Exercises for the lecture „Moderne Methoden der Datenanalyse”

Prof. Dr. S. Hansmann-Menzemer, M. Schiller
Physikalisches Institut der Universität Heidelberg

Apr 27 2010

Very short introduction into ROOT

Open ROOT session by typing

```
> root
```

This runs root C/C++ interpreter. ROOT commands are C/C++ statements, and ROOT can be e.g. used as a calculator:

```
root[] 1+sqrt(2)
```

To quit ROOT type (note the dot):

```
root[] .q
```

ROOT is an object-oriented package: every object has virtual method `Draw()`. Let's create a function object:

```
root[] TF1 f1("func1","sin(x)/x",0.0,10.0)
```

Here:

`"func1"`	unique object name
`"sin(x)/x"`	math formula, uses standard C++ syntax
`0.0,10.0`	limits

Let's draw it:

```
root[] f1.Draw()
```

This command opens a canvas with the default name `c1` and default title `c1`. Right mouse-click on the line. It opens a list of methods which class `TF1`

implements. Select `GetLineAttributes`. Change line color and width. This same can be done from the ROOT prompt:

```
root[] f1.SelLineColor(kRed)
root[] f1.SetLineWidth(3)
```

In a similar way you can access the properties of the canvas. E.g. change the canvas title:

```
root[] c1.SetTitle("new␣title")
```

Save the canvas into a file: go into upper menu, select `File > Save as`. To print to a printer, do `File > Print`.

Especially important are histograms (distributions). Let's define one:

```
root[] TH1D h1("h1", "my␣first␣histogram", 10, 0.0, 10.0);
```

Note `D` in the class name, it stands for `double`.

Here:

`"h1"`	unique object name
`"my␣first␣histogram"`	title to appear when the histogram is drawn
`10`	number of bins, of type `int`
`0.0, 10.0`	histogram limits, of type `double`

Let's fill in some values:

```
root[] h1.Fill(0)
root[] h1.Fill(3)
...
```

Now let's see how it looks:

```
root[] h1.Draw()
```

Now right-mouse-click on a histogram, select `SetLineAttributes`. The familiar dialog will open. Change line width and color. Change fill color and fill style. The fill color can be changed from the ROOT prompt as:

```
root[] h1.SetFillColor(kBlue)
```

Usually people work with macros, and want histograms to persist in memory, so let's define another histogram using `new` operator:

```
root[] TH1D* h2 = new TH1D("h2", "my␣second␣histogram", 10, 0., 10.);
```

Fill in some values

```
root[] h2->Fill(0)
...
```

Let's create a new Canvas using `new`:

```
root[] TCanvas* myC = new TCanvas("myC", "my␣title")
```

Now this canvas is the "current" one.

Draw both histograms in the same canvas:

```
root[] h1->Draw();
root[] h2->Draw("same");
```

Let's fit Gauss distribution to histogram `h2`. Right-click on a histogram, open the dialog with histogram methods, select `Fit`. Fit dialog will open, with the fields:

`formula`	name of fitting function, e.g. `gaus`
`option`	fitting option
`goption`	g(raphical) option
`xmin`, `xmax`	fitting range, default - all histogram range

For more on fitting functions see `http://www.physi.uni-heidelberg.de/~schiller/ex01/usefulROOT` and `http://root.cern.ch/download/doc/5FittingHistograms.pdf`.

By default the χ^2 fit is used. Let's try log likelihood (takes empty bins into account): set `option` = L and repeat the fit. Observe the difference!

If you want to have both fits plotted: `option` = L+ (adds the current fit to the list of fits).

See ROOT prompt for the details of fit results. This is an output from Minuit fitting package. Find parameter names, values, errors.

Now we can save the result of our work into a file. Open file `histo.root` for writing (if it exist, it will be overwritten):

```
root[] TFile* file = new TFile("histo.root", "RECREATE");
```

Write your histos into it:

```
root[] h1->Write()
root[] h2->Write()
```

Delete the pointer to the file object (closes the file):

```
root[] delete file
```

Now let's check the contents of the file. Open the file `histo.root` for reading:

```
root[] TFile* file = new TFile("histo.root", "READ");
```

Print info on the stored objects:

```
root[] file->Print()
```

Get an object by its name:

```
root[] TH1D* newh2 = (TH1D*)file->Get("h2")
```

Method `Get` returns a pointer to the object of type `TObject`. C++ requires us to explicitely cast it into the pointer pointing to `TH1D`.

Now the histogram with the name `h2` is in the memory, and we have `newh2` pointing to it. You can `Draw` it in the current canvas:

```
root[] newh2->Draw()
```

You can edit it:

```
root[] newh2->SetName("new name")
```

Now try to write it back into the file:

```
root[] newh2->Write()
root[] Error in <TFile::WriteTObject>:  Directory histo.root
root[] is not writable (Int_t)0
```

Ooops, of course, `histo.root` is open for `READ`-ing.

To simplify interactive work, open ROOT Browser

```
root[] TBrowser t
```

Navigate to `ROOT Files/histo.root`. Double click a histo to open it in the current canvas.

You can also browse the directory structure in `TBrowser`, open root files to view their contents, run root macros.

Now, if you have to type a lot of commands (maybe the whole analysis code!) you write them to a macro and call it e.g. `ex01.C`.

Then you load it like

```
root[] .L ex01.C
```

and call any function defined in it like

```
root[] function(<list of arguments>)
```

Now let's open

```
> emacs ex01.C &
```

and proceed with the exercises.