Outline 2017/04/07

- ROOT – “a gentle introduction”
- Histograms, TH1 class
- Data structure, ROOT files
- TTree (TBranch) and TFile classes
- Hands on software
  - try the four ways of using ROOT
  - understand how to use TH1
  - *change the script to get a new histogram*
  - *create a tree*
  - *read a tree*
  - *debug, add variables and create/read a new tree*

Communications et al.

April 2017

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
						1 April Fool's Day
2	3	4	5	6	7 OK	8
9 Palm Sunday	10 Passover	11	12	13	14 NO!	15
16 Easter Sunday	17	18	19	20	21 OK	22 Earth Day
23 30	24	25	26	27	28 OK	29

Debugging

First and more important rule of all:

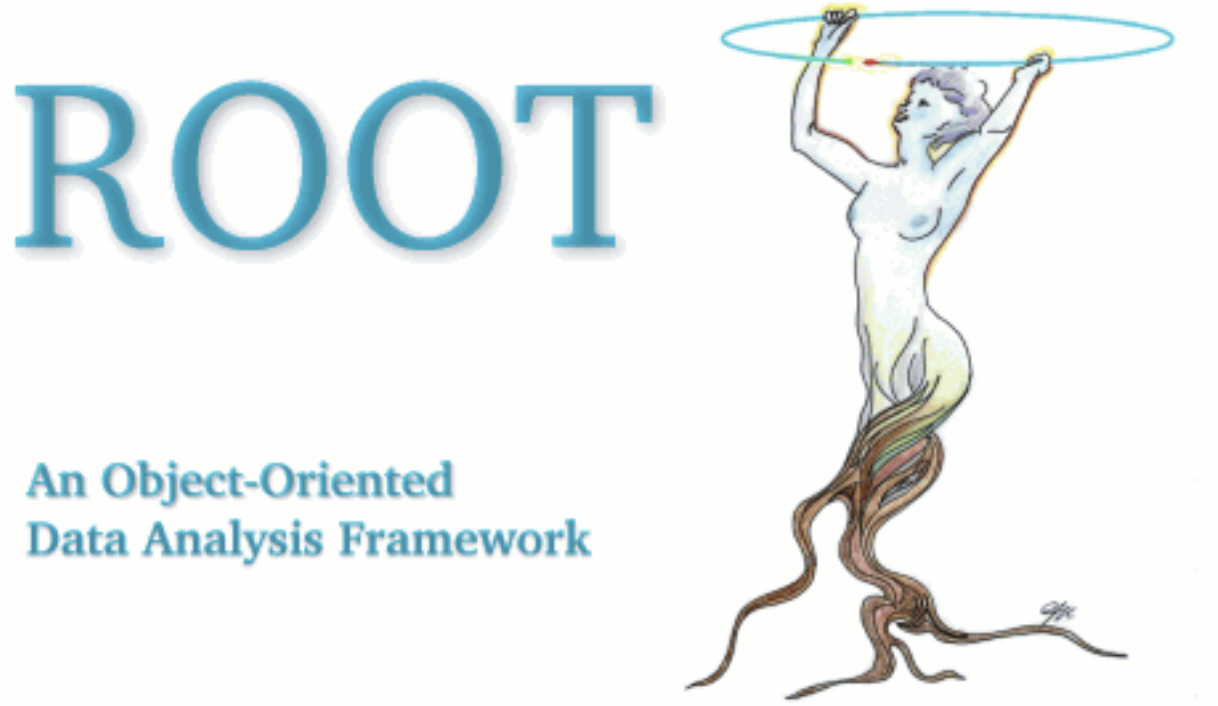
READ CAREFULLY

WARNINGS AND ERRORS!

```
g++ -Wall Ex1.cpp -o Ex1
Ex1.cpp: In function “int main()”:
Ex1.cpp:30:4: error: expected “,” or “;” before “Float_t”
Ex1.cpp:36:26: error: “qt” was not declared in this scope
```

ROOT

ROOT is a C++ framework – a set of classes and executables – built by physicists for physicists.



http://root.cern.ch

ROOT, documentation

ROOT is a C++ framework – a set of classes and executables – built by physicists for physicists.

http://root.cern.ch

http://root.cern.ch/drupal/content/users-guide

http://root.cern.ch/drupal/content/reference-guide

ROOT

- **Show results**. Results are best shown with histograms, scatter plots, fitting functions, etc. ROOT graphics may be adjusted real-time by few mouse clicks. High-quality plots can be saved in PDF or other format.
- **Interactive or built application**. You can use the CINT C++ interpreter or Python for your interactive sessions and to write macros, or compile your program to run at full speed. In both cases, you can also create a GUI.

ROOT

- **Save data**. You can save your data (and any C++ object) in a compressed binary form in a ROOT file. The object format is also saved in the same file. ROOT provides a data structure that is extremely powerful for fast access of huge amounts of data - orders of magnitude faster than any database.
- **Access data**. Data saved into one or several ROOT files can be accessed from your PC, from the web and from large-scale file delivery systems used e.g. in the GRID. ROOT trees spread over several files can be chained and accessed as a unique object, allowing for loops over huge amounts of data.
- **Process data**. Powerful mathematical and statistical tools are provided to operate on your data. The full power of a C++ application and of parallel processing is available for any kind of data manipulation. Data can also be generated following any statistical distribution, making it possible to simulate complex systems.

How to use ROOT

1. as classes in compiled programs
2. using the ROOT command line interface and C++ interpreter called “CINT” (since version 6.x it has been replaced by CLING/CLANG)
3. using *scripts*, files (usually with extension .C) containing list of command for CINT that can be loaded and run
4. compiling shared libraries out of your scripts using CINT (can be useful, not to be used as a standard way of working)

How to use ROOT

1. as classes in compiled programs:

```
#include <TH1D.h>

int main(){
 ...

 TH1D *h = new TH1D(“h”,“”,1000,0.,100.);
 h->Draw();
 etc. etc. ...
 return 0;
}
```

How to use ROOT

2. using the ROOT command line interface and C++ interpreter called “CINT” (soon it will be replace by CLING/CLANG)

```
prompt> root
root[0] TH1D *h = new TH1D(“h”,“”,1000,0.,100.)
root[1] h->Draw()
...
root[ ] .q
prompt>
```

quit root!

How to use ROOT

3. using *scripts*, files (usually with extension .C) containing list of command for CINT that can be loaded and run

file histo.C

```
void histo(){
 TH1D *h = new TH1D("h","",1000,0.,100.);
 h->Draw();
}
```

```
prompt> root
root[0] .L histo.C
root[1] histo()
```

How to use ROOT

4. compiling shared libraries out of your scripts using CINT (can be useful, not to be used as a standard way of working)

file histo2.C

```
#include <TH1D.h>
void histo(){
 TH1D *h = new TH1D("h","",1000,0.,100.);
 h->Draw();
}
```

```
prompt> root
root[0] .L histo.C++
root[2] histo()
root[1] .q
prompt> root
root[0] gSystem->Load("histo_C.so");
root[1] histo()
```

CINT, why?

Though compiled applications run much faster, interpreted scripts are easier to create and debug. This is why data analysis programs usually offer an interactive shell to the user.

Thanks to the C++ interpreter CINT, the ROOT framework provides a way to incrementally develop and debug your program and, at the same time, makes it smooth to transform your work into a compiled application.

CINT, why?

In addition, the user can choose to build a stand-alone application or a (static or, by default, shared) library.

In the latter case, the user can load such library from the ROOT shell (i.e. via the CINT interpreter), from a ROOT macro (that is a sequence of C++ statements to be executed step by step by the interpreter), or from a compiled application.

WARNING!! CINT IS _NOT_ C++

- the declaration of the type may be omitted when **“new”** is used:

```
root [0]  h = new TH1D(“h”,“”,1000,0.1,1.)
```

- we can access heap and stack objects with the same sintax:

```
root [1] h.Draw();
root [2] h->Draw();
```

- In case CINT cannot find an object being referenced, it will ask ROOT to search for an object with an identical <u>root name</u>;

WARNING!! CINT IS _NOT_ C++

- There is no need to put a semicolon at the end of a line. The difference between having it and leaving it off is that when you leave it off the return value of the command will be printed on the next line:

```
root [0] 2+3
(int)5
root [1] 2+3;
root [2]
```

- Incomplete data hiding, variables not removed from the stack when closing a block (!)

ROOT, basics

1. (almost) all ROOT class names start with “T”, for example: TObject, TFile, TGraph, ...
2. all private variable member names start with “f”, e.g. fPoints, fEntries, ... (we used “_” instead)
3. all global variable member names start with “g”, e.g. gStyle, gROOT, ...
4. all variable types have been re-defined in order to have the same bit lenght in any architecture and compiler

ROOT, basics

All variable types have been re-defined in order to have the same bit lenght in any architecture and compiler:

ROOT		C++
Int_t	↔	int
Float_t	↔	float
Double_t	↔	double
UInt_t	↔	unsigned int
...		...

http://root.cern.ch/root/html/ListOfTypes.html

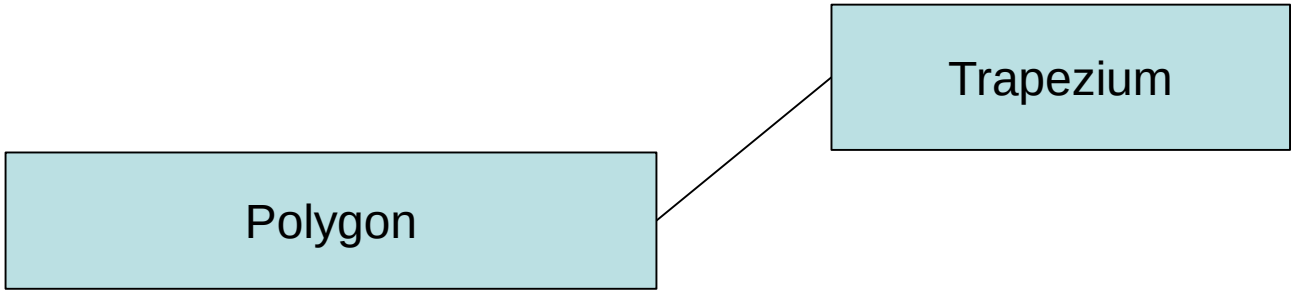
ROOT, TObject

In ROOT, almost all classes inherit from a common base class called TObject. The TObject class provides default behavior and protocol for all objects in the ROOT system. TObject provides protocol, i.e. (abstract) member functions, for:

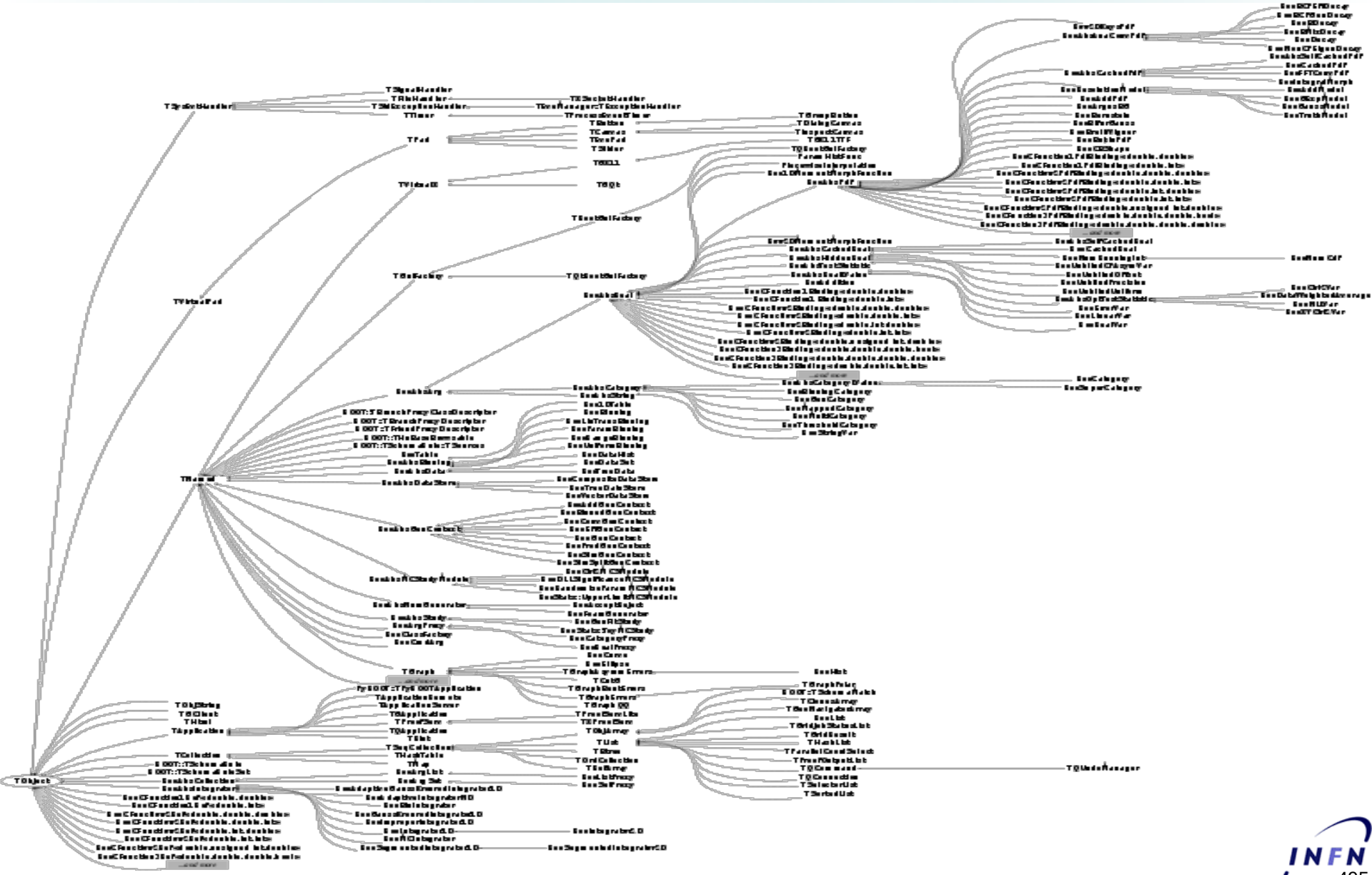
- Object I/O (Read(), Write())
- Error handling (Warning(), Error(), SysError(), Fatal())
- Sorting (IsSortable(), Compare(), IsEqual(), Hash())
- Inspection (Dump(), Inspect())
- Printing (Print())
- Drawing (Draw(), Paint(), ExecuteEvent())
- Bit handling (SetBit(), TestBit())
- Memory allocation (operator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

ROOT, hierarchy tree from TObject

Our example:



ROOT, hierarchy tree from TObject



TH1D class

- http://root.cern.ch/root/html532/TH1D.html
- [show by opening browser...]

In ROOT, the histogram classes are named THdp where d(dimension) can be 1, 2 or 3. The precision p can be C (for Char), S (for Short), F (for Float) or D (for Double).

TH1D is a class for 1-dimensional histograms where a Double_t is used to increment the sum of weights for each channel.

TH1D class, creating a histogram

A histogram is created by invoking one of the class constructors. For example:

```
TH1D::TH1D(TString,TString,Int_t,Float_t,Float_t);

TH1D *h = new TH1D("ROOTname","histogram title",nbins,xlow,xup);

TH1D *s1 = new TH1D("s1","signal",100,-6.,6.);
```

s1 is a histogram with range from -6 to 6 divided into 100 bins – each with dimension

[6-(-6)]/100 = 12 /100 = 0.12. Each bin can be filled with Double_t values as many times we need.

ROOT: memory handling

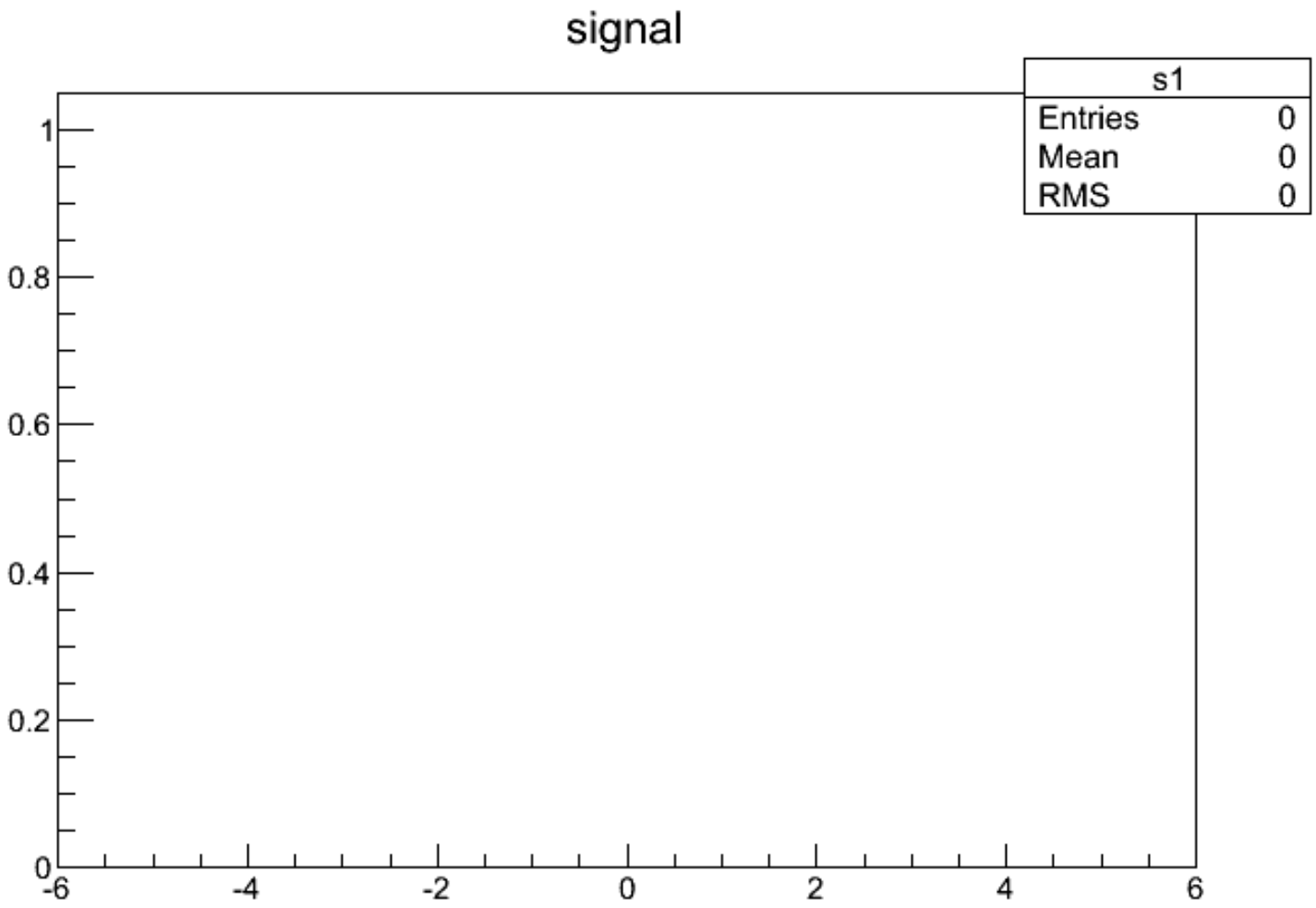
Almost all ROOT objects have an “internal” ROOT name.

ROOT maintains a list of its objects and:

1) objects are never lost if you remember their name or type (i.e. if you go out of scope and loose a pointer you can always search for the object in ROOT memory and recover the pointer);
2) objects are hierarchically stored (one can belong to another one)
3) ROOT sometimes deletes objects by its own (without warnings)!

TH1D class, creating a histogram

```
TH1D *s1 = new TH1D("s1","signal",100,-6.,6.);
```



TH1D class, filling a histogram

Histogram of all types are filled via the `hist->Fill(...)` functions:

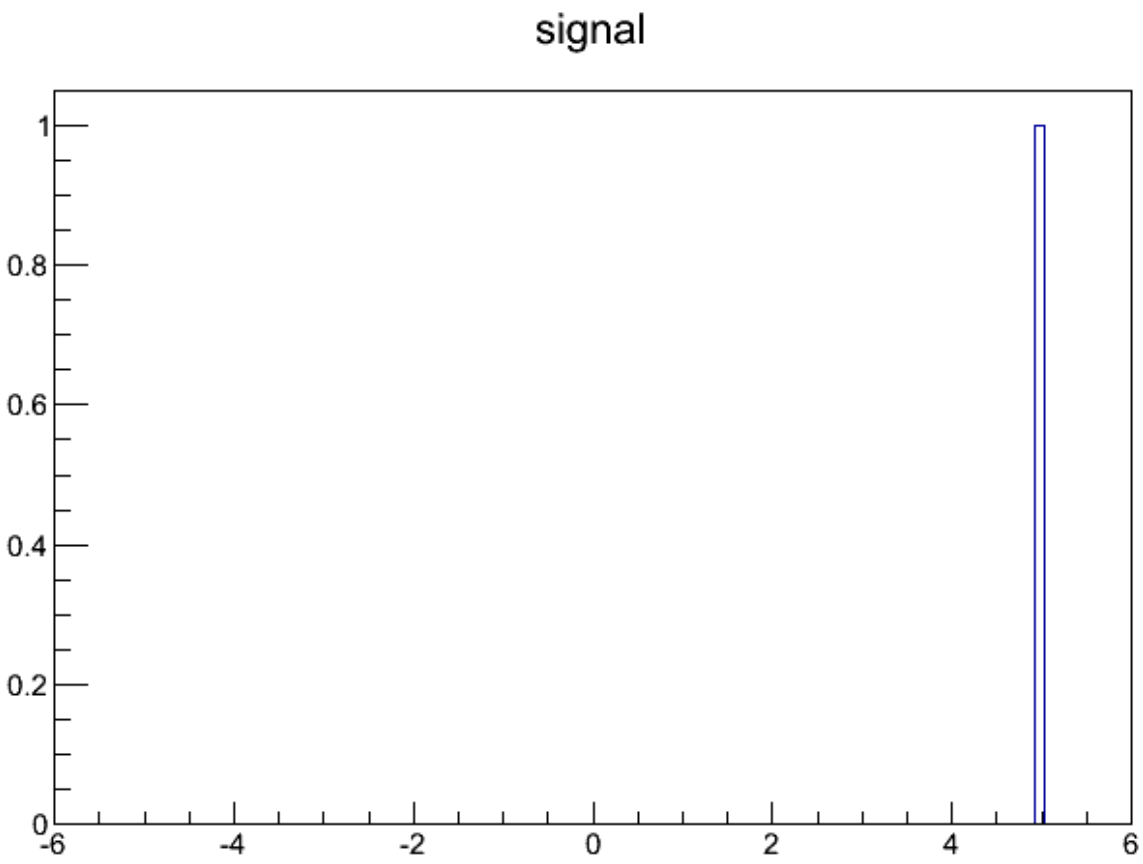
`TH1::Fill(x)` increment by `1` the bin corresponding to `x`.

`TH1::Fill(x,w)` increment by `w` the bin corresponding to `x`.

```
TH1D *s1 = new TH1D("s1","signal",100,-6.,6.);
s1->Fill(5.);
s1->Fill(-5.,666.);
```

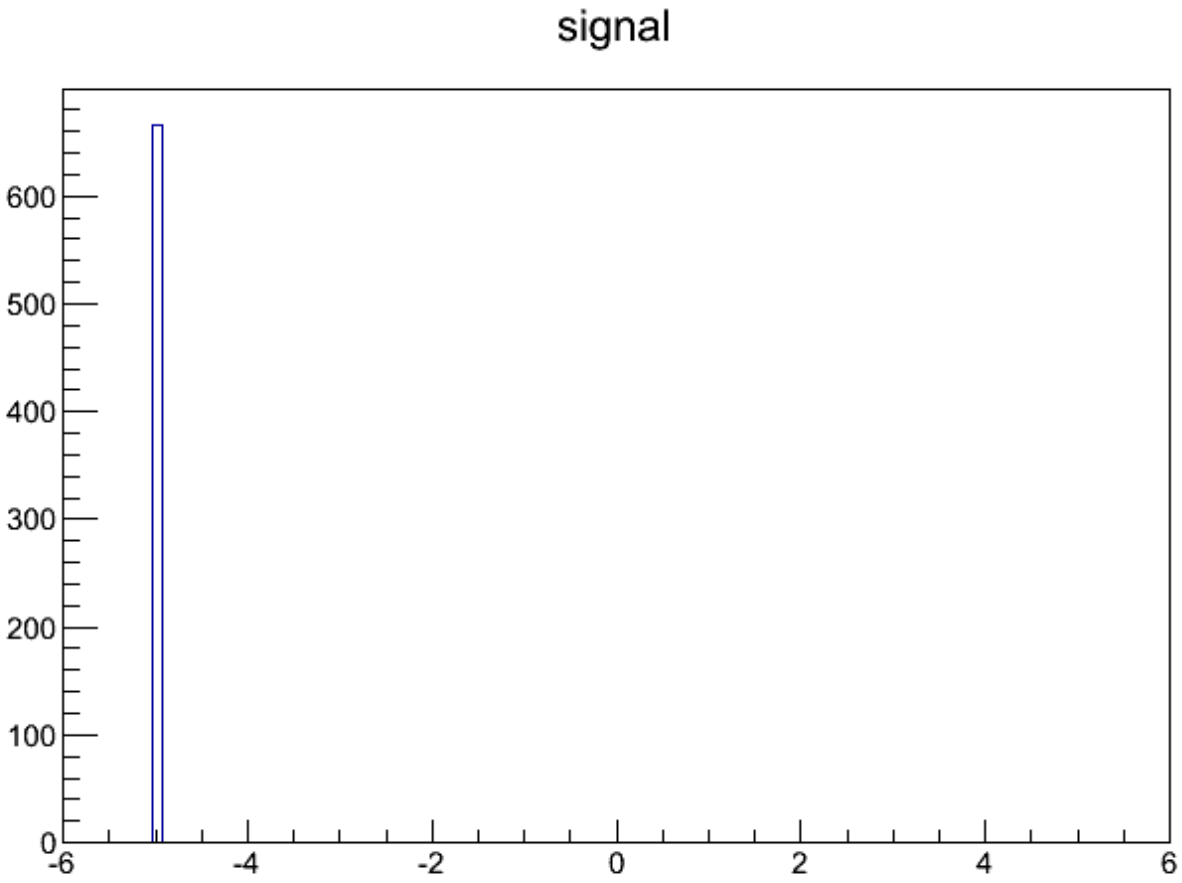
TH1D class, filling a histogram

```
TH1D *s1 = new TH1D("s1","signal",100,-6.,6.);
s1->Fill(5.);
```

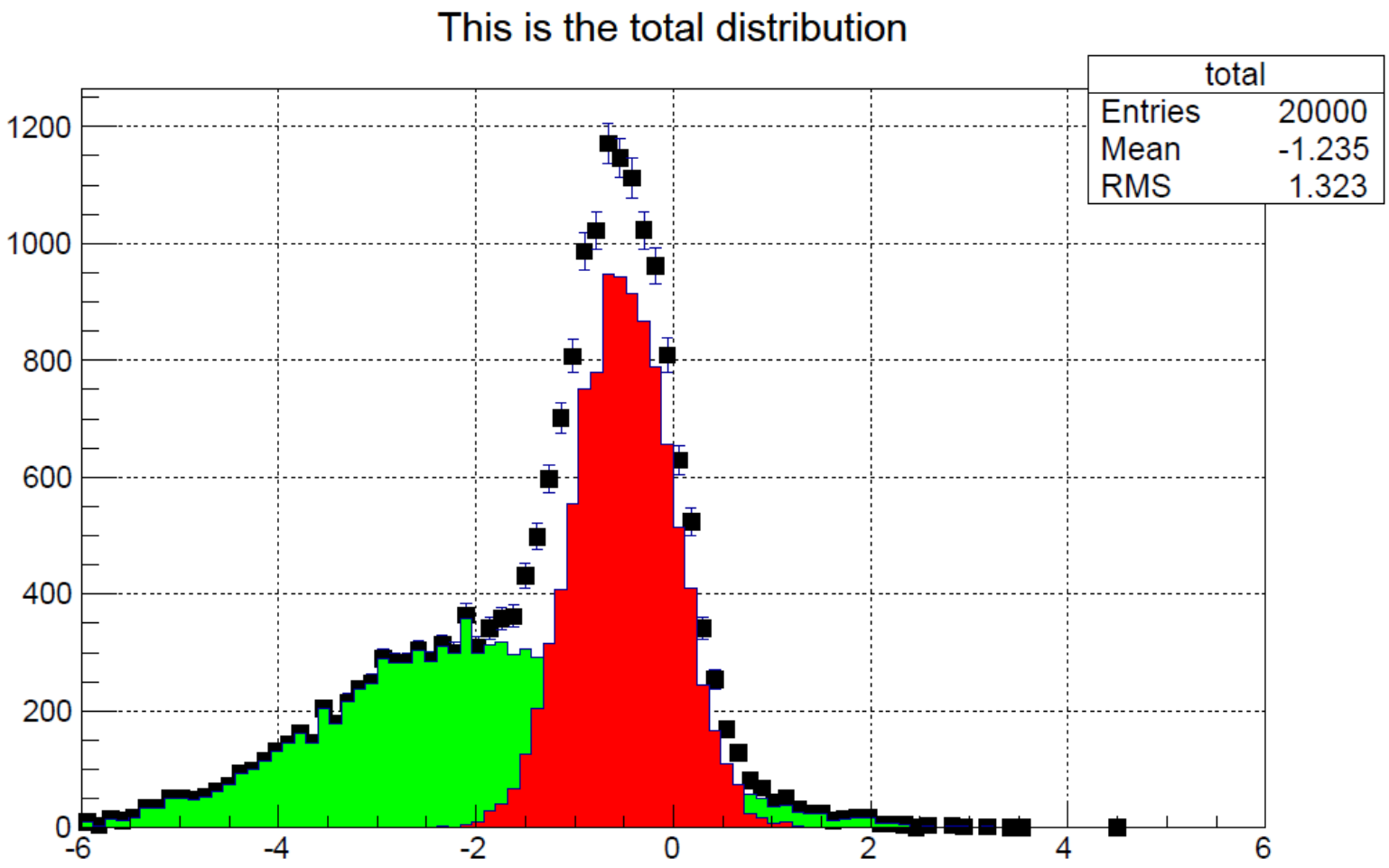


TH1D class, filling a histogram

```
TH1D *s1 = new TH1D("s1","signal",100,-6.,6.);
s1->Fill(5.);
s1->Fill(-5.,666.);
```



TH1D example: sum of histograms



TH1D example: sum of histograms, compiled

emacs@marte.infis.univ.trieste.it

File Edit Options Buffers Tools C++ Help

```
#include <TCanvas.h>
#include <TH1D.h>
#include <TRandom3.h>

void histosum(){

  TCanvas *cl = new TCanvas("cl","The histosum example",200,10,600,400);
  cl->SetGrid();
  cl->Draw();

  // Create some histograms.
  TH1D *total = new TH1D("total","This is the total distribution",100,-6.,6.);
  TH1D *main = new TH1D("main","Main contributor",100,-6.,6.);
  TH1D *s1 = new TH1D("s1","This is the first signal",100,-6.,6.);

  total->SetMarkerStyle(21);
  total->SetMarkerSize(0.8);
  main->SetFillColor(kGreen);
  s1->SetFillColor(kRed);

  // Fill histograms randomly
  TRandom3 *random= new TRandom3();

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  for ( Int_t i=0; i<10000; i++) {

    xmain = random->Gaus(-2,1.5);
    xs1 = random->Gaus(-0.5,0.5);

    main->Fill(xmain);
    s1->Fill(xs1);

    total->Fill(xmain);
    total->Fill(xs1);

  }

  total->Draw("elp");
  main->Draw("same");
  s1->Draw("same");
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
  cl->SaveAs("fighistosumMain.pdf");

  delete random;
  delete s1;
  delete main;
  delete total;
  delete cl;

}

int main(){
  histosum();
  return 0;
}
```

--:--- histosumMain.cpp All L47 (C++/l Abbrev)----

Beginning of buffer

TH1D example: sum of histograms, compiled

Compile with:

```
g++ -Wall histosumMain.cpp `root-config --cflags --ldflags --libs` -I`root-config --incdir`
    –o histosumMain
```

where:

```
prompt> root-config --cflags --ldflags --libs
-pthread -m32 -I/opt/root/include -m32 -L/opt/root/lib -lCore -lCint -lRIO -lNet -lHist -
    lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lMathCore -lThread
    -pthread -lm -ldl -rdynamic

> root-config --incdir
/opt/root/include
```

so it means:

```
g++ -Wall histosumMain.cpp -pthread -m32 -I/opt/root/include -m32 -L/opt/root/lib -lCore -
    lCint -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -
    lPhysics -lMathCore -lThread -pthread -lm -ldl –rdynamic -I/opt/root/include –o
    histosumMain
```

TH1D example: sum of histograms, compiled

Compile with:

```
g++ -Wall histosumMain.cpp `root-config --cflags --ldflags --libs` -I`root-config --incdir`
    -o histosumMain
```

execute the command in between before processing the full command

where:

```
prompt> root-config --cflags --ldflags --libs
-pthread -m32 -I/opt/root/include -m32 -L/opt/root/lib -lCore -lCint -lRIO -lNet -lHist -
    lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lMathCore -lThread
    -pthread -lm -ldl -rdynamic

> root-config --incdir
/opt/root/include
```

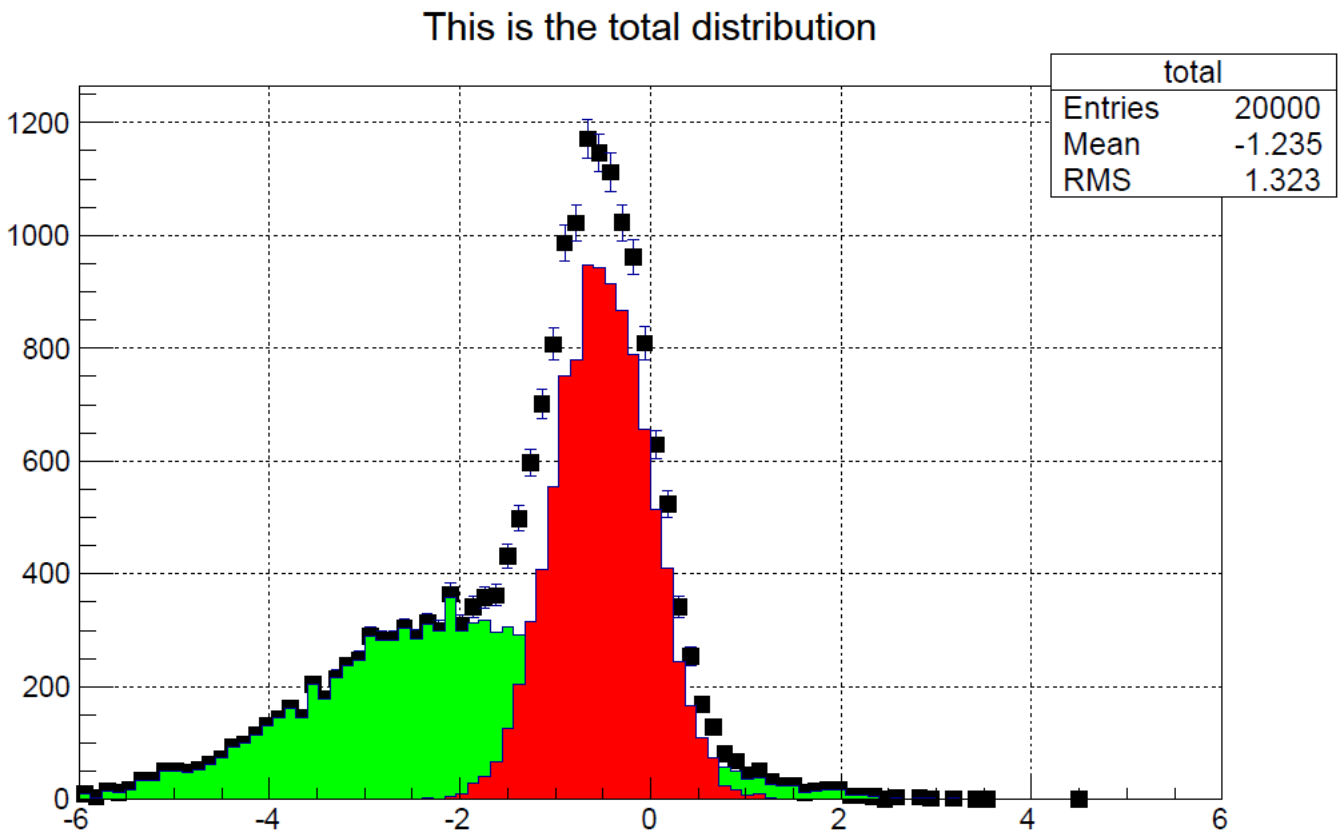
so it means:

```
g++ -Wall histosumMain.cpp -pthread -m32 -I/opt/root/include -m32 -L/opt/root/lib -lCore -
    lCint -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -
    lPhysics -lMathCore -lThread -pthread -lm -ldl –rdynamic -I/opt/root/include –o
    histosumMain
```

TH1D example: sum of histograms, compiled

Run with:

```
> export LD_LIBRARY_PATH=/opt/root/lib:$LD_LIBRARY_PATH
>./histosumMain
Info in <TCanvas::Print>: pdf file fighistosumMain.pdf has been created
> acroread fighistosumMain.pdf
```



TH1D example: sum of histograms, from command line

```
> root
root [0] TCanvas *c1 = new TCanvas("c1","The histosum
   example",200,10,600,400)
root [1] c1->SetGrid()
root [2] c1->Draw()
root [3] TH1D *total = new TH1D("total","This is the total
   distribution",100,-6.,6.)
...
```

TH1D example: sum of histograms, compiled script

emacs@marte.infis.univ.trieste.it

File Edit Options Buffers Tools C++ Help

```
#include <TCanvas.h>
#include <TH1D.h>
#include <TRandom3.h>

void histosum(){

  TCanvas *c1 = new TCanvas("c1","The histosum example",200,10,600,400);
  c1->SetGrid();
  c1->Draw();

  // Create some histograms.
  TH1D *total = new TH1D("total","This is the total distribution",100,-6.,6.);
  TH1D *main = new TH1D("main","Main contributor",100,-6.,6.);
  TH1D *s1 = new TH1D("s1","This is the first signal",100,-6.,6.);

  total->SetMarkerStyle(21);
  total->SetMarkerSize(0.8);
  main->SetFillColor(kGreen);
  s1->SetFillColor(kRed);

  // Fill histograms randomly
  TRandom3 *random= new TRandom3();

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  for ( Int_t i=0; i<10000; i++) {

    xmain = random->Gaus(-2,1.5);
    xs1 = random->Gaus(-0.5,0.5);

    main->Fill(xmain);
    s1->Fill(xs1);

    total->Fill(xmain);
    total->Fill(xs1);

  }

  total->Draw("elp");
  main->Draw("same");
  s1->Draw("same");
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
  c1->SaveAs("fighistosumMain.pdf");

}
```

--:--- histosumCompiled.C All L43 (C++/l Abbrev)

TH1D example: sum of histograms, compiled script

emacs@marte.infis.univ.trieste.it

File Edit Options Buffers Tools C++ Help

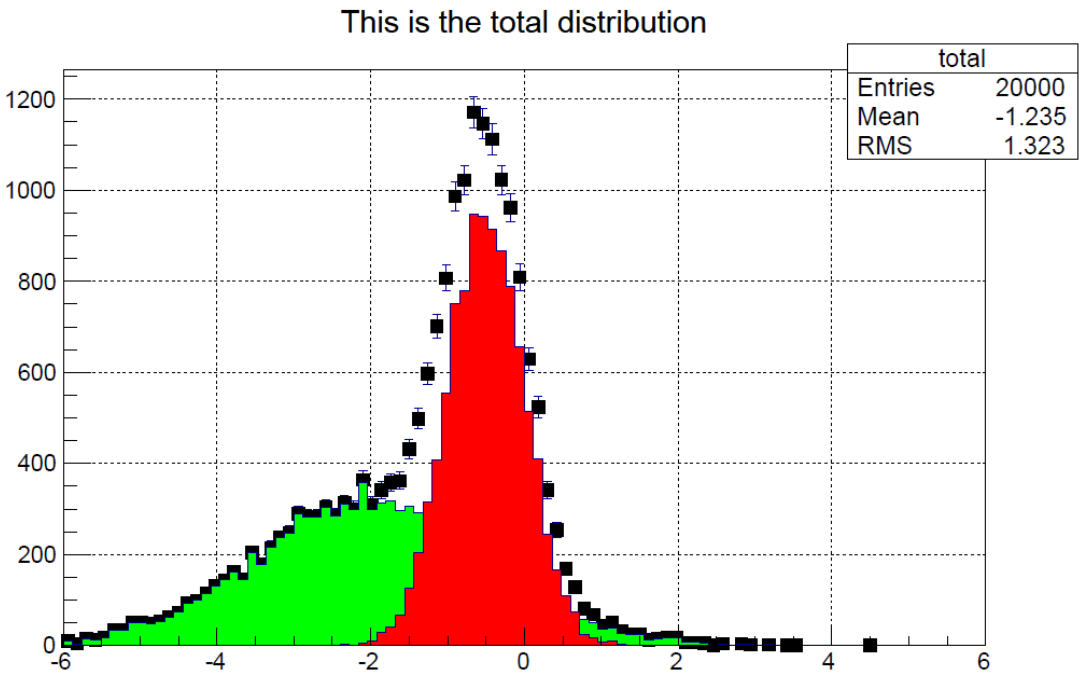
```
#include <TCanvas.h>
#include <TH1D.h>
#include <TRandom3.h>

void histosum(){

  TCanvas *c1 = new TCanvas("c1","The histosum example",200,10,600,400);
  c1->SetGrid();
  c1->Draw();

  // Create some histograms.
  TH1D *total = new TH1D("total","This is the total distribution",100,-6.,6.);
  TH1D *main = new TH1D("main","Main contributor",100,-6.,6.);
  TH1D *s1 = new TH1D("s1","This is the first signal",100,-6.,6.);

  total->SetMarkerStyle(21);
  total->SetMarkerSize(0.8);
  main->SetFillColor(kGreen);
  s1->SetFillColor(kRed);

  // Fill histograms randomly
  TRandom3 *random= new TRandom3();

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  for ( Int_t i=0; i<10000; i++) {

    xmain = random->Gaus(-2,1.5);
    xs1 = random->Gaus(-0.5,0.5);

    main->Fill(xmain);
    s1->Fill(xs1);

    total->Fill(xmain);
    total->Fill(xs1);

  }

  total->Draw("elp");
  main->Draw("same");
  s1->Draw("same");
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
  c1->SaveAs("fighistosumMain.pdf");

}
```

--:--- histosumCompiled.C All L43 (C++/l Abbrev)

like compiled program
but without main

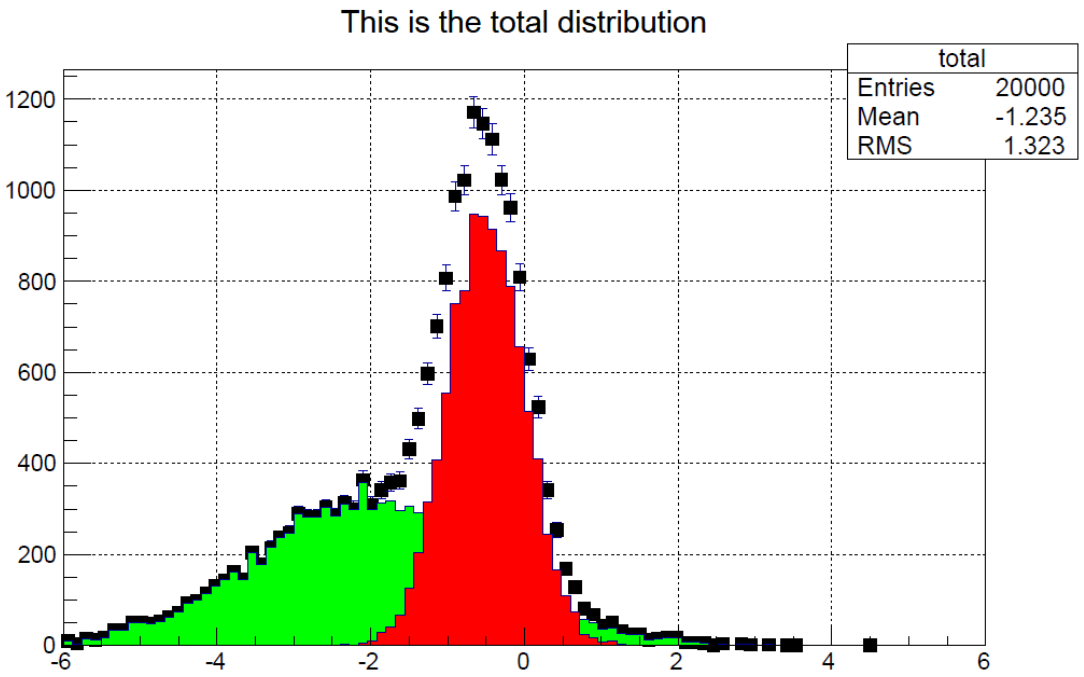
TH1D example: sum of histograms, compiled script

```
> root
root [0] .L histosumCompiled.C++
Info in <TUnixSystem::ACLiC>: creating shared library
   /home/mocchiut/scripts/./histosumCompiled_C.so
root [1] histosum()
Info in <TCanvas::Print>: pdf file fighistosumMain.pdf has been created
root [2]
```



TH1D example: sum of histograms, compiled script

```
> root
root [0] gSystem->Load("histosumCompiled_C.so")
root [1] histosum()
Info in <TCanvas::Print>: pdf file fighistosumMain.pdf has been created
root [2]
```



TH1D example: sum of histograms, interpreted script

emacs@marte.infis.univ.trieste.it

File Edit Options Buffers Tools C++ Help

```
void histosum(){

  TCanvas *cl = new TCanvas("cl","The histosum example",200,10,600,400);
  cl->SetGrid();
  cl->Draw();

  // Create some histograms.
  TH1D *total = new TH1D("total","This is the total distribution",100,-6.,6.);
  TH1D *main = new TH1D("main","Main contributor",100,-6.,6.);
  TH1D *s1 = new TH1D("s1","This is the first signal",100,-6.,6.);

  total->SetMarkerStyle(21);
  total->SetMarkerSize(0.8);
  main->SetFillColor(kGreen);
  s1->SetFillColor(kRed);

  // Fill histograms randomly
  TRandom3 *random= new TRandom3();

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  for ( Int_t i=0; i<10000; i++) {

    xmain = random->Gaus(-2,1.5);
    xs1 = random->Gaus(-0.5,0.5);

    main->Fill(xmain);
    s1->Fill(xs1);

    total->Fill(xmain);
    total->Fill(xs1);

  }

  total->Draw("elp");
  main->Draw("same");
  s1->Draw("same");
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
  cl->SaveAs("fighistosumMain.pdf");

}
```

--:--- histosum.C All L1 (C++/l Abbrev)

Wrote /home/mocchiut/scripts/1123/histosum.C

TH1D example: sum of histograms, interpreted script

emacs@marte.infis.univ.trieste.it

File Edit Options Buffers Tools C++ Help

```
void histosum(){

  TCanvas *cl = new TCanvas("cl","The histosum example",200,10,600,400);
  cl->SetGrid();
  cl->Draw();

  // Create some histograms.
  TH1D *total = new TH1D("total","This is the total distribution",100,-6.,6.);
  TH1D *main = new TH1D("main","Main contributor",100,-6.,6.);
  TH1D *s1 = new TH1D("s1","This is the first signal",100,-6.,6.);

  total->SetMarkerStyle(21);
  total->SetMarkerSize(0.8);
  main->SetFillColor(kGreen);
  s1->SetFillColor(kRed);

  // Fill histograms randomly
  TRandom3 *random= new TRandom3();

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  for ( Int_t i=0; i<10000; i++) {

    xmain = random->Gaus(-2,1.5);
    xs1 = random->Gaus(-0.5,0.5);

    main->Fill(xmain);
    s1->Fill(xs1);

    total->Fill(xmain);
    total->Fill(xs1);

  }

  total->Draw("elp");
  main->Draw("same");
  s1->Draw("same");
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
  cl->SaveAs("fighistosumMain.pdf");

}
```

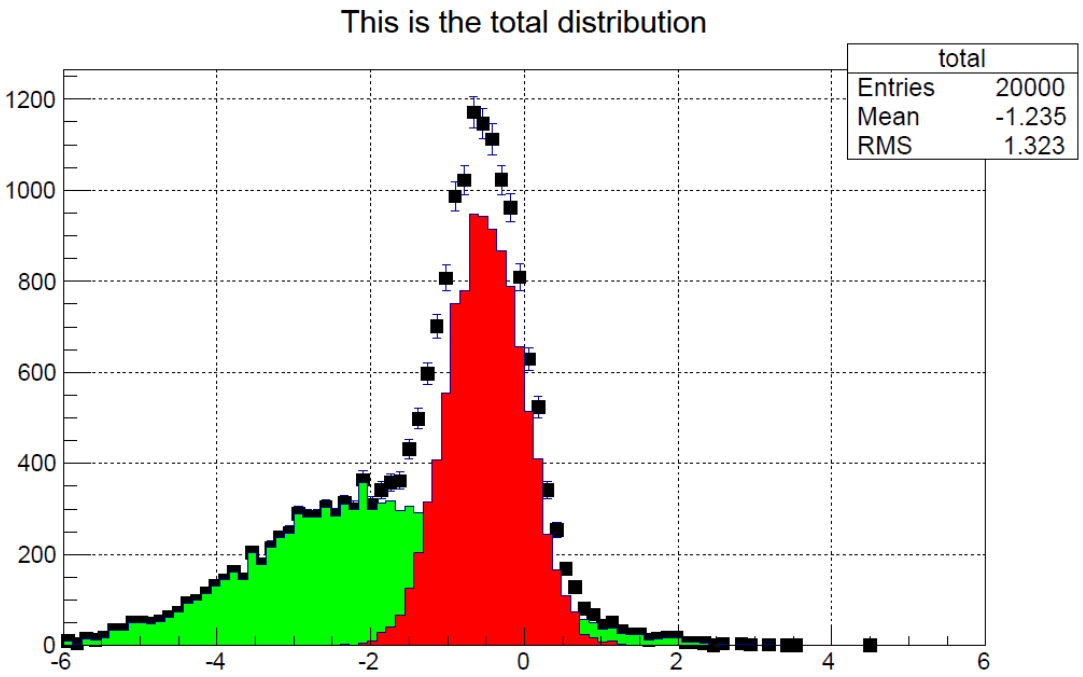
--:--- histosum.C All L1 (C++/l Abbrev)

Wrote /home/mocchiut/scripts/1123/histosum.C

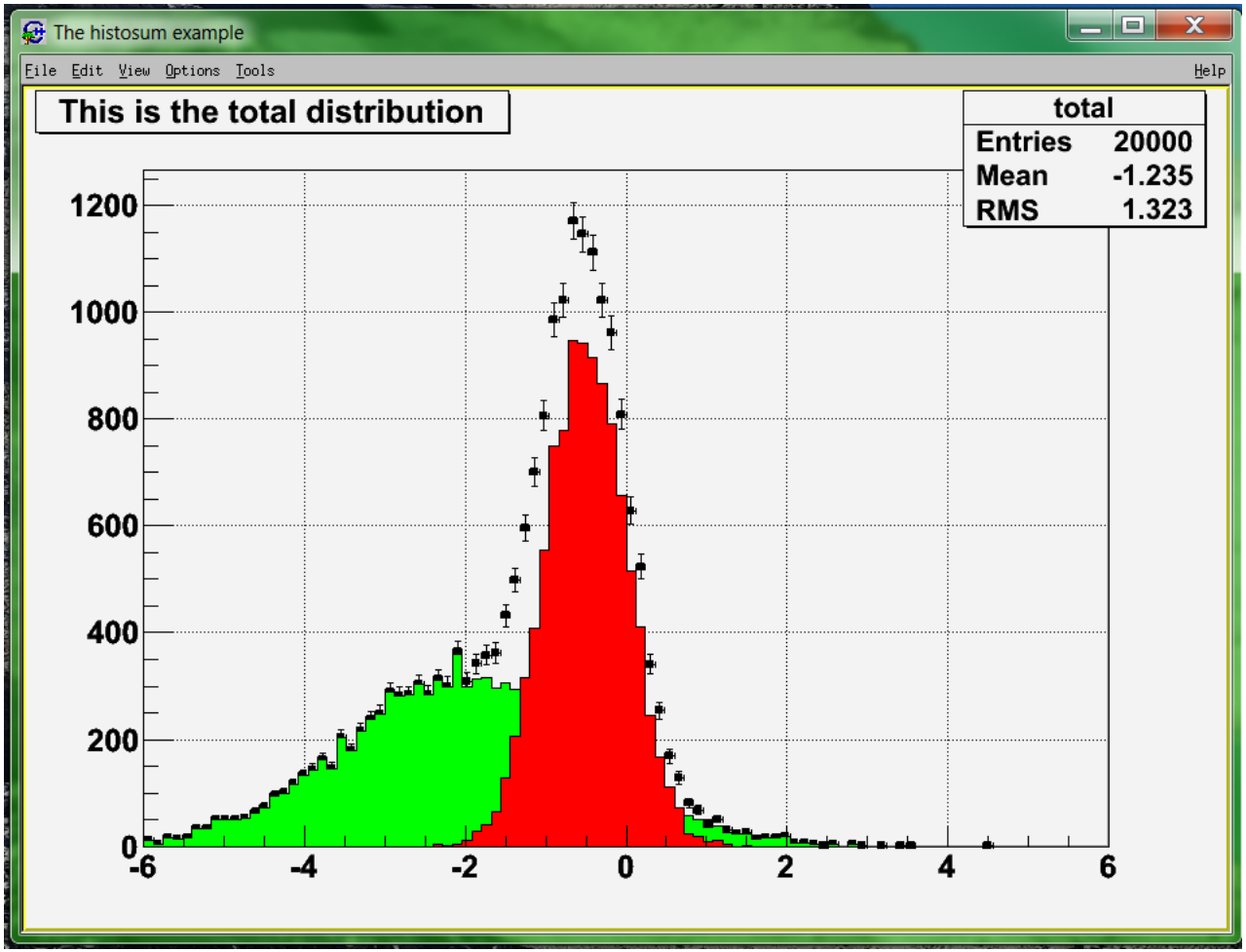
like compiled script
but without headers

TH1D example: sum of histograms, interpreted script

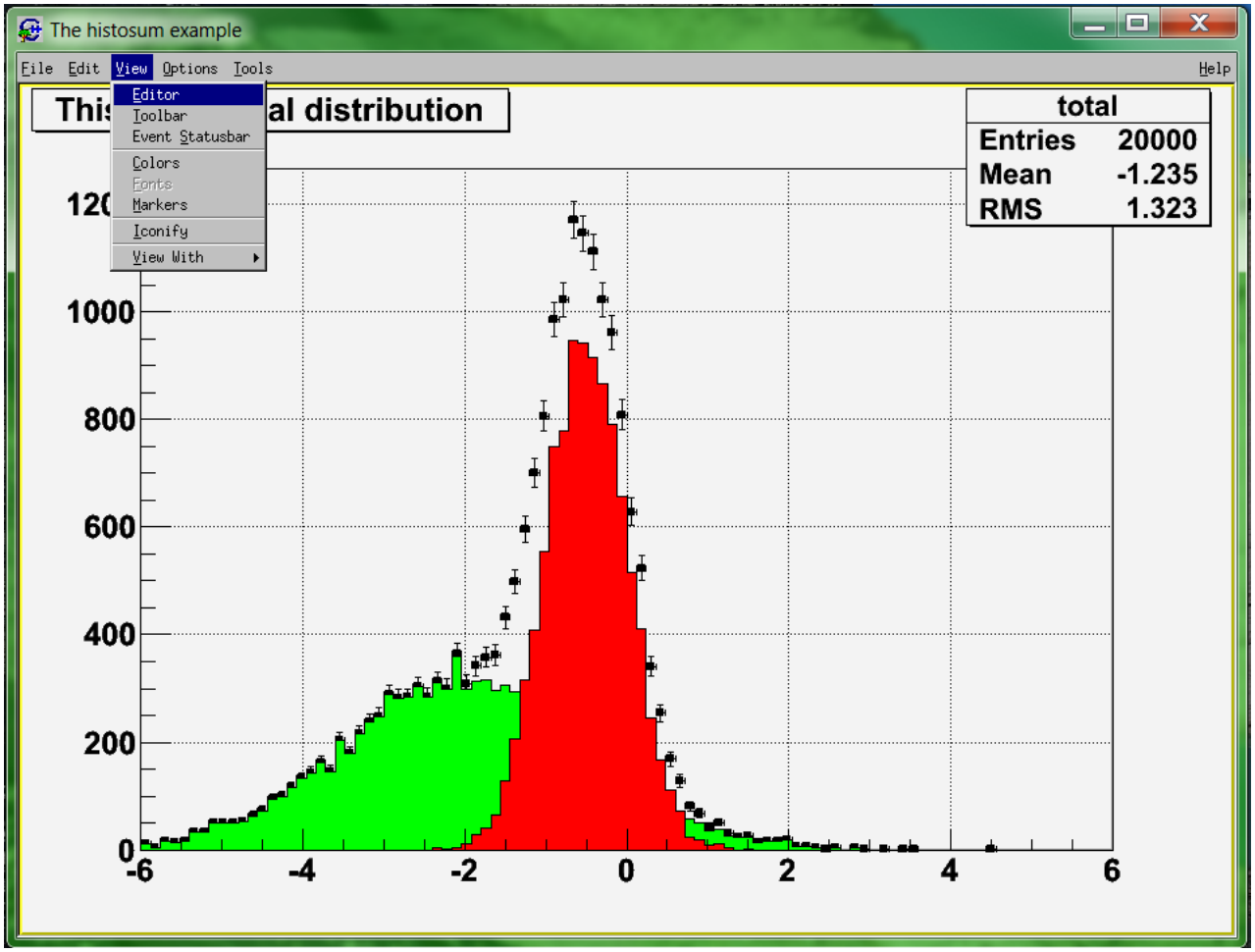
```
> root
root [0] .L histosum.C
root [1] histosum()
Info in <TCanvas::Print>: pdf file fighistosumMain.pdf has been created
root [2]
```



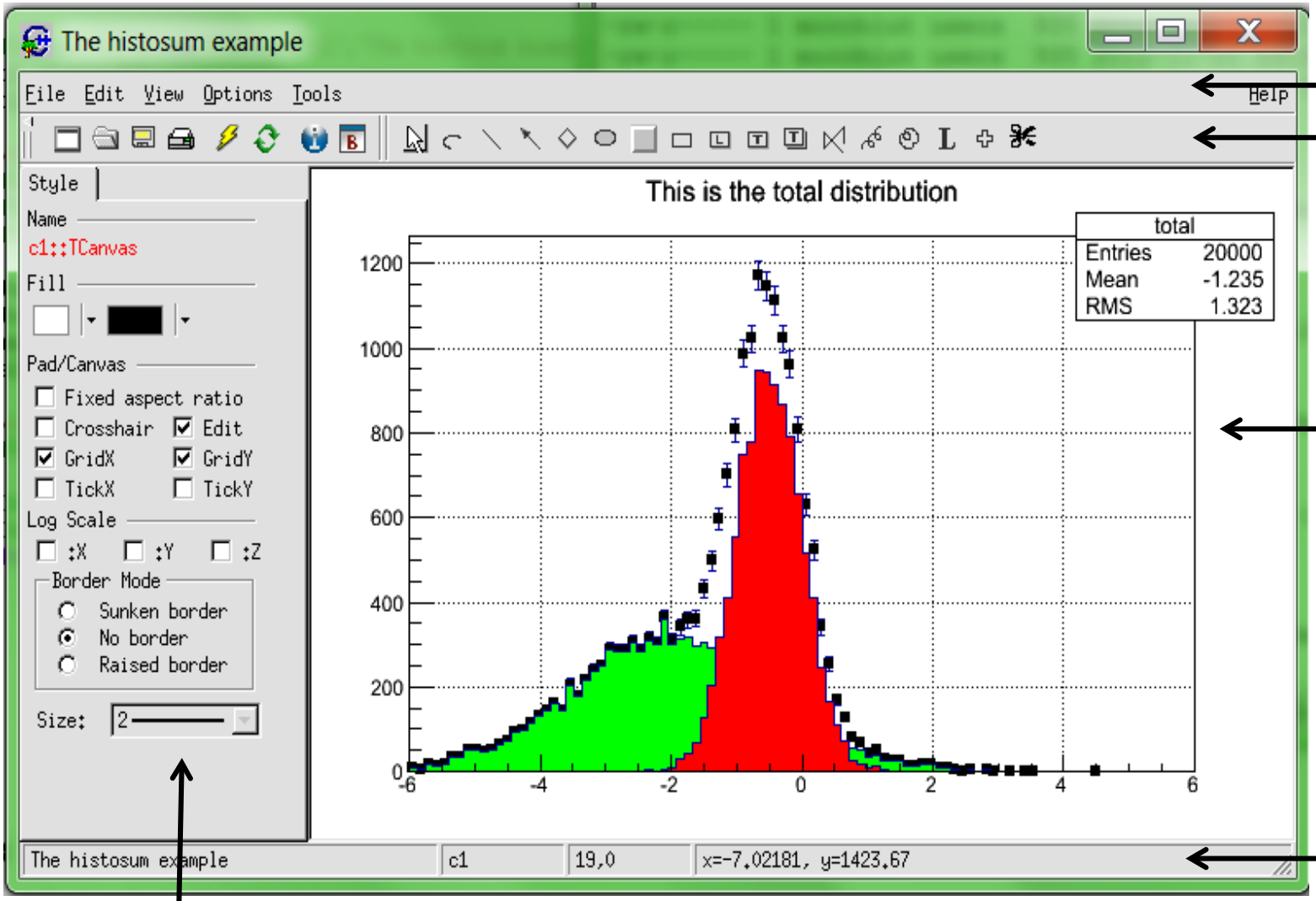
GUI


The histosum example
File Edit View Options Tools
Help
This is the total distribution
total
Entries 20000
Mean -1.235
RMS 1.323
1200
1000
800
600
400
200
0
-6
-4
-2
0
2
4
6

GUI


The histosum example
File Edit View Options Tools Help
Editor
Toolbar
Event Statusbar
Colors
Fonts
Markers
Iconify
View With
This al distribution
total
Entries 20000
Mean -1.235
RMS 1.323
1000
800
600
400
200
0
-6
-4
-2
0
2
4
6

GUI



Menu bar

Tool bar

Canvas

Status bar

Editor frame

ROOT

- **Save data**. You can save your data (and any C++ object) in a compressed binary form in a ROOT file. The object format is also saved in the same file. ROOT provides a data structure that is extremely powerful for fast access of huge amounts of data - orders of magnitude faster than any database.
- **Access data**. Data saved into one or several ROOT files can be accessed from your PC, from the web and from large-scale file delivery systems used e.g. in the GRID. ROOT trees spread over several files can be chained and accessed as a unique object, allowing for loops over huge amounts of data.

Data structure

- Data can be organized in many different ways. However, most applications (including database interfaces) adopt a simple model in which one has many copies of the same linear data structure (often called a "record"), ending up into a bidimensional data structure (often called a "table"). For example, data-sheet applications offer a graphical representation of the table in which the data structure is developed horizontally, across a number of columns, while the records are stacked vertically, spanning many rows.

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 210 x 20	4,14

Data structure, n-tuples

Of course, this widely adopted data organization is possible also in ROOT. The tables are named "*n-tuples",* as in mathematics, the records are called "*events",* as in physics, and the column headers are called "*variables",* as in computer science. However, ROOT can save tables, or n-tuples, in two ways.

The first one is the usual way, in which a file contains a sequence of records, or events:

variable0 ↓ variable1 ↓ variable2 ↓

event1 →
event2 →

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 210 x 20	4,14

Data structure, n-tuples problems

When the event data structure is quite larger than the subset on which the users normally focus on, reading such file is not very fast, because one has to "jump" from a subset of the current event to the corresponding subset of the next event, and so on, preventing any effective exploitation of the caching mechanisms implemented by the disk controllers and by the operative systems.

variable0 ↓ variable1 ↓ variable2 ↓

event1 →
event2 →

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 210 x 20	4,14

Data structure, n-tuples problems

When the event data structure is quite larger than the subset on which the users normally focus on, reading such file is not very fast, because one has to "jump" from a subset of the current event to the corresponding subset of the next event, and so on, preventing any effective exploitation of the caching mechanisms implemented by the disk controllers and by the operative systems.

variable0 variable1 variable2

event1
event2

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 210 x 20	4,14

Data structure, from n-tuples to trees

For this reason, instead of building a sequence of records, by default ROOT splits each event into its pieces (the variables) and builds a file by putting together all columns. This has two desirable effects: each column is a homogeneous sequence of the same variable (maximum internal compression); when looping over few variables of each event, as it is done most often, the time required to fetch all data is much smaller (fully exploiting OS and disk controller caching mechanisms).

variable0 ↓ variable1 ↓ variable2 ↓

event1 →
event2 →

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 210 x 20	4,14

Data structure, from n-tuples to trees

For this reason, instead of building a sequence of records, by default ROOT splits each event into its pieces (the variables) and builds a file by putting together all columns. This has two desirable effects: each column is a homogeneous sequence of the same variable (maximum internal compression); when looping over few variables of each event, as it is done most often, the time required to fetch all data is much smaller (fully exploiting OS and disk controller caching mechanisms).

variable0 variable1 variable2

event1
event2

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 200 x 20	4,14

Data structure, trees

In addition, ROOT provides more than plain n-tuples: the most powerful data structure is a "*tree*", the same data structure that is used by your operative system to save files into folders that may contain other folders. The structure of a ROOT tree can be arbitrarily complex, because a tree can contain, as "*branches*", simple variables or more complex objects, including other trees! A variable is always the end point of a branch in the ROOT jargon.

variable0 ↓ variable1 ↓ variable2 ↓

event1 →
event2 →

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 210 x 20	4,14

Data structure, trees

variable0 variable1 variable2

event1
event2

codice	articolo	prezzo iva esclusa
AMS 01.115	ADESIVO DIESEL	0,26
AMS 01.140	SALVAPORTA GOMMA MM.650 C/BIAD.	3,30
AMS 01.141	SALVAPORTA UNIV. MM.650	3,75
AMS 01.142	SALVAPORTA MM.650 UNIVERSALE	4,05
AMS 01.251	ANTISOL DEGRADE' FUME'	5,00
AMS 01.367/CP	ADESIVO "P" PRIVATISTA CP	3,20
AMS 01.506	ADESIVO PER TARGA LATERALE	1,04
AMS 01.544/OM	PELLIC.SOLARI OMOLOGATE cm.300x50 FUME'	31,60
AMS 01.545/OM	PELLIC.SOLARI OMOLOGATE cm.150x75 FUME'	28,80
AMS 01.702	PANNELLO TARGA RIPETITRICE	4,00
AMS 01.716	ADESIVO "R" RIFIUTI GIALLO	1,80
AMS 01.730	ANTISOL CAMION VERDE TRASP. cm 210 x 20	4,14

TTrees

- A ROOT tree is a collection of *branches (i.e. instances of the TBranch class*).
- When reading a variable from a tree stored into a TFile, ROOT only reads the branch containing it, to achieve the maximum performance.
- The TTree::Fill(...) method loops over all branches invoking TBranch::Fill(...).

How to create and fill a TTree:

```
TTree *mytree = new TTree("name","title");
Float_t myFloat;
mytree->Branch("myFloat",&myFloat,"myFloat/F");
myFloat = 1.2345;
mytree->Fill();
myFloat = 10.9876;
mytree->Fill();
```

http://root.cern.ch/root/html532/TTree.html

TTrees

How to read a TTree (when you have it already opened):

```
TTree *mytree = new TTree("name","title");
Float_t myFloat;
mytree->Branch("myFloat",&myFloat,"myFloat/F");
myFloat = 1.2345;
mytree->Fill();
myFloat = 10.9876;
mytree->Fill();
mytree->GetEntries();          ⇒  2
mytree->GetEntry(0);
cout << myFloat << "\n";       ⇒  1.2345
mytree->GetEntry(1);
cout << myFloat << "\n";       ⇒  10.9876
```

http://root.cern.ch/root/html532/TTree.html

TTrees

How to read a TTree (from scratch):

```
//somehow you have found in a file a tree; you called "ptree"
  the pointer to the tree. You know it contains a variable
  called "myFloat" as a leaf. Then you can read it this way:
Float_t flotty; // create a float variable (any name)
ptree->SetBranchAddress("myFloat",&flotty); // tell the program
// to put content of myFloat when reading the tree at the
// address of flotty
ptree->GetEntries();
ptree->GetEntry(0);
cout << flotty << "\n";
ptree->GetEntry(1);
cout << flotty << "\n";
```

→ 2

→ 1.2345

→ 10.9876

http://root.cern.ch/root/html532/TTree.html

TTrees, useful commands

How to see the content of a TTree:

```
mytree->Print();
```

How to see the content of a TTree entry:

```
mytree->Show(entryNumber);
mytree->Show(234);
```

How to scan the content of a TTree:

```
mytree->Scan(“leaf1:leaf2:...”,“leaf1<leaf2 && ...”);
mytree->Scan(“xs1:xmain”,“xs1>0.9”);
```

How to draw a histogram out of a leaf:

```
mytree->Draw(“xs1”)
```

http://root.cern.ch/root/html532/TTree.html

TFiles

- ROOT provides system-independent binary files in which the user can store objects;
- TDirectory inheritance: information stored into a ROOT file can be organized in several subfolders. The user can navigate it as he were browsing the file system of his operative system;
- data are usually compressed when written into a TFile.

ROOT files can be opened for reading and/or writing, and the most common way to open a file is via the TFile constructors:

```
TFile *myfile = TFile::Open("filename.root"); //READ ONLY
TFile *myfile = TFile::Open("filename.root", "NEW"); // WRITE,
   open only if not already existing
TFile *myfile = TFile::Open("filename.root", "UPDATE"); // UPDATE
   mode, to modify an existing files
TFile *myfile = TFile::Open("filename.root", "RECREATE"); // to
   OVER-WRITE and existing file (or create if not existing)
```

http://root.cern.ch/root/html532/TFile.html

TFiles

Immediately after opening a file, it becomes the current directory. If more than one file are open, one can change the active file by means of the TDirectory::cd(...) method. A common error is to forget what is the active file and save objects in the wrong one.

```
TFile *myfile = TFile::Open("filename.root");
myfile->Close();

TFile *myfile = new TFile("filename.root");
myfile->Close();

TFile *myfile = new TFile("filename.root");
delete myfile;

...
```

TFiles, how to write objects

```
TFile *myfile =
  TFile::Open("filename.root","NEW");

TTree *tree = new TTree("myname","mytitle");
...
myfile->cd();
tree->Write();
anyOtherTObject->Write();
myfile->Close();
```

TFiles, dictionary

A ROOT file can also contain the "*dictionary*" of all classes used to create the objects saved into the file. The dictionary provides a description of all class attributes and of the inheritance tree, so that it is possible to generate the corresponding C++ code and read back objects from the file. Indeed, when the user does not have the application that was used to save data into his ROOT file, he can generate this code on-fly and the result is automatically compiled and loaded in memory as a shared library. Of course, this is only valid for class attributes (the only things needed to fetch the data from the file): class methods can not automatically generated.

TFiles, why a dictionary?

Having the dictionary saved together with the data is an invaluable resource for the end user, because it allows fetching data also when the original application changed or disappeared. In addition, future class versions not backward compatible will not prevent the user from reading old data: ROOT has been written having in mind the needs of high-energy physics experiments, that have a life cycle of 10-20 years, and whose data are going to be re-analyzed for many years after their conclusion, despite of possible changes in their format.

Example 1

1. start root
2. create a file with a tree and some variables
3. fill the tree with random numbers
4. save the file and quit root

with a script

5. start root
6. open the file
7. look file content
8. look at tree structure
9. show tree content for some entries
10. scan
11. draw
12. quit root

interactive CINT session

Example 1: treeExample.C script

```
void treeExample(TString filename){
  cout << " Creating file and tree \n";
  // create file
  TFile *file = TFile::Open(filename,"RECREATE");

  // create tree
  TTree *tree = new TTree("mytree","Example of a TTree");

  // random generator
  TRandom3 *random= new TRandom3();

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  tree->Branch("xs1",&xs1,"xs1/D");
  tree->Branch("xmain",&xmain,"xmain/D");

  cout << " Start filling... \n";
  for ( Int_t i=0; i<10000; i++) {

    xmain = random->Gaus(-2,1.5);
    xs1 = random->Gaus(-0.5,0.5);

    tree->Fill();

  }
  cout << " ...done! \n";

  cout << " Saving file \n";
  // write tree into file and close file
  file->cd();
  tree->Write();
  file->Close();
  cout << " bye bye! \n";
}
```

Example 1: treeExample.C script

```
bash> root
root[0] .L treeExample.C
root[1] treeExample("emiliano.root")
 Creating file and tree
 Start filling...
 ...done!
 Saving file
 bye bye!
root[2] .q
bash> ls –lh emiliano.root
-rw-r----- 1 mocchiut mocchiut 147K Nov 29 17:42 emiliano.root
```

Example 1: playing with file and tree

```
bash> root
root[0] TFile *f = TFile::Open("emiliano.root")
root[1] f->ls()
TFile**         emiliano.root
 TFile*         emiliano.root
  KEY: TTree    mytree;1        Example of a TTree
root[2] mytree->GetEntries()
(const Long64_t)10000
root[3] mytree->Print()
******************************************************************************
*Tree    :mytree    : Example of a TTree                                     *
*Entries :    10000 : Total =          161751 bytes  File  Size =     145312 *
*        :          : Tree compression factor =   1.11                       *
******************************************************************************
*Br    0 :xs1       : xs1/D                                                  *
*Entries :    10000 : Total  Size=      80690 bytes  File Size  =      71864 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.12     *
*............................................................................*
*Br    1 :xmain     : xmain/D                                                *
*Entries :    10000 : Total  Size=      80704 bytes  File Size  =      72943 *
*Baskets :        3 : Basket Size=      32000 bytes  Compression=   1.10     *
*............................................................................*
```

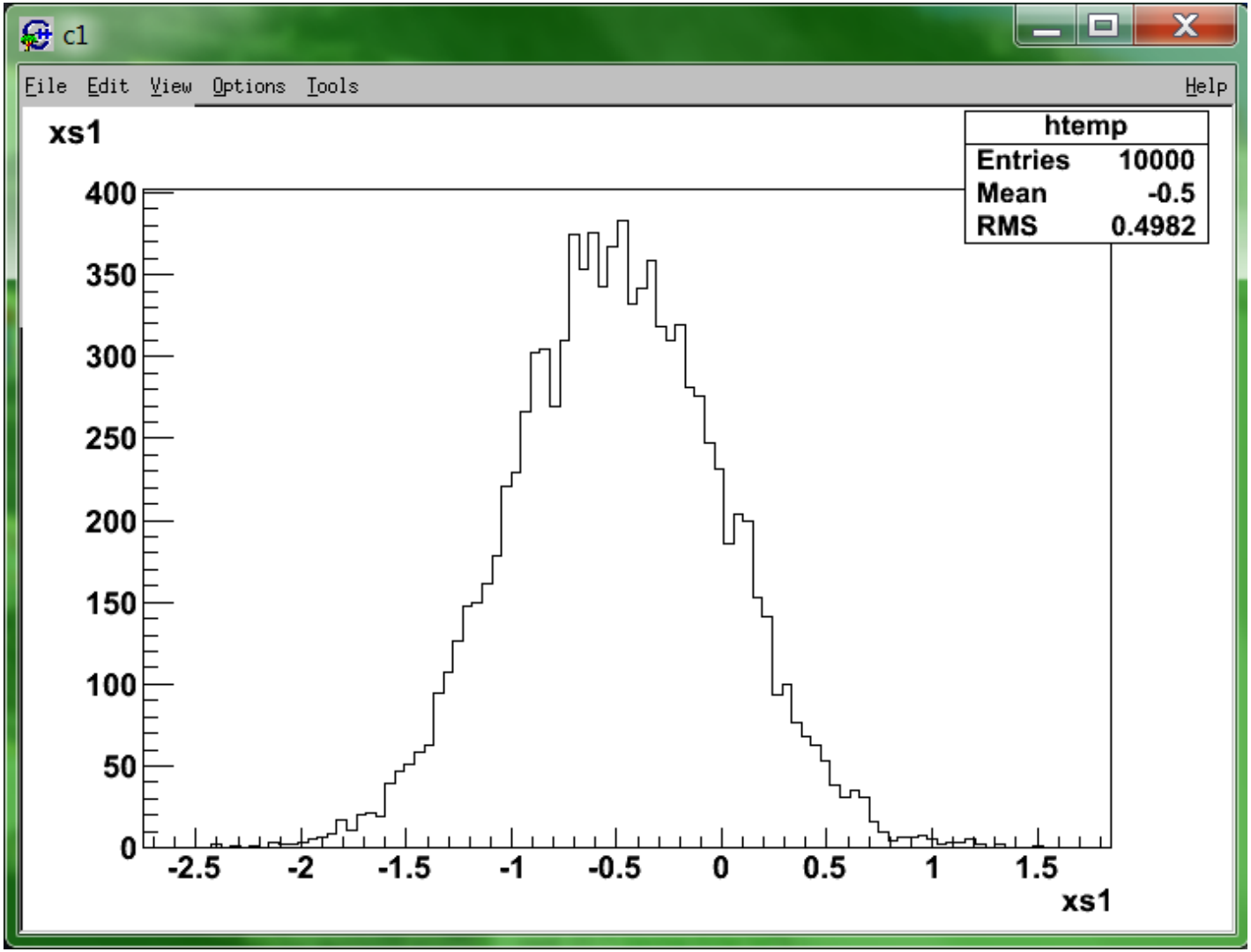
@2:CINT "cheating" pointer declaration!!

Example 1: playing with file and tree

```
bash> root
root[4] mytree->Show(0)
======> EVENT:0
 xs1              = -0.717382
 xmain            = -0.501601
root[5] mytree->Show(1234)
======> EVENT:1234
 xs1              = -0.143514
 xmain            = -3.06231
root[6] mytree->Scan("xs1:xmain","xs1>1.3")
************************************
*    Row   *       xs1 *     xmain *
************************************
*     1723 * 1.3004325 * -0.503121 *
*     2804 * 1.5032429 * -5.660506 *
************************************
==> 2 selected entries
(Long64_t)2
```

Example 1: playing with file and tree

```
root[6] mytree->Draw("xs1")
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
```



Example 2: readTree.C script

1. start root
2. read the previously created tree
3. create a plot with stored data
4. save the plot and quit root

} with a script

Example 2: readTree.C script

```
void readTree(TString inputFile){

  TCanvas *c1 = new TCanvas("c1","The histosum example",200,10,600,400);
  c1->SetGrid();
  c1->Draw();
  // Create some histograms.
  TH1D *total = new TH1D("total","This is the total distribution",100,-6.,6.);
  TH1D *main = new TH1D("main","Main contributor",100,-6.,6.);
  TH1D *s1 = new TH1D("s1","This is the first signal",100,-6.,6.);
  total->SetMarkerStyle(21);
  total->SetMarkerSize(0.8);
  main->SetFillColor(kGreen);
  s1->SetFillColor(kRed);

  // Open the file
  TFile *file = TFile::Open(inputFile);
  TTree *tree = (TTree*)file->Get("mytree");

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  tree->SetBranchAddress("xs1",&xs1);
  tree->SetBranchAddress("xmain",&xmain);

  for ( Int_t i=0; i<tree->GetEntries(); i++) {

    tree->GetEntry(i);

    main->Fill(xmain);
    s1->Fill(xs1);
    total->Fill(xmain);
    total->Fill(xs1);

  }

  total->Draw("e1p");
  main->Draw("same");
  s1->Draw("same");
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
  c1->SaveAs("figreadTree.pdf");

}
```

Example 2: readTree.C script

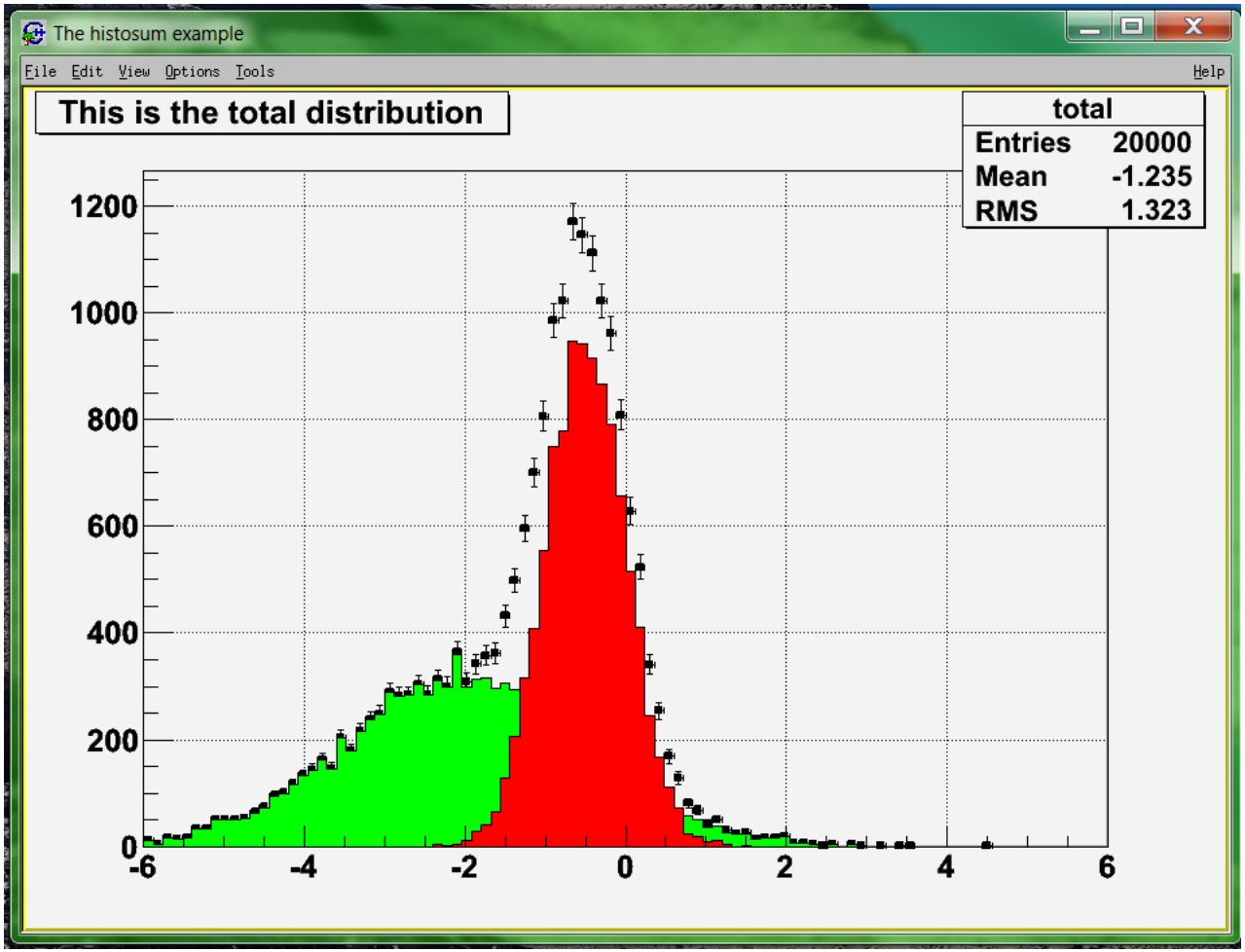
```
void readTree(TString inputFile){

  TCanvas *c1 = new TCanvas("c1","The histosum example",200,10,600,400);
  c1->SetGrid();
  c1->Draw();
  // Create some histograms.
  TH1D *total = new TH1D("total","This is the total distribution",100,-6.,6.);
  TH1D *main = new TH1D("main","Main contributor",100,-6.,6.);
  TH1D *s1 = new TH1D("s1","This is the first signal",100,-6.,6.);
  total->SetMarkerStyle(21);
  total->SetMarkerSize(0.8);
  main->SetFillColor(kGreen);
  s1->SetFillColor(kRed);

  // Open the file
  TFile *file = TFile::Open(inputFile);
  TTree *tree = (TTree*)file->Get("mytree");

  Double_t xs1 = 0.;
  Double_t xmain = 0.;
  tree->SetBranchAddress("xs1",&xs1);
  tree->SetBranchAddress("xmain",&xmain);

  for ( Int_t i=0; i<tree->GetEntries(); i++) {

    tree->GetEntry(i);

    main->Fill(xmain);
    s1->Fill(xs1);
    total->Fill(xmain);
    total->Fill(xs1);

  }

  total->Draw("e1p");
  main->Draw("same");
  s1->Draw("same");
  total->Draw("sameaxis"); // to redraw axis hidden by the fill area
  c1->SaveAs("figreadTree.pdf");

}
```

casting to a TTree!
"Get" is a polymorphic method
belonging to TDirectoryFile class

TDirectoryFile:
TObject * **Get**(const char* namecycle)

Example 2: running the script

```
root[0] .L readTree.C
root[1] readTree("emiliano.root")
Info in <TCanvas::Print>: pdf file figreadTree.pdf has been created
```



Exercises

Exercises – first part

1. Download from moodle the file

   `code20170407.tar.gz`

   and unpack it with:

   ```
   bash> tar zxvf code20170407.tar.gz
   ```

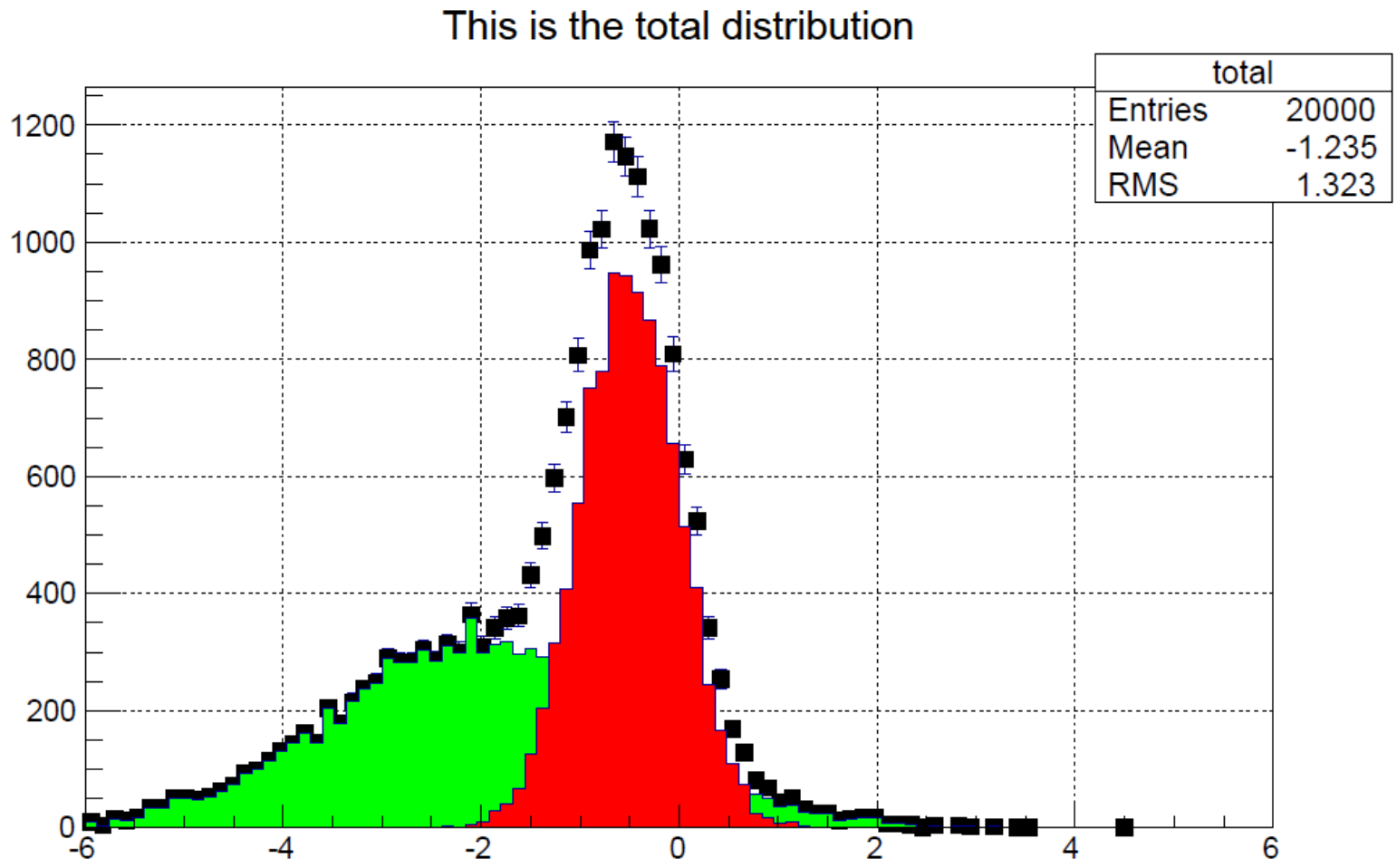
   it will create a directory called code20170407 with inside the following files:

   `histosumMain.cpp  histosum.C  histosumCompiled.C`

2. try to run the scripts in the three (four) ways
3. play with the GUI and interactive ROOT histogram

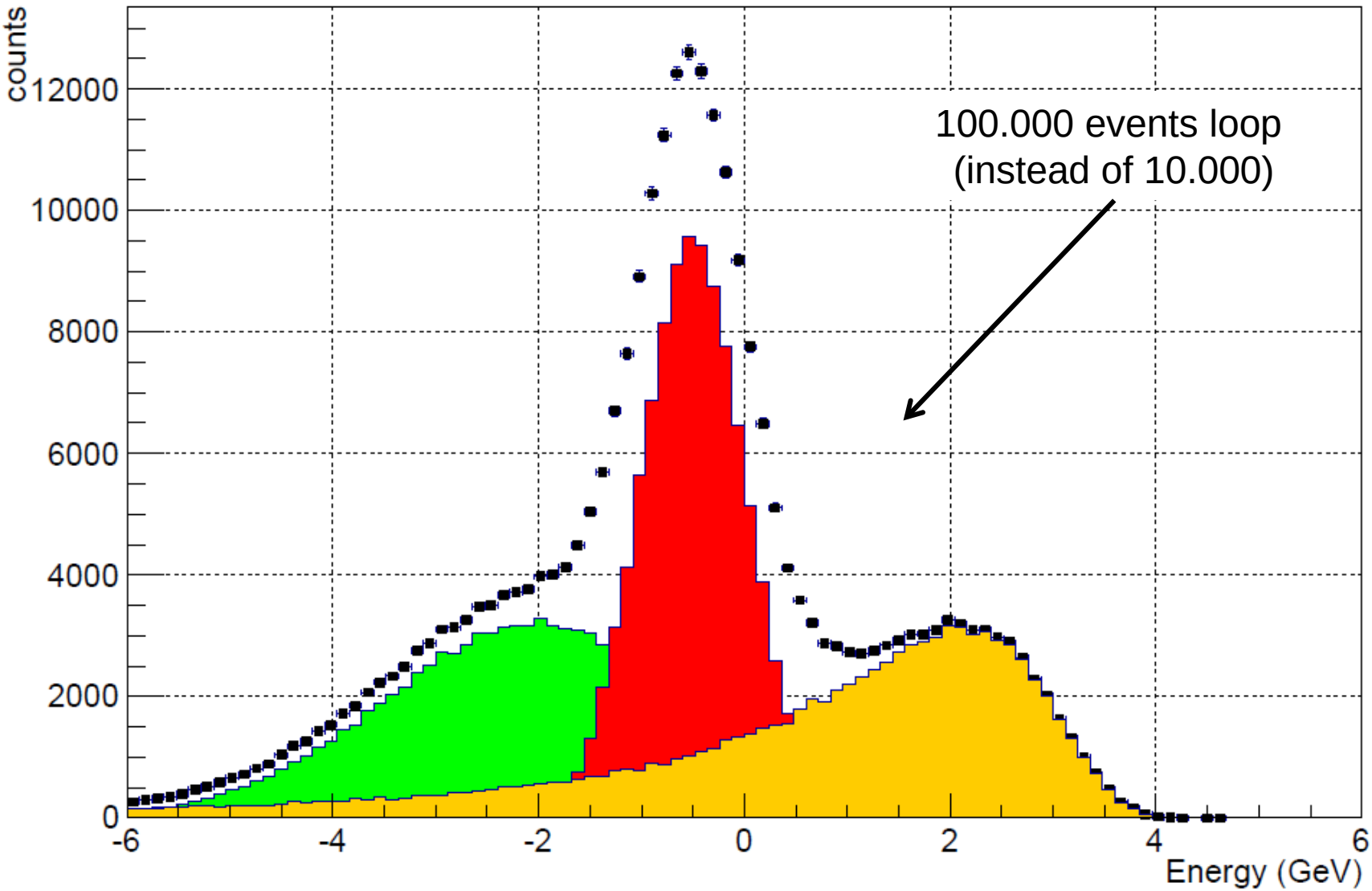
Exercise

Take the histosum.C script and change it in order to get instead of this figure:



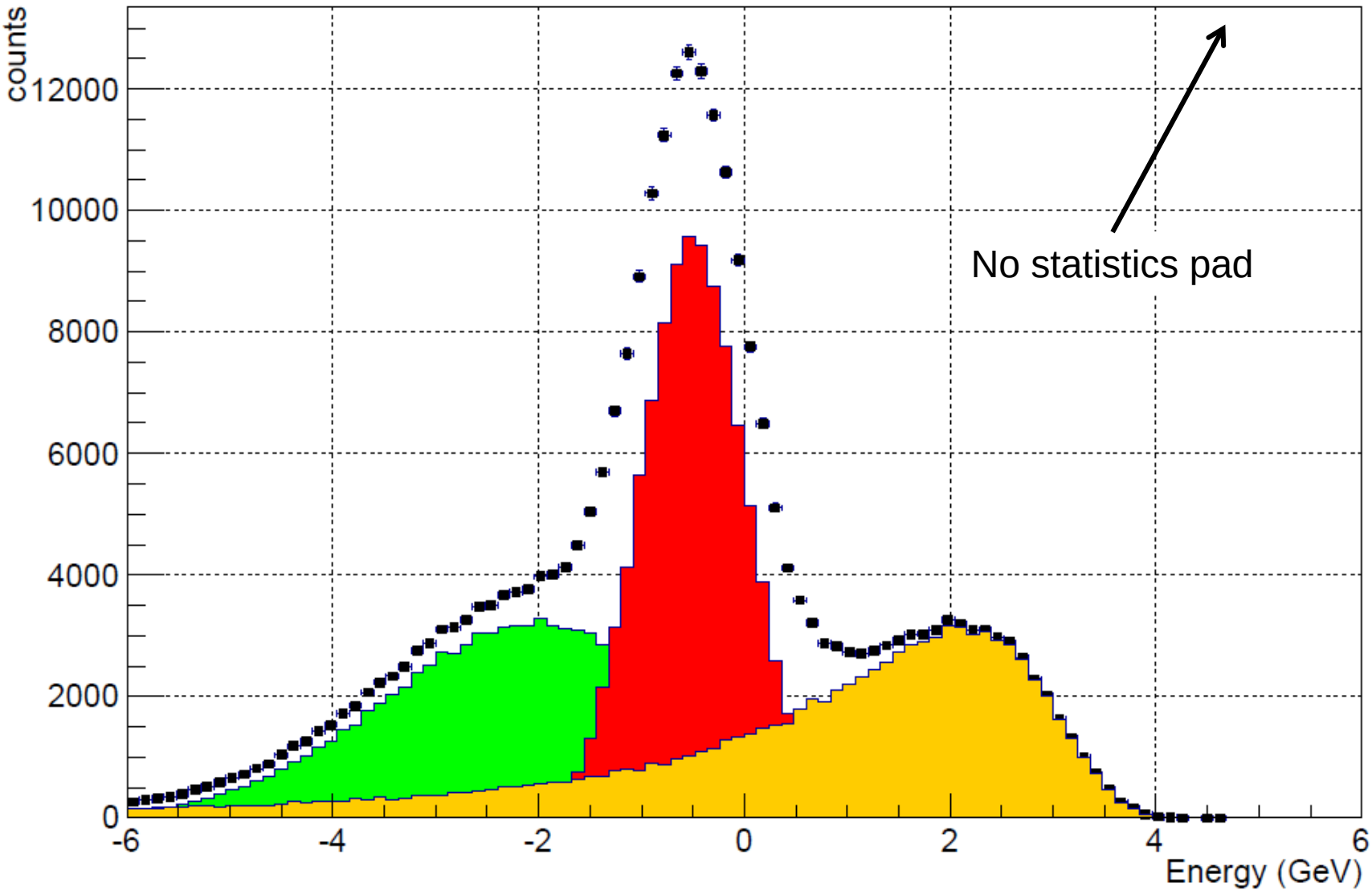
Exercise

This is the total distribution



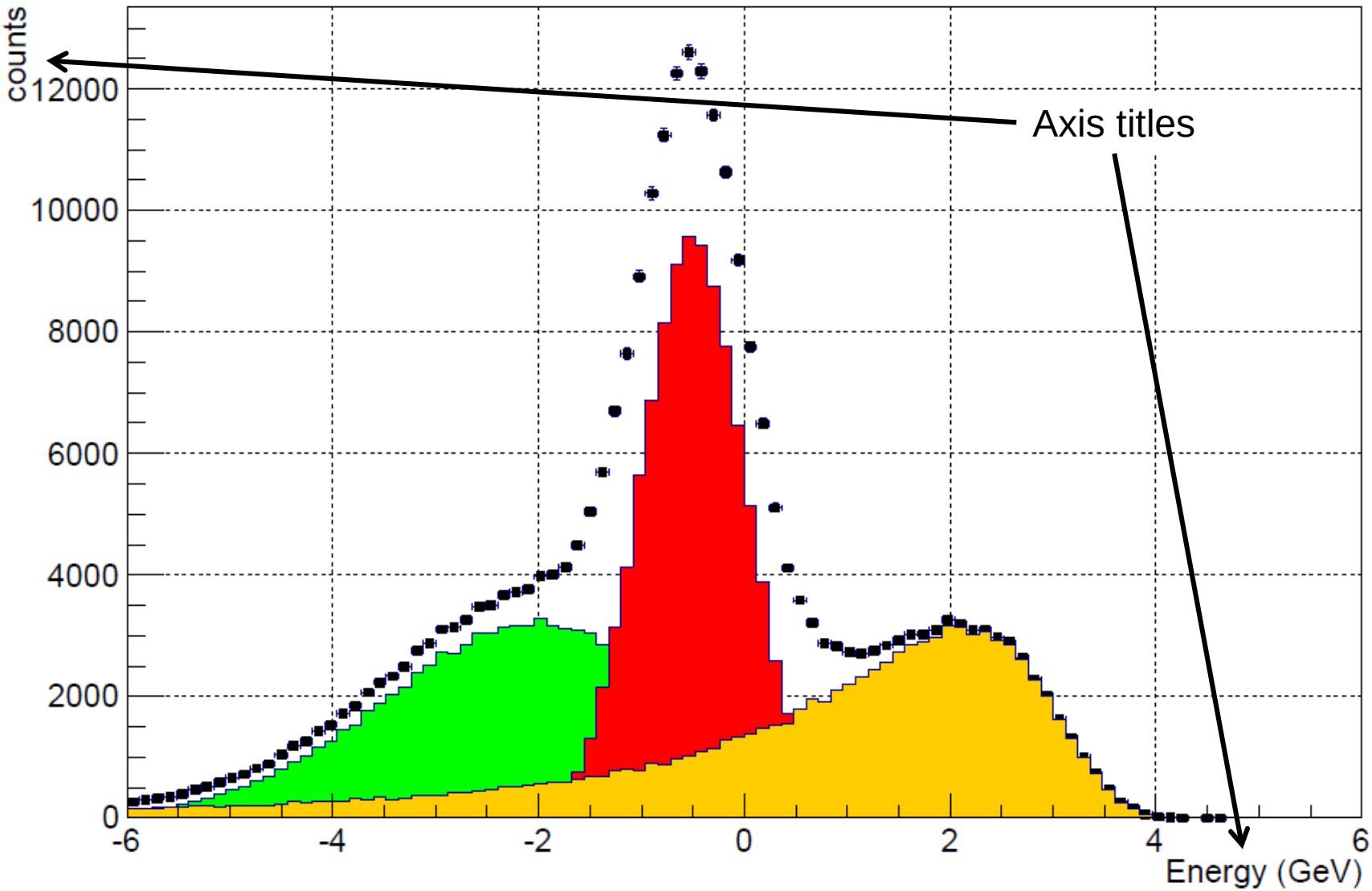
Exercise

This is the total distribution



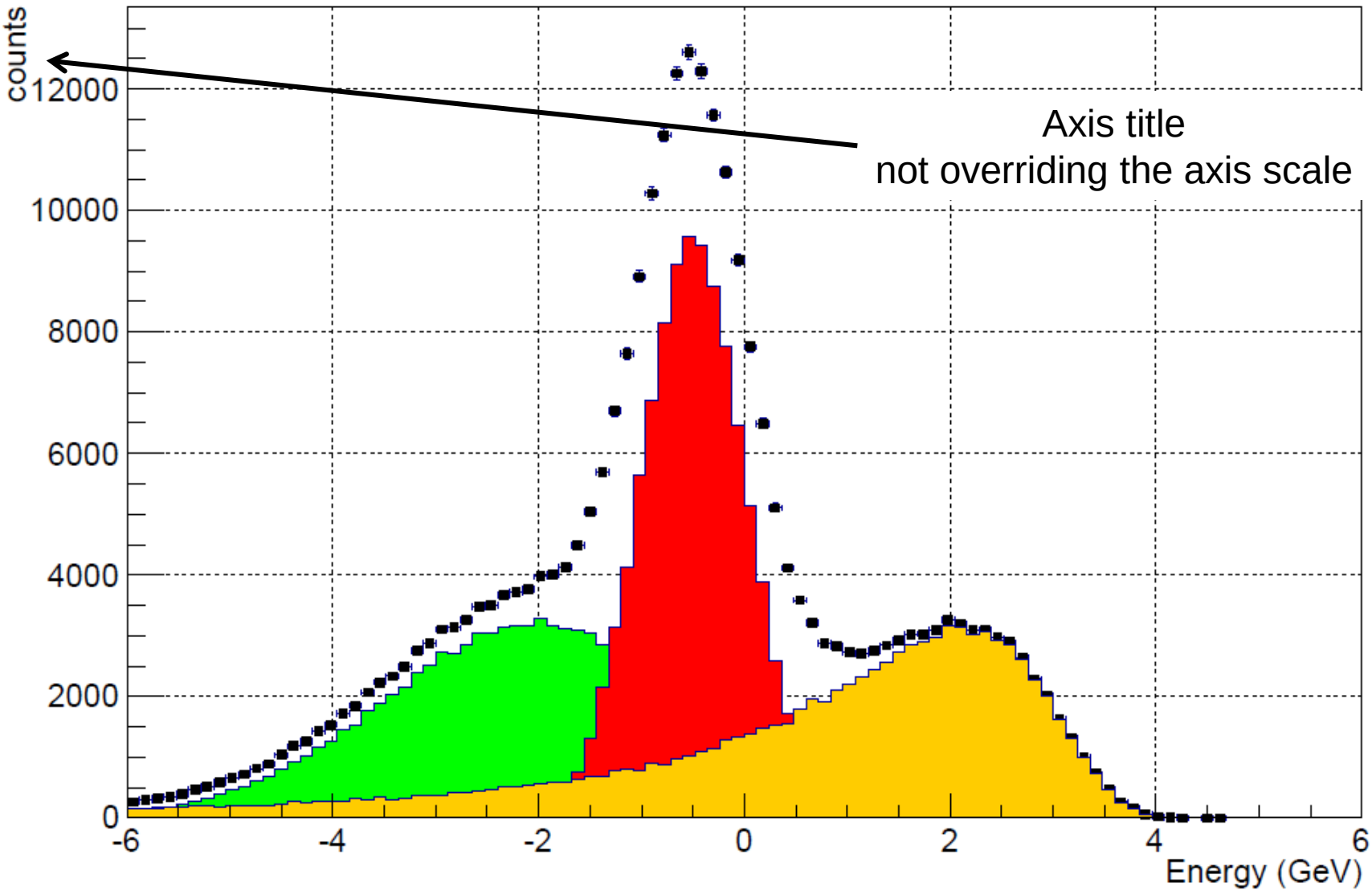
Exercise

This is the total distribution



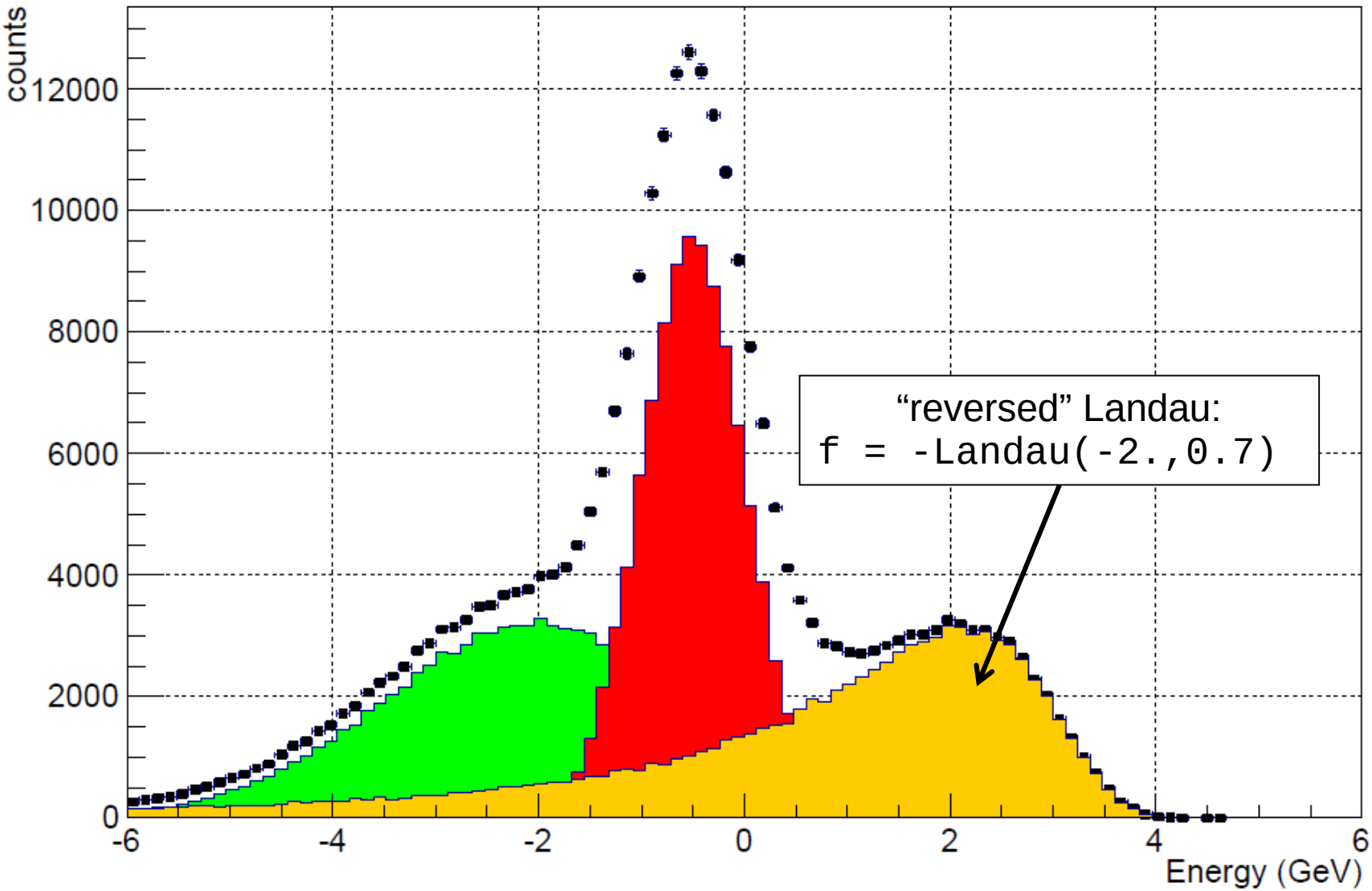
Exercise

This is the total distribution



Exercise

This is the total distribution



Exercises – second part

1. Download from moodle the file

   `code20170407.tar.gz`

   and unpack it with:

   ```
   bash> tar zxvf code20170407.tar.gz
   ```

   it will create a directory called code20170407 with inside the following files:

   `treeExample.C readTree.C treeExample2.C`

2. run the script `treeExample.C` to create a root file
3. play with it, try the `TFile::ls()`, `TTree::Print`, `TTree::Show`, `TTree::Scan`, `TTree:Draw` methods (slides 531, 540, 541);
4. run the `readTree.C` example;

Exercises

5. load and try to run `treeExample2.C` script. It crashes.
   - why?
   - at which point does the program crash (suggestion: add printout here and there in the code)?
   - is it the output file ok, does it make sense, is it corrupted?
   - what to do to avoid crashing?
   - where in the ROOT manual is described the reason why the program crashes?

NB:

```
root[0].L treeExample2.C
root[1] treeExample2()
```

```
 *** Break *** segmentation violation

===========================================================
There was a crash (#7 0x00a20f3d in SigHandler(ESignals) ()).
This is the entire stack trace of all threads:
===========================================================
#0  0x00802422 in __kernel_vsyscall ()
 etc. etc.
```

Exercises

5. load and try to run `treeExample2.C` script. It crashes.
   - why?
   - at which p
     there in the
   - is it the out
   - what to do t
   - where in t
     crashes?

```
void treeExample2(){
  // create file
  TFile *outputFile = new TFile("hihihihi.root","RECREATE");

  // create tree
  TTree *fullTree = new TTree("foolTree","Example of a TTree (hihihihi)");

  // random generator
  TRandom3 *random= new TRandom3();

  Float_t signal = 0.;
  Int_t neutrons = 0;
  fullTree->Branch("signal",&signal,"signal/F");
  fullTree->Branch("neutrons",&neutrons,"neutrons/I");

  for ( Int_t i=0; i<500000; i++) {
    signal = -random->Landau(-2,0.7);
    fullTree->Fill();
  }

  outputFile->cd();
  fullTree->Write();
  outputFile->Close();
  delete random;
  delete fullTree;
}
```

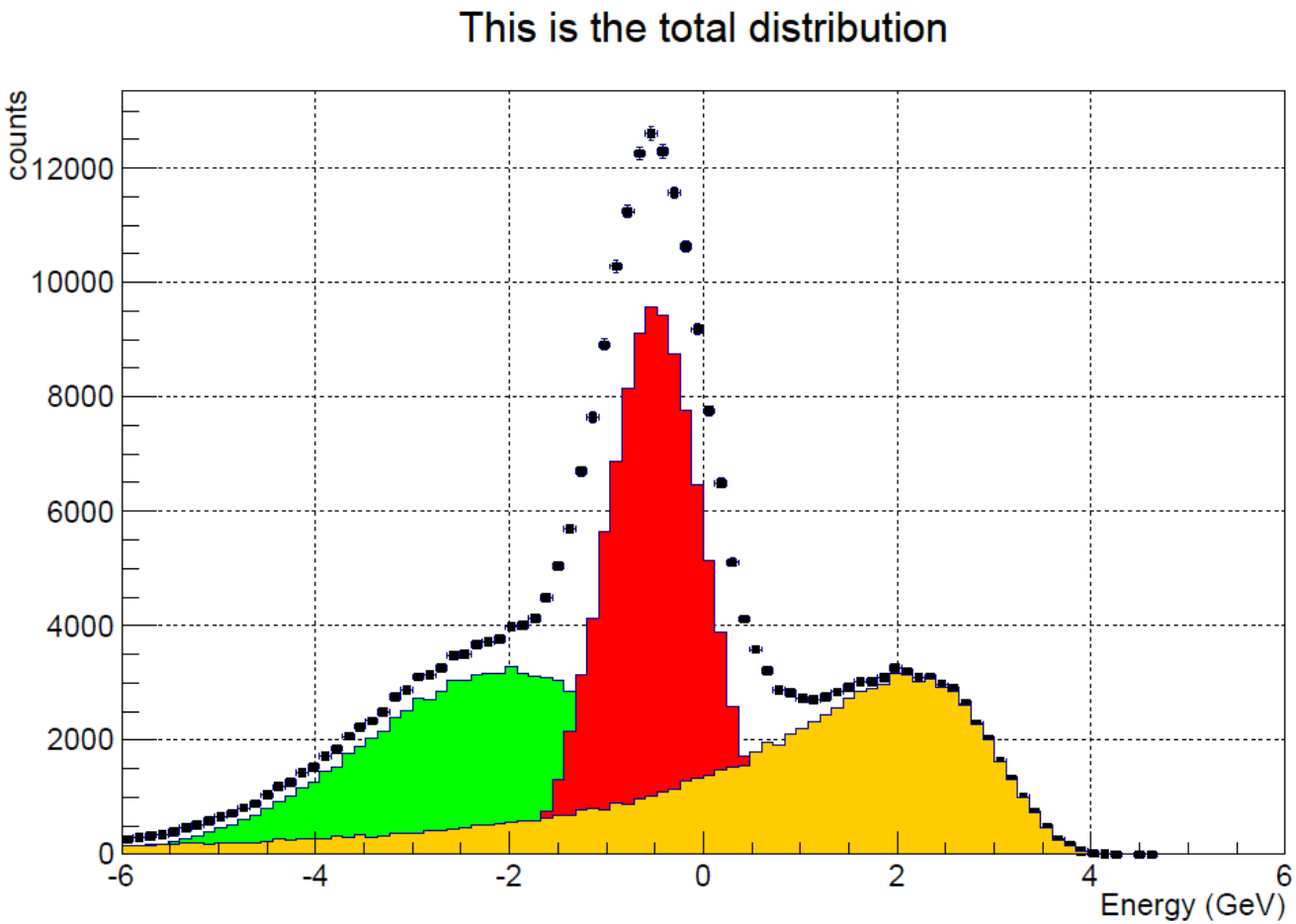
Exercises

5. load and try to run `treeExample2.C` script. It crashes.
   - why?
   - at which point does the program crash (suggestion: add printout here and there in the code)?
   - is it the output file ok, does it make sense, is it corrupted?
   - what to do to avoid crashing?
   - where in the ROOT manual is described the reason why the program crashes?

6. once you fixed the bug in `treeExample2.C` , change the script in order to:
   - remove useless variable(s);
   - add two new random generated gaussian variables (red and green distribution of the previous lesson exercise).

Exercises

7. write a `readTree2.C` script in order to re-create the last time histogram by reading data from the `treeExample2.C` output file:



2017/04/07 – take home messages

- ROOT useful for data analysis and storage
- ROOT CINT is not C nor C++!
- four ways of using ROOT (compiled, script, compiled script, command line)
- TFile and TTree structure, how to read and write a tree on a file
- pay attention to ROOT memory handling…