ROOT Tutorial

EIEIO Summer School
Will Parker

SNO+ Format of this Tutorial

UNIVERSITY OF OXFORD

Slides:

➡ What is ROOT

➡ Installation

➡ Basic usage

➡ Introductions to basic classes and concepts

Exercises (with demo):

➡ Basic ROOT command line usage

➡ Using macros

➡ Making plots

➡ Data storage and input/output

➡ Physics data simulation

Repo with example scripts and solutions from this tutorial

➡ https://github.com/willp240/rootTute/

This tutorial won't cover everything. But it will hopefully get you started, and point you to where to go for more information

What is ROOT

ROOT is software toolkit for:

➡ Data plotting (multiple 1 to N-dimensional histogram and graph formats)

➡ Data storage (data structures for event based analyses, I/O of any C++ object)

➡ Data analysis (math functions, fitting to histograms, statistical treatment)

➡ Data processing

It gives you everything you need to perform a physics analysis!

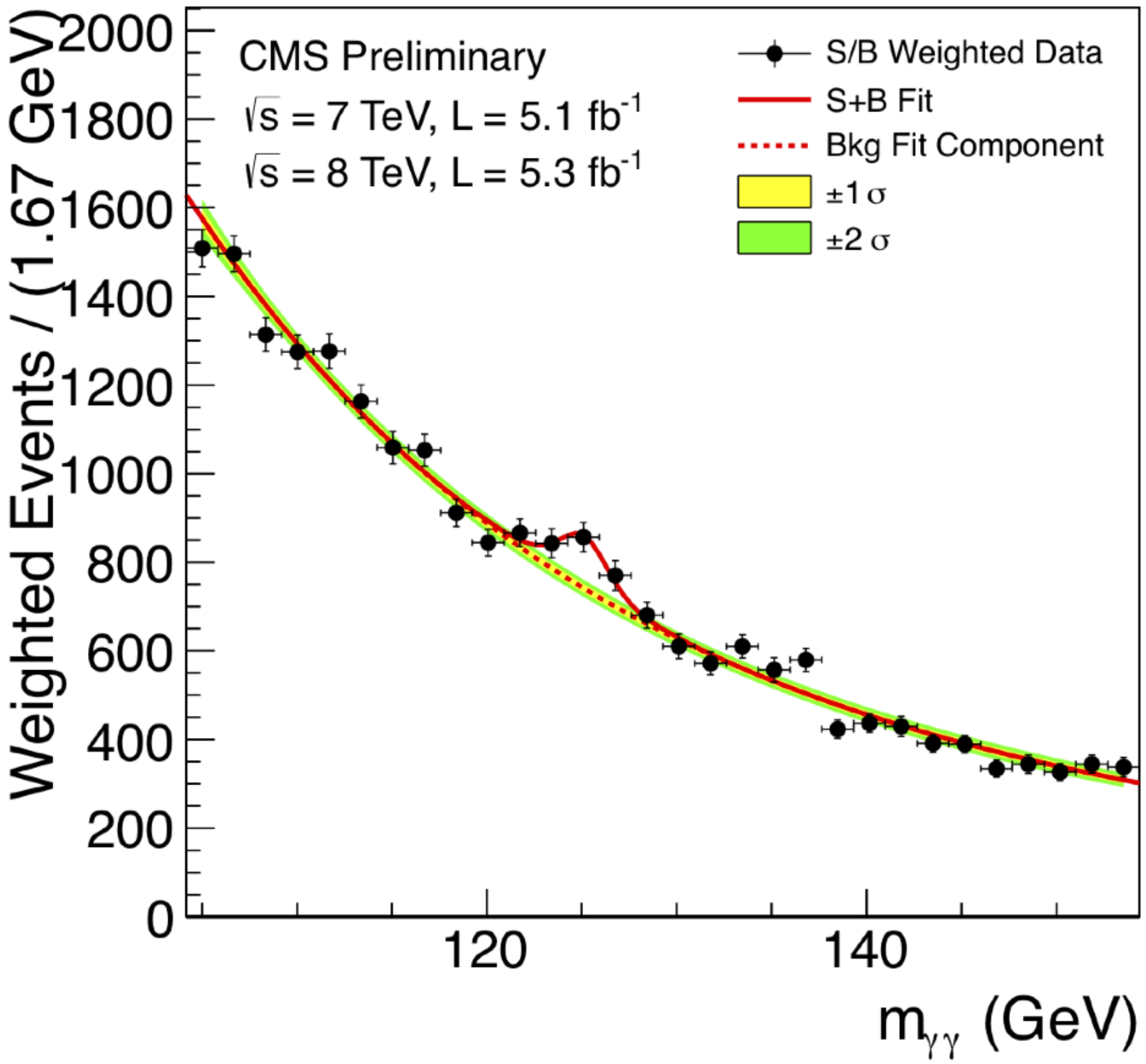
It's used by almost every high energy physics experiment

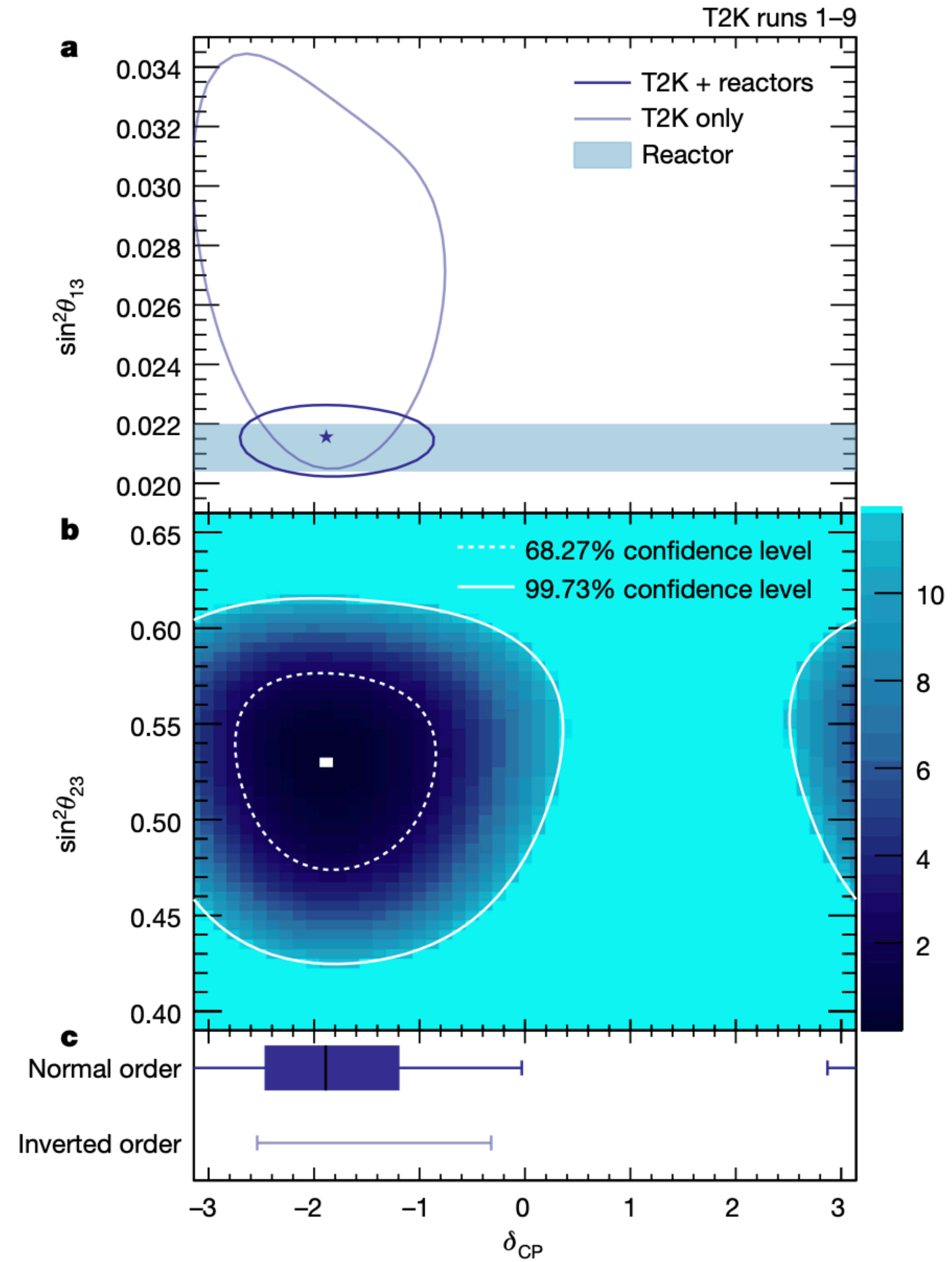
➡ Also now being used in other areas of physics, and industry

➡ Your joining a user base of tens of thousands!

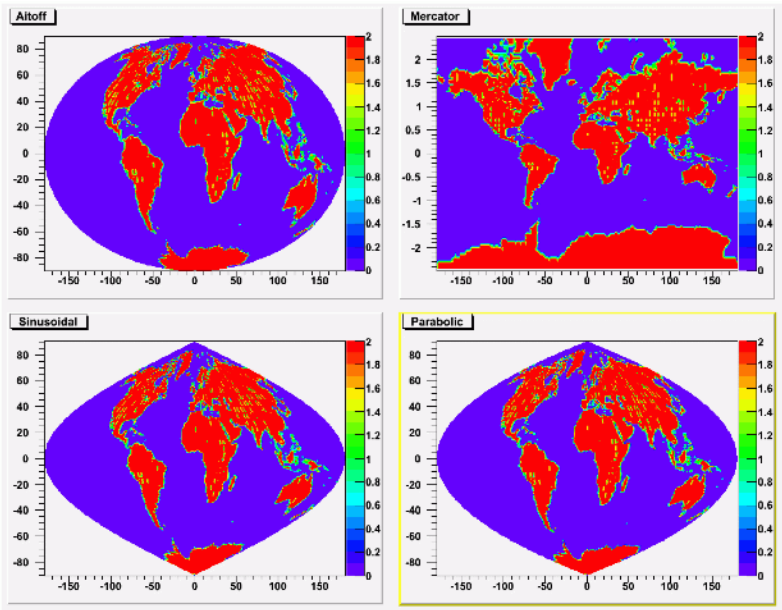
➡ Thousands of ROOT plots in scientific publications

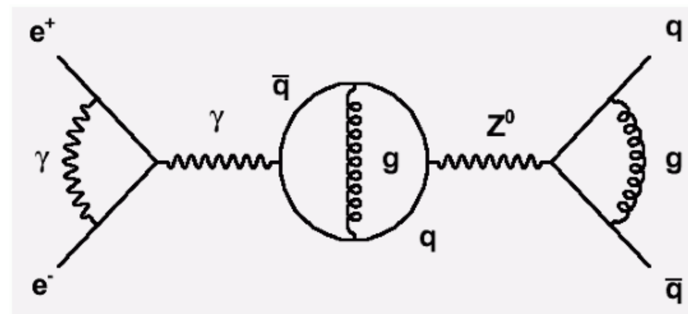
Example Plots

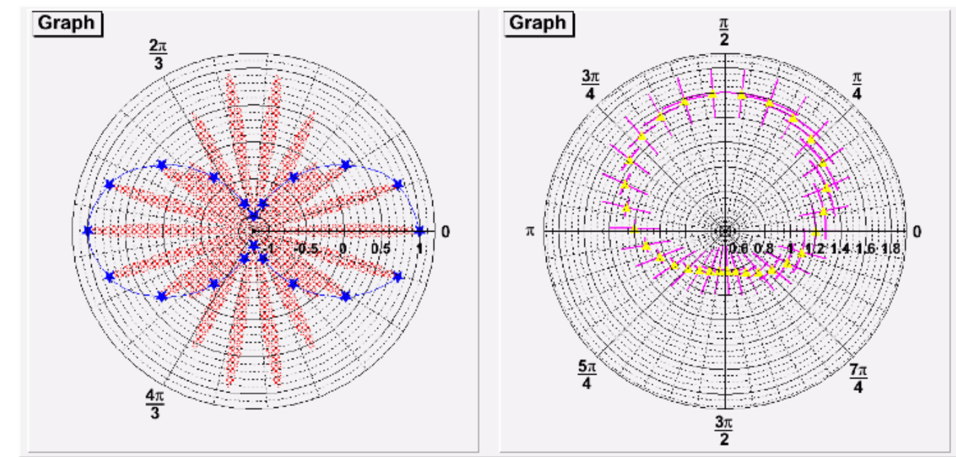


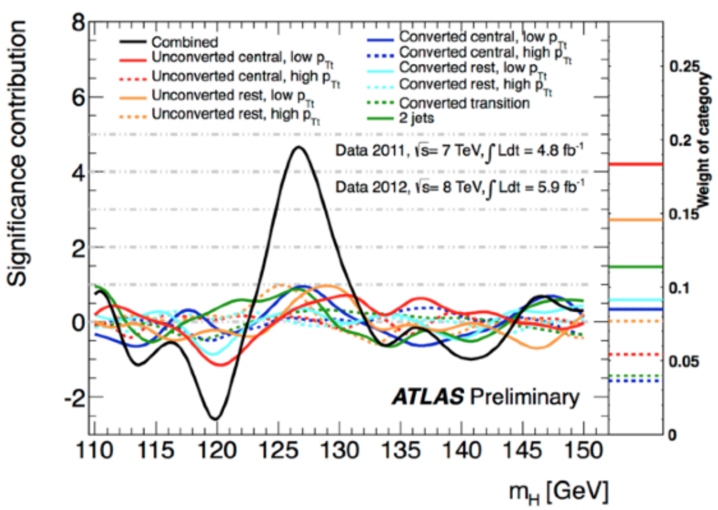


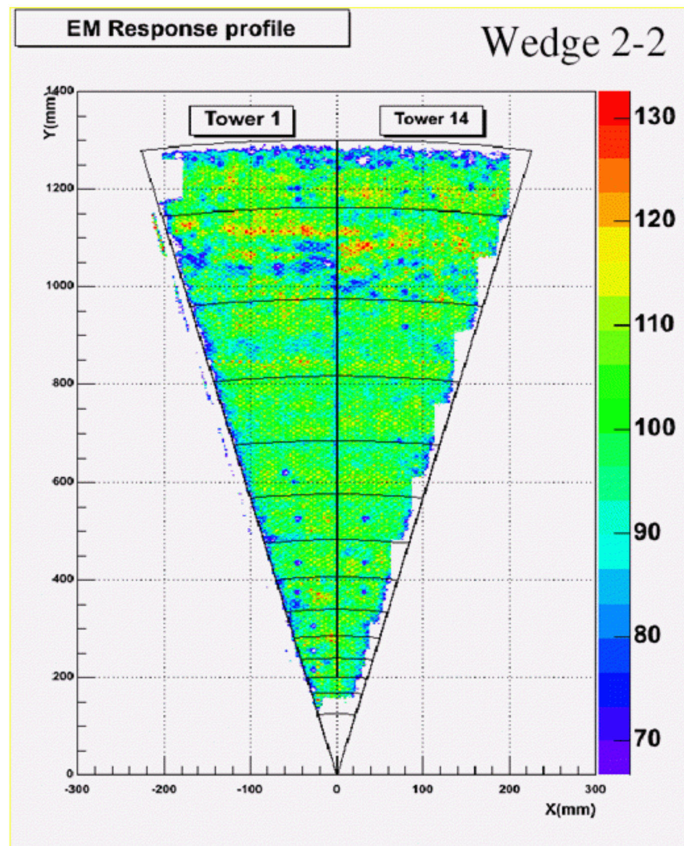
More Example Plots

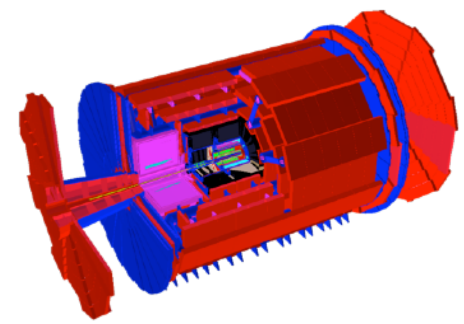


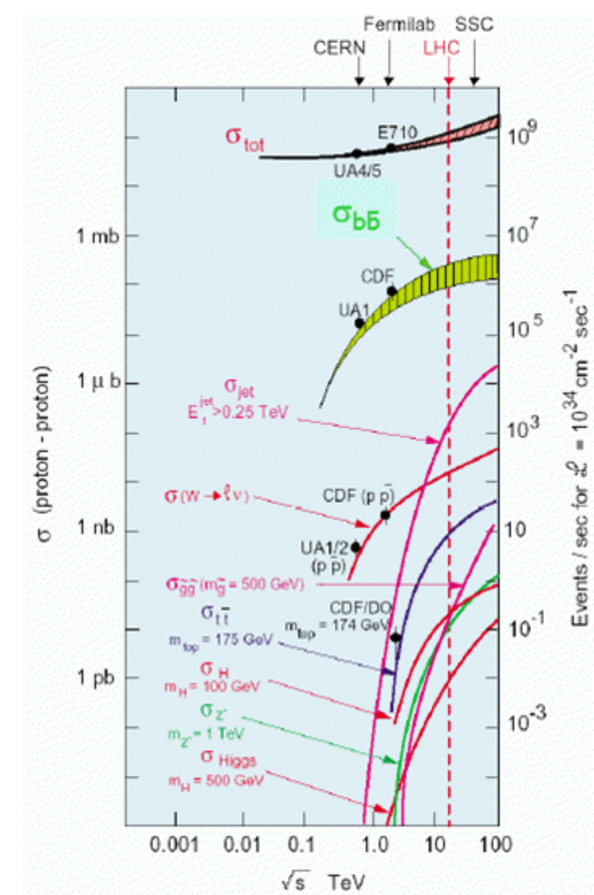


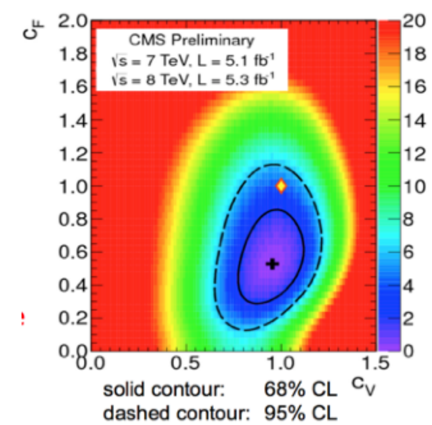


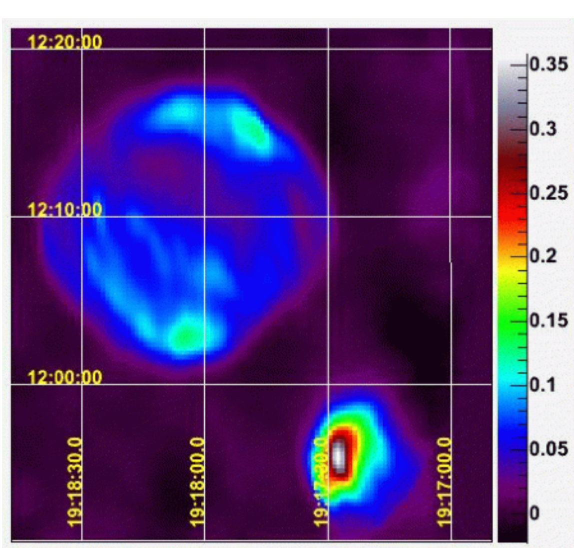


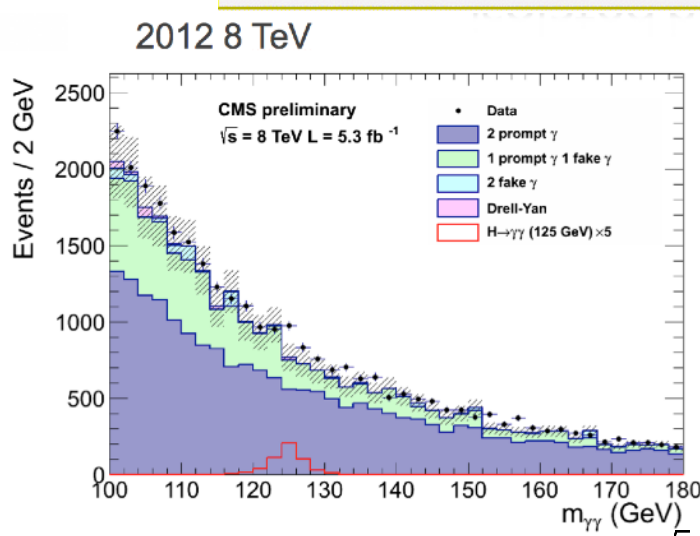


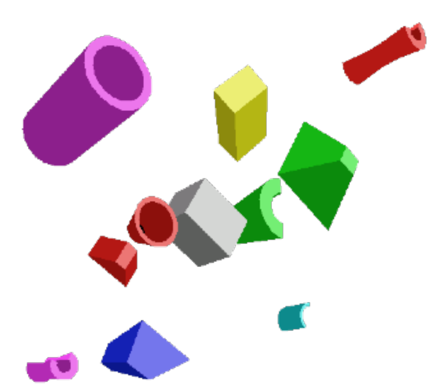


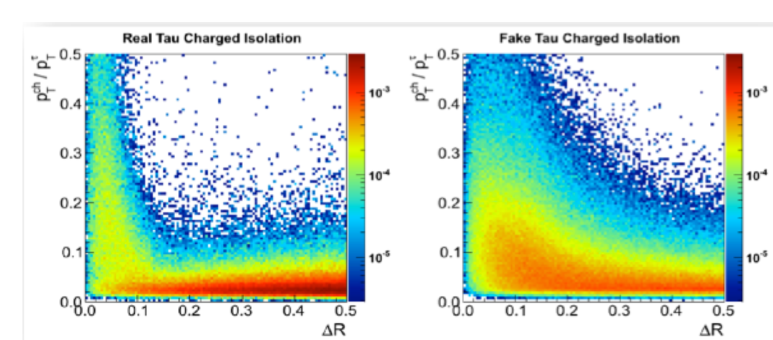




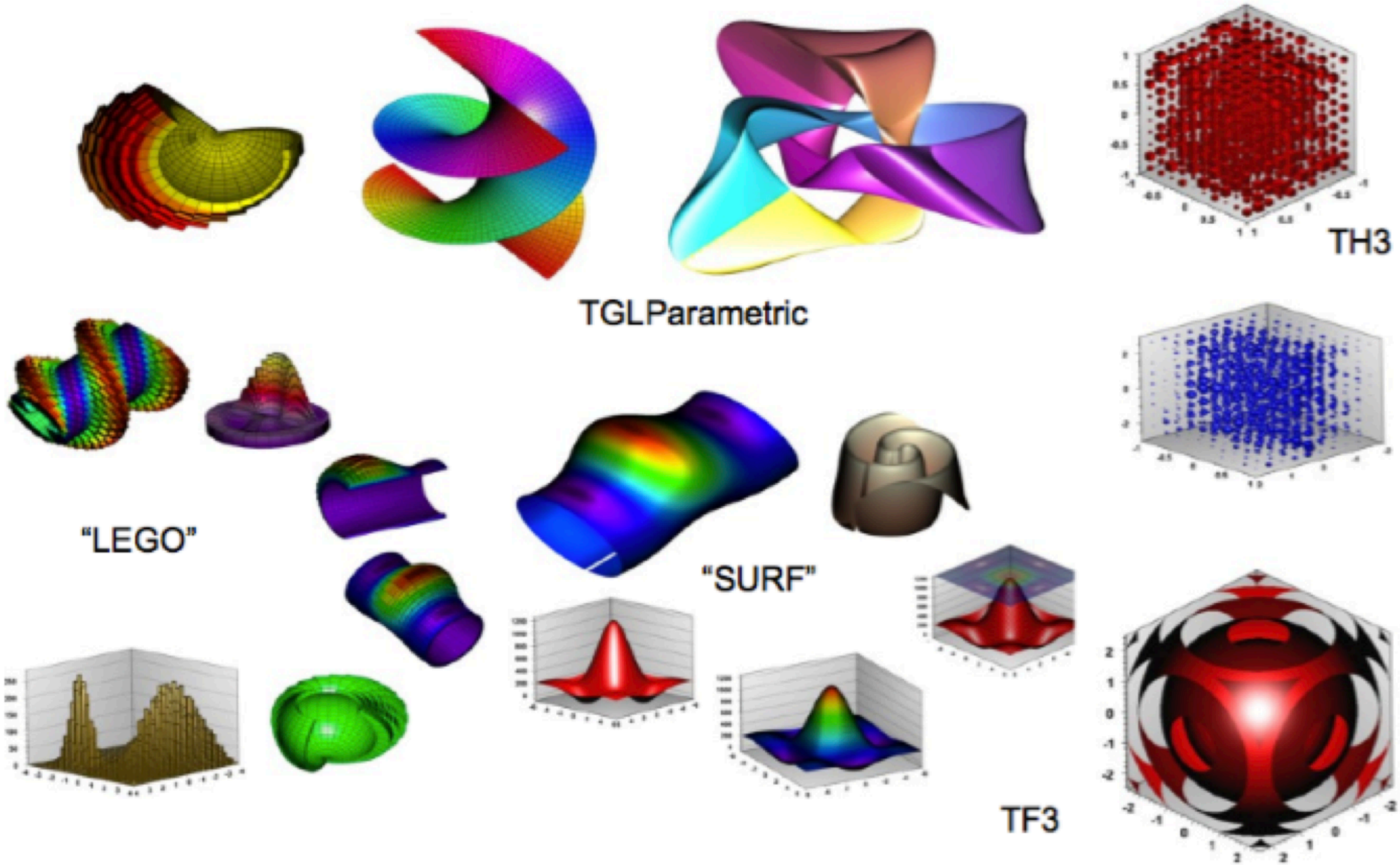








More Example Plots



ROOT Code

ROOT is mostly written in C++

- ➡ Object orientated data handling and analysis framework
- ➡ C++ interpreter
- ➡ Python bindings (PyRoot)

ROOT is Open Source Project

- ➡ Available under LGPL license
- ➡ First release 1995
- ➡ Many new releases since then!

ROOT is fully cross platform

- ➡ Windows
- ➡ MacOS
- ➡ Unix/Linux

Resources

28 year history + tens of thousands of users

= plenty of resources!

ROOT Primer:

➡ https://root.cern/primer/

ROOT User Guide:

➡ https://root.cern/manual/

ROOT Tutorial:

➡ https://root.cern/tutorials/

ROOT Class Reference:

➡ https://root.cern/doc/master/

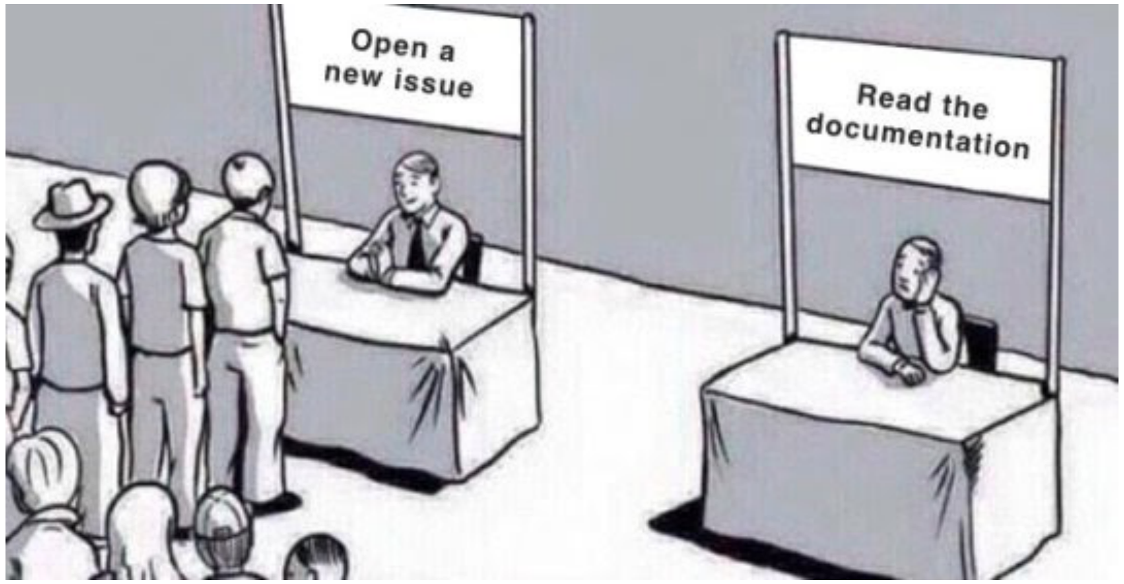
ROOT Forum:

➡ https://root-forum.cern.ch/

General Coding Help:

➡ http://www.cplusplus.com/doc/tutorial/

➡ https://stackoverflow.com/



Installation

Latest release is 6.28.04

➡ https://root.cern/releases/release-62804/

First install dependencies

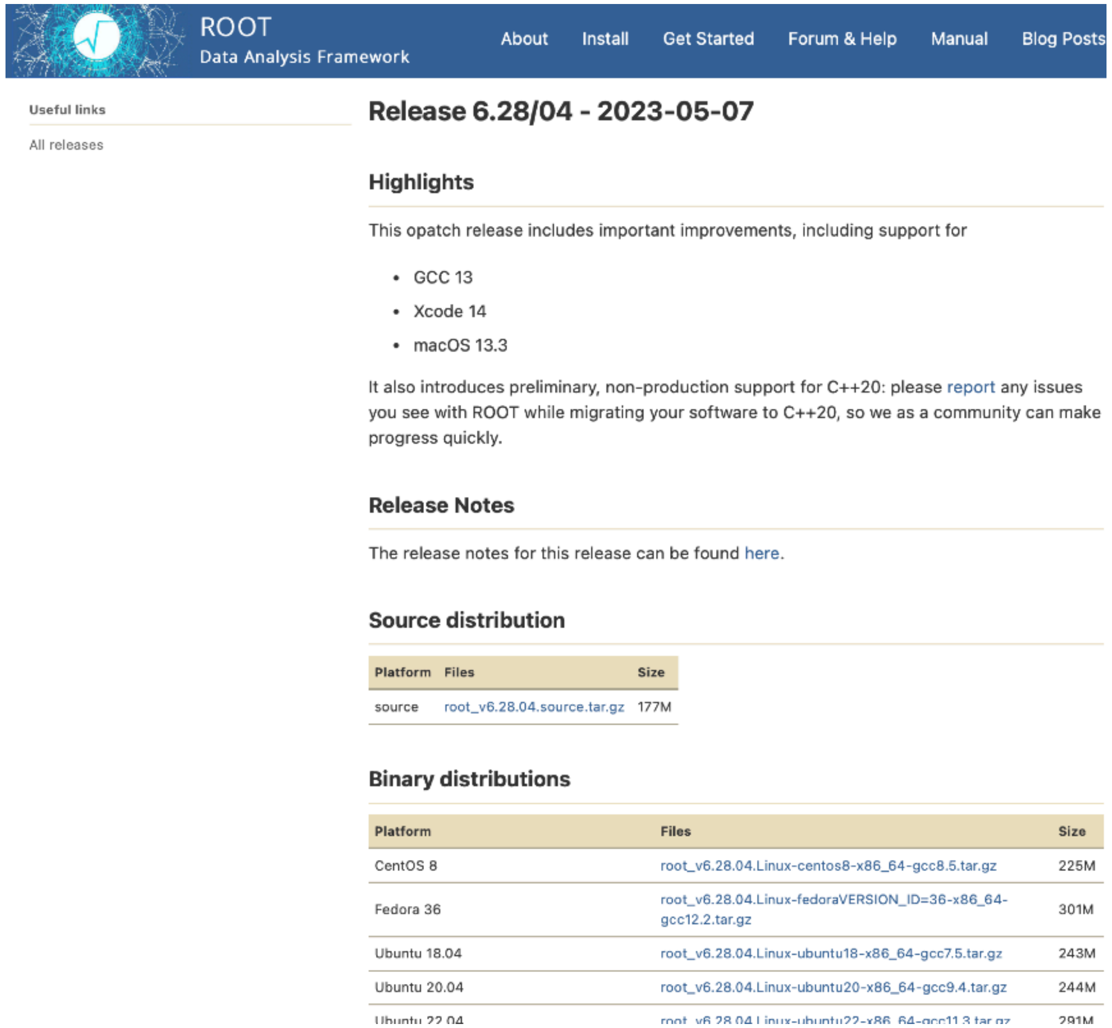
➡ https://root.cern/install/dependencies/

Download and install ROOT

➡ Get source code, or pre-compiled binaries for your operating system

➡ If using code: `./configure && make`

➡ `source bin/thisroot.sh`


ROOT
Data Analysis Framework
About Install Get Started Forum & Help Manual Blog Posts

Useful links
All releases

Release 6.28/04 - 2023-05-07

Highlights

This opatch release includes important improvements, including support for

- GCC 13
- Xcode 14
- macOS 13.3

It also introduces preliminary, non-production support for C++20: please report any issues you see with ROOT while migrating your software to C++20, so we as a community can make progress quickly.

Release Notes

The release notes for this release can be found here.

Source distribution

Platform	Files	Size
source	root_v6.28.04.source.tar.gz	177M

Binary distributions

Platform	Files	Size
CentOS 8	root_v6.28.04.Linux-centos8-x86_64-gcc8.5.tar.gz	225M
Fedora 36	root_v6.28.04.Linux-fedoraVERSION_ID=36-x86_64-gcc12.2.tar.gz	301M
Ubuntu 18.04	root_v6.28.04.Linux-ubuntu18-x86_64-gcc7.5.tar.gz	243M
Ubuntu 20.04	root_v6.28.04.Linux-ubuntu20-x86_64-gcc9.4.tar.gz	244M

More instructions

➡ https://root.cern/install/

C++ && Object Orientated Code Reminders

```
class Car {
  public:                          // Public access specifier
    std::string number_plate;      // Public attribute
  private:                         // Private access specifier
    int chassis_id;                // Private attribute
};

int main() {
  Car myCar;
  myCar.number_plate = "A123CDE";  // Allowed (public)
  myCar.chassis_id = 1234;         // Not allowed (private)
  return 0;
}
```

https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm

```
#include <iostream>

using namespace std;

// Base class
class Shape {
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }

   protected:
      int width;
      int height;
};

// Derived class
class Rectangle: public Shape {
   public:
      int getArea() {
         return (width * height);
      }
};

int main(void) {
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   return 0;
}
```

- ➡ A **class** is a combination of **methods** (functions) and **attributes** (data values)
- ➡ An **object** is an instance of a class
- ➡ **Members** (methods and attributes) can be accessible from outside the class (**public**), or only with other methods of the same class (**private**)
- ➡ Members accessed with `className.memberName()`
- ➡ Classes can inherit from other classes
  - ➡ **Protected** members cannot be accessed outside the class, but can be accessed by inherited classes
- ➡ Use **.** to access members of classes, **->** to access members of pointers to classes

Now let’s get started!

SNO+ Command Line Interpreter

UNIVERSITY OF OXFORD

ROOT has a built in C++ interpreter, CLING

➡ C++ interactive shell

➡ Just in time compilation

```
WillsNetwork:~ wparker$
WillsNetwork:~ wparker$ root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.26/06                        https://root.cern |
  | (c) 1995-2021, The ROOT Team; conception: R. Brun, F. Rademakers |
  | Built for macosx64 on Jul 28 2022, 18:08:51                      |
  | From tags/v6-26-06@v6-26-06                                      |
  | With Apple clang version 14.0.0 (clang-1400.0.29.202)            |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'       |
   ------------------------------------------------------------------

root [0]
```

Commands

- Exit: `.q`
- Show list of commands: `.?`
- Shell commands: `.!` Eg. `.!ls`
- Execute a macro: `.x <file_name>`
- Load a macro: `.L <file_name>`
- Compile a macro: `.L <file_name>+`
- `.help` for more!

Interpreting C++

```
root [0]
root [0] for (int i=0; i<5; i++) {cout << i << endl;}; cout << "done!" << endl ;
0
1
2
3
4
done!
root [1]
```

ROOT as a calculator

```
root [0]
root [0]
root [0] sqrt(36) + 4
(double) 10.000000
root [1] TMath::Pi()
(double) 3.1415927
root [2] pow(TMath::Pi(),2)
(double) 9.8696044
root [3]
```

Declaring and Using ROOT Classes

```
root [0] TH1D("histogram_name", "histogram title", 100, 0, 100)
(TH1D) Name: histogram_name Title: histogram title NbinsX: 100
root [1] histogram_name->GetXaxis()->GetNbins()
(int) 100
root [2]
```

Macros

- ➡ Running multiple lines of code on the command line can be time consuming
- ➡ Instead write macros in a text editor, and this gets interpreted my CLING

```
void exampleMacro( ) {

  TH1D* h = new TH1D("histogram_name", "histogram title", 100, 0, 100);

  std::cout << h->GetXaxis()->GetNbins() << std::endl;

  for (int i=0; i<5; i++) {
    std::cout << i << endl;
  }

  std::cout << "done!" << std::endl;

}
```

Load and Run

```
root [0] .L exampleMacro.C
root [1] exampleMacro()
100
0
1
2
3
4
done!
root [2]
```

Run from Outside ROOT

```
WillsNetwork:examples wparker$ root -l exampleMacro.C
root [0]
Processing exampleMacro.C...
100
0
1
2
3
4
done!
root [1]
```

Options

- -b batch mode (no graphics)
- -l don't show ROOT banner
- -q exit on completion
- -h show other options!

Compiling Macros

Use ACLiC (Automatic Compiler of Libraries for CLING)

- ➡ Faster execution. For longer code this outweighs overhead of compilation time
- ➡ If rapidly editing and rerunning, scripting often quicker than recompiling each time

```
#include "TH1D.h"
#include <iostream>

void example_function( int count_limit ){

  TH1D* h = new TH1D("histogram_name", "histogram title", 100, 0, 100);

  std::cout << h->GetXaxis()->GetNbins() << std::endl;

  for (int i=0; i<count_limit; i++) {
    std::cout << i << endl;
  }

  std::cout << "done!" << std::endl;

}
```

```
root [0] .L exampleMacro2.cc+
root [1] example_function(5)
100
0
1
2
3
4
done!
root [2]
```

Basic Classes / Namespaces

TMath

The `TMath` namespace provides a selection of common mathematical variables and functions

- ➡ Mathematical constants
- ➡ Trigonometric and other mathematical functions
- ➡ Statistical functions eg. mean + RMS of arrays, probability distributions
- ➡ Specialised mathematical functions eg. Bessel functions
- ➡ https://root.cern.ch/doc/master/namespaceMath.html

```
root [0] a = TMath::Pi()/2
(double) 1.5707963
root [1] b = TMath::Sin(a)
(double) 1.0000000
root [2] TMath::Gaus(a, 0, 1)
(double) 0.29121293
root [3] 
```

```
void exampleTMath( ){

  double a = TMath::Pi()/2;
  std::cout << a << std::endl;

  double b = TMath::Sin(a);
  std::cout << b << std::endl;

  std::cout << TMath::Gaus(a, 0, 1) << std::endl;
}
```

Histograms

Bin variables and plot their frequency

- ➡ Histograms are commonly used in high energy physics to visualise data
- ➡ `TH1D` is 1D histogram of doubles (`TH1F` for floats, `TH1I` for integers)
- ➡ `TH1D (“name”, “title”, nBins, minX, maxX)`
- ➡ Can have uniform or variable bin sizes
- ➡ Can `Add`, `Divide`, `Scale` etc.
- ➡ https://root.cern.ch/doc/master/classTH1D.html

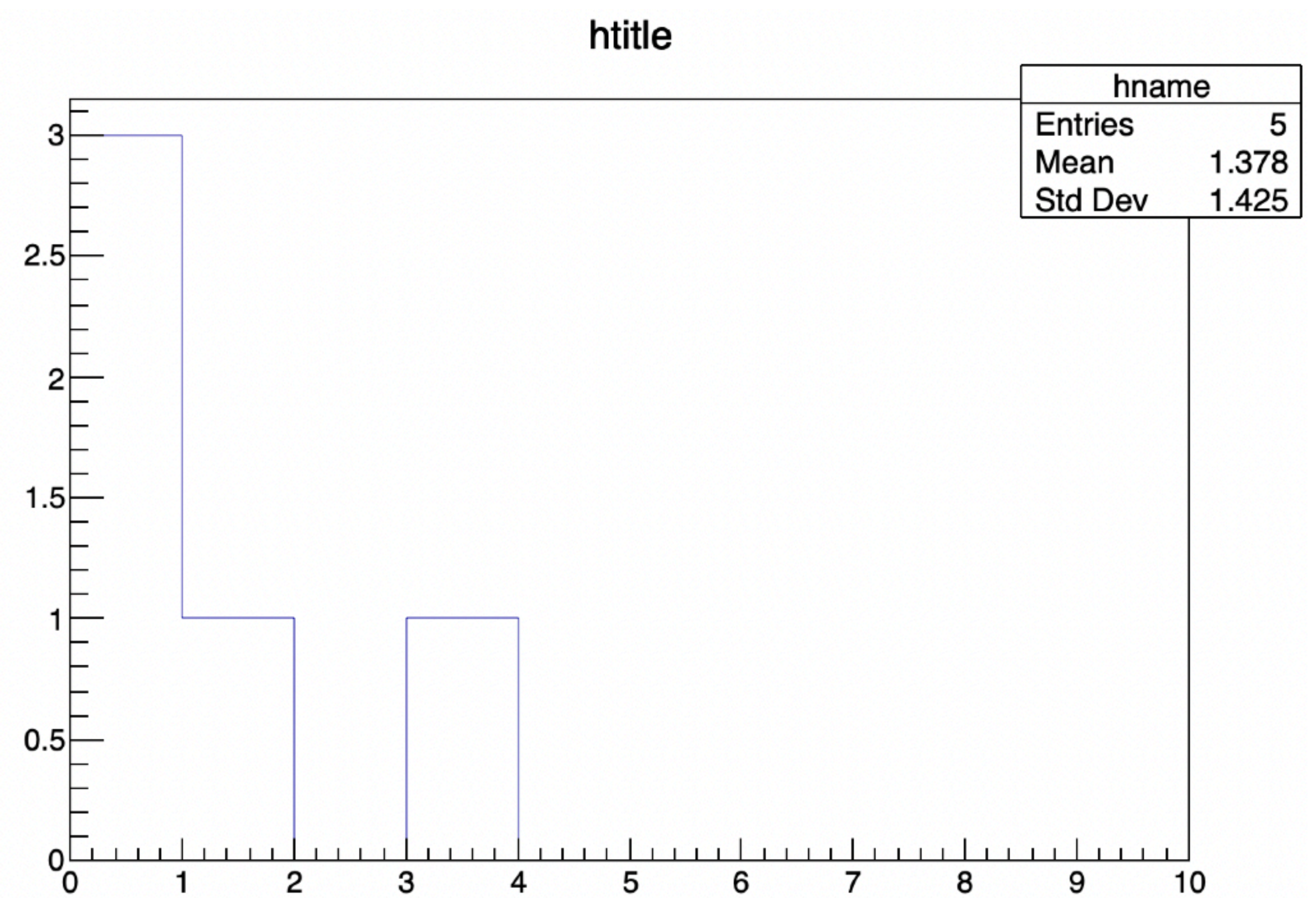
```
void exampleTH1D( ){

  TH1D* h1 = new TH1D("hname", "htitle", 10, 0, 10);

  h1->FillRandom("gaus", 5);

  h1->Draw();
}
```

Alternative Filling Methods

```
root [2] h->SetBinContent(1,4)
root [3] h->AddBinContent(2,5)
root [4] h->Fill(4)
```

Extracting Information

```
root [6] h->GetMean()
(double) 1.3000000
root [7] h->GetRMS()
(double) 1.1661904
root [8] h->Integral()
(double) 5.0000000
```

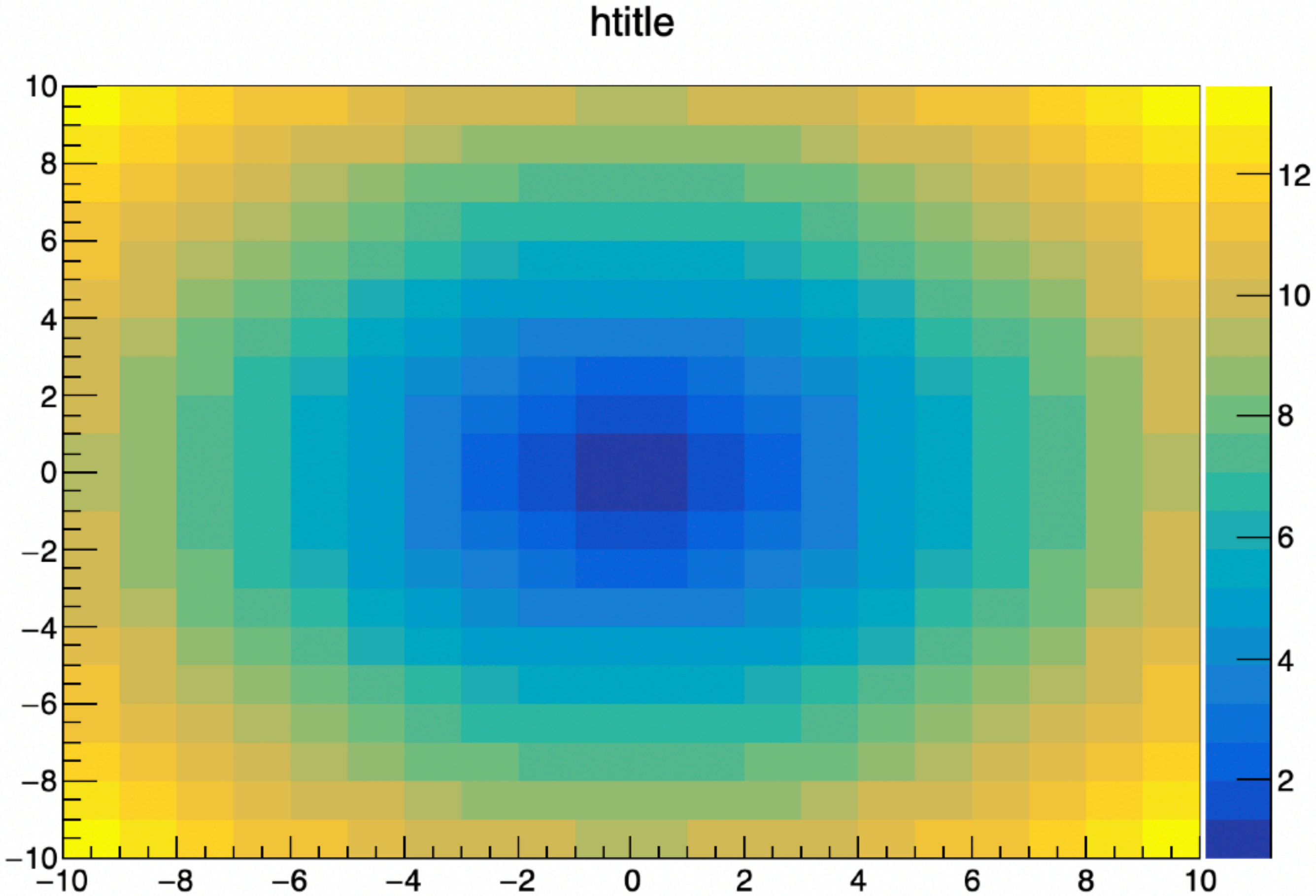


2D Histograms

We can also bin in 2D (and higher!)

- ➡ `TH2s` inherit from `TH1s`
- ➡ https://root.cern.ch/doc/master/classTH2.html

```
void exampleTH2D( ){

  TH2D* h1 = new TH2D("hname", "htitle", 20, -10, 10, 20, -10, 10);

  for(int xbin = 1; xbin <= h1->GetXaxis()->GetNbins(); xbin++) {
    for(int ybin = 1; ybin <= h1->GetYaxis()->GetNbins(); ybin++) {

      double x = h1->GetXaxis()->GetBinCenter(xbin);
      double y = h1->GetYaxis()->GetBinCenter(ybin);

      double r = TMath::Sqrt(x*x + y*y);


      h1->SetBinContent(xbin, ybin, r);

    }
  }

  h1->Draw("colz");
}
```



THistPainter

Draw histograms in different ways using different options: `h->Draw("option")`

➡ Table shows options for all 1D and 2D histograms

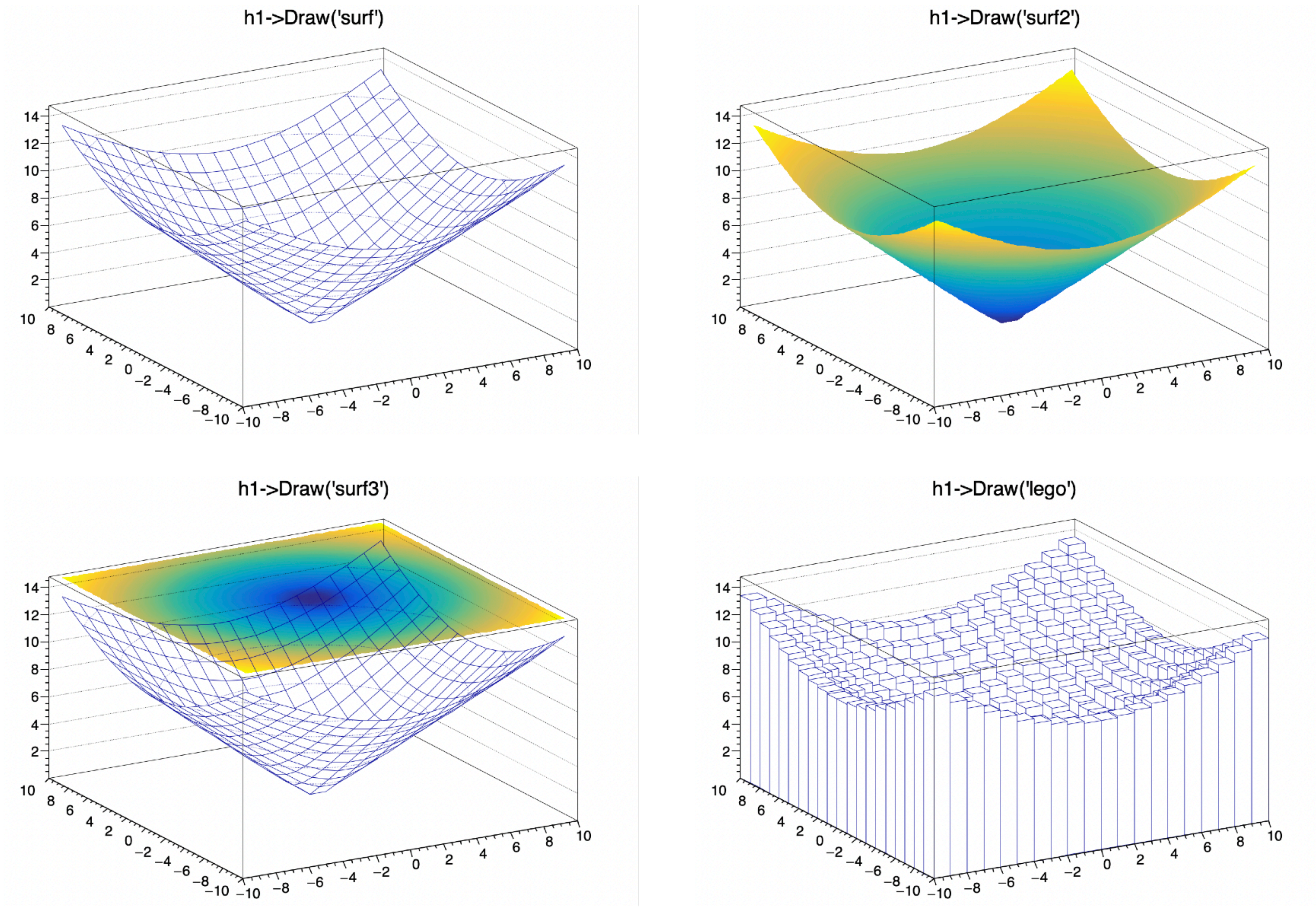
➡ Many more options for specific histogram types

➡ https://root.cern.ch/doc/v608/classTHistPainter.html

Options supported for 1D and 2D histograms

Option	Description
"E"	Draw error bars.
"AXIS"	Draw only axis.
"AXIG"	Draw only grid (if the grid is requested).
"HIST"	When an histogram has errors it is visualized by default with error bars. To visualize it without errors use the option "HIST" together with the required option (eg "hist same c"). The "HIST" option can also be used to plot only the histogram and not the associated function(s).
"FUNC"	When an histogram has a fitted function, this option allows to draw the fit result only.
"SAME"	Superimpose on previous picture in the same pad.
"LEGO"	Draw a lego plot with hidden line removal.
"LEGO1"	Draw a lego plot with hidden surface removal.
"LEGO2"	Draw a lego plot using colors to show the cell contents When the option "0" is used with any LEGO option, the empty bins are not drawn.
"LEGO3"	Draw a lego plot with hidden surface removal, like LEGO1 but the border lines of each lego-bar are not drawn.
"LEGO4"	Draw a lego plot with hidden surface removal, like LEGO1 but without the shadow effect on each lego-bar.
"TEXT"	Draw bin contents as text (format set via `gStyle->SetPaintTextFormat`).
"TEXTnn"	Draw bin contents as text at angle nn (0 < nn < 90).
"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.

TH2D Draw Options



This is just a selection!

TCanvas

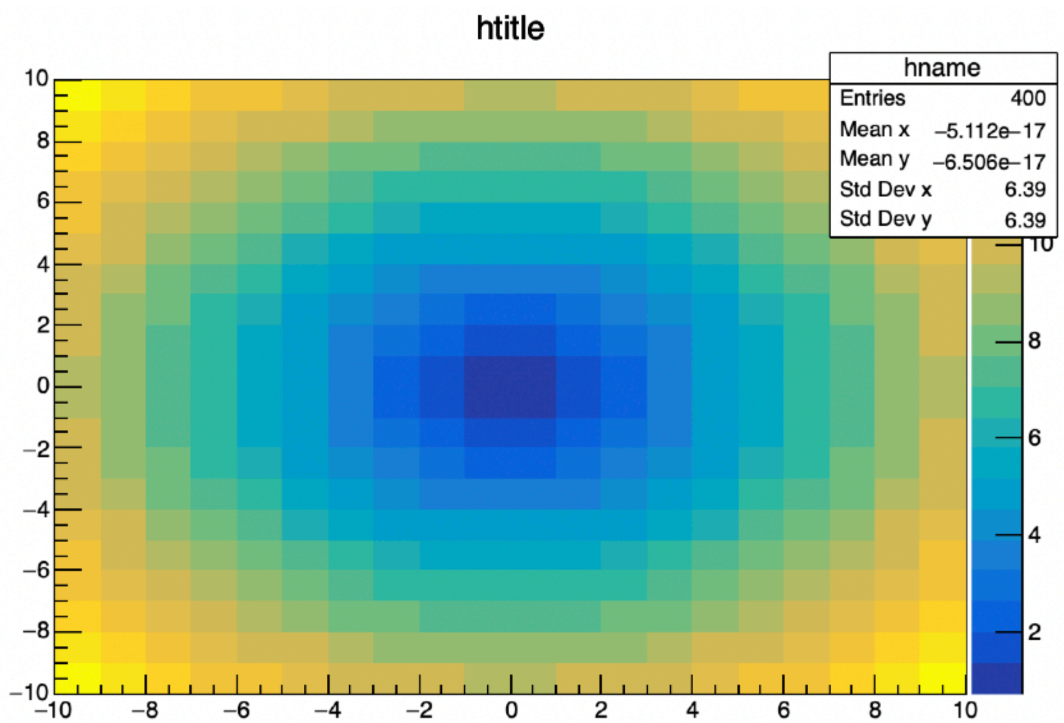
`TCanvas` is an area where objects can be drawn

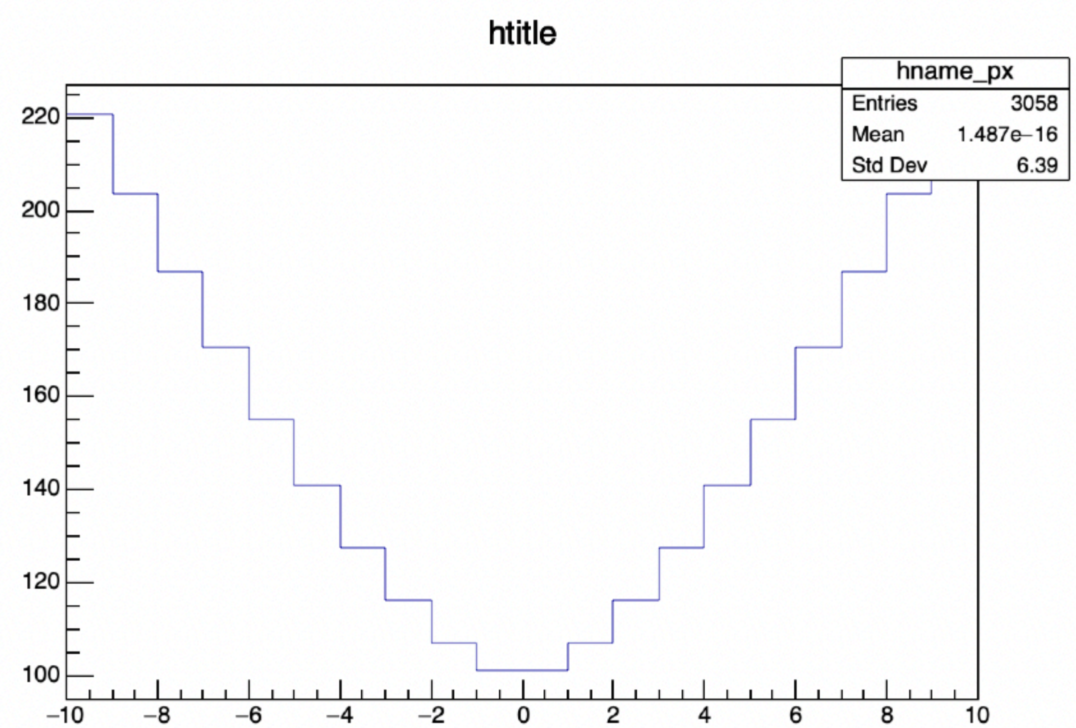
➡ `Draw()` makes a default `TCanvas`, but declaring your own gives you more control

➡ `TCanvas` can be divided into multiple pads

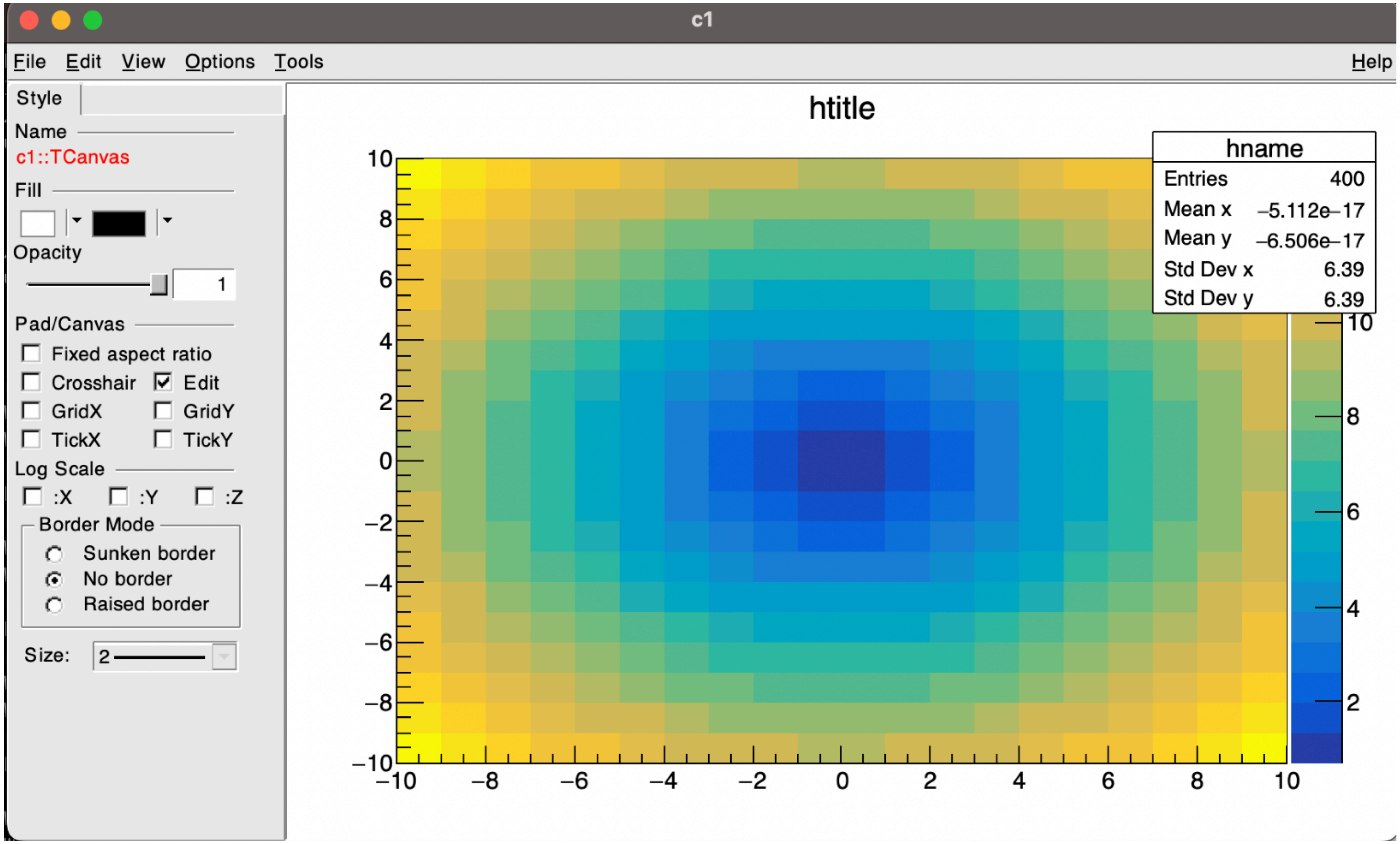
➡ https://root.cern/doc/master/classTCanvas.html

```
void exampleTCanvas( ){

  TCanvas* c1 = new TCanvas("cname", "ctitle", 600, 1500);
  c1->Divide(1,2);

  TH2D* h1 = new TH2D("hname", "htitle", 20, -10, 10, 20, -10, 10);

  for(int xbin = 1; xbin <= 20; xbin++) {
    for(int ybin = 1; ybin <= 20; ybin++) {

      double x = h1->GetXaxis()->GetBinCenter(xbin);
      double y = h1->GetYaxis()->GetBinCenter(ybin);

      double r = TMath::Sqrt(x*x + y*y);


      h1->SetBinContent(xbin, ybin, r);

    }
  }
  c1->cd(1);
  h1->Draw("colz");

  c1->cd(2);
  h1->ProjectionX()->Draw();
}
```





ROOT GUI

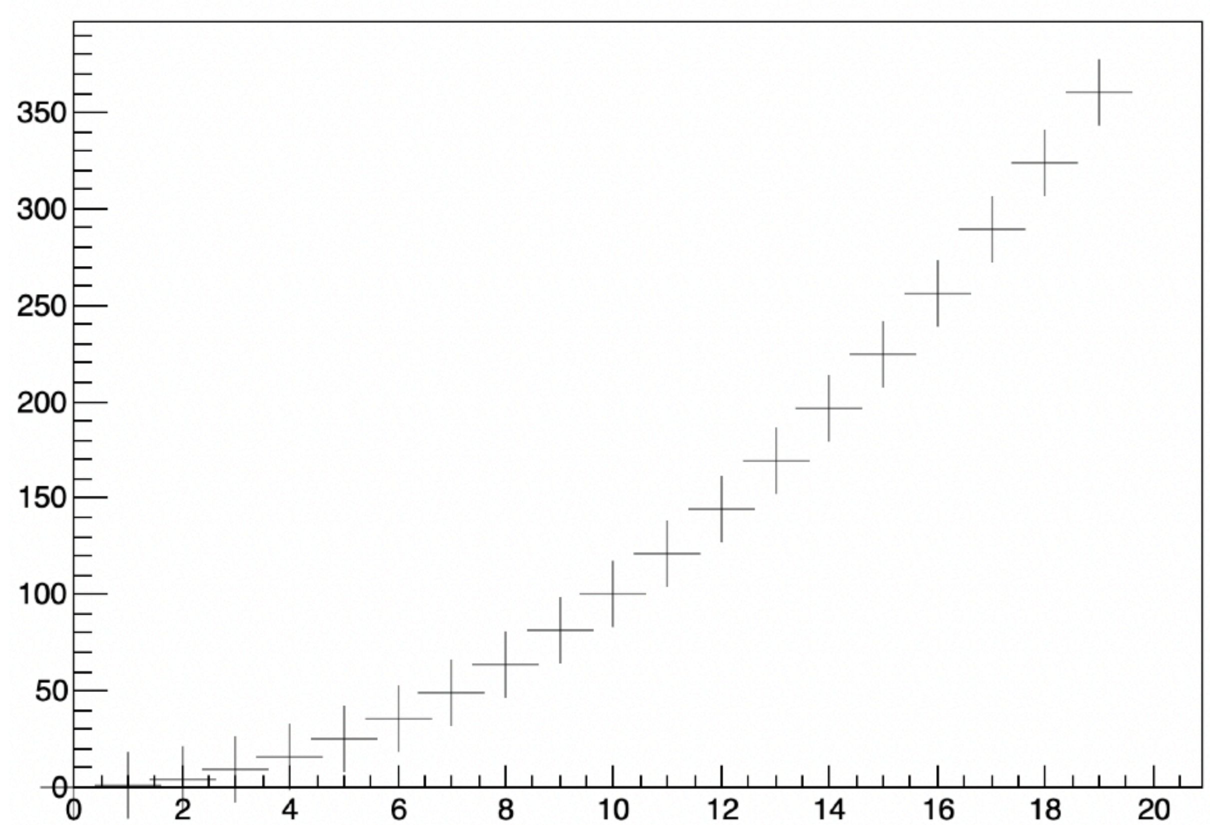

c1
File
Edit
View
Options
Tools
Help
Style
Name
c1::TCanvas
Fill
Opacity
1
Pad/Canvas
Fixed aspect ratio
Crosshair
Edit
GridX
GridY
TickX
TickY
Log Scale
:X
:Y
:Z
Border Mode
Sunken border
No border
Raised border
Size:
2
htitle
hname
Entries 400
Mean x −5.112e−17
Mean y −6.506e−17
Std Dev x 6.39
Std Dev y 6.39
10
8
6
4
2
0
−2
−4
−6
−8
−10
−10 −8 −6 −4 −2 0 2 4 6 8 10
10
8
6
4
2

TGraphs

`TGraphs` are a collection of distinct points

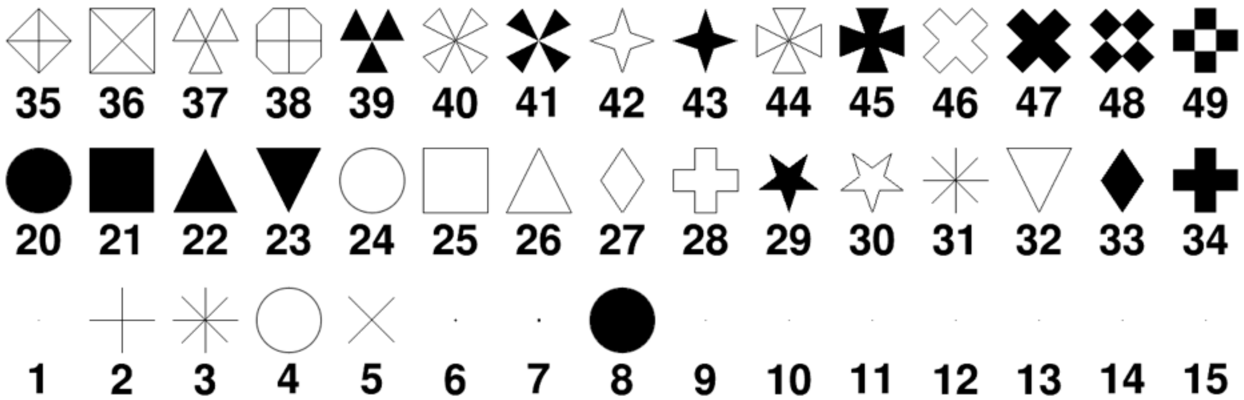
- ➡ `TGraph`: x-y plot with no error bars
- ➡ `TGraphErrors`: x-y plot with error bars
- ➡ `TGraphAsymmErrors`: x-y plot with asymmetric error bars
- ➡ https://root.cern.ch/doc/master/classTGraph.html

```
void exampleTGraph( ){

  TGraph* g = new TGraph();

  int npoints = 20;

  for(int ipoint = 0; ipoint < npoints; ipoint++) {

    double x = ipoint;
    double y = x*x;

    g->SetPoint(ipoint, x, y);
  }

  g->SetMarkerStyle(2);
  g->SetMarkerSize(4);
  g->Draw("AP");
}
```



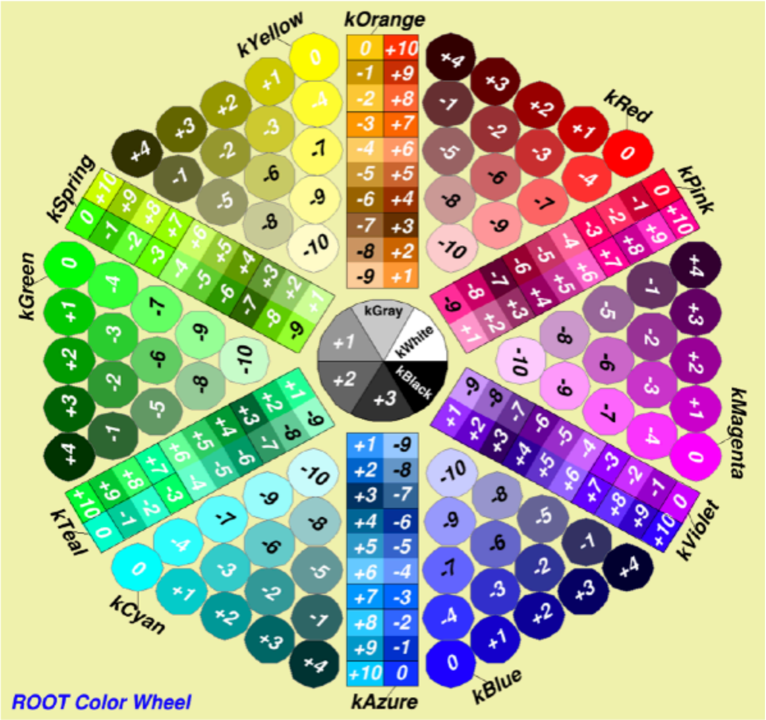
Drawing

TAttMarker



- `SetMarkerStyle( int )`
- `SetMarkerSize( int )`
- `SetMarkerColor( TColor )`
- https://root.cern.ch/doc/master/classTAttMarker.html

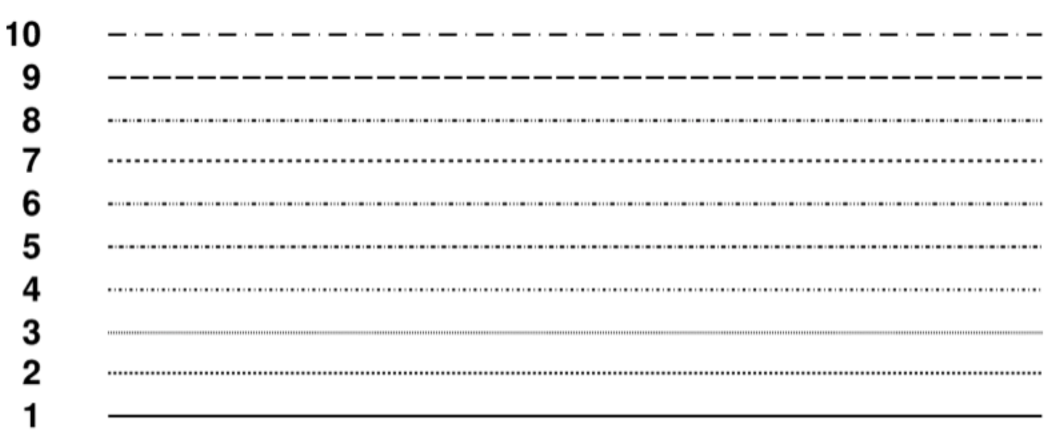
TColor



- https://root.cern.ch/doc/master/classTColor.html

TAttLine

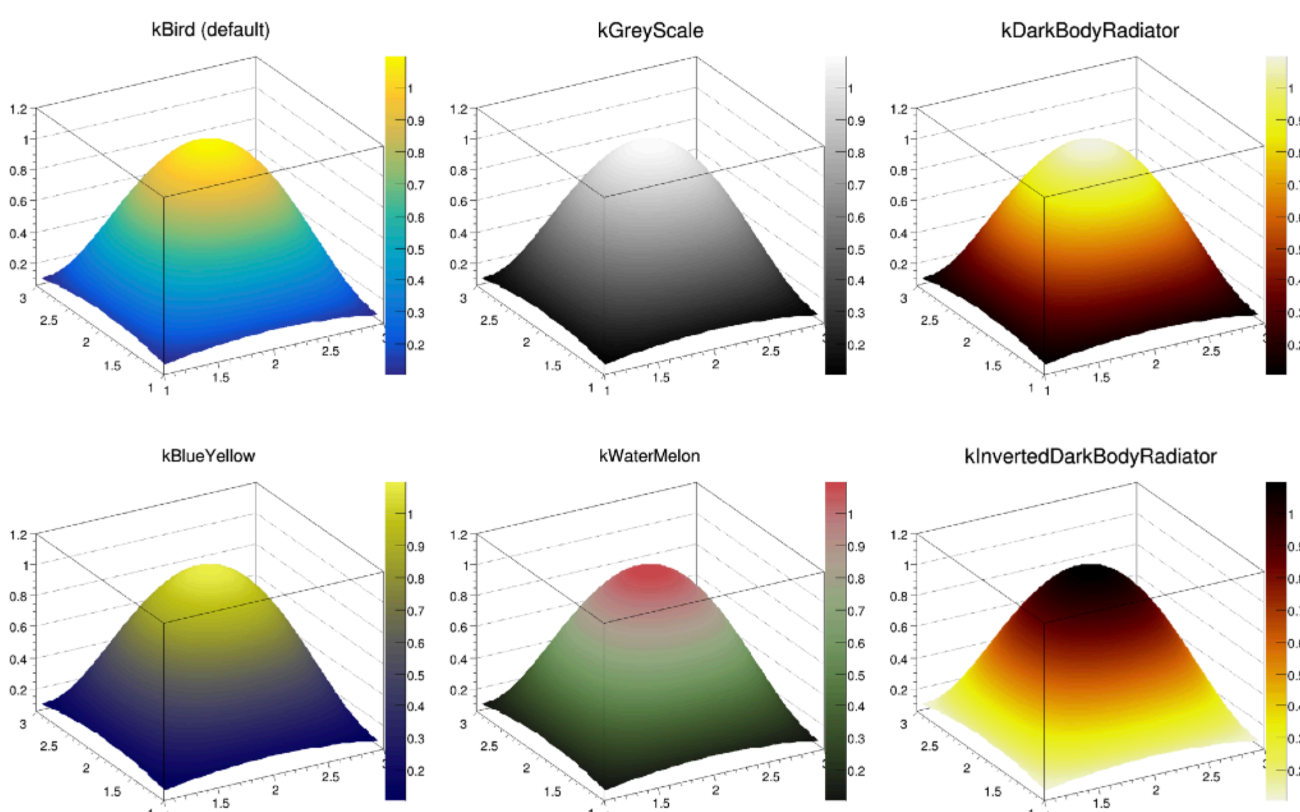
- `SetLineStyle( int )`
- `SetLineWidth( int )`
- `SetLineColor( TColor )`
- https://root.cern.ch/doc/master/classTAttLine.html



TColor Palettes

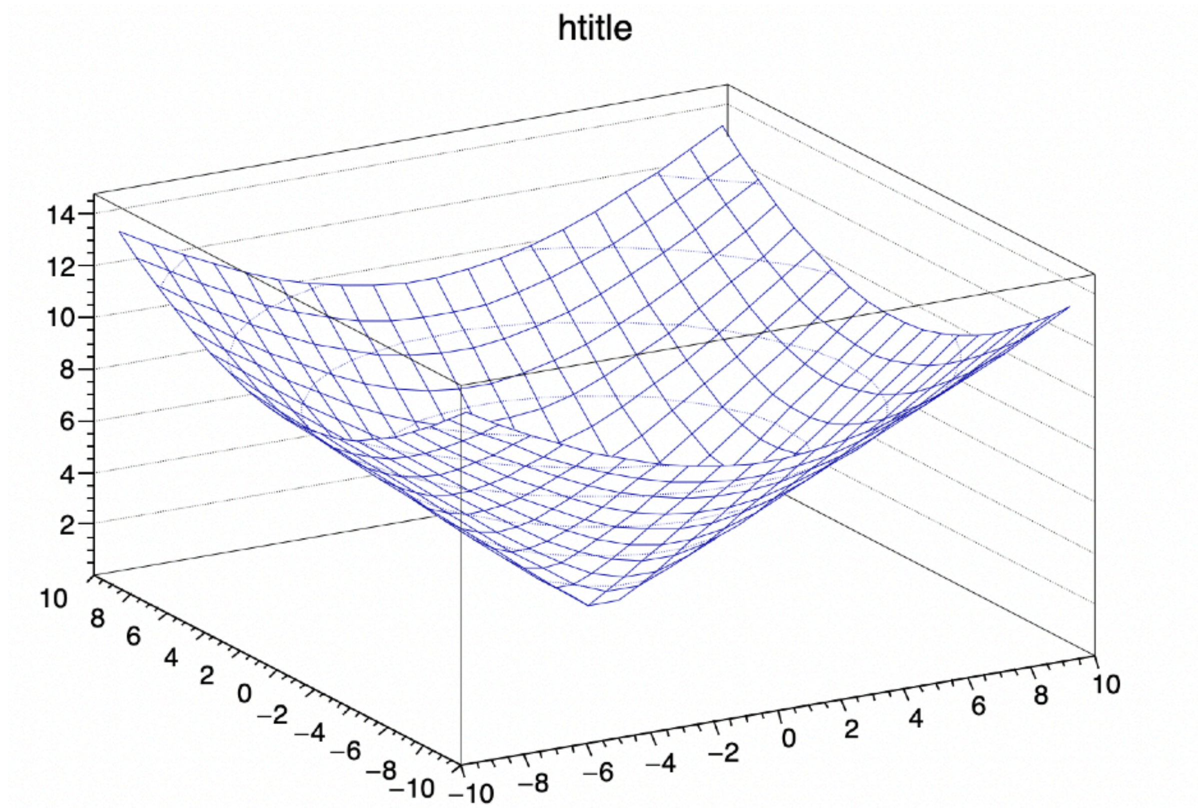
Many different Palettes to choose from

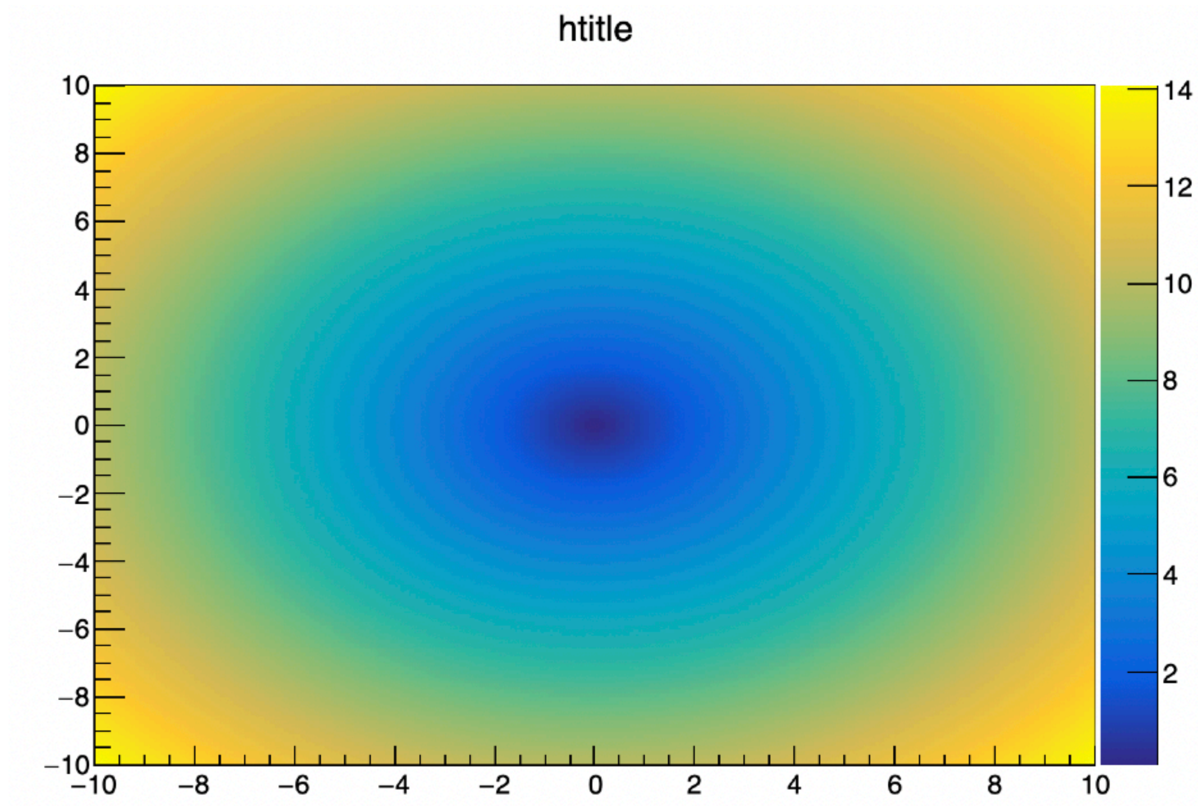
- ➡ `gStyle->SetPalette( int )`
- ➡ `gStyle` refers to current `TStyle`
- ➡ https://root.cern.ch/doc/master/classTStyle.html
- ➡ Can define your own `TStyle` and palette

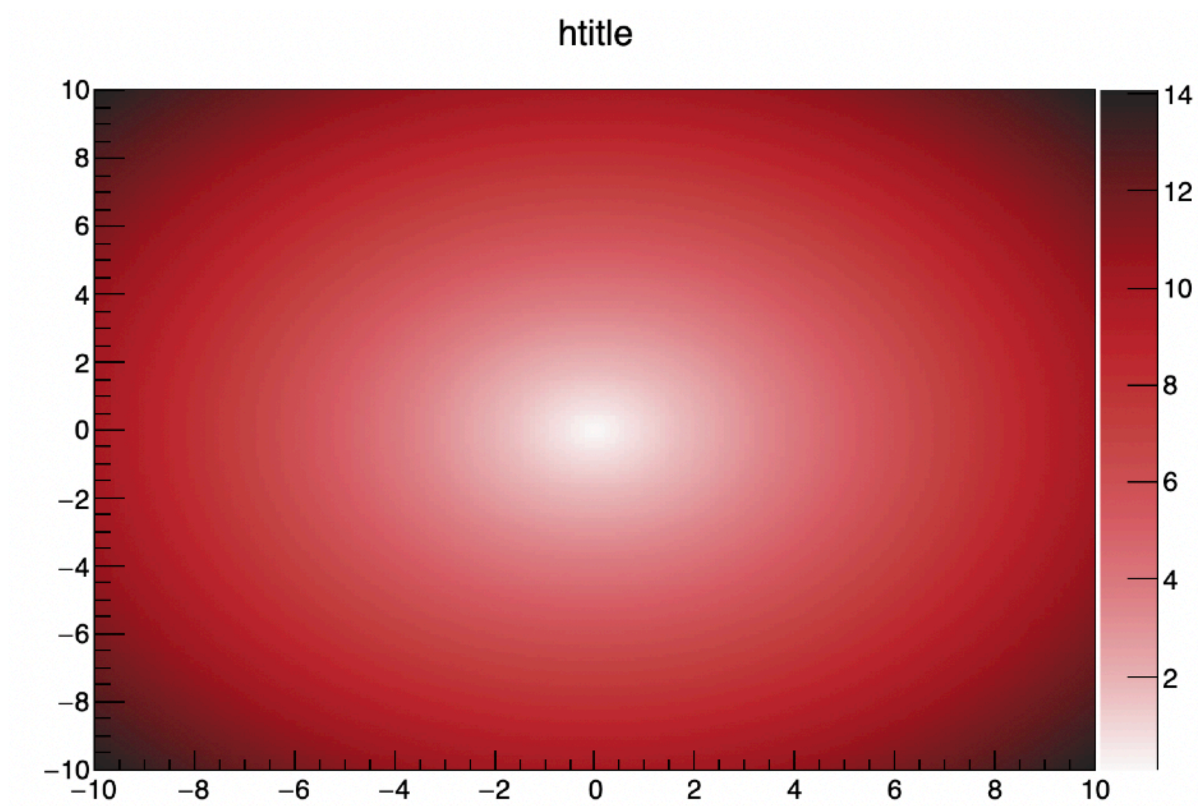



```
kDeepSea=51,            kGreyScale=52,      kDarkBodyRadiator=53,
kBlueYellow= 54,        kRainBow=55,        kInvertedDarkBodyRadiator=56,
kBird=57,               kCubehelix=58,      kGreenRedViolet=59,
kBlueRedYellow=60,      kOcean=61,          kColorPrintableOnGrey=62,
kAlpine=63,             kAquamarine=64,     kArmy=65,
kAtlantic=66,           kAurora=67,         kAvocado=68,
kBeach=69,              kBlackBody=70,      kBlueGreenYellow=71,
kBrownCyan=72,          kCMYK=73,           kCandy=74,
kCherry=75,             kCoffee=76,         kDarkRainBow=77,
kDarkTerrain=78,        kFall=79,           kFruitPunch=80,
kFuchsia=81,            kGreyYellow=82,     kGreenBrownTerrain=83,
kGreenPink=84,          kIsland=85,         kLake=86,
kLightTemperature=87,   kLightTerrain=88,   kMint=89,
kNeon=90,               kPastel=91,         kPearl=92,
kPigeon=93,             kPlum=94,           kRedBlue=95,
kRose=96,               kRust=97,           kSandyTerrain=98,
kSienna=99,             kSolar=100,         kSouthWest=101,
kStarryNight=102,       kSunset=103,        kTemperatureMap=104,
kThermometer=105,       kValentine=106,     kVisibleSpectrum=107,
kWaterMelon=108,        kCool=109,          kCopper=110,
kGistEarth=111,         kViridis=112,       kCividis=113
```

TColor Palettes







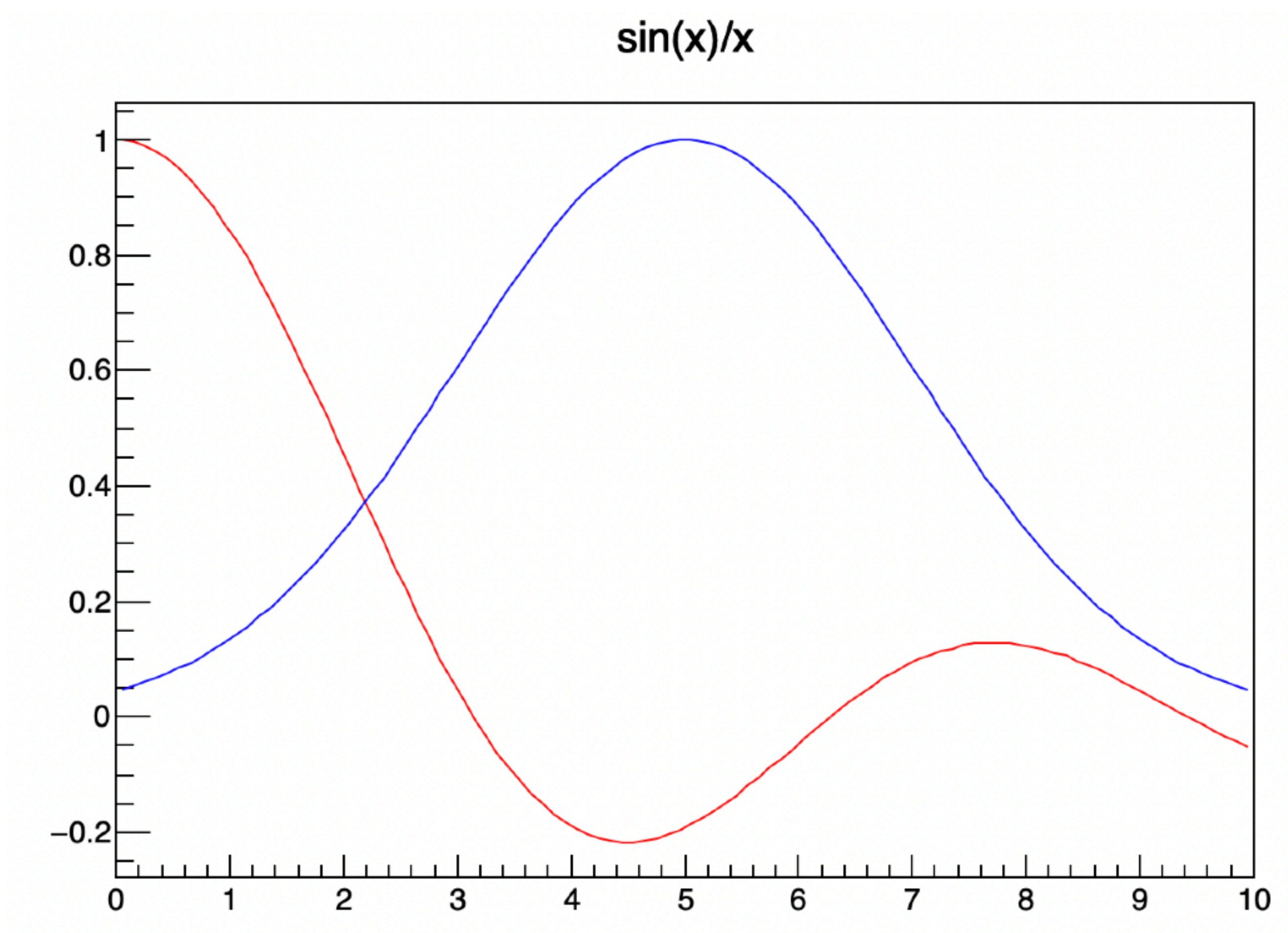
- ➡ With a badly chosen palette, the eye can see boundaries that aren't really there
- ➡ Also be careful to be colour vision deficiency friendly!

TFunctions

`TF1`, `TF2`, `TF3` classes for 1, 2, & 3 dimensional functions

- ➡ Can be “built-in” `TFormula` or user defined function
- ➡ https://root.cern.ch/doc/master/classTF1.html

```
void exampleTF1( ){

  TF1* f1 = new TF1("f1","sin(x)/x", 0., 10.);
  f1->SetLineColor(kRed);

  TF1* f2 = new TF1("f2","gaus", 0., 10.);
  f2->SetParameter(0, 1.); // Set normalisation of gaus
  f2->SetParameter(1, 5.); // Set mean of gaus
  f2->SetParameter(2, 2.); // Set wdith of gauus
  f2->SetLineColor(kBlue);

  f1->Draw();
  f2->Draw("same");
}
```

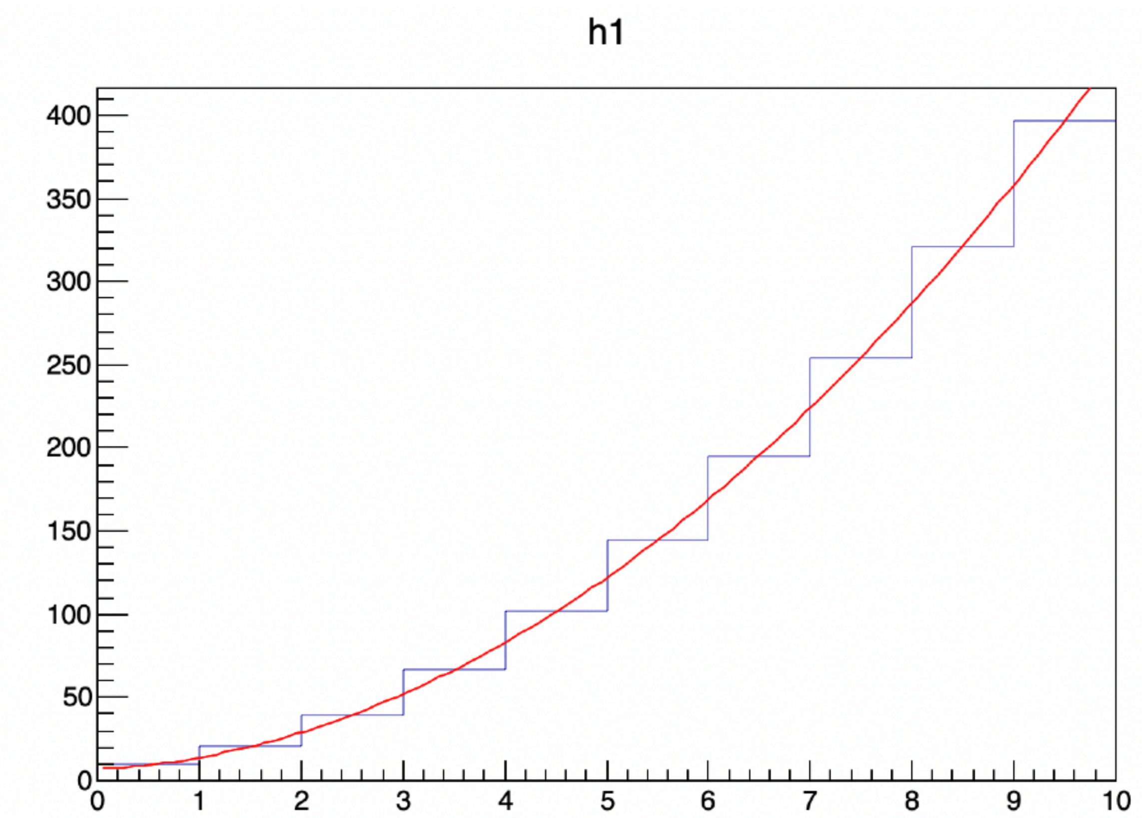


Fitting

Functions can be fit to histograms and graphs

- ➡ Can be “built-in” `TFormula` or user defined function
- ➡ Fit parameters are printed to screen
- ➡ Use `TF1::SetParameter( parameter_number, parameter_value)` for initial guess
- ➡ https://root.cern.ch/root/htmldoc/guides/users-guide/FittingHistograms.html

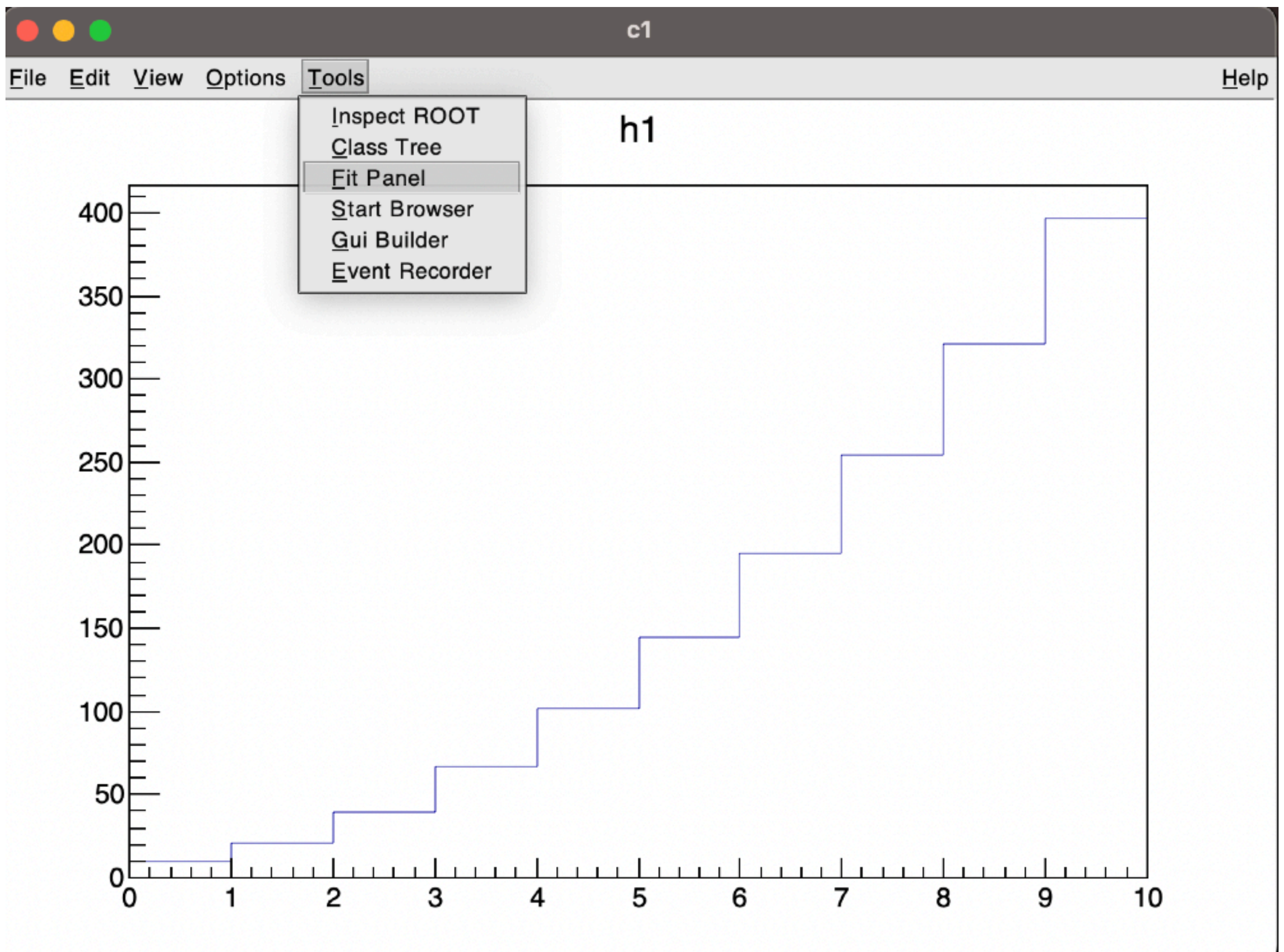
```
void exampleFit( ){

  TH1D* h1 = new TH1D("h1", "h1", 10, 0, 10);

  for(int ibin = 1; ibin <= h1->GetXaxis()->GetNbins(); ibin++) {

    double x = h1->GetXaxis()->GetBinCenter(ibin);
    double y = 4*x*x + 3*x +7;

    h1->SetBinContent(ibin, y);
  }


  TF1* f1 = new TF1("f1","pol2", 0., 10.);
  f1->SetLineColor(kRed);

  h1->Fit("f1");
  h1->Draw();
  f1->Draw("same");
}
```

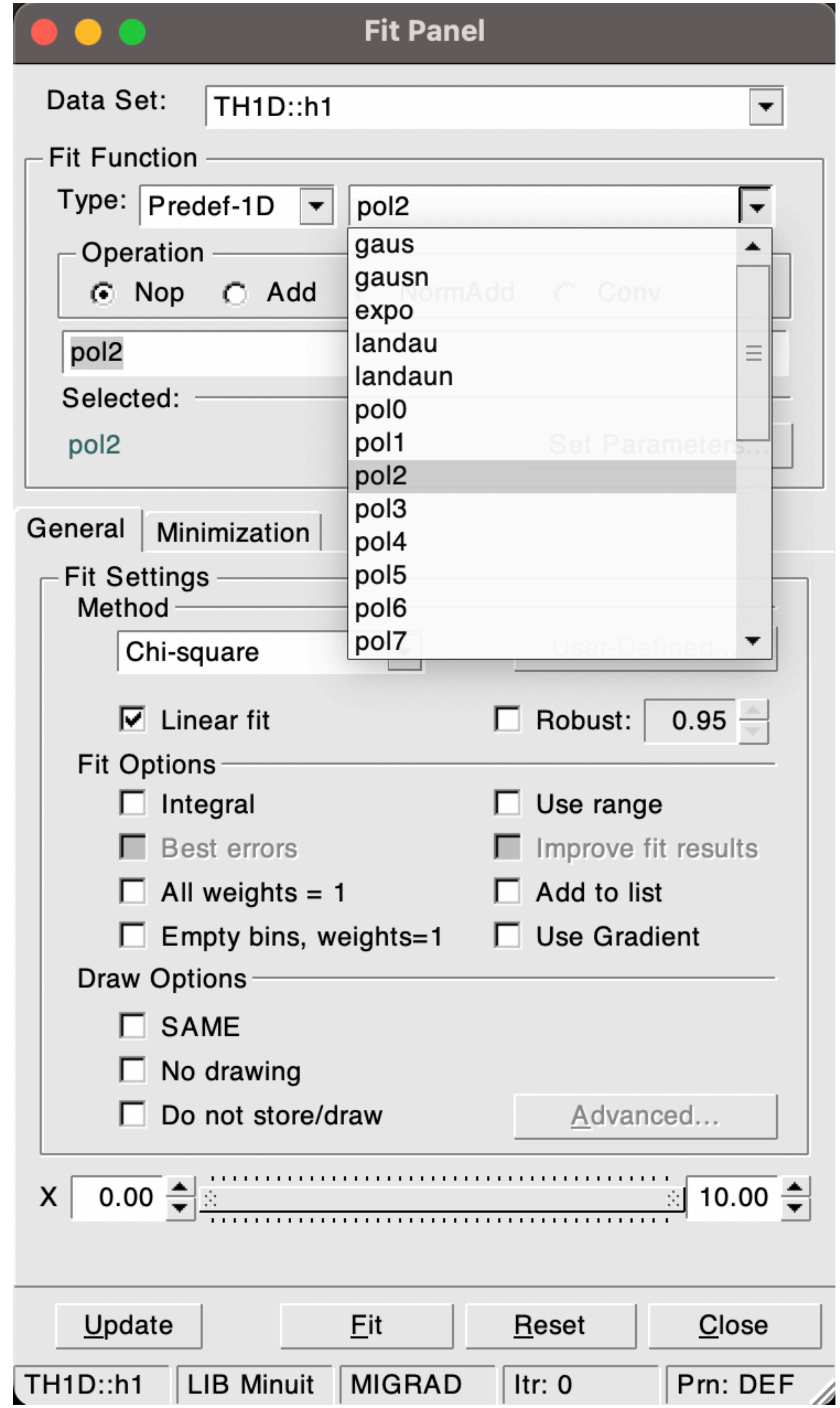



```
****************************************
Minimizer is Linear / Migrad
Chi2                      =  1.50378e-28
NDf                       =            7
p0                        =            7   +/-   3.95013
p1                        =            3   +/-   3.19478
p2                        =            4   +/-   0.403806
```

FitPanel

Can also use GUI to perform fit





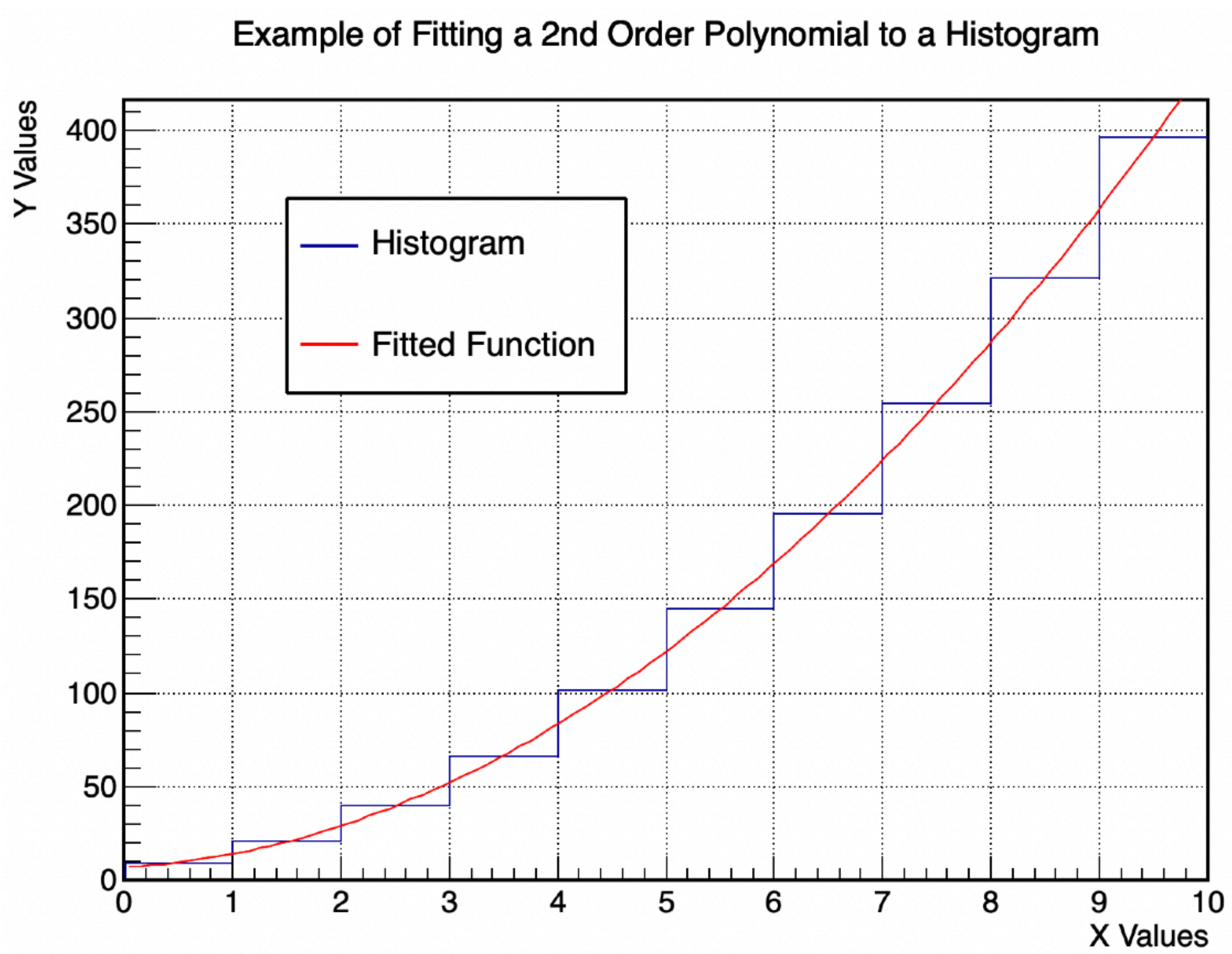
TLegend

`TLegend` class can be drawn onto `TCanvas`

➡ Each `TLegendEntry` is made of a reference to a ROOT object, a text entry, and an option of a graphical attribute

➡ https://root.cern.ch/doc/master/classTLegend.html

➡ Let's also tidy up axis labels and plot title while we're at it




```
void exampleTLegend( ){

  TCanvas* c1 = new TCanvas("cname", "ctitle", 800, 600);
  gStyle->SetOptStat(0);
  c1->SetFrameLineWidth(2);
  c1->SetGrid(1);

  TH1D* h1 = new TH1D("h1", "h1", 10, 0, 10);

  for(int ibin = 1; ibin <= h1->GetXaxis()->GetNbins(); ibin++) {

    double x = h1->GetXaxis()->GetBinCenter(ibin);
    double y = 4*x*x + 3*x +7;

    h1->SetBinContent(ibin, y);
  }


  TF1* f1 = new TF1("f1","pol2", 0., 10.);
  f1->SetLineColor(kRed);
  f1->SetLineWidth(2);
  f1->SetParameters(0, 10);
  f1->SetParameters(1, 5);
  f1->SetParameters(2, 5);

  TLegend* leg = new TLegend(0.22, 0.6, 0.47, 0.8);
  leg->AddEntry(h1, "Histogram", "l");
  leg->AddEntry(f1, "Fitted Function", "l");
  leg->SetLineWidth(2);

  h1->GetXaxis()->SetTitle("X Values");
  h1->GetYaxis()->SetTitle("Y Values");
  h1->SetTitle("Example of Fitting a 2nd Order Polynomial to a Histogram");
  h1->SetLineWidth(2);

  h1->Fit("f1");
  h1->Draw();
  f1->Draw("same");
  leg->Draw("same");

}
```

TFiles

C++ objects can be written to disk in ROOT

- ➡ All ROOT objects have `Write()` method
- ➡ Conventionally write to files with "`.root`" suffix
- ➡ Open on command line with:
  - ➡ `root file.root`
- ➡ Or within C++ code as:
  - ➡ `TFile* f = new TFile("filename.root","OPEN");`
- ➡ Code snippet examples in following slides
- ➡ https://root.cern.ch/doc/master/classTFile.html

- "`OPEN`" = open an existing root file
- "`CREATE`" = create a new root file
- "`RECREATE`" = create a new root file, overwrite if it already exists
- "`UPDATE`" = append to existing file

TTrees

`TTrees` are a data structure for storing large amounts of the same objects

- ➡ `TTrees` are optimised for reduced disk space and fast access
- ➡ `TTrees` can have many entries containing the same structure of objects
  - ➡ Often one entry == one event
- ➡ A `TTree` contains a list of `TBranches`
  - ➡ A `TBranch` contains a `TLeaf`
  - ➡ A `TLeaf` contain the data type and the data
- ➡ Analogous to `TTree` being a table, each entry is a row, each `TBranch` is a column
- ➡ https://root.cern.ch/doc/master/classTTree.html
- ➡ Also have:
  - ➡ `TNtuple`: a `TTree` containing only floats
  - ➡ `TNtupleD`: a `TTree` containing only doubles
  - ➡ `TChain`: a collection of files containing `TTrees`

Writing TTrees

Making a `TTree` and writing it to a `TFile`

```
void exampleTTreeOut( ){

  TTree* t = new TTree("eve_tree","Event Tree");

  TFile* f = new TFile("outfile.root","RECREATE");

  float var1, var2;
  int var3;

  t->Branch("var1", &var1, "var1/F");
  t->Branch("var2", &var2, "var2/F");
  t->Branch("var3", &var3, "var3/I");

  var1 = 0.5;
  var2 = 1.3;
  var3 = 5;
  t->Fill();

  var1 = -1.2;
  var2 = 4.5;
  var3 = 10;
  t->Fill();

  t->Print();
  t->Show(1);

  t->Write();

}
```

```
root [0]
Processing exampleTTreeOut.C...
******************************************************************************
*Tree    :eve_tree  : Event Tree                                             *
*Entries :        2 : Total =            2271 bytes  File  Size =          0 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :var1      : var1/F                                                 *
*Entries :        2 : Total  Size=        655 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    1 :var2      : var2/F                                                 *
*Entries :        2 : Total  Size=        655 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
*Br    2 :var3      : var3/I                                                 *
*Entries :        2 : Total  Size=        655 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*............................................................................*
======> EVENT:1
 var1            = -1.2
 var2            = 4.5
 var3            = 10
```

```
WillsNetwork:examples wparker$ root -l outfile.root
root [0]
Attaching file outfile.root as _file0...
(TFile *) 0x7f9ce7861ef0
root [1] .ls
TFile**         outfile.root
 TFile*         outfile.root
  KEY: TTree    eve_tree;1      Event Tree
root [2] eve_tree->Show(0)
======> EVENT:0
 var1            = 0.5
 var2            = 1.3
 var3            = 5
root [3]
```

Writing TTrees

Writing to a `TTree` from a data file

```
void exampleTTreeOutTxt( ){

  TTree* t = new TTree("eve_tree","Event Tree");

  t->ReadFile("data.txt","x:y:z");

  TFile* f = new TFile("outfile2.root","RECREATE");

  t->Write();
}
```

data.txt

```
2.43658 3.69984 7.21111
4.88294 9.65768 2.25769
1.04756 5.59091 2.61944
3.91112 2.81859 0.95969
1.34343 1.14249 1.72800
```

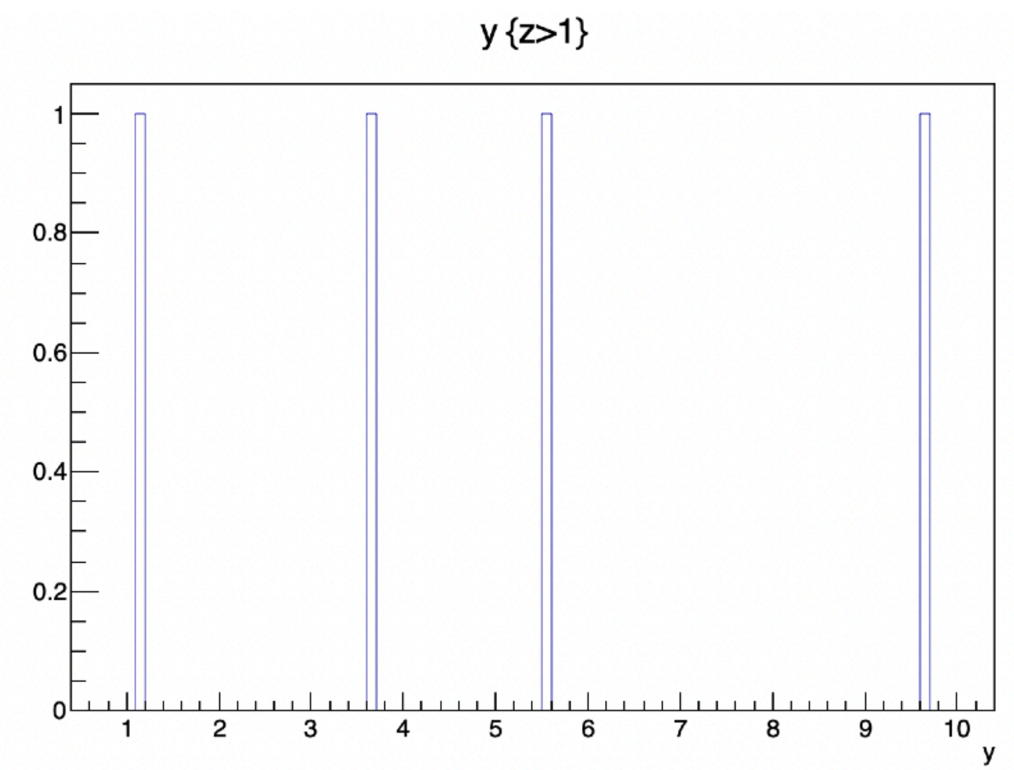
```
WillsNetwork:examples wparker$ root -l outfile2.root
root [0]
Attaching file outfile2.root as _file0...
(TFile *) 0x7f797cf5ebe0
root [1] .ls
TFile**         outfile2.root
 TFile*         outfile2.root
  KEY: TTree    eve_tree;1      Event Tree
root [2] eve_tree->Scan()
************************************************
*    Row   *       x.x *       y.y *       z.z *
************************************************
*        0 * 2.4365799 * 3.6998400 * 7.2111101 *
*        1 * 4.8829398 * 9.6576795 * 2.2576899 *
*        2 * 1.0475599 * 5.5909099 * 2.6194400 *
*        3 * 3.9111199 * 2.8185899 * 0.9596899 *
*        4 * 1.3434300 * 1.1424900 * 1.7280000 *
************************************************
(long long) 5
root [3]
```

Reading TTrees

Reading a `TTree` from a ROOT file

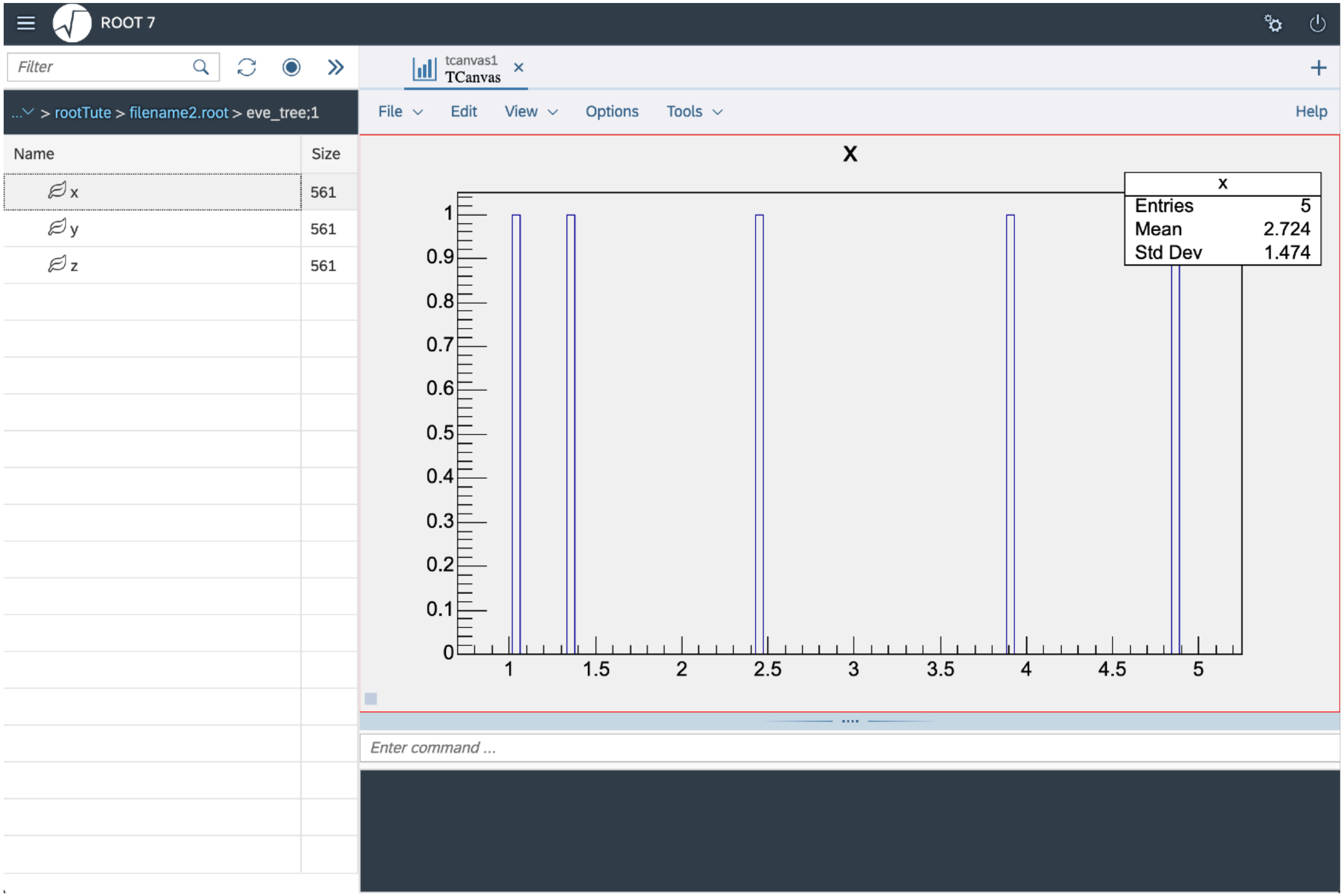
```
void exampleTTreeIn( ){

  TFile *f1 = TFile::Open("outfile2.root");

  TTree *t1 = (TTree*)f1->Get("eve_tree");

  float x,y,z;

  t1->SetBranchAddress("x", &x);
  t1->SetBranchAddress("y", &y);
  t1->SetBranchAddress("z", &z);

  for (int i_entry = 0; i_entry < t1->GetEntries(); i_entry++) {
    t1->GetEntry(i_entry);
    std::cout << x << " " << y << " " << z << std::endl;
  }

  t1->Draw("y", "z>1");

}
```

```
WillsNetwork:examples wparker$ root -l exampleTTreeIn.C
root [0]
Processing exampleTTreeIn.C...
2.43658 3.69984 7.21111
4.88294 9.65768 2.25769
1.04756 5.59091 2.61944
3.91112 2.81859 0.95969
1.34343 1.14249 1.728
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [1]
```



TBrowser

The `TBrowser` can be used to navigate files and make plots




```
WillsNetwork:examples wparker$ root -l
root [0] new TBrowser()
```

Other Tools

ROOT can do so much more than we've covered here!

- Random number generation
  - `TRandom3* r3 = new TRandom3();`
  - `double r = r3->Rndm(); // From Uniform Distribution`
  - `double g = r3->Gaus(); // From Gaussian Distribution`
- Physics vectors:
  - Lorentz vectors and 3D vectors in various coordinate systems
- ROOFit for modelling event distributions
- TMVA for machine learning
- ROOTStats for advanced statistical tools
- & more!

PyROOT

You can access the full ROOT C++ functionality from python with PyROOT

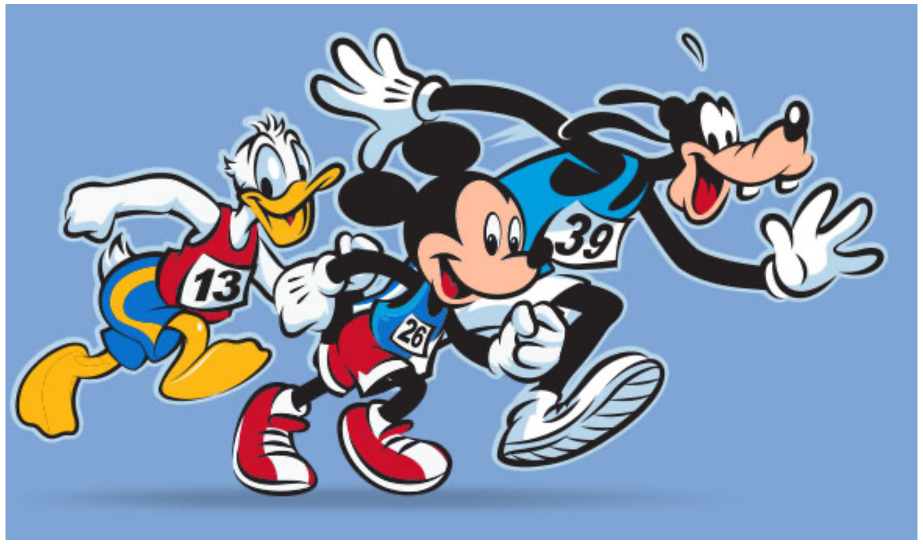
- ➡ Get the power of C++ compiled libraries with the flexibility of python (eg. dynamic typing)
- ➡ Can interoperate with standard data science python tools (eg. numPy, pandas)
- ➡ `import ROOT` to get started
- ➡ All the classes we've discussed can be accessed with `ROOT.TH1D`, `ROOT.TF1`, `ROOT.TGraphErrors` etc.
- ➡ Compatible with python >= 2.7
- ➡ https://root.cern/manual/python/

Summary

ROOT is a powerful toolkit for high energy physics analyses!

- We have looked at:
  - Using the command line interpreter and running macros
  - Making and plotting histograms and graphs
  - Fitting functions
  - Reading/writing to/from files
  - Where to find more information and help

Exercises!



Exercise 1

Getting used to the ROOT command line

- ➡ Open ROOT
- ➡ Declare some variables and do some calculations
- ➡ Draw a `TCanvas`
- ➡ Open the `TBrowser` and explore!

Exercise 2

Writing and running a basic macro

➡ In a macro, declare a histogram

➡ Fill it with some values, in any way you choose

➡ Declare a new `TFile`, and save the histogram

➡ Open the new ROOT file and draw the histogram

➡ Play with the aesthetics of the plot via the GUI

Exercise 3

Plotting the Gaussian approximation to the Poisson distribution

➡ Plot the Poisson distribution for $\lambda = 5, 10, 25, 50, 100$

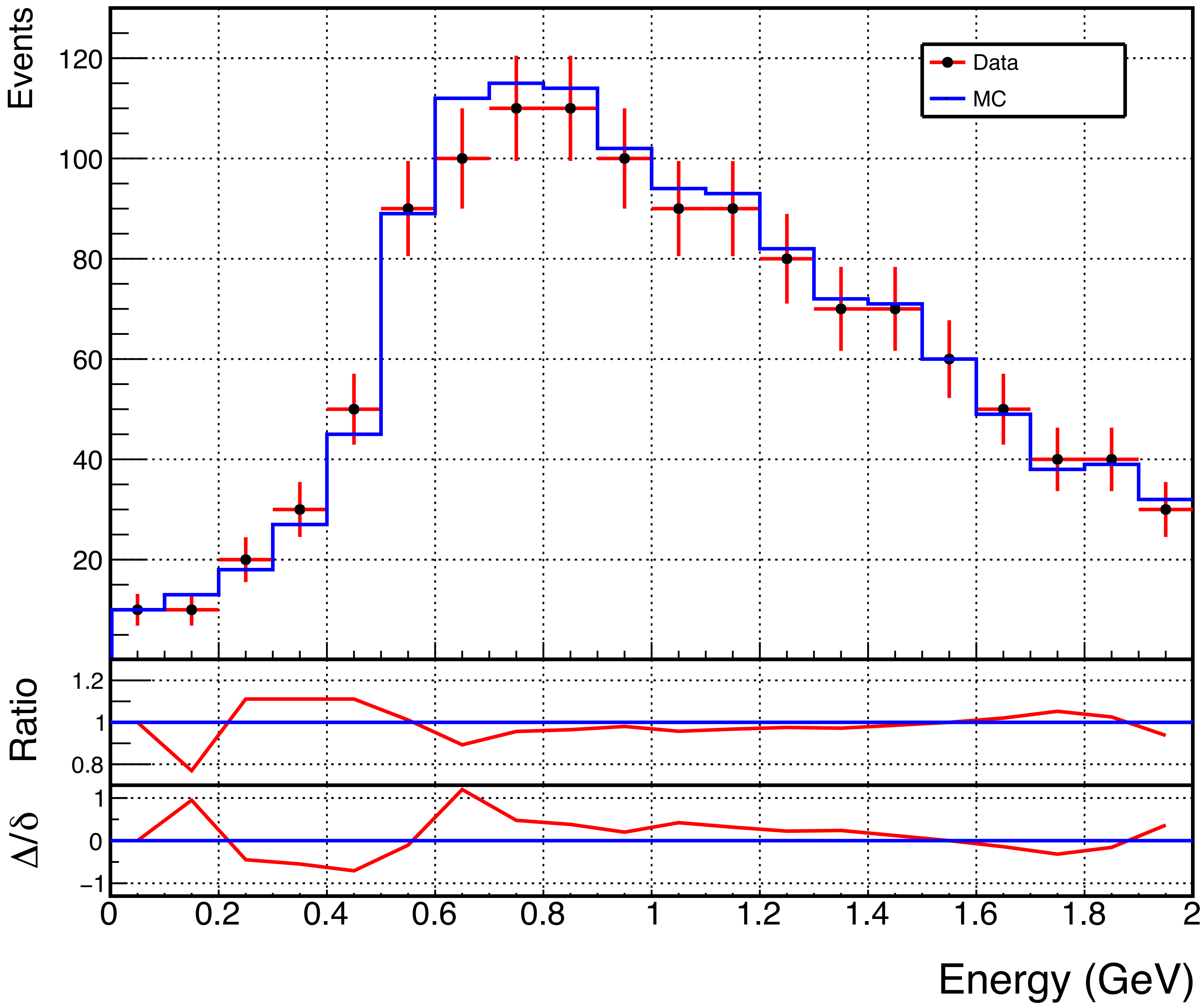
➡ For each, also plot the Gaussian distribution with $\mu = \lambda$, $\sigma = \sqrt{\lambda}$

➡ Use a different colour for each λ, and different line styles for Poisson and Gaussian

➡ Save the canvas to a PDF file

Exercise 4

Recreate this data - Monte Carlo comparison plot exactly (or as close as you can!)



Project

Simulating a dataset from a particle detector

Part A:

➡ Imagine you have a 5m radius spherical target mass, surrounded by PMTs

➡ Generate x, y, z and energy for 1000 events

- ➡ Assume this signal is uniformly distributed in position
- ➡ Assume an energy of, say, 2.5 MeV

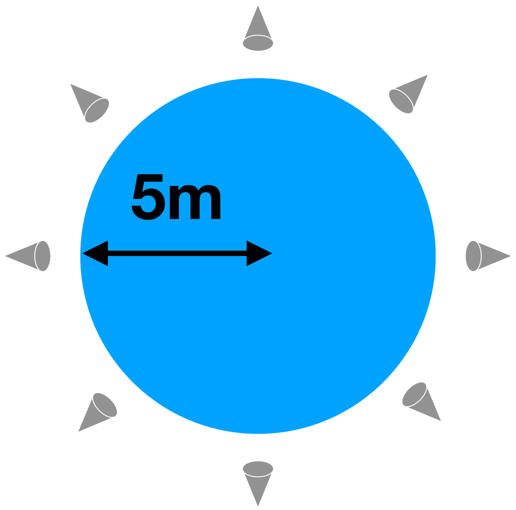
➡ Now generate x, y, z and energy for 1000 background events

- ➡ Let’s say it has the same energy as our signal
- ➡ But radially drops off as 1/r^2 away from the edge of the detector. Maybe there’s some external radiation that leaches into the target mass but only penetrates so far

➡ Now let’s generate some reconstructed values

- ➡ Assume a position resolution of 100 mm in each coordinate
- ➡ Assume an energy resolution of 3 %
- ➡ What if the resolution is a weak function of the radius?

➡ Now repeat for 100,000 events and save these values in a TTree in a file



Part B:

➡ Check the radial distributions look as expected

➡ Plot True vs Reconstructed R for all events

➡ Plot (Reconstructed E - True E) / True E as a function of True E

Part C:

➡ We are now going to try and reject those background events by cutting on radius

➡ Make a ROC curve (purity against sacrifice) for different values for a radial cut

Part D:

➡ Now repeat this whole process but for a background rate reduced by a factor of 10. How does the ROC curve change?

Bonus Exercise 5

Collecting real data and plotting correlations

- ➡ Ask your friends >= 5 questions about themselves with numeric answers (eg, birthdate, age, height, shoe size, favourite number, number of pets/ pairs of shoes/hammocks they own). Get creative!
- ➡ Record each answer in a text file, separated by a space, with a new line for a new friend
- ➡ Read this into a `TTree`, and save the `TTree` to file
- ➡ Open the file and plot different variables against each other. You can cut on variables (including ones your not plotting)
- ➡ See if you can pull out any amusing correlations!
  - ➡ But remember this does not necessarily imply causation!!!