ROOT

Pocket reference for 1st year course - BSc Physics, Unibo

2023

Contents

1 General structure

ROOT contains **interpreter** : *Just-In-Time* compilation → prompt : special commands (not standard C++ syntax) with `.`.

Base class `TObject` → `TNamed` → `TH1` (histograms) → `TH1F, TH1D, THIC, TH1S` according to **type representing entries** (not the type of data!!)

2 Basic shell & prompt commands

! Possible to use Tab

- `root` launch ROOT
- `.q` quit
- `.L <file.C>` load file (symbols defined in a macro)

- `.help` `.?` full help list
- `.! <cmd>` call any shell command ¡cmd¿ without leaving ROOT
- `.files` shows loaded libraries / sources
- `.x <macro>` loads & runs a macro
- `.U <file.C>` unload

Run a macro:

```
$ [0] .L <name>.C
$ [1] <name>()
```

Possible to type C++ commands directly in shell: **';' are unnecessary, object type can be omitted in declarations, possible to access members with obj name instead than pointer**:

```
$ [...] TH1F *histo=new TH1F(\histname"," Titolo", 100, 0, 10)
$ histname->Draw() // identical to histo->Draw()
```

Note: `#include <iostream>` and `namespace std;` are implicit!

Use prompt as calculator Ordinary operations + embedded library `TMath`:
`TMath::Abs(...), TMath::Exp(...), TMath::Gaus(...), TMath::Pi(), ...`

2.1 Recover session history

Saved in `$/home/.root_hist`

3 Histograms

1D

```
TH1F* <pt-name> = new TH1F( "<name>", "<title>", <NxBins>, <xmin>, <xmax>);
// declare new histogram
// range [xmin, xmax] is equally subdivided in N bins

<pt-name>->Fill(<x>);
// fill histo with variable <x> (e.g. from MC generation or read file, data)

<pt-name>->Fill(<x>, <n>);
// fill histo with n identical occurrences of x

<pt-name>->Draw(); // draw histo
```

2D

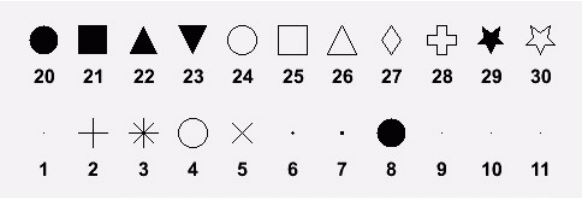
```
TH2F* <pt-name> = new TH2F( "<name>","<title>",<NxB>,<xmin>,<xmax>,<NyB>,<ymin>,<ymax>);

<pt-name>->Fill(x,y);
<pt-name>->Draw();

<pt-name>->ProjectionX(); // returns TH1F of projection w.r.t. x
<pt-name>->ProjectionY();
```

3D

```
TH3F* <pt-name> = new TH2F( "<name>","<title>",<Nx>,<xmn>,<xmx>,<Ny>,<ymn>,<ymx>,<Nz>,<zmn>,<zmx>);
```

N-D

```
THnSparse* pt = new THSparse( "<name>", "<title>", <Ndims>, <xmin>, <xmax>, <chuncksize>);
// min and max same for all dimensions
```



3.1 Overlap graphs

```
// declare and initialize two histos, with pointers h1, h2
h1->Draw();
h2->Draw("same"); // or h2->Draw("SameHist");
```

3.2 Fit a distribution on a histo

```
h1->Fit("gaus"); // gaussian fit for 1D histo
```

3.3 Draw options

`"E"` show error bars

`"hist"` show only histogram

`"lego"` lego plot

`"cont"` contour lines (linee di livello)

`"Surf"` surface

`"P"` draw marker (except empty bins)

`"AXIS"` draw only axis

`"AXIG"` draw only grid (if requested)

`"FUNC"` When histo has fitted function, draw the fit result only.

`"TEXT"` Draw bin contents as text (format set via `gStyle->SetPaintTextFormat`).

`"X+"` The X-axis is drawn on the top side of the plot.

`"Y+"` The Y-axis is drawn on the right side of the plot.

`"MIN0"` Set minimum value for the Y axis to 0, equivalent to `gStyle->SetHistMinimumZero()`.

OPTIONS ARE NOT CASE SENSITIVE

They can also be concatenated without spaces & commas: `"opt1 opt2"` `h1->SetMarkerStyle(<code>);` set style, see below for codes: Also `SetFillColor(...)`, `SetLineStyle(...)`. `SetLineColor(...)` avaiable

3.4 Other member functions for histos

`GetMean()` mean

`GerRMS()` `GetStdDev()` root of variance / SD

`GetMaximum()` maximum bin content

`GetMaximumBin()` location of maximum ($\neq$ former)

`GetBinCenter( <bin_number> )` center of bin

`GetBinContent( <bin_number> )` content of bin

`GetBinError( <bin_number> )`

`SetBinContent( <bin_number>, <value> )`

`SetBinError( <bin_number>, <value> )`

Note: for out-of-range entries:
`GetBinContent(0)` returns number of **underflow**
`GetBinContent(<Nbins + 1>)` return number of **overflow**

`GetEntries()` total entries (includes under/overflows)

`Integral( <bin_index1>, <bin_index2> )` integral on specified range

`Integral()` total integral

`GetIntegral()` array of cumulative entries

`GetMeanError()` error on mean estimate

`GetRMSError()` `GetStdDevError()` error on RMS estimate

3.5 Operations on histos

Form homologue histograms (**same range and number of bins**): overloads for **istances**, **NOT POINTERS**:

```
TH1F h1;
TH1F h2 = 3*h1;
TH1F h3 = h1+h2;
```

Otherwise through methods:

```
h->Add(<pt1>, <pt2>, <n1>, <n2>); // sum stored in *h, *h = n1*h1+n2*h2
h->Multiply(3);
h->Divide(<pt1>, <pt2>, <n1>, <n2>); // analogous to sum
```

4 Macros

Two types of script

Unnamed script all code between $\{\}$ + no declaration of classes, functions + no parameters (ok loops)

Named script like any C++ function + possible to define other functions, classes, use parameters
The executed function has the same name of the file (see Basics)

5 GUI

`TBrowser b` opens root files browser.
Double click on an object (e.g. histo) $\rightarrow$ opens new **TCanvas** and draws it

Handling TCanvas (if some of the followings not visible, click **View** and check out

Editor single left click on an object in graph $\rightarrow$ edit display parameters (color etc.)

Toolbar tools to insert text, symbols, etc.

Status bar shows object pointed by mouse & mouse position

Right click on object $\rightarrow$ contextual menu

Contextual menu

Rebin redefine binning

Fit (of FitPanel) fit a function on data (gaussian, exponential, polynomial etc.) $\rightarrow$ button `Set Parameters` for chosen distribution

To visualize fit on graph: right click on graph $\rightarrow$ open `TPaveStats::stats` $\rightarrow$ `SetOptFit` $\rightarrow$ se to `111`
`SetOptStat` allows do define other options

Canvas options Right click on canvas → `SetLogx`, `SetLogy` for logarithmic scale; `SetGridx`, `SetGridy` for grid

Saving file `File` ▶ `Save` (`Save As`)
Saving as `.C` file (containing the graph as C++ commands) enables to reproduce graph executing macro
Saving as `.root` file → saves canvas and all objects, double click on canvas inside .root (opened through TBrowser) to open and manipulate graph

6 Global variables

`gROOT` global info on current session: access to **every object created during session**

`gFile` current root file

`gStyle` access functionalities to manage graphic style

`gRandom` access random number generator (see MC)

Suggestion at the beginning of a macro, to eliminate copy created by multiple executions of code in a session:

```
delete gROOT->FindObject("<name>");
```

(`FindObject("<name>")` used to retrieve every object from gROOT

7 Managing .root files

`TFile *file = new TFile("<name>.root", "RECREATE");` open file. `RECREATE` creates new file if name not found, otherwise overwrites existing one. Alternative option: `"NEW"` (error if already existing!)

`h->Write();` write object (pointed by `h`) on file

`file->Close();` close

8 Graphs

Two classes: `TGraph` (series of N X-Y couples), `TGraphErrors` (derived from former, includes also errors on both X and Y)

TGraph Constructors

`TGraph (Int_t n, const Double_t *x, const Double_t *y)` n couples, x and y are **arrays**!

`TGraph (const char *filename, const char *format="%lg %lg", Option_t *option="")` input file **must contain 2 separate columns of values** (divided by blank delimiter)
Default format: `"%lg %lg"` (2 double), to skip columns: `%lg %*lg %lg"`
Additional options to interpret different delimiters: can be explicitly specified in option argument (`option = "<symbol>"`)

TGraphErrors Constructors

`TGraphErrors (Int_t n, const Double_t *x, const Double_t *y, const Double_t*ex=0, const Double_t *ey=0)`
analogous to TGraph, ex, ey = arrays of errors

`TGraphErrors (const char *filename, const char *format="%lg %lg %lg %lg", Option_t *option="")`
input file **must contain at least 3 columns**. If there are 4 (or more, only first 4 read): X, Y, EX, EY. If only 3: X,Y,EY.

COMMA FOR DECIMALS MUST BE REPLACED WITH DOT

8.1 Graph member functions

(graph here is the pointer) Cosmetics:

graph->SetTitle("<title>")

graph->SetMarkerStyle(kOpenCircle (here kOpenCircle is default code)

graph->SetMarkerColor(kBlue) (kBlue also default)

graph->SetLineColor(kBlue) ...

Statistical properties:

graph->GetCorrelationFactor()

graph->GetCovariance()

graph->GetPoint(<i>,<x>,<y>) returns i-th point

graph->GetX() / graph->GetY() returns pointer to array of x / y values

graph->GetN

graph->Integral()

graph->AddPoint(<x>,<y>)

graph->SetPoint(<i>, <x>, <y>)

8.2 Drawing graphs

graph->Draw(<options>)

"A" draws axes

"P" draws points markers

"E" draws error bars

8.3 Fit

graph->Fit(<string>) <string> contains C++ expression

graph->Fit(TF1* f1 with previously defined function (see after)

TF1* fitFunc = h->GetFunction("f1") recover fit function from histo (analogous for graph)

fitFunc->GetChiSquare()

fitFunc->GetParameter(<i>) i-th parameter value

fitFunc->GetParError(<i>) error on i-th parameter

8.3.1 Statistics & fit parameters

gStyle->SetOptStat(<ksiourmen>) choose which statistics param. to display (each mode with a value - default if omitted): k (kurtosis), s (skewness), i (integral), o (overflows), u (underflows), r (1 rms, 2 +rms error), m (1 mean, 2 +mean error), e (entries), n (name)
gStyle->SetOptFit(<pcev>) analogous for fit parameters: p (probability), c (chisquare / dof), e (errors), v (name/value of params: 1 only non-fixed, 2 all)

9 Functions

In 1 variable (x): class **TF1**. User-defined function (and function objects, lambda) or built-in function objects → **TFormula**
For more dimensions (variables) **TF2, TF3**.

`TF1 *f1 = new TF1("f1", "sin(x)/x\,<xmin>,<xmax>)`

`TF1 *f2 = new TF1("f2", "f1 * 2",0,10)` previously defined functions can be used in definition of new ones

`TF1 *f3 = new TF1("f3","[0]*x*sin([1]*x)",-3,3)` possible to use parameters: to **initialize** them,
`f3->SetParameter(<value0>, <value1>)`

```
Double_t MyFunction(Double_t *x, Double_t *par){
        Float_t xx = x[0];
        Double_t val = TMath::Abs(par[0]*sin(par[1]*xx)/xx);
        return val;
}
```

Note: important to follow this signature!

`TF1 *f4 = new TF1("f4",MyFunction,0,10,2);` last constructor parameter is **number of parameters in MyFunction**

Cosmetics

`f1->SetLineColor(kRed)`

`f1->SetLineStyle(2)` 2 = dashed, 3 = dotted, 4 = dasheddotted

Member functions:

`f1->Eval(<x_value>)` evaluate on a point

`f1->Integral(<a>, <b>)`

10 Legend

```
TLegend *leg = new TLegend(.1,.7,.3,.9,\ <title> ");
leg->AddEntry(graph,"Punti sperimentali");
leg->AddEntry(f,\Fit Lineare");
leg->Draw("Same");
```

11 Canvas syntax

`myCanvas->Print("<image-file>")`