

ROOT

Pocket reference for 1st and 2nd year courses - BSc Physics, Unibo

2023-2024

Contents

1	General structure	2
2	Basic shell & prompt commands	2
2.1	Recover session history	3
2.2	Check versions	3
2.3	L ^A T _E X	3
3	Macros	3
4	GUI	3
5	Global variables	4
6	Managing .root files	4
6.1	Decoupling from file	4
6.2	Make user defined types writeable	4
7	TList	5
8	Histograms	5
8.1	Overlap histos	6
8.2	Histo Draw options	6
8.3	Cosmetics for histos	6
8.4	Other member functions for histos	8
8.5	Operations on histos	8
8.6	Filling a histo from ascii file	9
9	Graphs	9
9.1	Graph member functions	9
9.2	Color reference	10
9.3	Drawing TGraph	10
9.4	Drawing TGraphErrors	11
9.5	Additional styling	11
10	Fit	11
10.1	Statistics & fit parameters	12
11	Functions	13
12	TMath and TFormula	14
13	Legend	16
14	Canvas syntax	17
15	Pseudo-Random number generation: TRandom	18
15.1	Methods for generic distributions	18
15.2	Random generation from a generic function	18
15.3	Filling an histo with randomly generated values	18
16	Benchmark	19

17 Trees	19
17.1 Filling	19
17.2 Reading and representing	19

A general note: strings between `< ... >` are meant to be replaced by suitable ones (without the two kets)

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`, `TH1C`, `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
- `.! wslview <image-file>` (for WSL users) open image with default photo viewer from inside ROOT

Run a macro:

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

Load a library (class def) - only with **implementation file**:

```
$ [0] gROOT->LoadMacro(\MyClass.cxx+")
```

or

```
$ [0] .L MyClass.cxx+
```

Produces a **dynamic library** file (`.so`).

To automatically reload when running main macro put at beginning of latter: `R__LOAD_LIBRARY(MyClass.so)`

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:**

```
$ [0] TH1F *histo=new TH1F(\histname"," Titolo", 100, 0, 10)
$ [1] 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`

2.2 Check versions

`gcc -v` gcc version

`lsb_release -a` linux OS

2.3 L^AT_EX

Can be used for labels etc. Same syntax as normal L^AT_EX

- $x_{\{1\}} = x_1$
- $x^{\{1\}} = x^1$

but commands are called with '`#`' instead of '`\`'

3 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)

4 GUI

`TBrowser b` opens root files browser.

Double click on an object (e.g. histo) → 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 → 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 → contextual menu

Contextual menu

Rebin redefine binning

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

To visualize fit on graph: right click on graph → open `TPaveStats::stats` → `SetOptFit` → set to `111`
`SetOptStat` allows to 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

5 Global variables

List of useful global pointers.

`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 PRNG)

`gPad` current pad (see Canvas)

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

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

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

General styling

`gROOT->SetStyle("<style>");` set window style. Can be custom one or chosen between default ones:

Classic, Plain, Modern, Bold, Video, Pub

6 Managing .root files

`TFile *file = new TFile("<name>.root", "RECREATE");` open file. Options:

`RECREATE` creates new file if name not found, otherwise overwrites existing one

`NEW` or `CREATE` (error if already existing, returns empty ptr!)

`UPDATE`

`READ` (default)

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

`file->Write();` write all histos in memory in current session

`file->ls();` list content, with indentation for file tree

`TH1F* hs = (TH1F*)file->Get("<name>");` retrieve object from name
Explicit cast suggested

`file->Close();` close

6.1 Decoupling from file

To avoid disappearance of histos from canvas after closing read file(s):

`TH1::AddDirectory(kFALSE);` static! (default value is `kTrue`)

`h->SetDirectory(0);` `h->DrawCopy();` non static

6.2 Make user defined types writeable

In header:

```
#ifndef MYCLASS_H
#define MYCLASS_H
...
class MyClass: public TObject{
public:
    ...
private:
```

```

    ...
    ClassDef(MyClass,1)
};
#endif

In implementation:
#include "MyClass.h"
...
ClassImp(MyClass)

```

7 TList

A container class to store objects of every type inheriting from TObject

```

TList * list= new TList();

list->Add(<ptr>); with pointer!

list->At(<idx>) indexing starts from 0, returns empty pointer if out of range

list->At(<idx>)->InheritsFrom("TGraph") check inheritance (boolean)

```

8 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>->FillRandom("<funcname>",<Ntimes>,TRandom* rng = nullptr) fill N times generating ac-
    cording to auto normalized function (using rng)

<pt-name>->Draw(); draw histogram

```

2D

```

TH2F* <pt-name> = new TH2F( "<name>","<title>",<Nx>,<xmin>,<xmax>,<Ny>,<ymin>,<ymax>);

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

<pt-name>->Draw();

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

<pt-name>->ProjectionY()

<pt-name>->GetNbinsX()

<pt-name>->GetNbinsY()

```

3D

```

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

```

ND

```

THnSparse* pt = new THSparse( "<name>", "<title>", <Ndims>, <xmin>, <xmax>, <chuncksize>);

min and max same for all dimensions

```

8.1 Overlap histos

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

8.2 Histo 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.

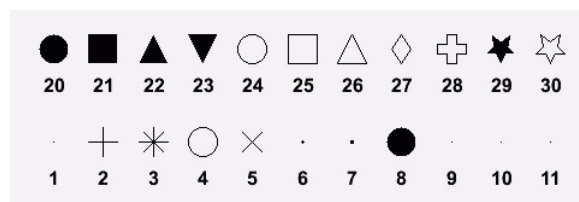
"MIN0" Set minimum value for the Y axis to 0, equivalent to `gStyle->SetHistMinimumZero()`.

OPTIONS ARE NOT CASE SENSITIVE

They can also be concatenated without spaces & commas: `"opt1 opt2"`

8.3 Cosmetics for histos

`h1->SetMarkerStyle(<code>);` set style, see ↓ for codes:



`h1->SetMarkerSize(<rel_sz>);`

`h1->GetXaxis()->SetTitle("<title>");` change axis title, same for y with `GetYaxis()`

`h1->GetXaxis()->SetTitleSize(<sz>);` expressed as percent of pad size
unless if precision = 3, when it's in pixels

`h1->GetXaxis()->SetTitleOffset(<ofs>);` 0 is default, 1 is standard offset, 1.x adds 10*x%

`SetFillColor(<color>);` see after for codes

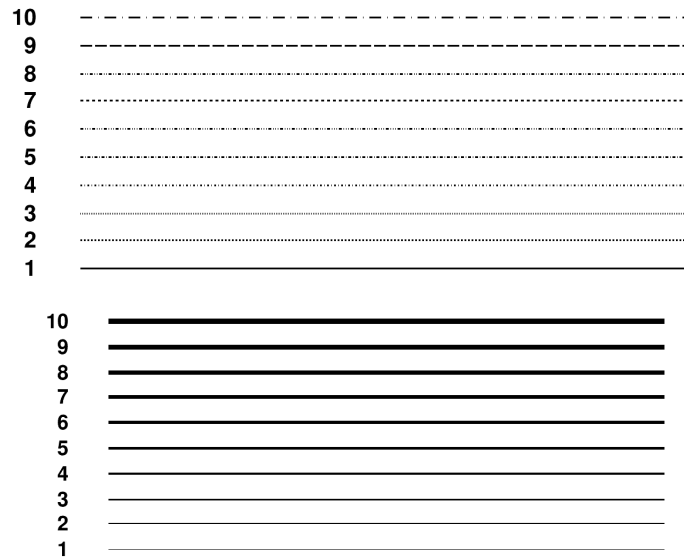
`SetFillColorAlpha(<color>, <transparency ratio>);` allows to manipulate opacity,

e.g. `(kBlue, 0.35)`

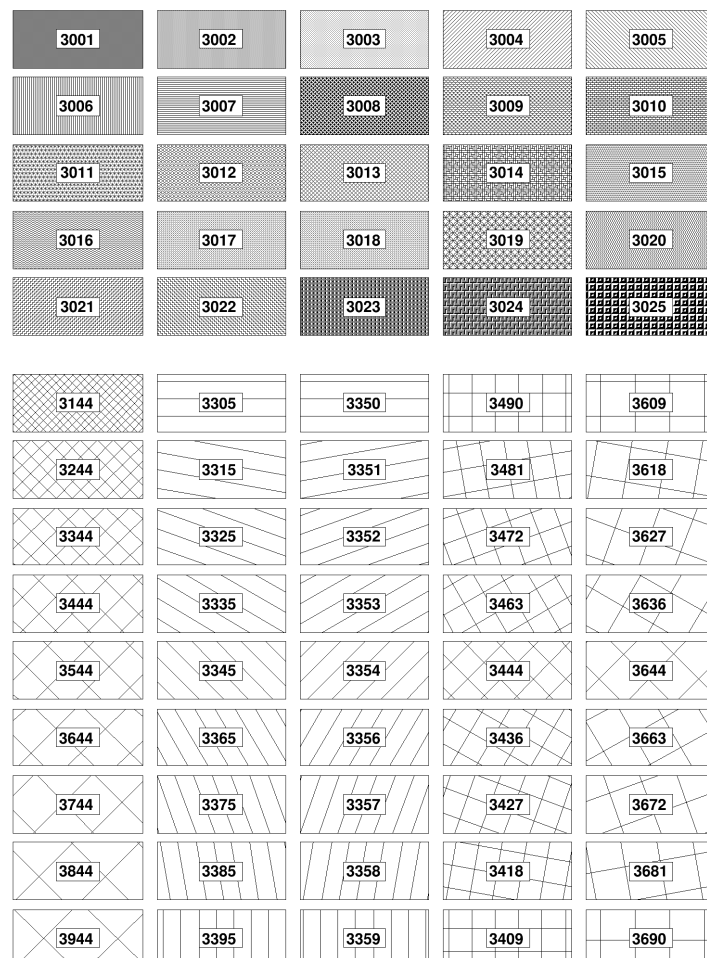
SetLineColor(<color>); or SetLineColorAlpha(<color>, <transp>);

SetLineStyle(<code>); see below

SetLineWidth(<width>); see below



SetFillStyle(<code>); 0 for hollow, 1001 for solid, 3000 + pattern number see below



8.4 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>)`

`GetNbinsX()` number of bins

Note: for out-of-range entries:

`h->GetBinContent(0)` returns number of **underflow**

`h->GetBinContent(h->GetNbinsX()+ 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

`Sumw2()` create structure to store sum of square of weights. **Strongly suggested** before performing operations on histos

8.5 Operations on histos

Form homologue histograms (**same range and number of bins**): overloads for **instances**, **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(<int>);`

`h->Divide(<pt1>, <pt2>, <n1>, <n2>);` analogous to sum

8.6 Filling a histo from ascii file

```
TH1F *h1 = new TH1F("h1","Tempi di Caduta",8,-0.5,15.5);

ifstream in;
in.open("maxwell.dat");
Float_t x,y;
while (1) { // always true condition: iterates until break called
    in >> x >> y;
    if(!in.good()) break;
    h1->Fill(y);
}
in.close();
```

9 Graphs

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

TGraph Constructors Derived class!

`TGraph (<int_N>, <db*_x>, <db*_y>)` N couples, `x` and `y` are **arrays** of size N

`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: explicitly specified in option argument (`option = "<symbol>"`)

TGraphErrors Constructors

`TGraphErrors (<int_N>,<db*_x>,<db*_y>,<db*_ex> =0, <db*_ey> =0)` analogous to `TGraph`
`ex`, `ey` = arrays of errors (for negligible/null uncertainty: substitute with `0`)

`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

9.1 Graph member functions

(`graph` here is the pointer) — inherited by `TGraphErrors` !

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();`

Other:

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

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

`graph->GetXaxis();` pointer to X axis ► `graph->GetXaxis()->SetTitle("title")`

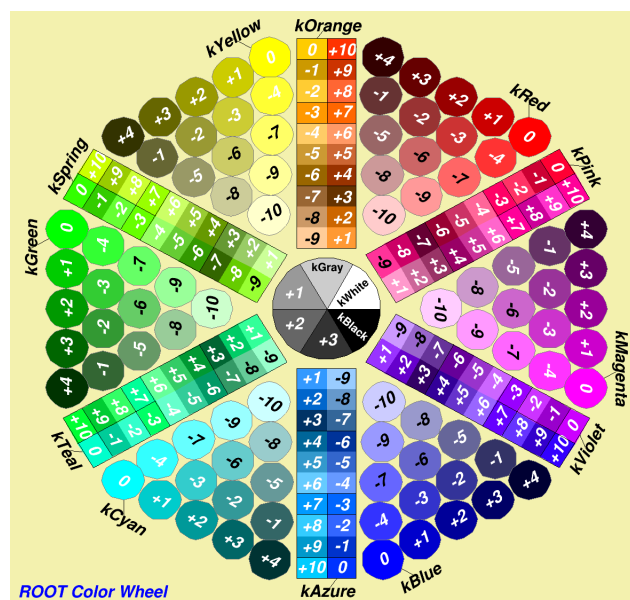
`graph->GetYaxis();` pointer to Y axis ► `graph->GetYaxis()->SetTitle("title");`

`graph->SetMinimum(<double>);` set minimum on Y

`graph->SetMaximum(<double>);` set maximum on Y

9.2 Color reference

See figure



9.3 Drawing TGraph

`graph->Draw(<options>)`

"A" draws axis

"P" draws points markers (the current one set)

"E" draws error bars

"AI" draws invisible axis (no labels)

* draws star at each point (alternative to P)

C draws a smooth curve connecting points

X+ X axis drawn on the top side

Y+ Y axis drawn on the right side

RX reverse the X axis

RY reverse the Y axis

9.4 Drawing TGraphErrors

Along with previous options, some specific ones:

- `Z` do **not** draw horizontal and vertical lines at the end of error bars
- `>` draw arrow at the end
- `|>` filled arrow
- `X` do **not** draw error bars
- `||` draw only lines at the end of bars, **not** bars themselves
- `0` force error bars drawing also for points outside visible range along Y (by default they're not drawn)
- `2` draw error rectangles
- `3` filled area through the end points
- `4` smoothed filled area
- `5` like `2`, but countour lines are drawn.

9.5 Additional styling

`gStyle->SetErrorX(<dx>)` if set to 0 removes error along x

`gStyle->SetEndErrorSize(<n_px>)` size of line at the end of error bars. Default = 1.

10 Fit

The following syntax is valid both for histos and graphs.

```
<pt>->Fit( "<name>", "<option>", "<graphic_opt>", <xmin>, <xmax>);
```

where `name` is one of the defaults:

`gaus` `gausn` (normalized)

`landau` `landaun` `expo` `pol1` `pol2` ... `pol9` (polynomial of degree n)

`chebyshev1` `chebyshev2` ... `chebyshev9`

To print list of available functions:

```
TF1::InitStandardFunctions(); // not needed if 'gROOT->GetFunction()' is called before
gROOT->GetListOfFunctions()->ls();
```

`name` can also be a formula accepted by the linear fitter with the operator `++`
(e.g. `x++sin(x)`, for fitting `[0]*x+[1]*sin(x)`)

```
<pt>->Fit( TF1* f1, "<option>", "<graphi_opt>", <xmin>, <xmax>) with previously defined function (see after)
```

`graphic_opt` is analogous to the one for `Draw`, whereas `option` can contain one (or more) of the following:

FOR HISTOS ONLY

- `L` use logarithmic likelihood method (instead of default Chi square)
- `WIDTH` scales histogram bin content by bin width (useful for variable bins)
- `MULTITHREAD` forces employment of multithreading whenever possible

FOR GRAPHS ONLY

- `W` ignore point errors when fitting TGraphErrors

FOR BOTH HISTOS AND GRAPHS

R use fitting range specified in the function range (default is histo's)
C in case of linear fit, disables calculation of Chi square (saves CPU time)
Q quiet mode: print minimum data
V verbose mode: print everything
S stores full fit result and returns a `TFitResultPtr` for access
`TF1* fitFunc = <pt>->GetFunction("f1")` recover fit function from histo (analogous for graph)
`fitFunc->GetChisquare()`
`fitFunc->GetNDF();` degrees of freedom
`fitFunc->GetParameter(<i>);` `i`-th parameter value
`fitFunc->GetParError(<i>);` error on `i`-th parameter

ONLY IF "S" OPTION USED:

```
TFitResultPtr r = h->Fit("\fitFunc", \S");  
TMatrixD cor = r->GetCorrelationMatrix();  
TMatrixD cov = r->GetCovarianceMatrix();  
cor.Print();  
cov.Print();
```

10.1 Statistics & fit parameters

`gStyle->SetOptStat(<ksiourmen>)` choose statistics parameters to be displayed
(each mode with a value - default **0** if omitted):

- k** **1** = print kurtosis, **2** = print kurtosis + k. error
- s** **1** = print skewness, **2** = print skewness + s. error
- i** **1** = print integral of bins, **2** = print integral of bins with option
- o** **1** = print number of overflows
- u** **1** = print number of underflows
- r** **1** = print SD **2** = print SD + SD error ¹
- m** **1** = print mean **2** = print mean + mean error
- e** **1** = print number of entries
- n** **1** = print histogram name

STARTS FROM THE END:

```
gStyle->SetOptStat(11); // only name + entries  
gStyle->SetOptStat(1101); // name, mean, RMS  
gStyle->SetOptFit(<pcev>) analogous for fit parameters:
```

- p** **1** = print Probability
- c** **1** = print Chisquare / Number of d.o.f.
- e** **1** = print errors
- v** **1** = print name/values of parameters (only non-fixed) **2** = print name/value of *all* parameters

`gStyle->SetOptFit(1)` is **equivalent** to `gStyle->SetOptFit(111)` (!)

¹Actually **r** stands for Root Mean Square, defined according to

$$x_{RMS} = \sqrt{\frac{1}{n} \sum x_i^2}$$

11 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

```
f3->SetParameter(<index>, <value>);
```

 to **initialize** one of them

```
f3->SetParameters(<value1>, <value2>, ..., <valuek>);
```

 following order!

See TFormula for more info

For user-defined:

```
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**

```
TF1 *f5 = new TF1("f5", [](double *x, double *p){ <function body> }, <xmin>, <xmax>, <npar>);
```

 use of lambdas is also possible

```
TF1 *f6 = new TF1("f6", "[0](double *x, double *p){ <function body> }", <xmin>, <xmax>, <npar>);
```

 also as string expression (JIT will do the rest)

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>);
```

 compute $\int_a^b f1$

```
f1->SetMaximum(<value>);
```

 set maximum for Y axis

```
f1->SetMinimum(<value>);
```

 minimum for Y axis

```
f1->SetRange( <x_min> , <x_max> );
```

 set interval for independent variable to $[x_min, x_max]$

12 TMath and TFormula

TMath

`TMath::Abs(<x>)`

`TMath::AreEqualAbs(<x>,<y>,<eps>)` returns true if `TMath::Abs(x-y) < eps`

`TMath::ASin(<x>)` ; `TMath::ASinh(<x>)`; `TMath::ATan(<x>)`; `TMath::ACos(<x>)`; `TMath::ACosh(<x>)`

`TMath::Cos(<x>)` ; `TMath::Cosh(<x>)`

`TMath::Sin(<x>)` ; `TMath::Sinh(<x>)`

`TMath::Tan(<x>)` ; `TMath::Tanh(<x>)`

`TMath::Ln10()` returns $\ln 10$

`TMath::LogE()` returns $\log_{10} e$

`TMath::Ldexp(<x>, <exp>)` where `exp` is integer. Returns $x \cdot 2^{exp}$

`TMath::Log(<x>)` natural logarithm

`TMath::Log10(<x>)` returns $\log_{10} x$

`TMath::Log2(<x>)` returns $\log_2 x$

`TMath::Max(<x>,<y>)` returns maximum value between `x` and `y` (for integers, doubles... everything)

`TMath::Min(<x>,<y>)` same but for minimum

`TMath::Nint(<x>)` rounds `x` to nearest integer

`TMath::Power(<x>,<y>)` returns x^y

`TMath::Prob(<chi2>,<ndof>)` returns probability for $\chi^2 = \text{chi2}$ with `ndof` degrees of freedom

`TMath::Sq(<x>)` returns x^2

`TMath::Sqrt(<x>)` returns \sqrt{x}

CONSTANTS

`TMath::E()` returns e

`TMath::G()` returns G

`TMath::Gn()` returns g

`TMath::H()` returns h `TMath::Hbar()` returns \hbar

`TMath::K()` returns k_B

`TMath::Na()` returns N_A

`TMath::Pi()` returns π

`TMath::R()` returns R

`TMath::Sigma()` returns σ (Stefan-Boltzmann)

`TMath::Sqrt2()` returns $\sqrt{2}$

TFormula

`gaus(<const>,<mean>,<sigma>)` not normalized

`landau(<mpv>,<sigma>)`

`expo(<const>,<slope>)` e^{A+Bx}

`pol<N>(<p1>, ... ,<pN>)` polynomial $\sum_{i=1}^N p_i \cdot x^i$

`sqrt(<x>)`

`sq(<x>)` x^2

`pow(<x>,<y>)` x^y

`<x>*<y>`

`<x>^<n>` or `<x>**<n>`

`<x>/<y>`

`sin(<x>)` `cos(<x>)` `tan(<x>)`

`asin(<x>)` `acos(<x>)` `atan(<x>)`

`sinh(<x>)` `cosh(<x>)` `tanh(<x>)`

`asinh(<x>)` `acosh(<x>)` `atanh(<x>)`

`exp(<x>)`

`log(<x>)`

`log10(<x>)`

`e` | `pi`

`ln10` | `sqrt2`

In function initialization, values can be replaced either with the variable `x` or parameters `[i]`. Some special ones, where `n` is the starting index for numbering.

`gaus(<n>)` not default normalized gaussian with three parameters

`gausn(<n>)` normalized gaussian with three parameters

`expo(<n>)` exponential with three parameters

`pol<N>(<n>)` polynomial with `N` parameters

13 Legend

```
TLegend *leg = new TLegend(<x1>,<y1>,<x2>,<y2>,"<title>");
```

(x_1, y_1) = bottom left corner, (x_2, y_2) = upper right corner **in normalized coordinates**

so 1 = **pad** height / width

$$x = \frac{\text{absolute horizontal position}}{\text{screen width}} \quad y = \frac{\text{absolute vertical position}}{\text{screen height}}$$

x goes from left to right, y from bottom to top

Careful when using more pads (see **Canvas syntax**)

```
leg->AddEntry(graph,"Punti sperimentali");
```

```
leg->AddEntry(f,"Fit Lineare");
```

```
leg->AddEntry(<object>,"<description>");
```

```
leg->AddEntry(<object>,"<description>", "<option>"); // alternative syntax
```

Possible options (even more than one of these):

l line

e error bar

p point

```
leg->Draw("Same");
```

```
leg->SetTextAlign(<nm>); with nn = 11, ..., 13
```

11	21	31
12	22	32
13	23	33

Cosmetics (**gStyle** member functions)

```
gStyle->SetLegendBorderSize(<n>);
```

```
gStyle->SetLegendFillColor(<color>);
```

```
gStyle->SetLegendFont(<n>); // see below
```

```
gStyle->SetLegendTextSize(<size>); // see below
```

font code ($\langle n \rangle$) = 10 × font number + precision

Example of fonts with precision = 2:

```
12 : ABCDEFGH abcdefgh 0123456789 @#$
22 : ABCDEFGH abcdefgh 0123456789 @#$
32 : ABCDEFGH abcdefgh 0123456789 @#$
42 : ABCDEFGH abcdefgh 0123456789 @#$
52 : ABCDEFGH abcdefgh 0123456789 @#$
62 : ABCDEFGH abcdefgh 0123456789 @#$
72 : ABCDEFGH abcdefgh 0123456789 @#$
82 : ABCDEFGH abcdefgh 0123456789 @#$
92 : ABCDEFGH abcdefgh 0123456789 @#$
102 : ABCDEFGH abcdefgh 0123456789 @#$
112 : ABCDEFGH abcdefgh 0123456789 @#$
122 : ABXΔEΦΓΗ αβχδεφγη 0123456789 ≡#≡
132 : ABCDEFGH abcdefgh 0123456789 @#$
142 : 𐀀𐀁𐀂𐀃 𐀄𐀅𐀆𐀇 𐀈𐀉𐀊𐀋 𐀌𐀍𐀎𐀏 𐀐𐀑𐀒𐀓
152 : ABXΔEΦΓΗ αβχδεφγη 0123456789 ≡#≡
```


14 Canvas syntax

```
TCanvas* myCanvas = new TCanvas()
```

```
TCanvas* myCanvas = new TCanvas()
```

`myCanvas->Print("<file-name>.<extension>", "<option>");` prints canvas to file. Possible formats:

`.ps` (Postscript, default one) with options `Portrait` or `Landscape`,

`.eps` (encapsulate Postscript), `.pdf` with option `Title: <title>`,

`.svg`, `.tex`, `.gif`, `.gif+<N>` (animated gif, where `N` is the delay in units of 10ms)

`.xpm`, `.png`, `.jpg`, `.tiff`, `.cxx`, `.xml`, `.json`, `.root`

```
myCanvas->SetCanvasSize(<x_px>,<y_px>);
```

```
myCanvas->SetWindowSize(<x_px>,<y_px>);
```

If canvas size exceeds window size, scrollbars are displayed

```
myCanvas->GetWh();
```

 get value of window height

```
myCanvas->GetWw();
```

 get value of window width

```
myCanvas->ToggleToolBar();
```

 hides if shown or vice versa

Pads

```
myCanvas->Divide(<nx>, <ny>)
```

 divides equally into `nx×ny` pads

```
myCanvas->Divide(<nx>, <ny>, <x_margin>, <y_margin>, <color>)
```

 Same; margins are given as **percent of canvas**. `color` is the color of new pads;

Pads can be divided in sub-pads.

```
myCanvas->cd(<pad_num>)
```

 sets current pad. Starts from `1`, `0` is parent pad (frame). Current pad can be retrieved through `gPad`. It goes by rows, so the numbering looks like:

$$\begin{bmatrix} 1 & \rightarrow & n \\ n+1 & \rightarrow & 2n \\ \vdots & \vdots & \vdots \\ (m-1)n+1 & \rightarrow & mn \end{bmatrix}$$

15 Pseudo-Random number generation: TRandom

Classes with algorithms employed:

- TRandom Linear Congruential Generator

↖ *It's a **very bad** generator, not to be used!*

- TRandom1 RANLUX
- TRandom2 Tausworthe
- TRandom3 Mersenne Twister

15.1 Methods for generic distributions

Called on global pointer `gRandom` or on `TRandom*` pointer after declaration

`Uniform(<double_1>, <double_2>)` uniform distribution on `]double_1, double_2]` based on `Rndm()`

`Rndm()` uniform in `]0, 1]`

`Uniform(<double>)` uniform distribution on `]0, double]`

`Integer(<int_max>)` uniform **integer** distribution on `[0, int_max - 1]`

`Gaus(<mean>, <sigma>)` (careful, just one '**s**')

`Poisson(<mean>)` **integer** poissonian distribution

`PoissonD(<mean>)` **double** poissonian distribution

`Binomial(<n_tot>, <prob_of_succ>)`

`Exp(<tau>)`

`Landau(<mpv>, <sigma>)` Landau distribution: `mpv` is the *most probable value* (moda) and `sigma` is *not* the SD (which is undefined)

15.2 Random generation from a generic function

```
TF1 *f1 = new TF1("f1", "<expression>", <xmin>, <xmax>);  
double rd = f1->GetRandom();
```

`r` is now a random variable distributed according to the PDF defined by `f1`. It is not necessary to manually ensure that `f1` is normalized: **normalization is carried out automatically**, the only requirement on the function is **continuity**.

`histo->FillRandom("<function>", <n>)` fills the histogram with `n` extractions from a random variable distributed according to `f1` (name). Same considerations as before apply.

15.3 Filling an histo with randomly generated values

```
for(Int_t j=0;j<ngen;j++){ // generation loop  
    Double_t x = gRandom->Uniform(xmin,xmax); // extraction  
    h->Fill(x); // filling  
}
```

16 Benchmark

```
TBenchmark *b = new TBenchmark(); initialize  
  
b->Start("<name>"); start benchmark with specified name  
    (more with different names from same b)  
  
b->Stop("<name>"); stop benchmark with specified name  
  
b->Print("<name>"); print parameters  
  
b->Show("<name>"); = stop + print
```

17 Trees

Data structures for storage of homologous units made of *heterogeneous* objects. Useful for data persistence, optimization of memory usage (thanks to compression) and I/O speed - especially in WORM - and flexibility in analysis.

Can handle **any** data type. Direct access to any point; only object (even partially) placed on memory. Organised in TBranches, themselves structured in TLeafs

17.1 Filling

```
TTree *T = new TTree("T","test"); // create the tree  
Float_t x,y,z;  
T->Branch("x",&x,"x/F"); // create branches  
T->Branch("y",&y,"y/F"); // <name>/<type>  
T->Branch("z",&z,"z/F");  
for(Int_t i=0;i<100;i++){  
    ... // read or generate x y z, very costly operations  
    T->Fill(); // fill the tree, for each entry  
}
```

17.2 Reading and representing

```
T->Print(); // list of contained variables  
  
T->Draw("\x"); // show variable. Histo layout chosen by ROOT  
  
TH1F *h1=new TH1F("h1","hist from tree",50,-4, 4);  
T->Draw("x>>h1"); // to fill user defined histo  
  
T->Draw("x","x>0"); // selection on 1v  
T->Draw("x","y>0 && x<10") // selection on 2vv  
  
T->Draw("\sqrt(x*x+y*y)"); // operation on variables  
  
T->Draw("\y:x"); // 2d correlation plot  
T->Draw("\z:y:x"); // 3d
```