ALMA MATER STUDIORUM - Università di Bologna

69703 - ANALISI STATISTICA DEI DATI NELLA FISICA NUCLEARE E SUBNUCLEARE

Modulo 3 : Laboratorio di analisi statistica per la fisica delle alte energie
docente: G. Sirri
http://www.unibo.it/docenti/gabriele.sirri2

ESERCIZIO 0 : Introduzione a ROOT

- Abbiamo tempo. usatelo per «giocare» con ROOT.
  Per favore non fate un banale copia-incolla per finire in 10 minuti.
- Fate domande (a voi stessi, al vostro vicino o a me)
- Questo dovrebbe essere solo un semplice «riscaldamento», non discuteremo di tutti i dettagli
  QUINDI fate domande, anche le più semplici !

1 Introduction to ROOT

Open a root session in a shell by typing

```
> root
```

(where the > symbol stands for your terminal prompt). This should start root. If not, check your environment again... You now have the root C++ interpreter running, so you can enter almost any valid C++ code. Or just use root as a calculator:

```
root[0] 1 + log(2)
```

The object in root we will use most is the histogram, so let's create one:

```
root[1] TH1F myhisto("myhisto","My most fabulous histogram",10,0,100)
```

This creates a histogram (of type `TH1F`), to which root will refer by its name ("myhisto") and which will be displayed with the title given. It has ten bins of equal size, spanning the range [0, 100]. Now we can fill some values into the histogram (root will return the corresponding bin number, if you do not like this, terminate the statements with a semicolon):

```
root[2] myhisto.Fill(42)
root[3] myhisto.Fill(3.141592)
root[4] myhisto.Fill(66)
root[5] myhisto.Fill(99)
root[6] myhisto.Fill(69)
root[7] myhisto.Fill(17.7)
```

Now let us draw the histogram:

```
root[8] myhisto.Draw()
```

This opens a canvas, the surface root draws on. By default, the canvas will be named `c1`. In the canvas window, you can open the Editor from the View menu. It allows you to change the style of various objects on the canvas (which you can also move around with the mouse); be aware though that there is no undo function. All the things that can be set from Editor, can also be set from the command line, *e.g.*,

```
root[9] myhisto.SetLineColor(2)
```

for the change to become visible, you have to draw the histogram again:

```
root[9] myhisto.Draw()
```

If you want the histogram to be drawn with error bars, try

```
root[9] myhisto.Draw("E1")
```

If you do not want to rewrite all the code every time you start root, you can use macro files, *i.e.*, files with program code (in this case CINT-macros). They end in .C and can be executed directly from root. Make it a habit to start them with a comment stating the exercise they are for and your name and e-mail. Create a file exercise0.C in your exercise directory and fill it similar to the following:

```
// exercise0.C: Example root program for the root tutorial
//
// Nomen Nominandum, 15.10.2014
// nomen.nominandum@stud.uni-heidelberg.de
void fillHistogram(unsigned int nentries)
  {
    TH1F * histo = new TH1F("histo","Another histogram",10,0,100);
    for(unsigned int i=0; i < nentries; i++)
      {
        histo->Fill(fmod(i*777,100));
      }
  }
```

Note that we create the histogram on the heap using `new`, to make sure it does not go out of scope when the function returns. In root, we can now load the macro file using

```
root[10] .L exercise0.C
```

(all root comments start with the "."). Now we can call the function like any other:

```
root[11] fillHistogram(100)
```

Let's now draw our new histogram:

```
root[12] histo->Draw()
```

(now `histo` is actually not the variable name, but the name we gave it in the constructor - the variable name is not defined any more, as it is out of scope). We can also draw the old histogram on the same canvas

```
root[13] myhisto.Draw("same")
```

be aware that CINT, the root C++ interpreter, does not really differentiate between the . (for objects) and -> (for pointers to objects) member access, which is one of its drawbacks. Compiled C++ of course does, so better do it right from the start. Now let us save the histograms to a file

```
root[14] TFile* file = new TFile("exercise0.root","RECREATE")
```

and now we can write out the histograms and close the file

```
root[15] myhisto.Write()
root[16] histo->Write()
root[17] file->Close()
root[18] delete file
```

Now try to read the histograms back in:

```
root[19] TFile* fileagain = new TFile("exercise0.root","READ")
root[20] TH1F * histoagain = (TH1F*)fileagain->Get("histo");
```

You retrieve objects from root files using their name (usually the first parameter in the constructor) with the `Get()` function, which will always return a pointer to a `TObject`, the base class of everything in root. You have to manually cast to the type you are expecting (here a `TH1F` pointer). This is not particularly safe and cumbersome... Draw your re-loaded histogram

```
root[21] histoagain->Draw()
```

For quick checks, you can also use the root file browser:

```
root[22] TBrowser b
```

where `b` is just the variable name of the new object of the class `TBrowser`. A browser window will appear. Browse to your file exercise0.root and have a look at its content. Double-click on the histograms to see what happens.
To quit root do

```
root[22] .q
```

Extra exercises – if you have the time

Note that you can execute root macros directly from the command line by including them as an argument, *i.e.*:

```
> root -l exercise0_macro.C
```

For the macro file `exercise0.C`, this does not work directly for two reasons. First, the name of the function is different from the name of the file. Second, because we need the argument `nentries`. After changing the file name from `exercise0.C` to `fillHistogram.C`, we can do:

```
> root -l fillHistogram.C\(100\)
root[1] histo->Draw()
```

where the \ is to escape the special meaning of the parenthesis in the shell.
If you still have time have a look at the root documentation: `http://root.cern.ch/drupal/content/documentation`, do some basic Tutorials or HowTos (*e.g.*, on histograms).