write in a text sheet the most simple following script: as first line, we need to include all the needed libraries (a library is included giving the `include` command with a hash sign (`#`) which is a directive for the preprocessor. In this easiest case the directive `#include <iostream>` tells the preprocessor to include the `iostream` standard file. This specific file (`iostream`) includes the declarations of the basic standard input-output library in C++.

```
#include <iostream>
using namespace std;
int hello()
{
    cout<<"Hello everybody!!"<<endl;
    return 0;
}
```

and save it as `hello.cc`. Please note the `using namespace std`: all the elements of the standard C++ library are declared within a *namespace*,

for you may want or you eventually finish up to have the same names in different namespaces. The `std` uses the standard library, and in fact it will be included in most of the source codes you'll use. A final note is about the `return 0` statement that causes the main function to finish. Return may be followed by a return code (in our case code 0) which is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.

Now let's load the script with the `.L hello.cc++` command and run it by calling the main function `hello` from the shell

```
root [] .L hello.cc+
***eventual Info messages***
root [] hello()
Hello everybody!!
(int)0
```

That's it. Let's now try to write a more complex one: we have a set of data and we want to perform a linear fit and also to have a χ^2 probability on the statistic box.

```
#define N 6
#include <iostream>
#include <TCanvas.h>
#include <TH1F.h>
#include <TStyle.h>

void DataFit() {

    //definition with initialization of the vectors
    double x[N] = {0.2, 2., 3.3, 4.1, 6.2, 7.};
    double y[N] = {1., 1.5, 1.9, 2.5, 2.7, 3.7};
    double dy[N] = {0.4, 0.2, 0.5, 0.3, 0.9, 0.5};

    //creating the Canvas
    TCanvas *data = new TCanvas("Data", "Data", 800, 600);
    data->cd(1);
    gStyle->SetOptStat(1);
    gStyle->SetOptFit(11111);

    //Preparing the histogram
    TH1F* h1 = new TH1F("Data Fit", "Data Set Fit", 15, 0., 15.);
    //Filling the histo
    for (int i = 0; i<N; i++){
        h1->SetBinContent(h1->FindBin(x[i]), y[i]);
        h1->SetBinError(h1->FindBin(x[i]), dy[i]);
    };
    //Draw
    h1->Draw();
    h1->Fit("pol1","E+","");
}
```

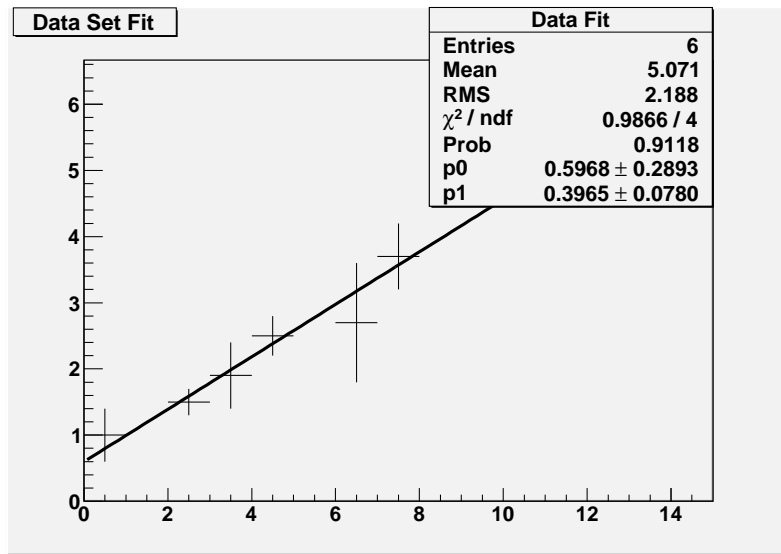


Figure 5: A linear Fit.

We now save this file as DataFit.cc (is always a very good idea to save with the same name as stated in the void DataFit() declaration) and we load it with ROOT.

```
root [] .L DataFit.cc++
***Info messages***
root [] DataFit()
FCN=0.98662 FROM MINOS      STATUS=SUCCESSFUL      22 CALLS      53 TOTAL
                        EDM=3.94011e-23      STRATEGY= 1      ERROR MATRIX ACCURATE
EXT PARAMETER
NO.  NAME      VALUE      ERROR      STEP      FIRST
1    p0      5.96826e-01  2.89256e-01  5.18264e-09  -5.67033e-12
2    p1      3.96466e-01  7.80323e-02  7.80323e-02  1.75160e-11
```

In this way we directly inserted the data values as vectors at the beginning, but sometimes we want to use an external list like this:

```
0.2  1.  0.4
2.   1.5 0.2
3.3  1.9 0.5
4.1  2.5 0.3
6.2  2.7 0.9
7.   3.7 0.5
```

let's assume it saved in a list.txt file. How can we do? Of course there are many ways to do it but our favourite one is the following:

```
#include <iostream>
#include <fstream>
#include <TCanvas.h>
```

```

#include <TH1F.h>
#include <TStyle.h>
using namespace std;

void DataFit() {

    //creating the Canvas
    TCanvas *data = new TCanvas("Data", "Data", 800, 600);
    data->cd(1);
    gStyle->SetOptStat(1);
    gStyle->SetOptFit(11111);

    //Preparing the histogram
    TH1F* h1 = new TH1F("Data Fit", "Data Set Fit", 15, 0., 15.);
    //Read from file and Filling the histo
    ifstream datain;
    datain.open("list.txt");
    double x, y, dy;
    while(datain>> x >> y >> dy)
    {
        cout << "x= " << x << " " << "y= " << y << " " << "sigmay= " << dy << endl;
        h1->SetBinContent(h1->FindBin(x), y);
        h1->SetBinError(h1->FindBin(x), dy);
    }
    datain.close();

    //Draw
    h1->Draw();
    h1->Fit("pol1", "E+", "");
}

```

where we used the streaming to fill the histogram. We also added a cout (we took the data from an external list file -which must be in the same folder where the DataFit.cc is- and it's nice to check if the list is the right one) so the screen now displays

```

root [] DataFit()
x= 0.2 y= 1 sigmay= 0.4
x= 2 y= 1.5 sigmay= 0.2
x= 3.3 y= 1.9 sigmay= 0.5
x= 4.1 y= 2.5 sigmay= 0.3
x= 6.2 y= 2.7 sigmay= 0.9
x= 7 y= 3.7 sigmay= 0.5
FCN=0.98662 FROM MINOS      STATUS=SUCCESSFUL      24 CALLS      66 TOTAL
                        EDM=2.32486e-21      STRATEGY= 1      ERROR MATRIX ACCURATE
EXT PARAMETER
NO.   NAME      VALUE      ERROR      STEP      FIRST
1     p0      5.96826e-01  2.89256e-01  -2.80463e-08  1.22857e-11
2     p1      3.96466e-01  7.80324e-02  7.80324e-02  3.50320e-12

```

Now we note that even if we didn't filled in any error on the x values in Fig.5 we see an error bar for the abscissa. This is due to the binning. Please note that the binning in the TH1 is 15; if we set a higher number of

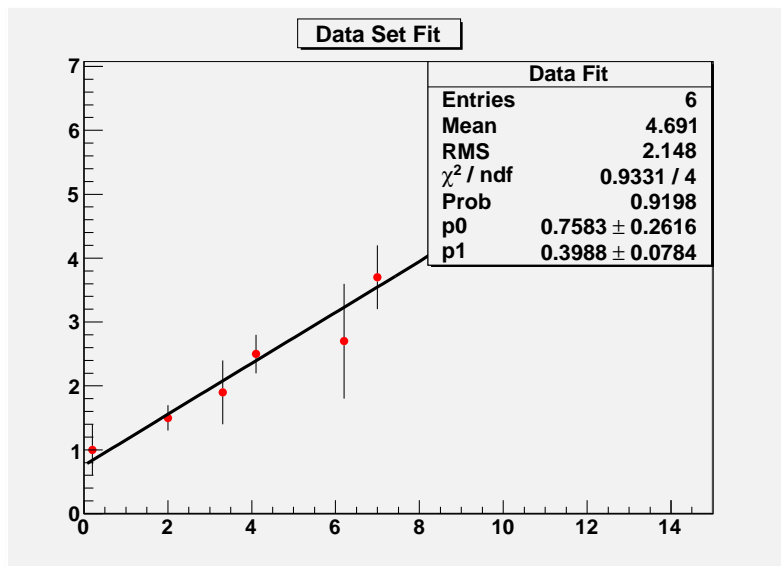


Figure 6: The binning causes the parameters to change. Here we added some style settings such as `h1->SetMarkerStyle(20);` and `h1->SetMarkerColor(2);`

bins these “error bars” will become smaller and smaller like in Fig.6 where we used a binning of 1000 making the abscissa bars practically disappear. These affects all the fitting parameters and, of course, the χ^2 .