ROOT

Data Analysis Framework

PH2282 extra 1: ROOT

Applied Multi-Messenger Astronomy 2:
Statistical and Machine Learning Methods in Particle and Astrophysics

Matteo Agostini

TUM - summer term 2019

What's ROOT?

ROOT is a software toolkit which provides building blocks for:

- Data processing
- Data analysis
- Data visualisation
- Data storage

ROOT is written mainly in C++ (C++11 standard)

- Bindings for Python and other languages provided

Adopted in High Energy Physics and other sciences (but also industry)

- 250 PetaBytes of data in ROOT format on the LHC Computing Grid
- Fits and parameters' estimations for discoveries (e.g. the Higgs)
- Thousands of ROOT plots in scientific publications

https://root.cern.ch/

ROOT
Data Analysis Framework

Google Custom Search

Download Documentation News Support About Development Contribute

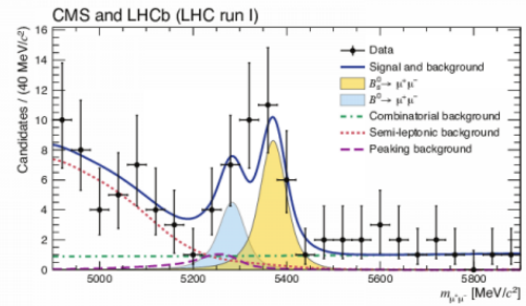
Getting Started | Reference Guide | Forum | Gallery

ROOT is ...

A modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

Try it in your browser! (Beta)

Download ROOT or Read More ...



Previous Pause Next

Under the Spotlight

08-03-2017 **Development release 6.09/02 is out!**

This is the first ROOT development release of the 6.09 series! It is meant to offer a preview of the many features which will be included in the 6.10 production release.

05-09-2016 **Get the most out of the ROOT tutorials!**

Other News

16-04-2016 The status of reflection in C++

05-01-2016 Wanted: A tool to 'warn' user of inefficient (for I/O) construct in data model

03-12-2015 ROOT::TSeq::GetSize() or ROOT::seq::size()?

ROOT in a Nutshell

ROOT can be imagined as a family of building blocks for a variety of activities, for example:

- C++ interpretation: fully C++11 compliant
- Math: non trivial functions (e.g. Erf, Bessel)
- Data analysis: histograms, graphs, trees
- I/O: row-wise, column-wise storage of any C++ object inference
- Statistical tools (RooFit/RooStats): rich modeling and statistical inference
- Multivariate Analysis (TMVA): boosted decision trees, neural networks...
- And more: HTTP servering, JavaScript visualisation, advanced graphics (2D, 3D, event display).
- PROOF: parallel analysis facility

Interpreter

ROOT is shipped with an interpreter, CLING:

- C++ interpretation: highly non trivial and not foreseen by the language!
- One of its kind: Just In Time (JIT) compilation
- A C++ interactive shell.

Can interpret "macros" on the fly (non compiled programs)

- Rapid prototyping possible

ROOT provides also Python bindings:

- Can use Python interpreter directly after a simple import ROOT
- Possible to mix the two languages

Interpreter

```
~$ root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.06/02                        http://root.cern.ch |
  |                                  (c) 1995-2014, The ROOT Team     |
  | Built for linuxx8664gcc                                           |
  | From heads/master@v6-07-02-437-gb06340c, Mar 02 2016, 19:01:57    |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'        |
   ------------------------------------------------------------------

root [0] 3*3
(int) 9
root [1] a = 4
(int) 4
root [2] b = 5.5
(double) 5.50000
root [3] a*b
(double) 22.0000
root [4] sqrt(a*b)>2
(bool) true
```

```
Cling (C/C++ interpreter) meta commands usage
All commands must be preceded by a '.', except
for the evaluation statement { }
================================================================
Syntax: .Command [arg0 arg1 ... argN]

  .L <filename>                  - Load the given file or library

  .(x|X) <filename>[args]          - Same as .L and runs a function with
                                   signature: ret_type filename(args)

  .> <filename>                  - Redirect command to a given file
     '>' or '1>'                  - Redirects the stdout stream only
     '2>'                          - Redirects the stderr stream only
     '&>' (or '2>&1')                 - Redirects both stdout and stderr
     '>>'                          - Appends to the given file

  .U <filename>                  - Unloads the given file

  [...]

  .help                          - Shows this information

  .q                                 - Exit the program
```

A bit of C++: variables, if, cout

```
1  // this line is commented out
2  /* this block of text is commented out */
3
4  // c++ has many variables:
5  int myInteger = 1;
6  double myDouble = 1.1;
7  bool myYesNoAnswer = false;
8
9  // if statements
10 if (myYesNoAnswer == true) myInteger = 2; else myInteger = -1;
11
12 // print on screen
13 cout << "myInteger is now equal to " << myInteger << endl;
```

A bit of C++: variables, if, cout

```
root [0] // c++ has many variables:
root [1] int myInteger = 1;
root [2] double myDouble = 1.1;
root [3] bool myYesNoAnswer = false;
root [4] // if statements
root [5] if (myYesNoAnswer == true) myInteger = 2; else myInteger = -1;
root [6] // print on screen
root [7] cout << "myInteger is now equal to " << myInteger << endl;
myInteger is now equal to -1
```

```
root [0] // c++ has many variables:
root [1] myInteger = 1
(int) 1
root [2] myDouble = 1.1
(double) 1.10000
root [3] myYesNoAnswer = false
(bool) false
root [4] // if statements
root [5] if (myYesNoAnswer == true) myInteger = 2; else myInteger = -1;
root [6] // print on screen
root [7] cout << "myInteger is now equal to " << myInteger << endl;
myInteger is now equal to -1
```

A bit of C++: loops

```
root [0] // standard for loop
root [1] for (int i = 0; i<5; i++) cout << i << endl;
0
1
2
3
4
root [3] // new c++11 range-based loop
root [4] for (auto i : {0,4,3}) cout << i << endl;
0
4
3
root [5] // this is a vector
root [6] vector<int> myVector = {1,3,5};
root [7] for (int i = 0; i < myVector.size(); i++) {
root (cont'ed, cancel with .@) [8]   cout << myVector.at(i) << endl;
root (cont'ed, cancel with .@) [9]}
1
3
5
root [8] for (auto i : myVector) cout << i << endl;
1
3
5
```

A bit of C++: lambda functions

```
root [0] vector<int> v1 = {1,2,3,4,5,6,7,8,9};
root [1] vector<int> v2 = {9,8,7,6,5,4,3,2,1};
root [2] auto SmartPrint = [] (vector<int>& v) {
root (cont'ed, cancel with .@) [3] for (auto i : v) cout << i << " ";
root (cont'ed, cancel with .@) [4] cout << endl;
root (cont'ed, cancel with .@) [5] };
root [6] SmartPrint(v1)
1 2 3 4 5 6 7 8 9
root [7] SmartPrint(v2)
9 8 7 6 5 4 3 2 1
```

Lambda functions are very powerful objects also when coding a macro. They can capture any variable visible when they are defined and are typically used whenever the same block of lines is repeated multiple times on different objects.

A bit of C++: namespaces, classes and objects

- ROOT's namespace are just containers of functions. E.g. `TMath` encapsulates all math functions. The functions are accessible through the scope operator "::"

```
TMath::Gaus(...)
```

- ROOT's classes are containers of data and functions, e.g. the graph is a class named `TGraph`
- many graphs can be created from `TGraph`. Each of them is called an object (i.e. a concrete instance of a class). An object is constructed with special functions called **constructors**:

```
TGraph myGraph (``dataFile.txt'')
```

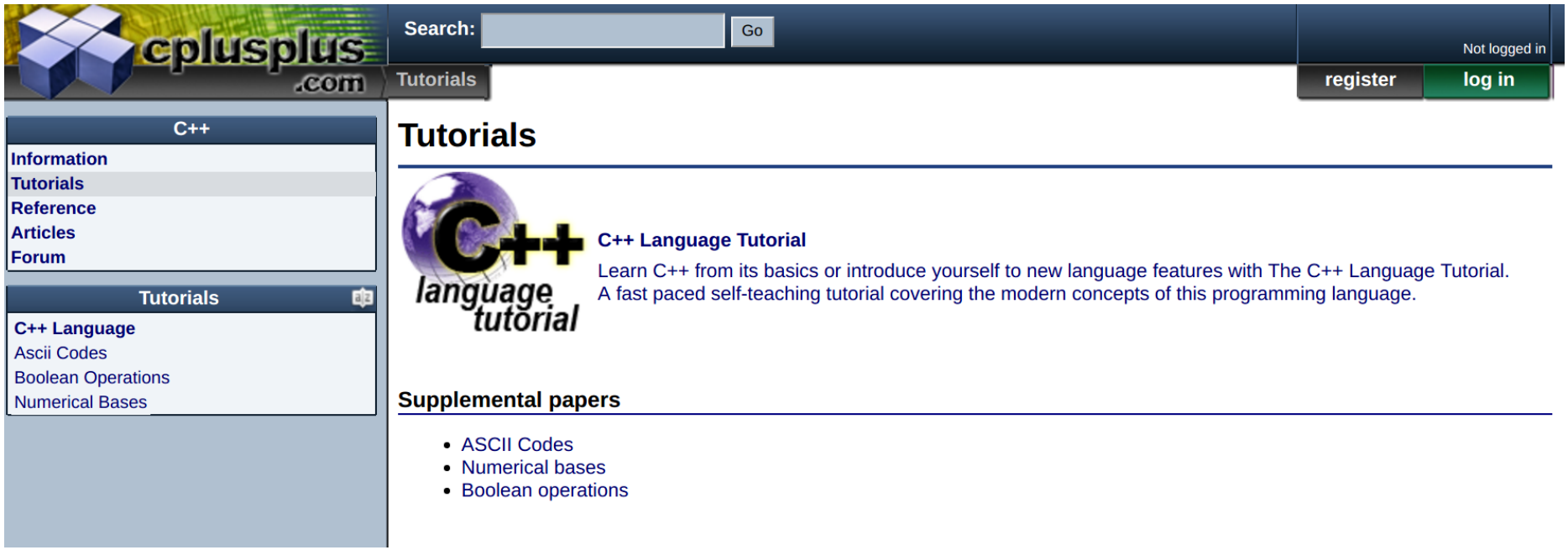
- The internal functions of an object are accessible through the "." operator:

```
myGraph.SetTitle(``theTitle'')
```

- Sometimes we end up having a pointer to an object, in this case the object functions are accessible through:

```
PointerToMyGraph->SetTitle(``theTitle'')
```

http://www.cplusplus.com



TMath functions

Tab-completion available in the interpreter!

```
root [0] TMath::
ACos
ACosH
ASin
ASinH
ATan
ATan2
ATanH
Abs
AreEqualAbs
AreEqualRel
BesselI
BesselI0
BesselI1
BesselJ0
BesselJ1
BesselK
BesselK0
[..]
```

TMath functions

A few examples (list also available trough tab-completion):

```
root [0] TMath::Gaus(
Double_t Gaus(Double_t x, Double_t mean = 0, Double_t sigma = 1, Bool_t norm =

root [0] TMath::Power(
Double_t Power(Double_t x, Double_t y)
Double_t Power(Double_t x, Int_t y)
LongDouble_t Power(Long64_t x, Long64_t y)
LongDouble_t Power(LongDouble_t x, Long64_t y)
LongDouble_t Power(LongDouble_t x, LongDouble_t y)

root [0] TMath::Min(
Double_t Min(Double_t a, Double_t b)
Float_t Min(Float_t a, Float_t b)
Int_t Min(Int_t a, Int_t b)
Long64_t Min(Long64_t a, Long64_t b)
Long_t Min(Long_t a, Long_t b)
Short_t Min(Short_t a, Short_t b)
UInt_t Min(UInt_t a, UInt_t b)
ULong64_t Min(ULong64_t a, ULong64_t b)
ULong_t Min(ULong_t a, ULong_t b)
UShort_t Min(UShort_t a, UShort_t b)
```

TMath functions

Short list:

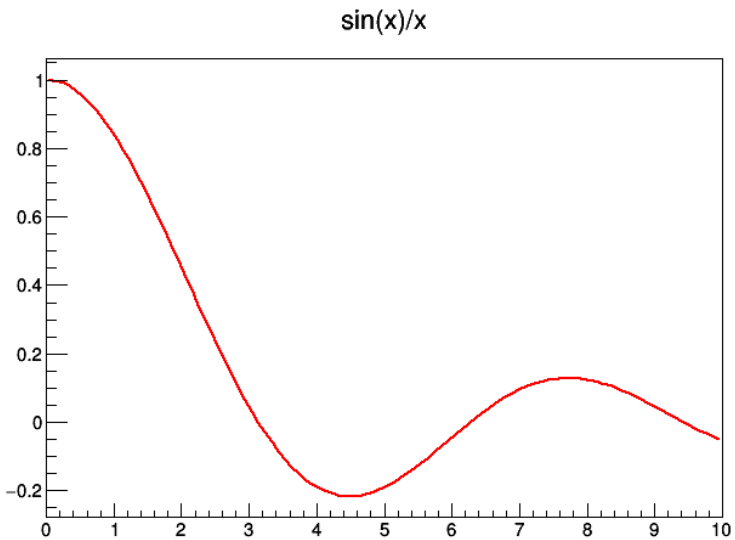
```
1   Abs                    // absolute value
2   Binomial               // binomial distribution
3   Ceil                   // down rounding to the closest integer
4   ChisquareQuantile      // chi-square quantile for a given probability
5   Erf                    // error function
6   Exp                    // exponential function
7   Floor                  // up rounding to the closest integer
8   Gaus                   // Gaussian function
9   Log                    // logarithm
10  Log10                  // logarithm with base 10
11  Max                    // max between two
12  Min                    // min between two
13  Pi                     // 3.1415...
14  Power                  // power
15  Poisson                // Poisson distribution
16  Sqrt                   // square root
```

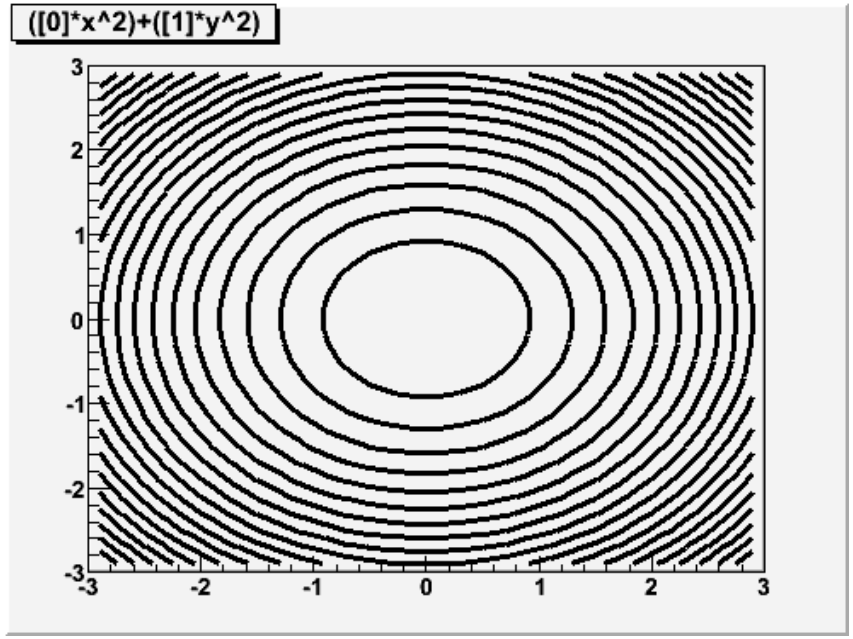
full list available at https://root.cern.ch/doc/v608/namespaceTMath.html

Data analysis objects: functions, hists and graphs

Functions:

- `TF1` A 1-Dim function with parameters.
- `TF2` A 2-Dim function with parameters.
- `TF3` A 3-Dim function with parameters.

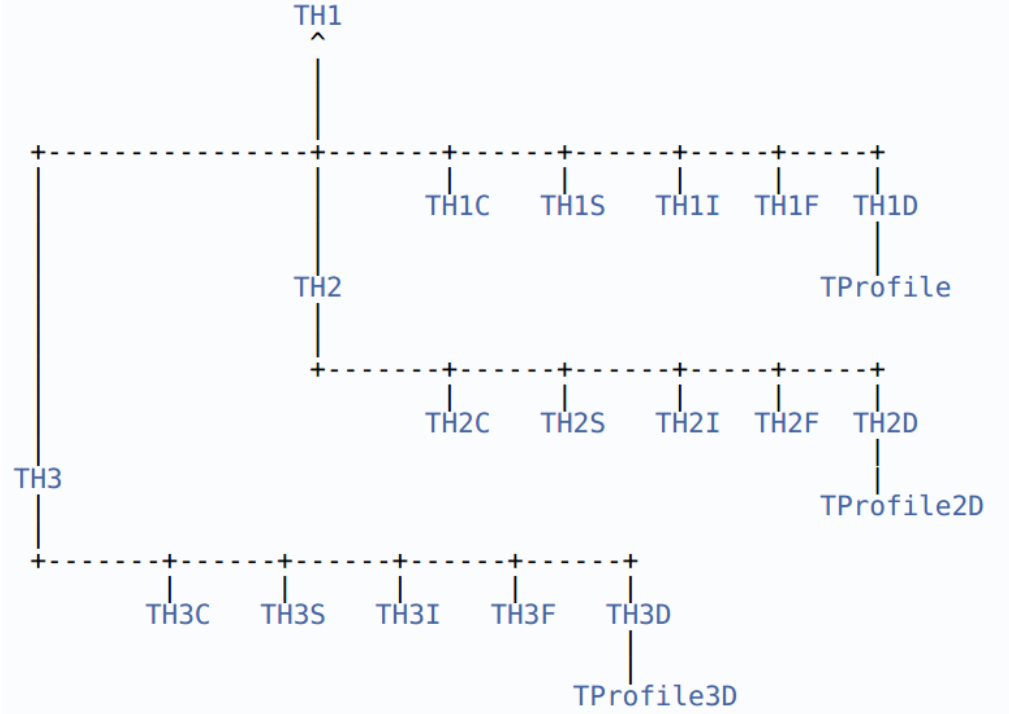




Data objects: functions, hists and graphs

Histograms (virtual classes):

- `TH1` 1-D hist per channel
- `TH2` 2-D hist per channel
- `TH3` 3-D hist per channel
- `THn` multidimensional hist
- `TProfile` profile hist
- `THSpare` spare multidim hist



Note:

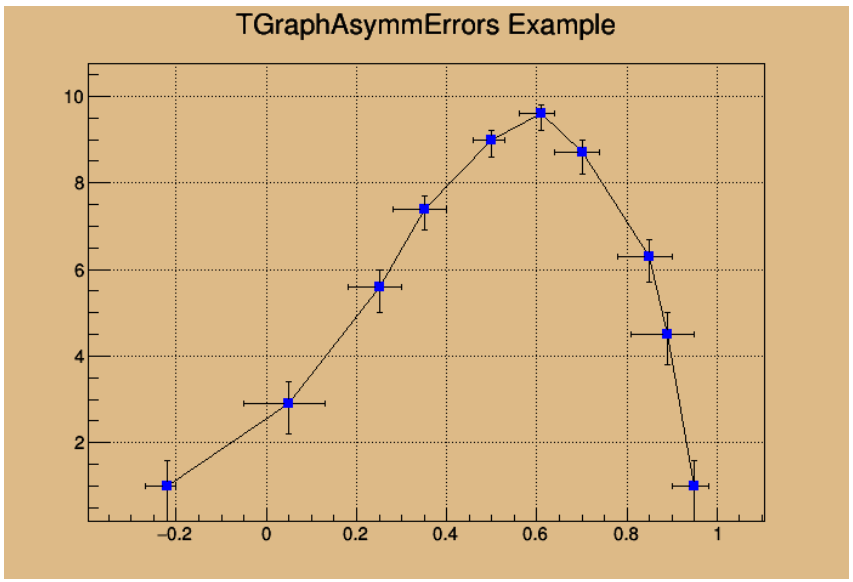
- concrete histograms are of a specific type, e.g. TH1D or TH1I
- Profile histograms are used to display the mean value of Y and its error for each bin in X.
- THn and THSpare are template classes

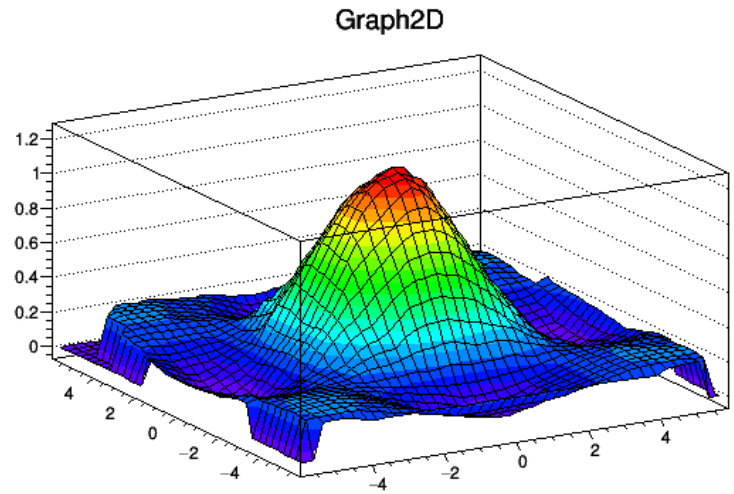
```
C -> char
S -> short
I -> int
F -> float
D -> double
```

Data analysis objects: functions, hists and graphs

Graph: object made of two (or more) arrays X and Y with npoints each.

- `TGraph` 1-Dim graph with only points
- `TGraph2D` 2-Dim graph with only points
- `TGraphErrors` 1-Dim graph with points and errors
- `TGraph2DErrors` 2-Dim graph with points and errors
- `TGraphAsymmErrors` 1-Dim graph with points and asymmetric errors





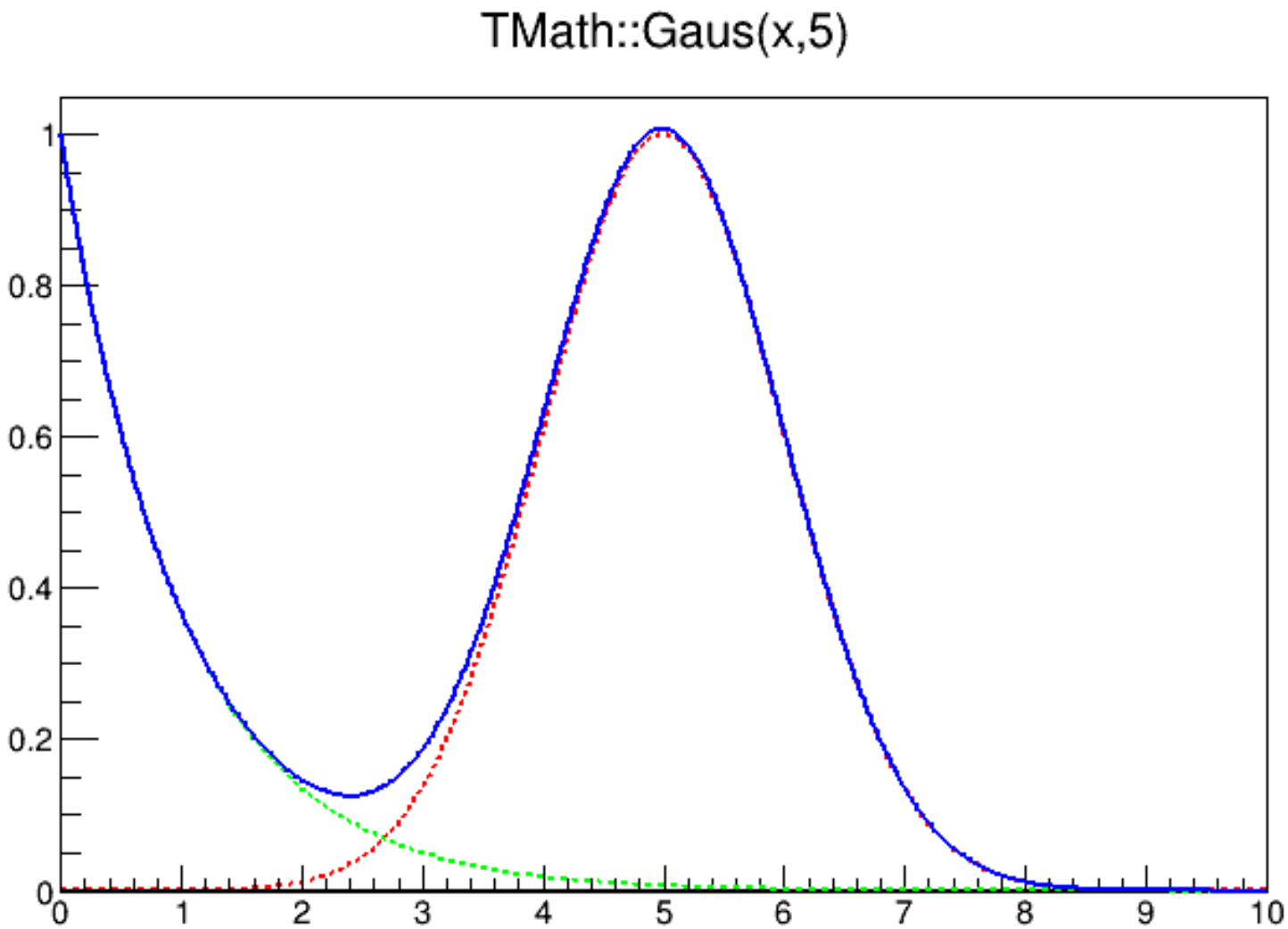
graphs are used also for drawing bands

TF1 (1)

```
// Constructor of a TF1
TF1::TF1(const char* name, const char* formula,
         Double_t xmin = 0, Double_t xmax = 1);
```

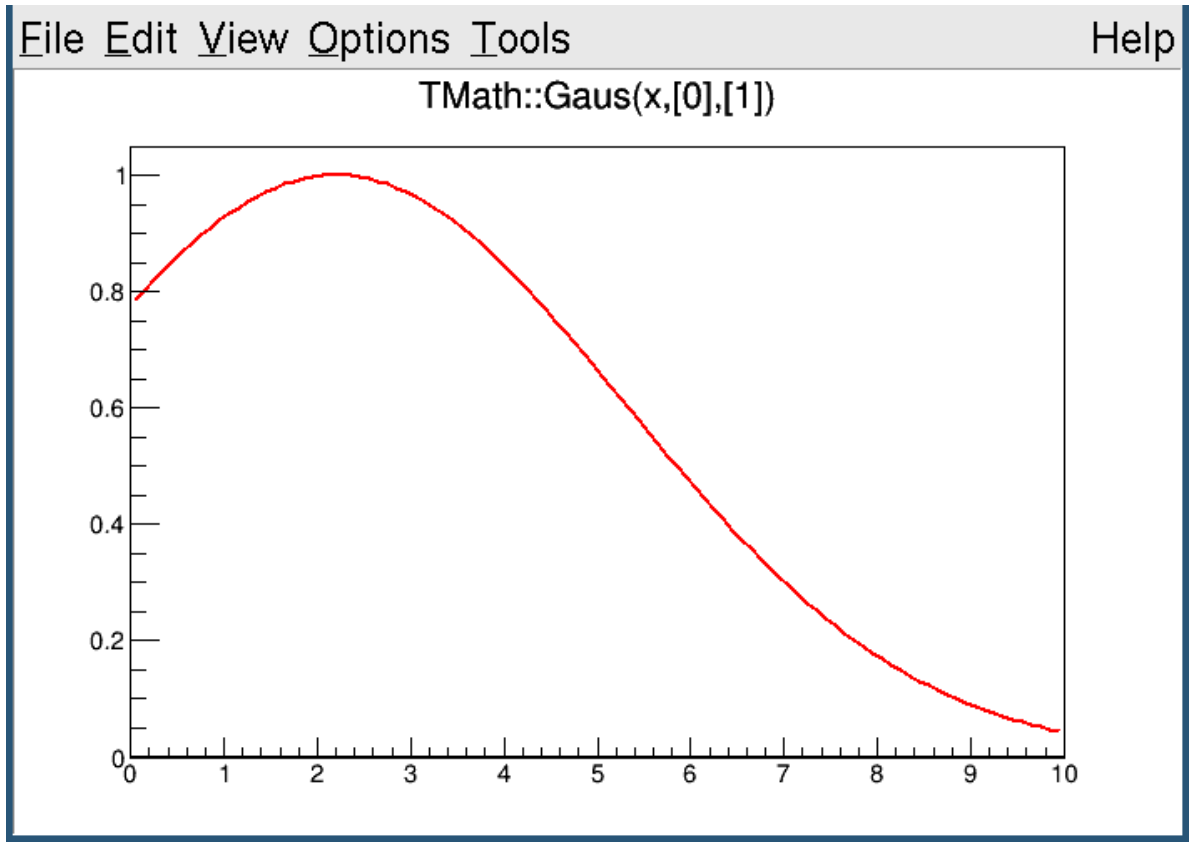
```
// Some commands
// cut and paste them into the interpreter!
TF1 f1 ("f1", "TMath::Gaus(x,5)",0,10.);
f1.SetLineColor(kRed);
f1.SetLineStyle(2); // dashed
f1.SetNpx(1e5); // increase number of displayed points
f1.DrawCopy(); // draw obj

TF1 f2 ("f2", "TMath::Exp(-x)",0,10.);
f2.SetLineColor(kGreen);
f2.SetLineStyle(2);
f2.SetNpx(1e5);
f2.DrawCopy("same"); // same = add obj to plot

TF1 f3 ("f2", "TMath::Gaus(x,5)+TMath::Exp(-x)",0,10.);
f3.SetLineColor(kBlue);
f3.SetLineStyle(1);
f3.SetNpx(1e5);
f3.DrawCopy("same");
```

TF1 (2)



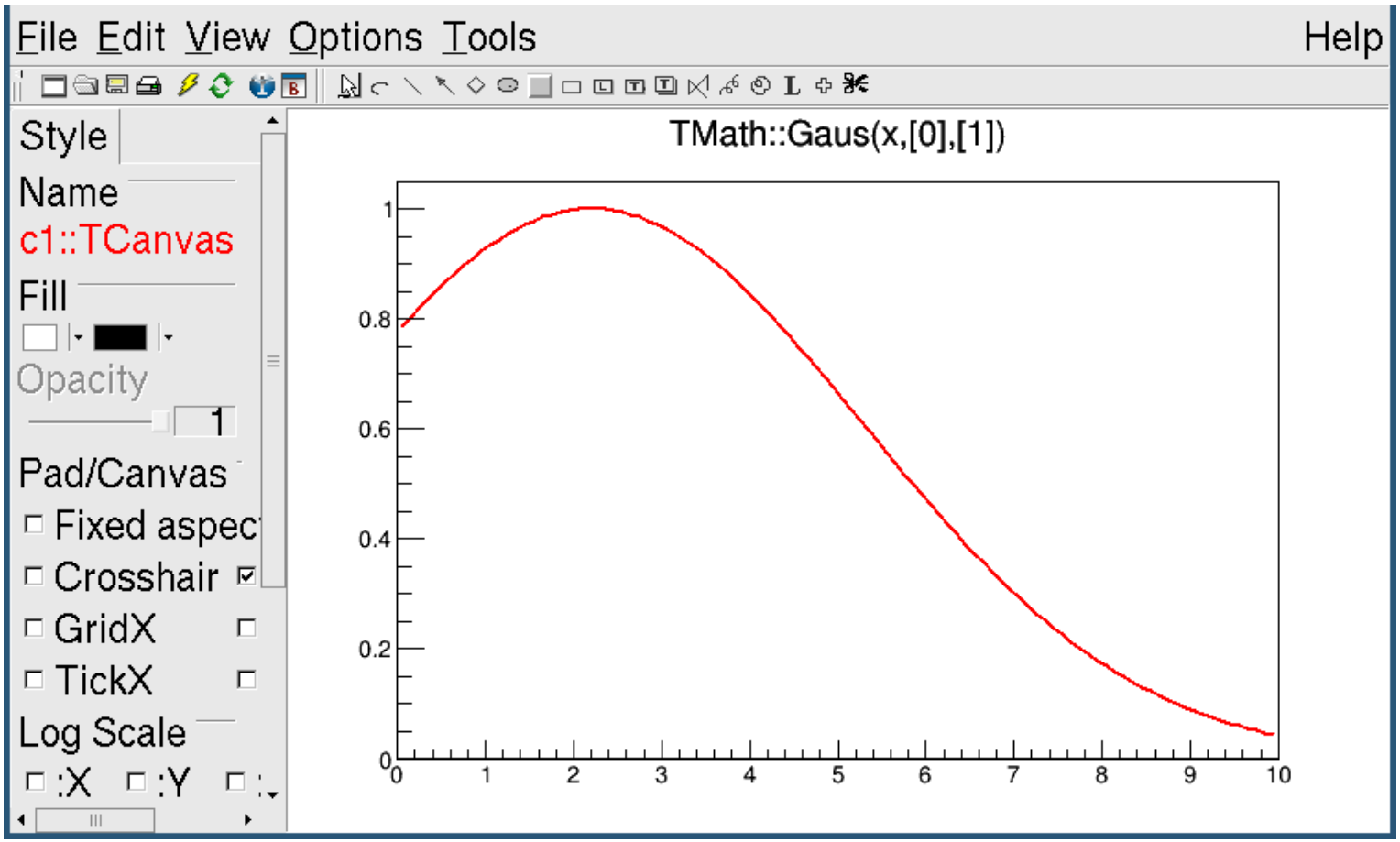
TF1 (3)

```
1  TF1 f1 ("f1", "TMath::Gaus(x,[0],[1])",0,10.);
2  f1.SetParNames("mean", "sigma"); // set names in sequence
3  f1.SetParameter("mean", 2.2); // set  mean value
4  f1.SetParameter("sigma", 3.1); // set sigma value
5  f1.DrawCopy();
```



TF1 (3)

```
1  TF1 f1 ("f1", "TMath::Gaus(x,[0],[1])",0,10.);
2  f1.SetParNames("mean", "sigma"); // set names in sequence
3  f1.SetParameter("mean", 2.2); // set  mean value
4  f1.SetParameter("sigma", 3.1); // set sigma value
5  f1.DrawCopy();
```



TF1 (4)

Generation of Random Numbers:

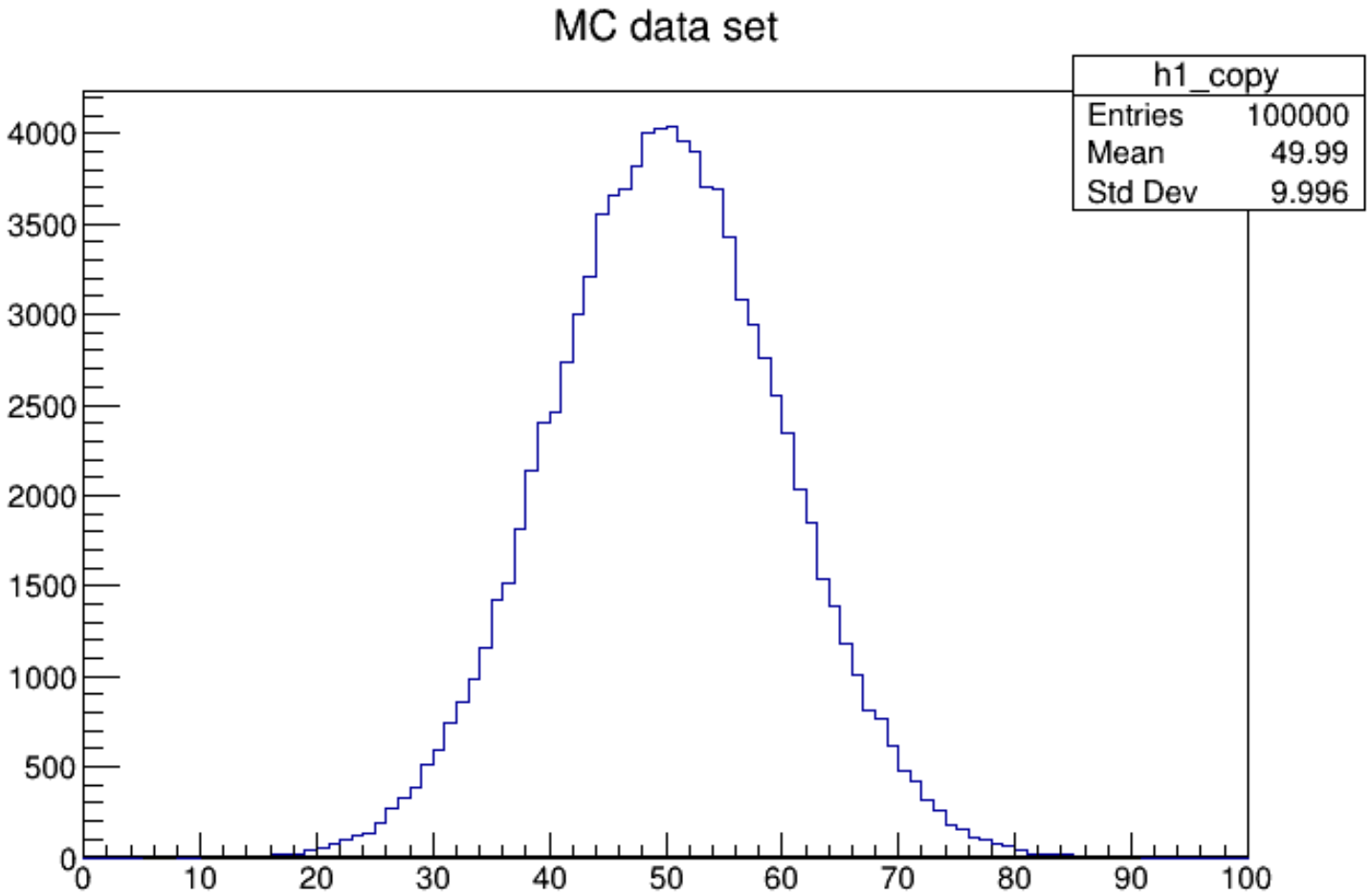
```
root [0] TF1 f1 ("f1", "TMath::Gaus(x,[0],[1])",0,10.);
root [1] f1.SetParNames("mean", "sigma"); // set names in sequence
root [2] f1.SetParameter("mean", 2.2); // set  mean value
root [3] f1.SetParameter("sigma", 3.1); // set sigma value
root [4] f1.GetRandom(2 /*range min*/, 5 /*range max*/)
(Double_t) 2.45505
root [5] f1.GetRandom(2 /*range min*/, 5 /*range max*/)
(Double_t) 2.79242
root [6] f1.GetRandom(2 /*range min*/, 5 /*range max*/)
(Double_t) 4.97270
```

TH1D

```
// Constructor of a TH1D
TH1D::TH1D (const char* name, const char* title,
      Int_t nbinsx, Double_t xlow, Double_t xup);
```

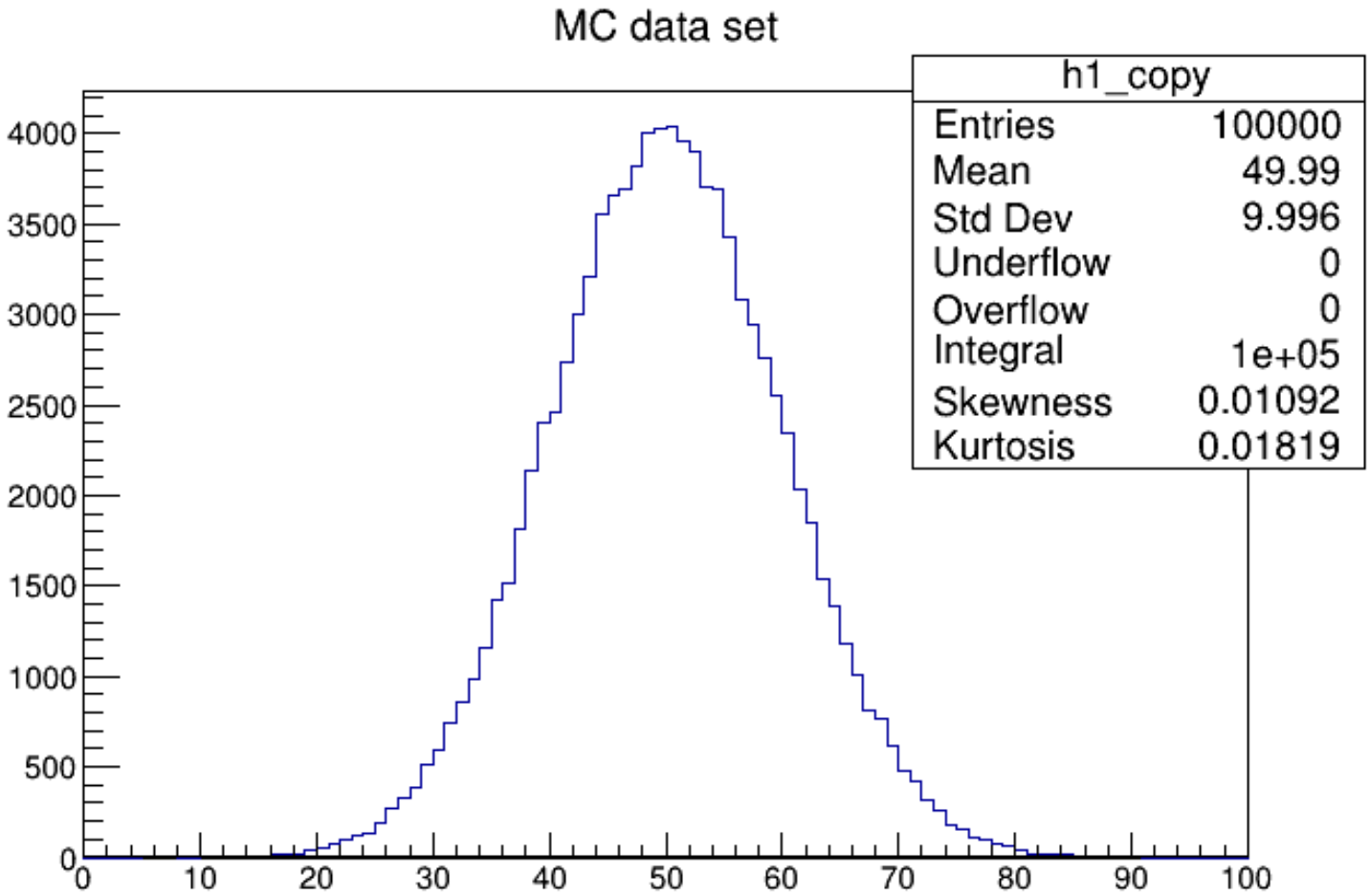
```
// Some useful commands
// cut and paste them into the interpreter!
TH1D h1 ("h1","MC data set",100,0,100);
h1.Fill(4);
h1.Fill(4);
h1.Fill(12);
h1.DrawCopy();

h1.Reset();
h1.DrawCopy();

TF1 f1 ("f1", "TMath::Gaus(x,50,10)",0,100);
for (int i = 0; i < 1e5; i++) h1.Fill(f1.GetRandom(0,100));
h1.DrawCopy();
```

TH1D



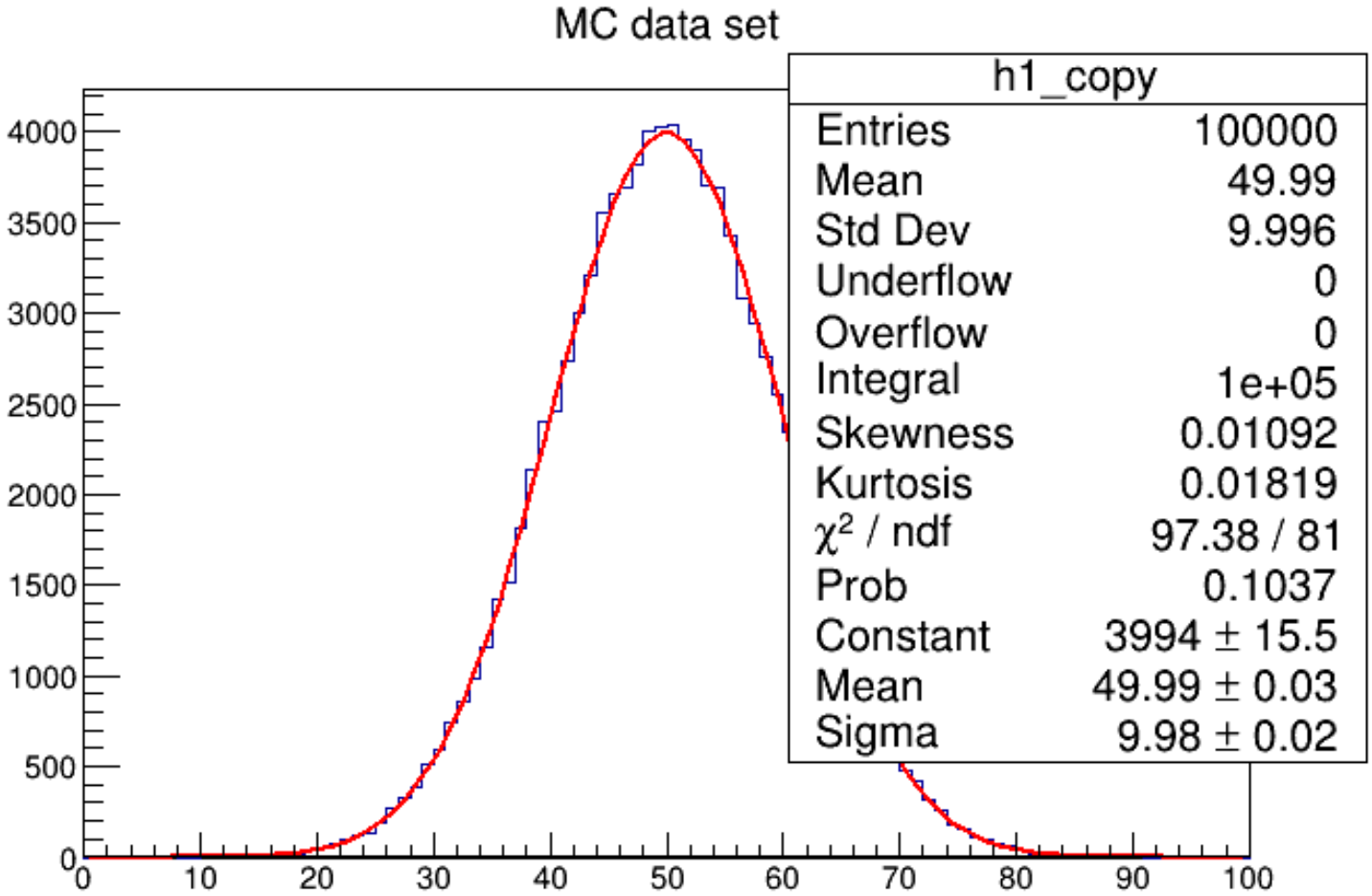

```
root [14] h1.GetMean() // Get mean of the histogram
(Double_t) 49.9902
root [15] h1.GetRMS()  // Get root mean square
(Double_t) 9.99557
```

TH1D




```
root [14] h1.GetMean() // Get mean of the histogram
(Double_t) 49.9902
root [15] h1.GetRMS()  // Get root mean square
(Double_t) 9.99557
```

TH1D




```
root [14] h1.GetMean() // Get mean of the histogram
(Double_t) 49.9902
root [15] h1.GetRMS()  // Get root mean square
(Double_t) 9.99557
```

How to fit? TH1D and TF1

```
1  // This is my histogram
2  TH1D h1 ("h1","MC data set",100,0,100);
3
4  // This is my function to generate the data
5  TF1 f1 ("f1", "TMath::Gaus(x,50,10)",0,100);
6  for (int i = 0; i < 1e5; i++) h1.Fill(f1.GetRandom(0,100));
7
8  // This is my function to fit the data
9  TF1 f2 ("f2", "[0]*TMath::Gaus(x,[1],[2],kTRUE)",0,100);
10 f2.SetParameter( 0, 5e5); // Set parameter starting value
11 f2.SetParLimits( 0, 1e4, 1e6); // Set parameter range
12
13 f2.SetParameter( 1, 10);
14 f2.SetParLimits( 1, 0, 100);
15
16 f2.SetParameter( 2, 5);
17 f2.SetParLimits( 2, 0, 50);
18
19 // finally perform the fit
20 // parse to the histogram the ref to the fitting function
21 h1.Fit(&f2);
```

More details on the fit

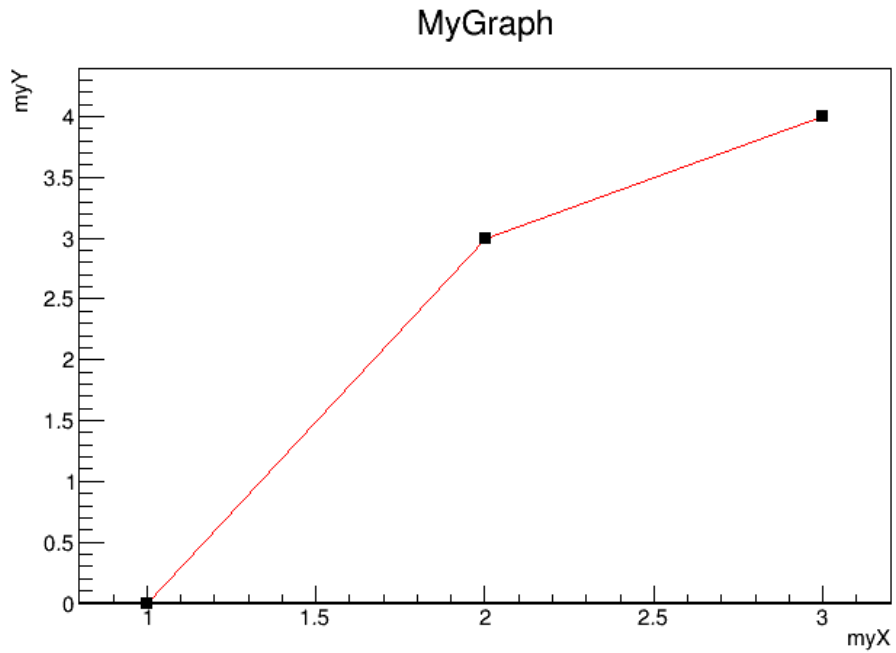
```
1  //
2  // Some of the options ares:
3  //  - Q: quite output
4  //  - E: use minos techniques for error estimation
5  //  - S: return fit results
6  //  - R: fit in range defined by TF1
7  //  - L: use a binned likelihood fit instead of a chi-square
8  //
9
10 TFitResultPtr fitRes = h1.Fit(&f2, "SLR");
11 fitRes->Print();
12
13 cout << fitRes->Parameter(0) << "+-" << fitRes->ParError(0) << endl;
14 cout << fitRes->Parameter(1) << "+-" << fitRes->ParError(1) << endl;
15 cout << fitRes->Parameter(2) << "+-" << fitRes->ParError(2) << endl;
16 cout << "likelihood value " << fitRes->MinFcnValue() << endl;
```

We discuss later why here we ues the symbol "->" instead of "."...

TGraph

```
1  // constractor without data
2  TGraph::TGraph()
3  // constract from existing arrays
4  TGraph::TGraph(Int_t n, const Double_t* x, const Double_t* y)
5  // constract from file
6  TGraph::TGraph(const char* filename, const char* format = "%lg %lg",
7                 Option_t* option = "")
```

```
TGraph g1;
g1.SetTitle("MyGraph; myX; myY");
g1.SetPoint(0,1,0);
g1.SetPoint(1,2,3);
g1.SetPoint(2,3,4);
g1.SetMarkerStyle(kFullSquare);
g1.SetMarkerColor(kBlack);
g1.SetLineColor(kRed);
g1.Draw("APL");
```

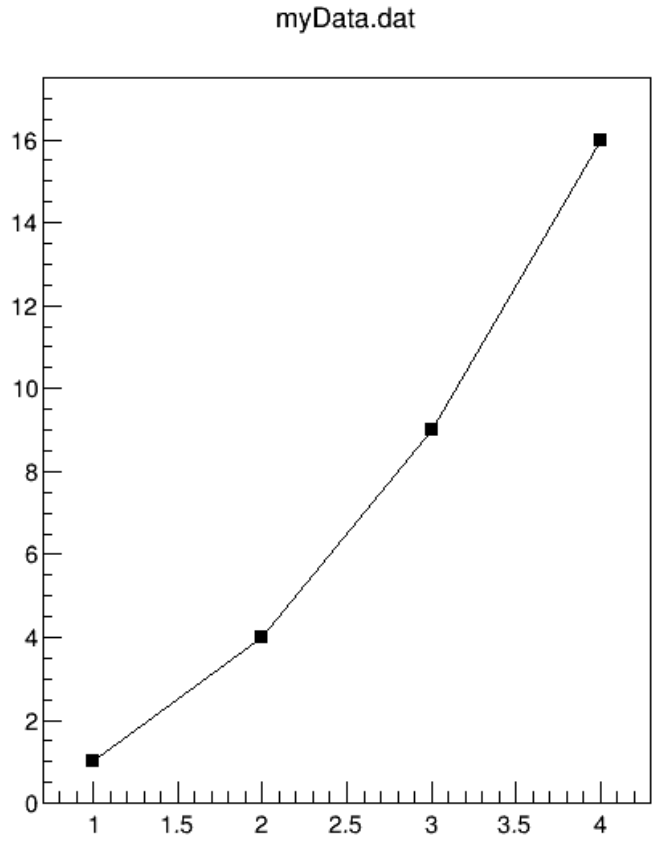


TGraph

```
1  // constractor without data
2  TGraph::TGraph()
3  // constract from existing arrays
4  TGraph::TGraph(Int_t n, const Double_t* x, const Double_t* y)
5  // constract from file
6  TGraph::TGraph(const char* filename, const char* format = "%lg %lg",
7                 Option_t* option = "")
```

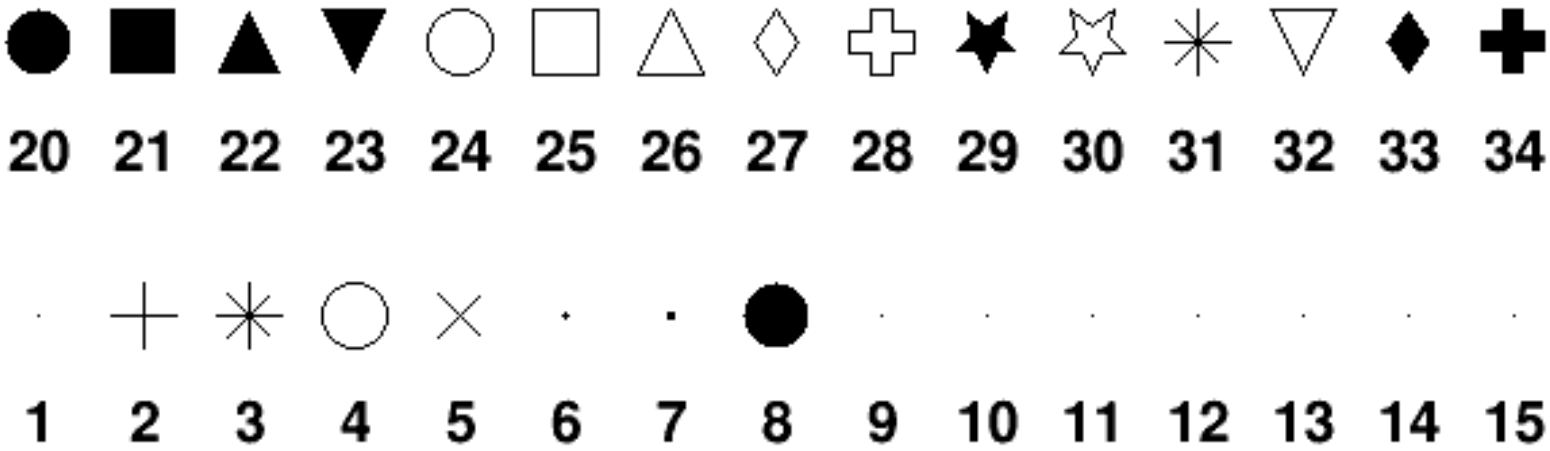
```
~$ cat myData.dat
1   1
2   4
3   9
4  16
```

```
root [0] TGraph g1("myData.dat");
root [1] g1.SetMarkerStyle(kFullSquare);
root [2] g1.Draw("APL");
root [3] // Evaluate the graph for x=1.5
root [4] g1.Eval(1.5)
(Double_t) 2.50000
root [5] // Get access to X of point 2
root [6] g1.GetX()[2]
(Double_t) 3.00000
```



Graphics: the markers

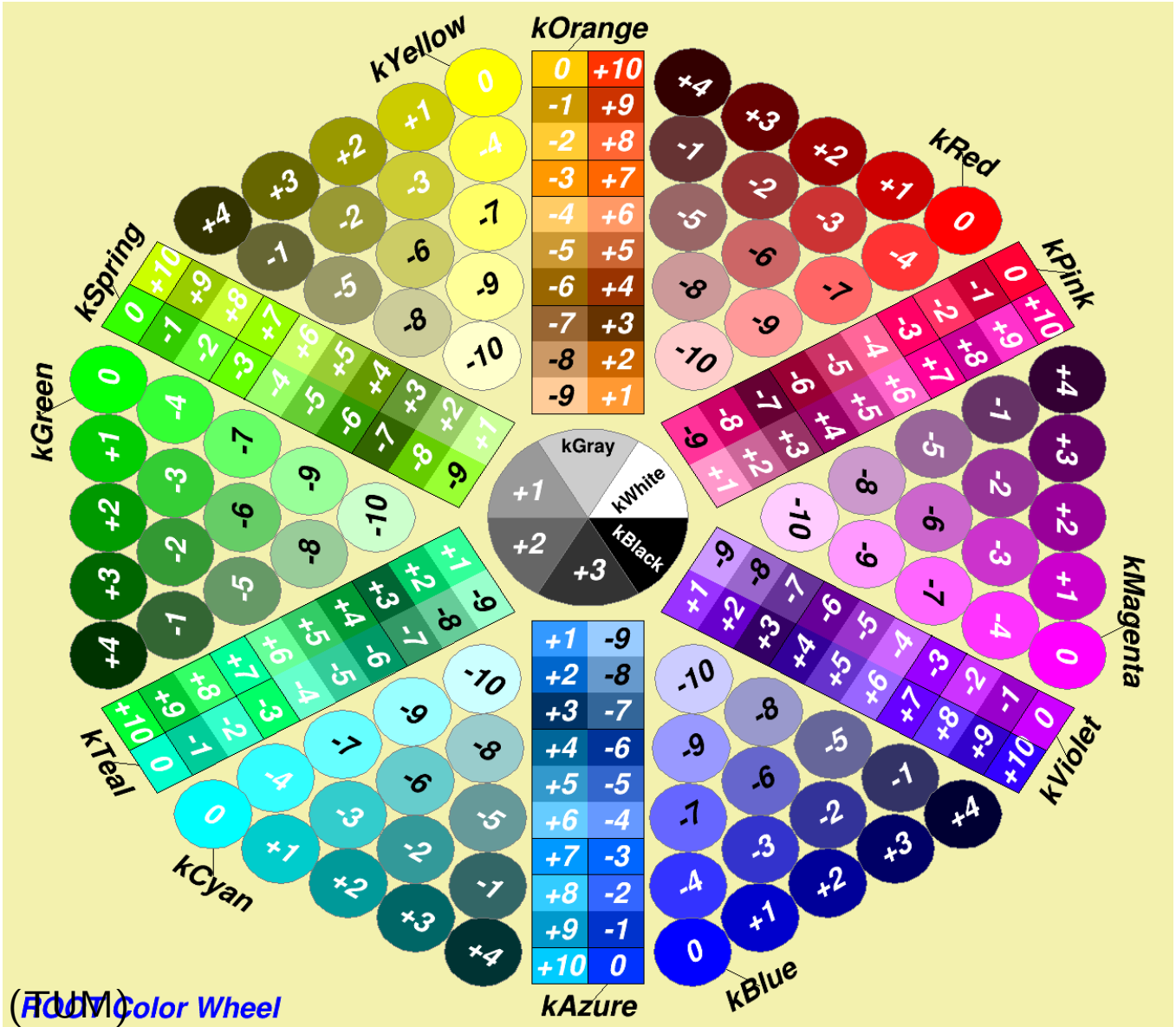
```
g1.SetMarkerStyle(???)
```




```
kDot=1, kPlus, kStar, kCircle=4, kMultiply=5,
kFullDotSmall=6, kFullDotMedium=7, kFullDotLarge=8,
kFullCircle=20, kFullSquare=21, kFullTriangleUp=22,
kFullTriangleDown=23, kOpenCircle=24, kOpenSquare=25,
kOpenTriangleUp=26, kOpenDiamond=27, kOpenCross=28,
kFullStar=29, kOpenStar=30, kOpenTriangleDown=32,
kFullDiamond=33, kFullCross=34
```

Graphics: the colors

`f1.SetLineColor(?); f1.SetFillColor(?); g1.SetMarkerColor(?)`



Macros and compilation

Marcro from file (HelloWorld.C):

```
void HelloWorld () {
   cout << "HelloWorld" << endl;
}
```

```
void HelloWorld (int i) {
   cout << i << endl;
}
```

```
#include <iostream>
using namespace std;

void HelloWorld (int i) {
   cout << i << endl;
}
```

Execute in ROOT

```
matteo ~$ root HelloWorld.C
root [0]
Processing HelloWorld.C...
HelloWorld
```

```
matteo ~$ root HelloWorld.C"(1000)"
root [0]
Processing HelloWorld.C(1000)
1000
```

```
root [0] // load macro
root [1] .L HelloWorld.C
root [2] // compile and load
root [3] .L HelloWorld.C+
root [4] // force compilation and load
root [5] .L HelloWorld.C++
root [6] // compile with optimization
root [7] .L HelloWorld.C+O
root [8] HelloWorld(1000)
```

Stand-alone executable

```
#include <iostream>
using namespace std;

void HelloWorld (int i) {
   cout << i << endl;
}

int main () {
   HelloWorld(1);
   return 0;
}
```

```
~$ g++ -o HelloWorld HelloWorld.C `root-config --cflags --libs`
~$ ./HelloWorld
1
```

Conclusion and exercises

- get used to ROOT6 new functionalities!
- interpreter as a calculator
- new C++ functionalities (auto, ranged-based loops, lambda)
- explore TMath namespace
- Basic containers: TF1, TH1D, TGraph
- explore graphical tools
- macros and compilation

Goal of the day: build a macro for fitting Toy-MC generated data sets:

1) build a TF1 to generate data sets from a simple model (Gaussian signal + flat background)
2) build a TH1D and fill it with a TF1::GetRandom
3) use a second TF1 to fit the histogram
4) Retrieve the results

ROOT coding convention

Classes begin with T	TLine, TTree
Non-class types end with _t	Int_t
Data members begin with f	fTree
Member functions begin with a capital	Loop()
Constants begin with k	kInitialSize, kRed
Global variables begin with g	gEnv
Static data members begin with fg	fgTokenClient
Enumeration types begin with E	EColorLevel
Locals and parameters begin with a lower case	nbytes
Getters and setters begin with Get and Set	SetLast(), GetFirst()

Taligent's conventions: https://root.cern.ch/TaligentDocs/TaligentOnline/DocumentRoot/1.0/Docs/books/WM/WM_63.html

TObject

In ROOT all classes inherit from a common base class called `TObject`

`TObject` provides a common interface (i.e. abstract member functions), for:

- Object I/O: `Read()`,`Write()`
- Error handling: `Warning()`,`Error()`,`SysError()`,`Fatal()`
- Sorting: `IsSortable()`,`Compare()`,`IsEqual()`,`Hash()`
- Inspection: `Dump()`,`Inspect()`
- Printing: `Print()`
- Drawing: `Draw()`,`Paint()`,`ExecuteEvent()`
- Bit handling: `SetBit()`,`TestBit()`
- Memory allocation: operator `new/delete`, `IsOnHeap()`
- Access to meta information: `IsA()`,`InheritsFrom()`
- Object browsing: `Browse()`,`IsFolder()`

Global Pointer Variables

ROOT has a set of global variables that apply to the session and are available in each point of the code and at any time:

- `gROOT`: pointer to the actual running instance of ROOT. It provides access to any named object created in a ROOT program:
  ```
  TObject* obj = gROOT->FindObjectAny("objName");
  ```
  Usefull also to reset the session:
  ```
  gROOT->Reset();
  ```
- `gFile` pointer to the currently opened file
- `gDirectory` pointer to current directory
- `gPad` pointer to the active pad
- `gRandom` pointer to the active random number generator
- `gEnv` pointer to `TEnv` (environment settings for the current session)

Object Ownership

An object has ownership of another object if it must delete it! General rules:

- objects created by the user (histograms, trees,. . .) are owned by current directory (`gDirectory`)
- objects automatically created are typically own by `gROOT`
- objects created by another object (e.g. the TF1 created by the `TH1::Fit("gaus")`) is owned by the histogram
- An object created by `DrawCopy()` is owned by the pad it is drawn in

TFile

The format of file used by ROOT:

```
TFile::TFile (const char *  fname1,       // file name
              Option_t*  option = "",     // new/recreate/update/read
              const char *  ftitle = "", // file ttle
              Int_t  compress = 1);       // compression level from 0 to 9
```

Frequently used methods:

```
TFile inputFile ("myFile", "read/create/new/update");
inputFile.IsOpen();     // check if it is open
inputFile.IsZombie();   // check if it is open
inputFile.cd();         // similarly to TDirectory
inputFile.Close();      // close file
```

Typicall work flow:

```
// writing
TFile iFile ("myFile.root", "new");
TH1D h1 ("h1","myHist",100,0,100);
h1.Write();
iFile.Close();
```

```
// loading
TFile iFile ("myFile.root", "new");
TH1D* h1Ptr = nullptr;
iFile.GetObject("h1", h1Ptr)
if (!h1Ptr)
   cerr << "object not found\n";
```

TFile

How to merge the content of two TFiles containing objects with different names? How to merge two trees residing into two files?

```
~$ hadd outputFile.root inputFile1.root inputFile2.root

~$ hadd --help
Usage: hadd [-f[fk][0-9]] [-k] [-T] [-O] [-a]
            [-n maxopenedfiles] [-cachesize size] [-v [verbosity]]
            targetfile source1 [source2 source3 ...]

This program will add histograms from a list of root files and write them
to a target root file. The target file is newly created and must not
exist, or if -f ("force") is given, must not be one of the source files.
Supply at least two source files for this to make sense... ;-)
[...]
```

TDirectory

A TFile is formally a TDirectory:

```
root [0] gDirectory->pwd()
Rint:/
root [1] TFile myFile ("myFile.root","new");
root [2] gDirectory->pwd()
myFile.root:/
root [3] myFile.Close();
root [4] gDirectory->pwd()
Rint:/

root [0] TH1D h1 ("h1","h1",100,0,100);
root [1] .ls
 OBJ: TH1D      h1      h1 : 0 at: 0x7f89db849028
root [2] TFile myFile ("myFile.root","new");
root [3] TH1D h2 ("h2","h2",100,0,100);
root [4] .ls
TFile**         myFile.root
 TFile*         myFile.root
  OBJ: TH1D     h2      h2 : 0 at: 0x7f89db849708
root [5] myFile.Close()
root [6] .ls
 OBJ: TH1D      h1      h1 : 0 at: 0x7f89db849028
```

TTree

- the data structures provided by ROOT to store large quantities of data
- it store informations in a table-wise format: different fields as columns, different events as lines
- each column is a branch
- fields can be elementary variables or complex objects

```
1  // Constructor:
2  TTree::TTree (const char * name, const char *  title);
3
4  // Fill tree from ascii file
5  TTree t ("t","myTree");
6  // The numerical format of each column in the file must be
7  // specified, e.g. D->double, I->Integer, C->Char
8  t.ReadFile ("myAsciiFile.txt", "col1/D:col2/I:col3/C");
```

TTree: writing

```
1  TFile ofile("outputFile.root","recreate");
2  TTree t("t", "myTree");
3
4  double x; int y;
5  t.Branch("x", &x);
6  t.Branch("y", &y);
7
8  for (int i = 0; i < 100 ; i++){
9     cout << i << endl;
10    x = gRandom->Gaus();
11    y = gRandom->Poisson(1);
12    t.Fill();
13 }
14
15 t.Write();
16 ofile.Close();
```

TTree: interactive usage

```
~$ root outputFile.root
root [0]
Attaching file outputFile.root as _file0...
(TFile *) 0x20ed1b0
root [1] t->Show(0)
======> EVENT:0
 x               = 0.998933
 y               = 0
root [2] t->Scan("x:y")
************************************
*    Row   *         x *         y *
************************************
*        0 * 0.9989327 *         0 *
*        1 * 1.1055978 *         0 *
*        2 * -3.126301 *         2 *
*        3 * -0.410763 *         0 *
*        4 * 0.1916502 *         1 *
[...]
```

TTree: drawing

```
Long64_t Draw(const char* varexp,                // what do draw, e.g. "x:y" or "x"
              const TCut& selection,             // which cut to apply, e.g. "x>0"
              Option_t* option = "",             // options as in other objects
              Long64_t nentries = kMaxEntries, // how many entries to read
              Long64_t firstentry = 0);          // read starting from this entry
```

```
root [0] TFile iFile("outputFile.root");
root [1] t->Draw("x:y","y>1","colz");                  // draw x vs y
root [2] t->Draw("x>>(1000)");                         // specify hist binning
root [3] t->Draw("x>>(1000,0,100)");                   // specify hist range and binning
root [4] t->Draw("x>>hh (100,0,100)","", "goff"); // create hh and do not plot it
root [5] .ls
TFile**         outputFile.root
TFile*          outputFile.root
OBJ: TTree      t       myTree : 0 at: 0x1cd36b0
OBJ: TH1F       hh      x : 0 at: 0x2a6c2a0
root [6] hh->Draw();
root [7] t->Draw("x>>+hh"); // add more to an existing hist
```

TTree: advanced analysis

```
// new looping method based on TTreeReader
// useful when the branch contain complicated objects
TFile iFile ("outputFile.root");
TTree* tPtr = nullptr;  iFile.GetObject("t", tPtr);

TTreeReader reader (tPtr);
TTreeReaderValue<double> xPtr (reader, "x");
TTreeReaderValue<int>    yPtr (reader, "y");
while(reader.Next()) {
   const double x = *xPtr;
   const int    y = *yPtr;
   cout << x <<  "  " << y << endl;
}
```

```
// Old but still supported  solution
TFile iFile ("outputFile.root");
TTree* tPtr = nullptr; iFile.GetObject("t", tPtr);

double x; int y;
tPtr->SetBranchAddress("x",&x); // note the &
tPtr->SetBranchAddress("y",&y);

for (int i = 0; i < tPtr->GetEntries(); i++) {
   tPtr->GetEntry(i); // load entry by hand
   cout << x <<  "  " << y << endl;
}
```

TNtuple

A TTree containing only float values, very handy for a quick analyis. No need to connect branches:

```
1  TFile oFile ("myFile.root","recreate");
2  // parse branch names directly from the constructor
3  TNtuple t ("t","t","minuitStatus:N1:N2:N3");
4
5  for (..) {
6     // fill like an histogram
7     t.Fill(/*float*/, /*float*/, /*float*/);
8  }
9
10 t.Write();
11 oFile.Close();
```