



Root&RooFit

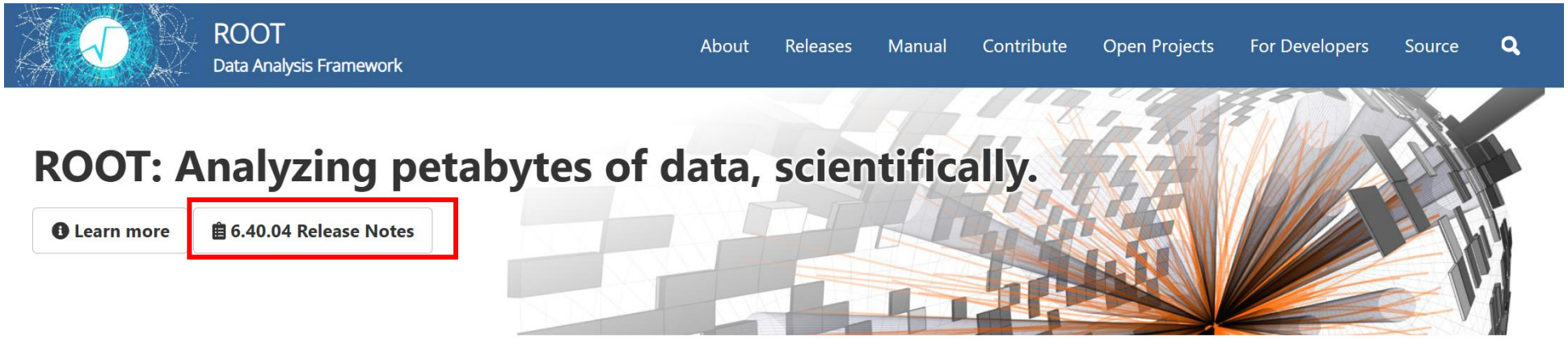
第五届Belle II冬季学校 南京 东南大学

2026年11月13-19日

李素娴 (郑州大学)



网址：https://root.cern/





Get ROOT



Learn



Reference



Forum



Gallery



Enables processing and scientific analysis of large amounts of data: today, more than 2 exabytes are stored in ROOT files. The Higgs was discovered with ROOT!



Open source, which means that you can use it freely and modify it. It adopts an open development process, inviting its users to contribute to it.



Delivered with a C++ interpreter, ideal for *fast prototyping*. ROOT also offers a Python interface to all its components with dynamic bindings. Jupyter notebooks are supported too.

About ROOT

ROOT is a software framework born at CERN, at the heart of the research on high-energy physics.
Every day, thousands of physicists use ROOT applications to analyze their data or to perform simulations.

Installation：

（1） KEKCC: Have already installed.

1. Setup Belle II environment

```
[lisx@cw05 ~]$ source /cvmfs/belle.cern.ch/tools/b2setup  light-2609-luna
Warning: Changing existing PYTHONPATH from /sw/belle/cern/root/lib to /cvmfs/belle.cern.ch/tools
Belle II software tools set up at: /cvmfs/belle.cern.ch/tools
Environment setup for release: light-2609-luna
Central release directory    : /cvmfs/belle.cern.ch/el9/releases/light-2609-luna
```

2. Start ROOT

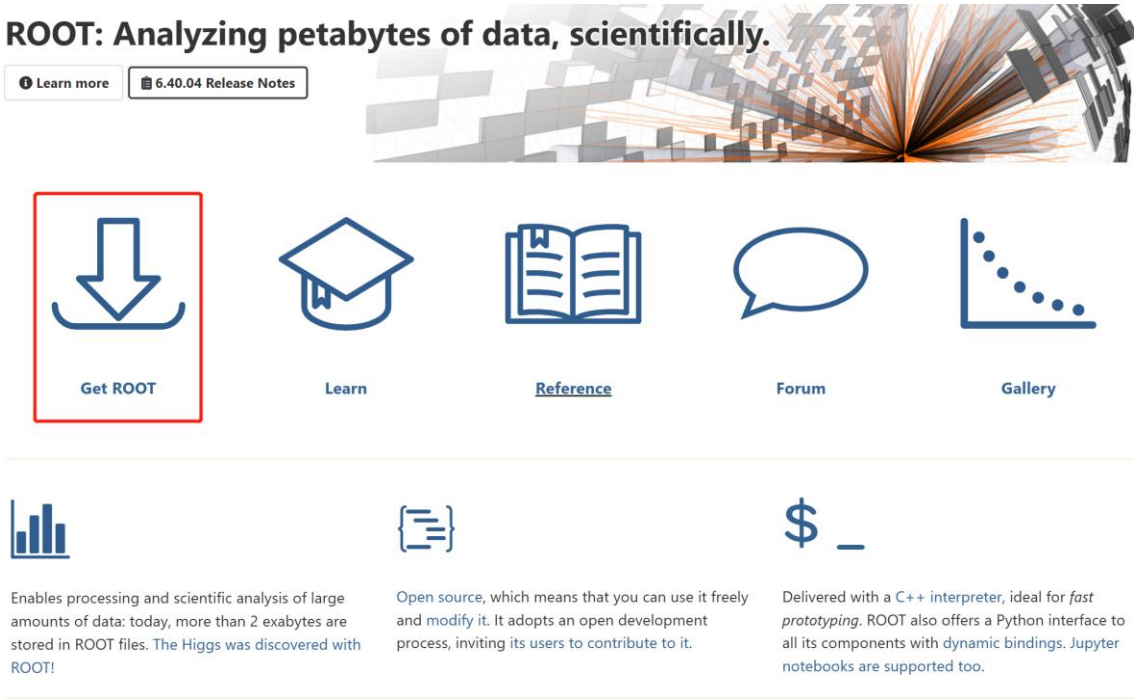
```
[lisx@cw05 ~]$ root
root [0] .q
```

3. Check installed version

```
[lisx@cw05 ~]$ root --version
ROOT Version: 6.36.06
Built for linuxx8664gcc on Jun 04 2026, 14:54:02
From tags/6-36-06@6-36-06
```

Installation：

（2） local laptop.



Get ROOT

Curated by the ROOT Core developers

- **Conda-Forge (Linux, macOS):** `mamba env create -n root-640 "root=6.40.*"`
- **Binaries (Linux, macOS, Windows):** instructions per release here
- **CVMFS ROOT only (Linux, macOS):** instructions per release here
- **CVMFS full LCG package suites (Linux, macOS):** `. /cvmfs/sft.cern.ch/lcg/views/LCG_<release number>/<platform>/setup.sh`
- **Sources:** Instructions here

In alpha mode:

- **PyPi (Linux, from 6.42.00 macOS):** `pip install root`

ROOT installed at KEKCC is recommended!

Learn

Planning to learn more about ROOT? This is the right page!

We offer ROOT courses and exercises, in Python based on Jupyter Notebooks. For these courses, you don't need to install ROOT on your machine. You can directly run all the examples and exercises on SWAN (if you have a CERN computing account), or otherwise using GitHub Codespaces or Binder.

Three levels are available:

- Beginners: ROOT Introductory Python Course, which can be accompanied by its the video version. This course covers the ROOT basics.
- Intermediate: ROOT Advanced Python Course, which can be accompanied by its the video version. This course is about RNTuple, UHI, RooFit, Pythonizations and some more advanced RDataFrame features
- Advanced: ROOT Advanced C++ Course. This course is fully in C++ and covers aspects such as using basic ROOT objects, dealing with RDataFrame using only compiled code, dataset schema with custom classes using ROOT dictionaries and writing it to TTree and RNTuple datasets, likelihood evaluation with RooFit, as well as parallelism and performance considerations.

Would you like to install ROOT on your local system? Have a look to the instructions.

Finally, if you wish to delve further into ROOT functionalities, check the following:

- The ROOT beginners' guide (aka "Primer")
- The Manual where more in depth information can be found
- The full API Documentation
- The Tutorials

Lastly, in case you have a problem or a question, don't hesitate to use the ROOT Forum.



Public RN OPEN-VIDEO-2025-169-001

ROOT Summer Student Course 2025

© 2025 CERN Conditions of Use 28 July 2025 01:23:28 2957 Views

This is an introduction to ROOT which is based on the recordings of the 2025 ROOT Summer Student courses given by the members of the ROOT team. We encourage anyone watching the video to follow along with the material in Jupyter notebooks. For the instructions how to do so, please visit the GitHub page with the course repository before watching the video: https://github.com/root-project/student-course/tree/25.07 and follow the the README section. Note, the tag 25.07 of the repository (as in the link) should be used together with this video.

Command Line

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + \ldots = \sum_{n=0}^{\infty} x^n$$

ROOT can be used as a simple calculator, but we let's make a step forward: declare **variables** and use a ***for*** control structure.

```
root [0] double x=.5
(double) 0.5
root [1] int N=30
(int) 30
root [2] double gs=0;
```

```
root [3] for (int i=0;i<N;++i) gs += pow(x,i)
root [4] std::abs(gs - (1/(1-x)))
(Double_t) 1.86265e-09
```

Macros

- We have seen how to interactively type lines at the prompt
- The next step is to write "ROOT Macros" – lightweight programs
- The general structure for a macro stored in file *MacroName.C* is:

Function, no main, same name as the file

```
void MacroName() {
        <           ...
          your lines of C++ code
                    ...             >
}
```

Running a macro:

- A macro is executed at the system prompt by typing:

```
> root MacroName.C
```

- or executed at the ROOT prompt using .x:

```
> root
root [0] .x MacroName.C
```

- or it can be loaded into a ROOT session and then be run by typing:

```
root [0] .L MacroName.C
root [1] MacroName();
```

Interpretation and Compilation

- We have seen how ROOT interprets and "just in time compiles" code. ROOT also allows to compile code "traditionally". At the ROOT prompt:

```
root [1] .L macro1.C+
root [2] macro1()
```

Generate shared library and execute function

- ROOT libraries can also be used to produce standalone, compiled applications:

Advanced Users

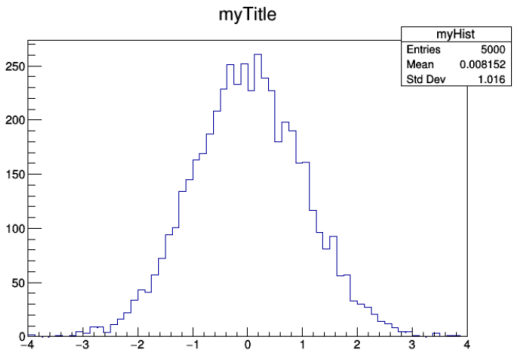
```
int main() {
  ExampleMacro();
  return 0;
}
```

```
> g++ -o ExampleMacro ExampleMacro.C `root-config --cflags --libs`
> ./ExampleMacro
```

Histograms, Graphs and Functions

- ROOT provides a rich set of histograms
  - Focus on the class **TH1D today**: one dimensional histogram filled with doubles
  - but also available:
    - multiple dimensions histogram **TH{1,2,3}** classes
    - histograms holding different precision types: floats F, integers I, strings S

```
TH1D::TH1D ( const char *  name,
             const char *  title,
             Int_t         nbinsx,
             Double_t      xlow,
             Double_t      xup
           )
```



C++

```
root [0] TH1D h("myHist", "myTitle", 64, -4, 4)
root [1] h.FillRandom("gaus")
root [2] h.Draw()
```

Python

```
> python
>>> import ROOT
>>> h = ROOT.TH1F("myHist", "myTitle", 64,
-4, 4)
>>> h.FillRandom("gaus")
>>> h.Draw()
```

https://root.cern/doc/v606/classTHistPainter.html

The histogram painter class.

Implements all histograms' drawing's options.

- Introduction
- Histograms' plotting options
  - Options supported for 1D and 2D histograms
  - Options supported for 1D histograms
  - Options supported for 2D histograms
  - Options supported for 3D histograms
  - Options supported for histograms' stacks (THStack)
- Setting the Style
- Setting line, fill, marker, and text attributes
- Setting Tick marks on the histogram axis
- Giving titles to the X, Y and Z axis
- The option "SAME"
  - Limitations
- Superimposing two histograms with different scales in the same pad
- Statistics Display
- Fit Statistics
- The error bars options
- The bar chart option
- The "BAR" and "HBAR" options
- The SCATter plot option (default for 2D histograms)
- The ARRow option
- The BOX option
- The COLor option
- The CANDLE option
- The VIOLIN option

Histograms tutorials: https://root.cern/doc/master/group__tutorial__hist.html

Shows how to create, fill and write a histogram to a **ROOT** file.

C++

```
void hist000_TH1_first()
{
   // Open the file to write the histogram to
   auto outFile = std::unique_ptr<TFile>(TFile::Open("outfile.root", "RECREATE"));

   // Create the histogram object
   // There are several constructors you can use (\see TH1). In this example we use the
   // simplest one, accepting a number of bins and a range.
   int nBins = 30;
   double rangeMin = 0.0;
   double rangeMax = 10.0;
   TH1D histogram("histogram", "My first ROOT histogram", nBins, rangeMin, rangeMax);

   // Fill the histogram. In this simple example we use a fake set of data.
   // The 'D' in TH1D stands for 'double', so we fill the histogram with doubles.
   // In general you should prefer TH1D over TH1F unless you have a very specific reason
   // to do otherwise.
   const std::array values{1, 2, 3, 3, 3, 4, 3, 2, 1, 0};
   for (double val : values) {
      histogram.Fill(val);
   }

   // Write the histogram to `outFile`.
   outFile->WriteObject(&histogram, histogram.GetName());

   // When the TFile goes out of scope it will close itself and write its contents to disk.
}
```

Shows how to create, fill and write a histogram to a **ROOT** file.

Python

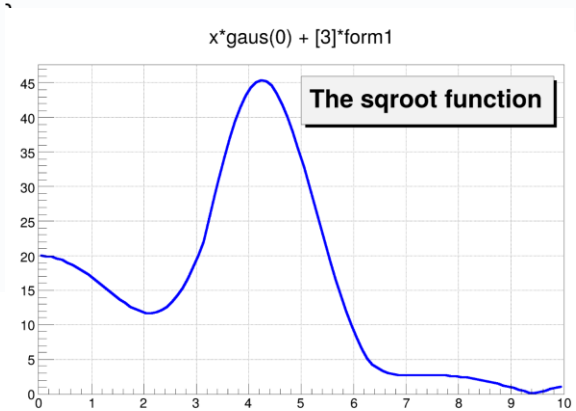
```
import ROOT

# Open the file to write the histogram to
with ROOT.TFile.Open("outfile.root", "RECREATE") as outFile:
   # Create the histogram object
   # There are several constructors you can use (see TH1). In this example we use the
   # simplest one, accepting a number of bins and a range.
   histogram = ROOT.TH1D("histogram", "My first ROOT histogram", nbinsx = 30, xlow = 0.0, xup = 10.0)

   # Fill the histogram. In this simple example we use a fake set of data.
   # The 'D' in TH1D stands for 'double', so we fill the histogram with doubles.
   # In general you should prefer TH1D over TH1F unless you have a very specific reason
   # to do otherwise.
   values = [1, 2, 3, 3, 3, 4, 3, 2, 1, 0]
   for val in values:
      histogram.Fill(val)

   # Write the histogram to `outFile`.
   outFile.WriteObject(histogram, histogram.GetName())

   # When `with` block exits, `outFile` will close itself and write its contents to disk.
```

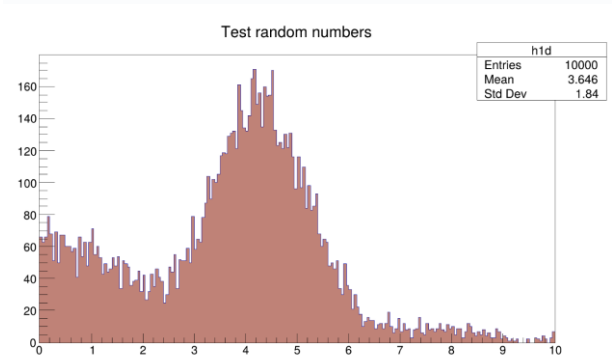
Fill a 1D histogram with random values using predefined functions.

```
void hist001_TH1_fillrandom()
{
   // Create a one dimensional histogram and fill it with a gaussian distribution
   int nBins = 200;
   double rangeMin = 0.0;
   double rangeMax = 10.0;
   TH1D h1d("h1d", "Test random numbers", nBins, rangeMin, rangeMax);

   // "gaus" is a predefined ROOT function. Here we are filling the histogram with
   // 10000 values sampled from that distribution.
   h1d.FillRandom("gaus", 10000);

   // Open a ROOT file and save the histogram
   auto myfile = std::unique_ptr<TFile>(TFile::Open("fillrandom.root", "RECREATE"));
   myfile->WriteObject(&h1d, h1d.GetName());
}
```

Draw a 1D histogram to a canvas.

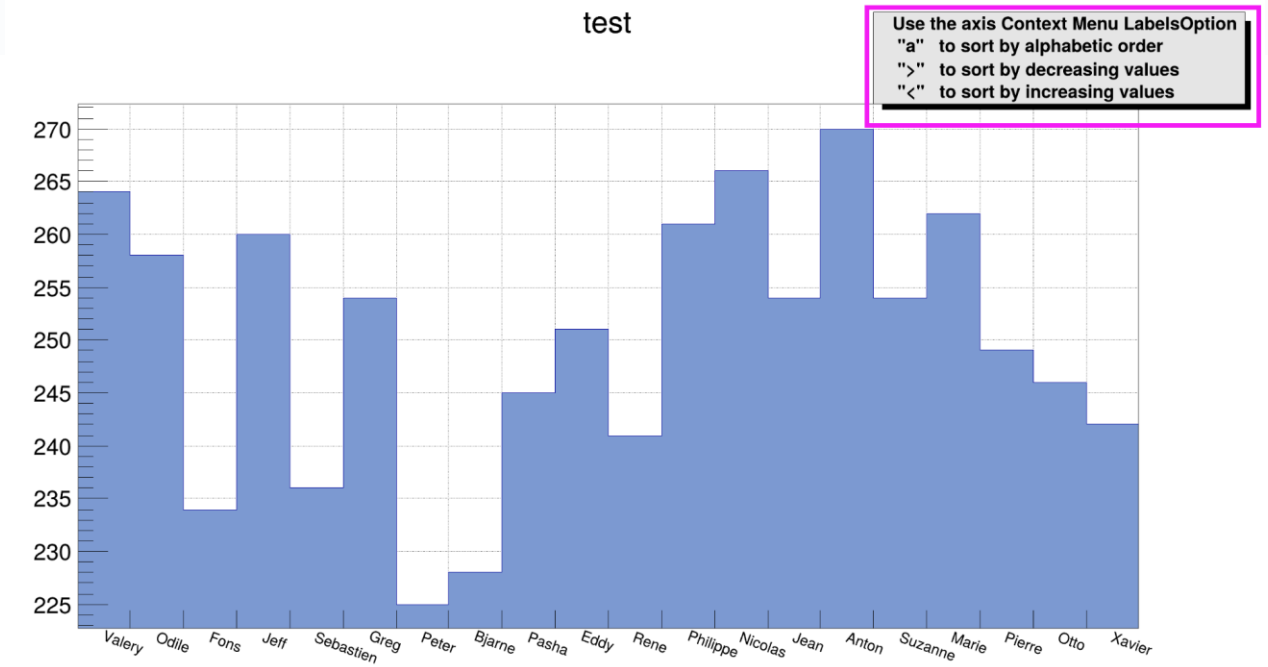
```
void hist003_TH1_draw()
{
   // Create and fill the histogram.
   // See hist002_TH1_fillrandom_userfunc.C for more information about this section.
   auto *form1 = new TFormula("form1", "abs(sin(x)/x)");
   double rangeMin = 0.0;
   double rangeMax = 10.0;
   auto *sqroot = new TF1("sqroot", "x*gaus(0) + [3]*form1", rangeMin, rangeMax);
   sqroot->SetLineColor(4);
   sqroot->SetLineWidth(6);
   sqroot->SetParameters(10.0, 4.0, 1.0, 20.0);

   int nBins = 200;
   auto *h1d = new TH1D("h1d", "Test random numbers", nBins, rangeMin, rangeMax);

   h1d->FillRandom("sqroot", 10000);

   // Create a canvas and draw the histogram
   int topX = 200;
   int topY = 10;
   int width = 700;
   int height = 900;
   auto *c1 = new TCanvas("c1", "The FillRandom example", topX, topY, width, height);

   // Split the canvas into two sections to plot both the function and the histogram
   // The TPad's constructor accepts the relative coordinates (0 to 1) of the pad's boundaries
   auto *pad1 = new TPad("pad1", "The pad with the function", 0.05, 0.50, 0.95, 0.95);
   auto *pad2 = new TPad("pad2", "The pad with the histogram", 0.05, 0.05, 0.95, 0.45);

   // Draw the two pads
   pad1->Draw();
   pad2->Draw();

   // Select pad1 to draw the next objects into
   pad1->cd();
   pad1->SetGridx();
   pad1->SetGridy();
   pad1->GetFrame()->SetBorderMode(-1);
   pad1->GetFrame()->SetBorderSize(5);

   // Draw the function in pad1
   sqroot->Draw();
   // Add a label to the function.
   // TPaveLabel's constructor accepts the pixel coordinates and the label string.
   auto *lfunction = new TPaveLabel(5, 39, 9.8, 46, "The sqroot function");
   lfunction->Draw();
   c1->Update();

   // Select pad2 to draw the next objects into
   pad2->cd();
   pad2->GetFrame()->SetBorderMode(-1);
   pad2->GetFrame()->SetBorderSize(5);

   h1d->SetFillColor(45);
   h1d->Draw();
   c1->Update();
}
```





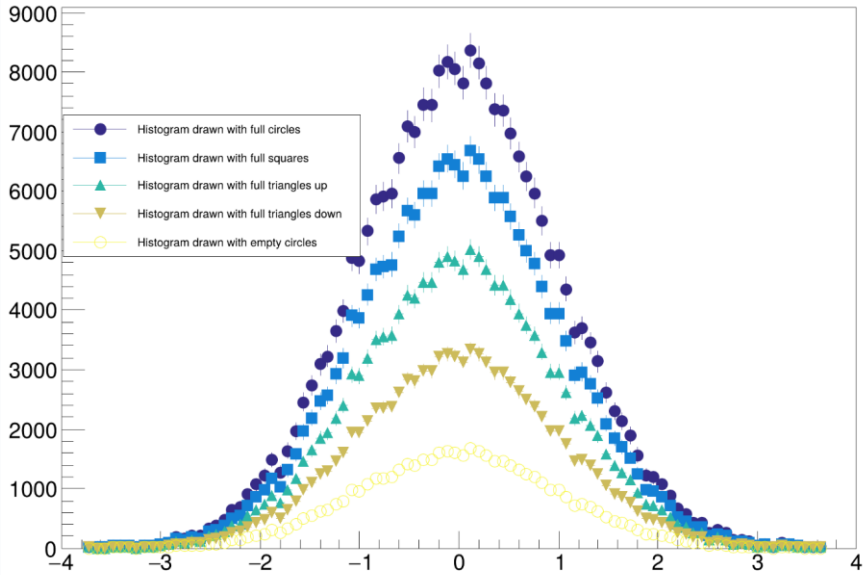
1D histograms with alphanumeric labels.

```
// Draw the histogram
h->Draw();

// Draw a boxed text
// "brNDC" = coordinates draw bottom-right shadow and pass the coordinates in normalized device coordinates
auto *pt = new TPaveText(0.7, 0.85, 0.98, 0.98, "brNDC");
pt->SetFillColor(18);
pt->SetTextAlign(12);
pt->AddText("Use the axis Context Menu LabelsOption");
pt->AddText(" \"a\"   to sort by alphabetic order");
pt->AddText(" \">\"   to sort by decreasing values");
pt->AddText(" \"<\"   to sort by increasing values");
pt->Draw();
```



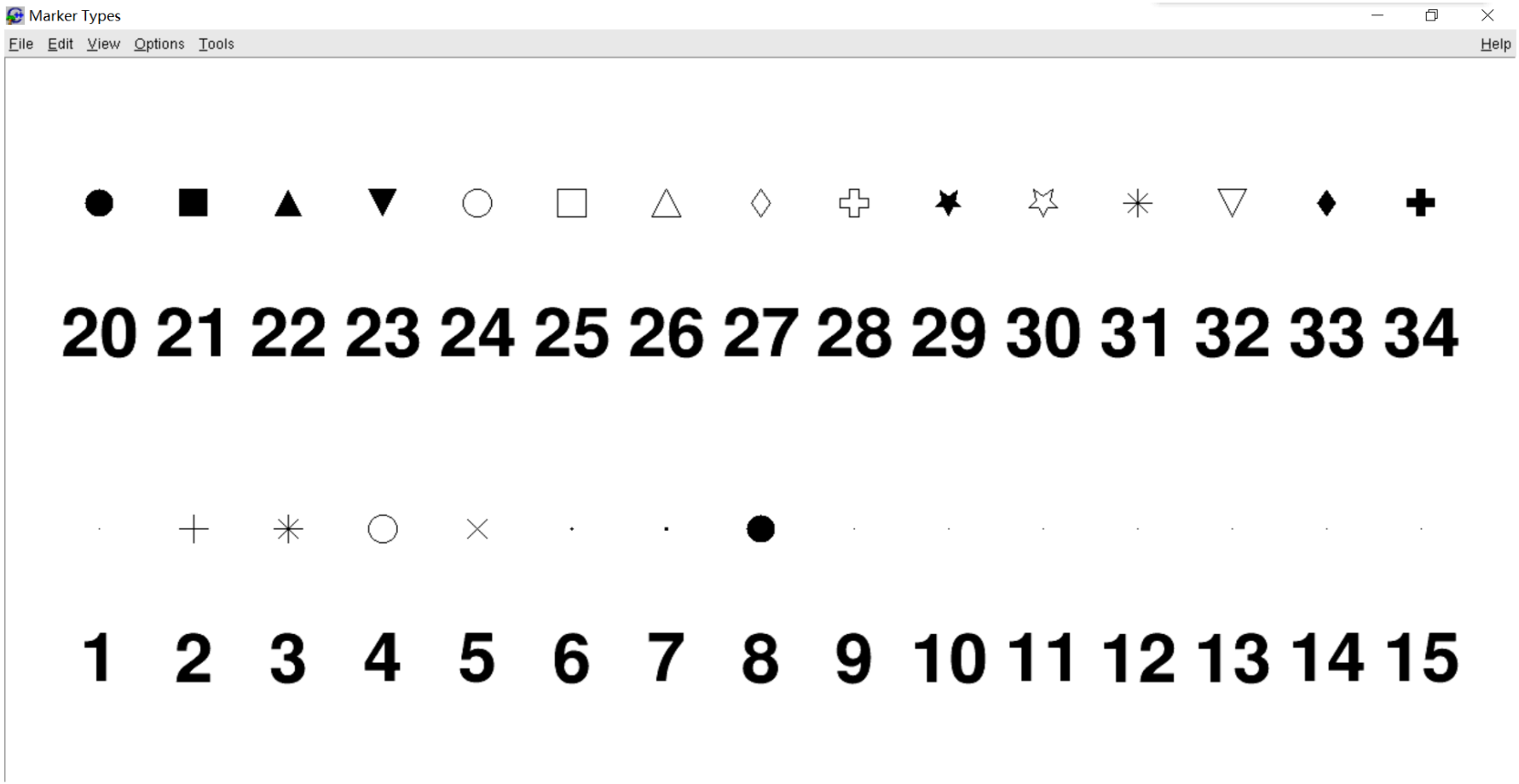
Palette coloring for TH1. options PFC (Palette Fill Color), PLC (Palette Line Color) and PMC (Palette Marker Color)

```
void hist005_TH1_palettecolor()
{
   auto *canvas = new TCanvas();

   // Disable drawing the title of the canvas
   gStyle->SetOptTitle(kFALSE);
   // Disable drawing the stats box
   gStyle->SetOptStat(0);

   auto *h1 = new TH1D("h1", "Histogram drawn with full circles", 100, -4, 4);
   auto *h2 = new TH1D("h2", "Histogram drawn with full squares", 100, -4, 4);
   auto *h3 = new TH1D("h3", "Histogram drawn with full triangles up", 100, -4, 4);
   auto *h4 = new TH1D("h4", "Histogram drawn with full triangles down", 100, -4, 4);
   auto *h5 = new TH1D("h5", "Histogram drawn with empty circles", 100, -4, 4);

   // Use Mersenne-Twister random number generator
   TRandom3 rng;
   for (int i = 0; i < 25000; i++) {
      // "Rannor" fills the two parameters we pass with RANdom numbers picked from a NORmal distribution.
      // In this case we ignore the second value.
      double val, ignored;
      rng.Rannor(val, ignored);
      // Fill() called with 2 arguments adds the given value (first arg) with the specified weight (second arg)
      h1->Fill(val, 10.);
      h2->Fill(val, 8.);
      h3->Fill(val, 6.);
      h4->Fill(val, 4.);
      h5->Fill(val, 2.);
   }

   // Set different styles for the various histograms
   h1->SetMarkerStyle(kFullCircle);
   h2->SetMarkerStyle(kFullSquare);
   h3->SetMarkerStyle(kFullTriangleUp);
   h4->SetMarkerStyle(kFullTriangleDown);
   h5->SetMarkerStyle(kOpenCircle);

   // Draw all histograms overlapped in the same canvas (thanks to the "SAME" option)
   h1->Draw("PLC PMC");
   h2->Draw("SAME PLC PMC");
   h3->Draw("SAME PLC PMC");
   h4->Draw("SAME PLC PMC");
   h5->Draw("SAME PLC PMC");

   // Build a legend from the objects drawn in the pad, using their description that we specified when constructing
       them
   gPad->BuildLegend();
}
```

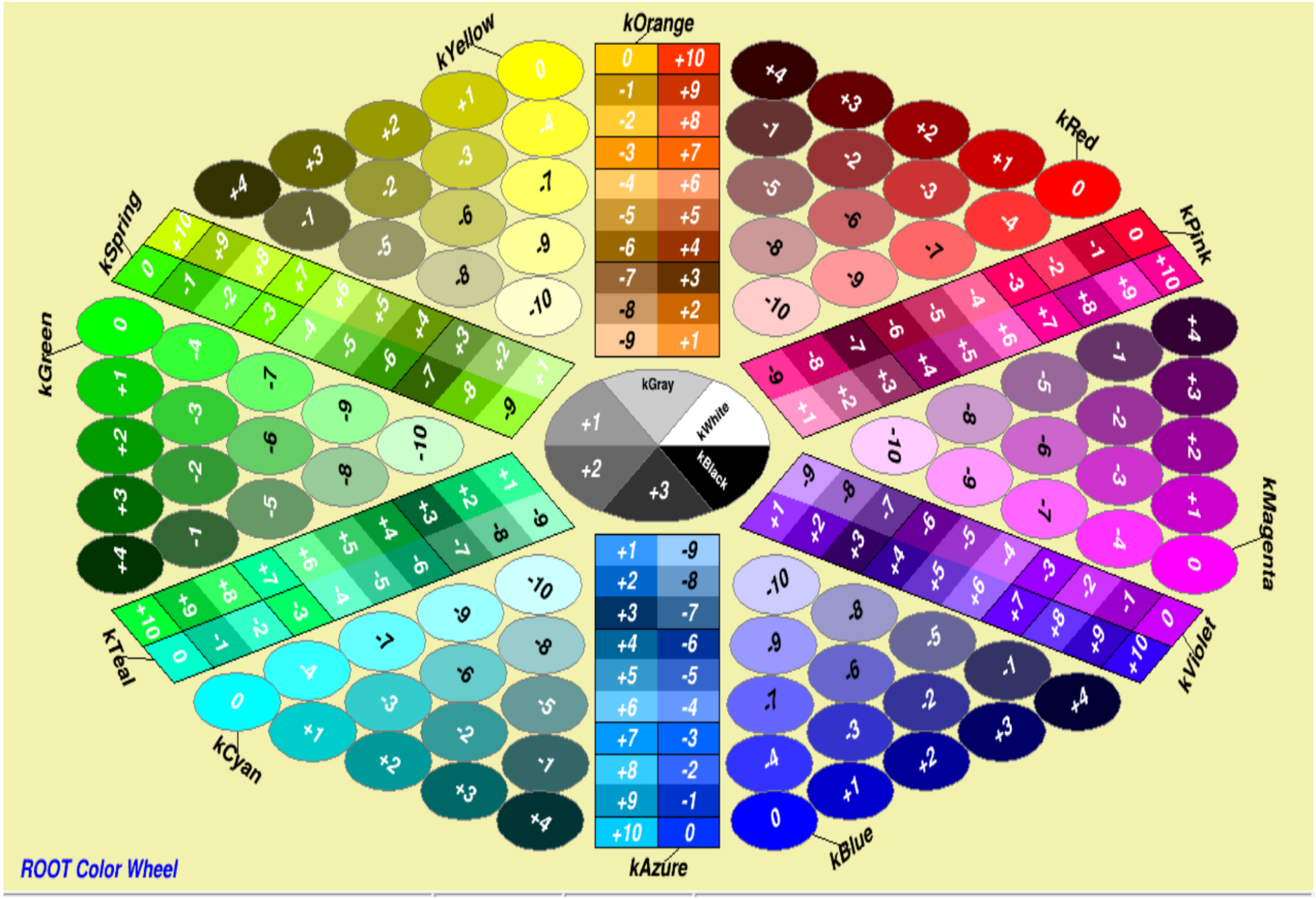


SetMarkerStyle(20)

marker



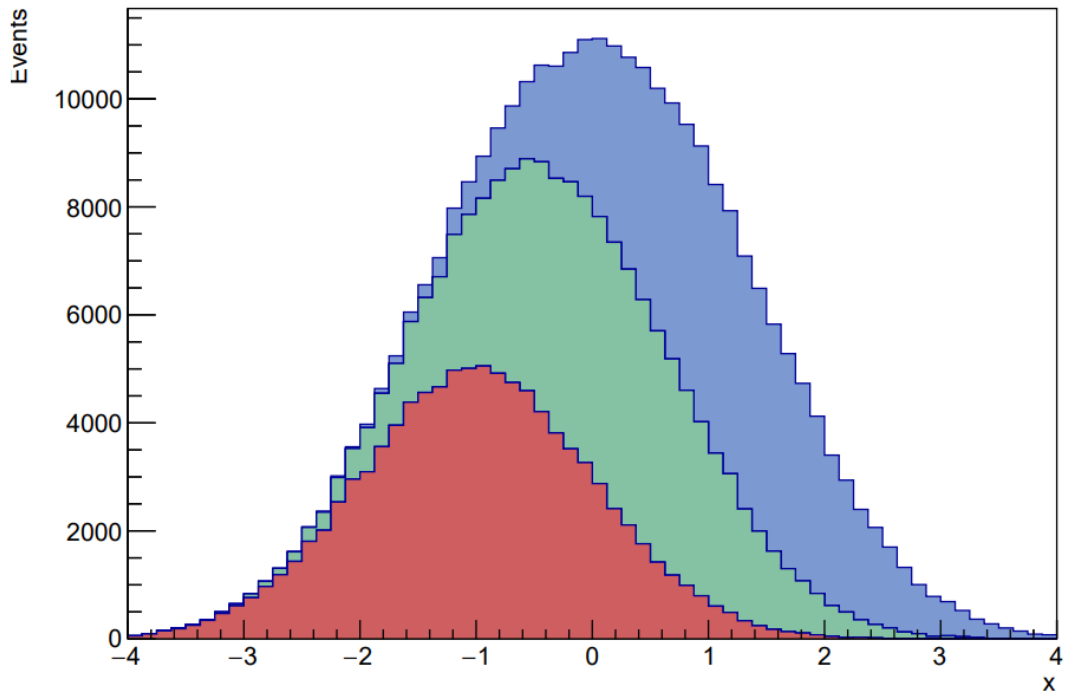
SetLineColor(kRed+12)



Example: stacked histograms

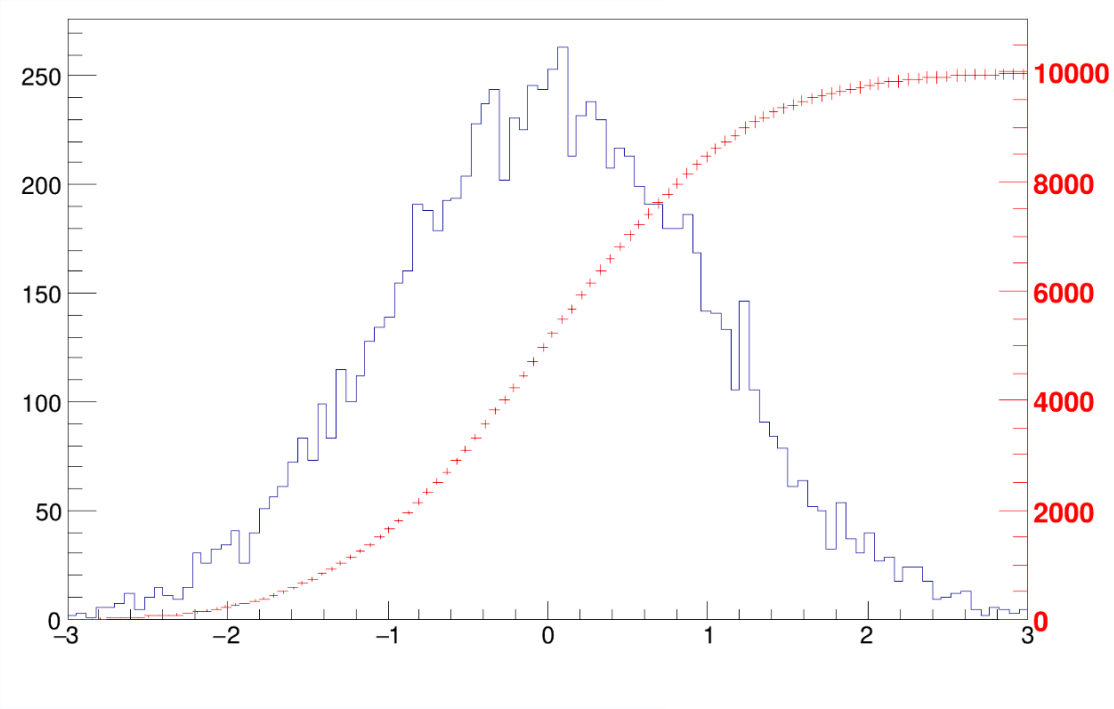
In high energy physics, we often plot stacked histograms, for example to show the contributions of different processes. This can be doe with the THStack.

```
TF1 f1{"f1", "gaus", -4.0, 4.0};

TH1D h1("h1", "x", 64, -4.0, 4.0);
TH1D h2("h2", "x", 64, -4.0, 4.0);
TH1D h3("h3", "x", 64, -4.0, 4.0);

THStack hs("hs","");
hs.SetTitle(";x;Events");

std::vector<TH1D*> histos{&h1, &h2, &h3};
std::vector<int> colors{46, 30, 38};

for(int i = 0; i < histos.size(); ++i) {
    TH1D & h = *histos[i];
    f1.SetParameters(1.0, i - 1, 1.0);
    h.FillRandom("f1", 100000);
    h.SetFillColor(colors[i]);
    hs.Add(&h);
}

hs.Draw();
```



two histograms with different scales in the "same" pad.

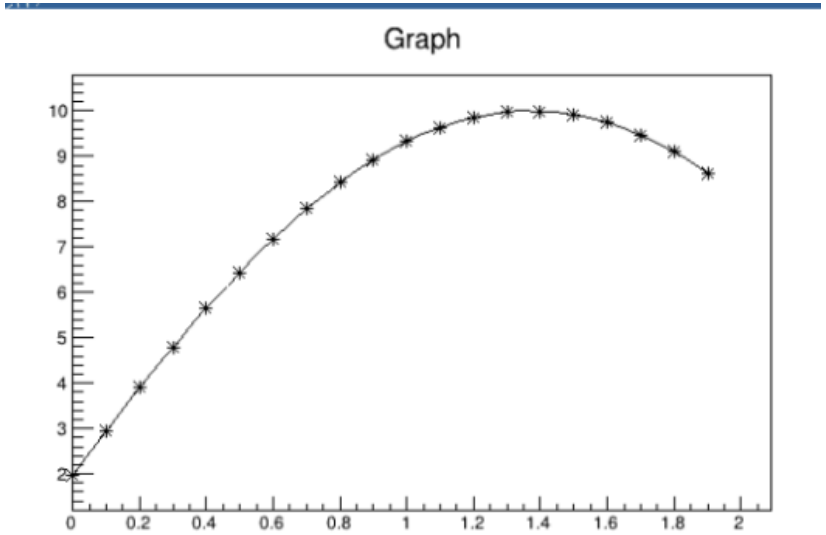
```
void hist010_TH1_two_scales()
{
   TCanvas *c1 = new TCanvas("c1", "hists with different scales", 600, 400);

   // create/fill draw h1
   gStyle->SetOptStat(kFALSE);
   TH1D *h1 = new TH1D("h1", "my histogram", 100, -3, 3);
   TRandom3 rng;
   for (int i = 0; i < 10000; i++)
      h1->Fill(rng.Gaus(0, 1));
   h1->Draw();
   c1->Update();

   // create hint1 filled with the bins integral of h1
   TH1D *hint1 = new TH1D("hint1", "h1 bins integral", 100, -3, 3);
   double sum = 0;
   for (int i = 1; i <= 100; i++) {
      sum += h1->GetBinContent(i);
      hint1->SetBinContent(i, sum);
   }

   // scale hint1 to the pad coordinates
   double rightmax = 1.1 * hint1->GetMaximum();
   double scale = gPad->GetUymax() / rightmax;
   hint1->SetLineColor(kRed);
   hint1->Scale(scale);
   hint1->Draw("same");

   // draw an axis on the right side
   TGaxis *axis =
      new TGaxis(gPad->GetUxmax(), gPad->GetUymin(), gPad->GetUxmax(), gPad->GetUymax(), 0, rightmax, 510, "+L");
   axis->SetLineColor(kRed);
   axis->SetLabelColor(kRed);
   axis->Draw();
}
```



Graphs

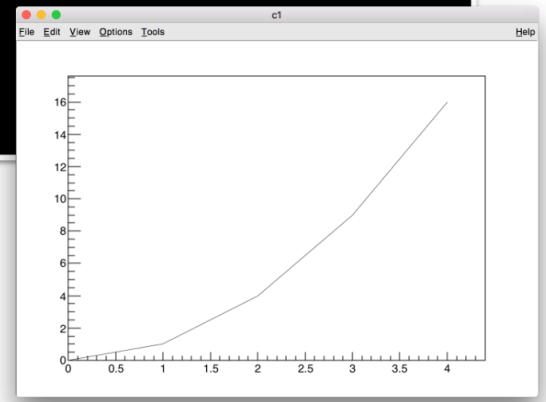
https://root.cern/doc/master/group__tutorial__graphs.html

- Display points and associated errors
- **Fundamental to display trends**




```
>>> g = ROOT.TGraph()
>>> for i in range(5): g.SetPoint(i,i,i*i)
>>> g.Draw("APL")
```

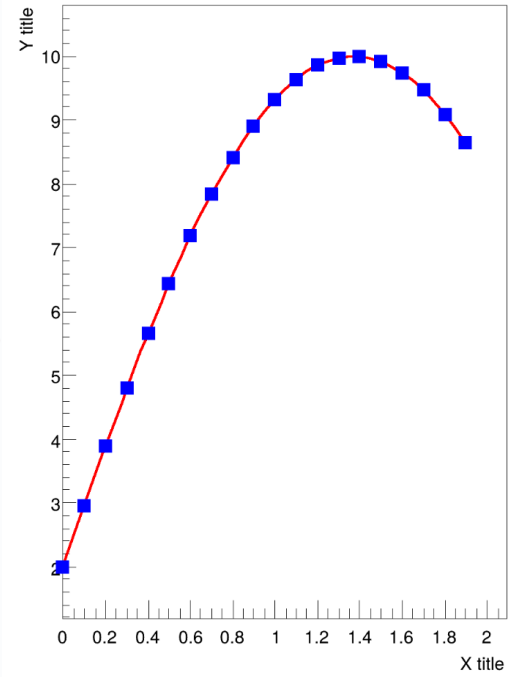
Python




```python
import ROOT
import numpy as np

# Create a canvas to draw the graph
canvas = ROOT.TCanvas("c1", "TGraph Example", 800, 600)

# Define the number of data points
n = 10
# Generate X values as a sequence from 0 to 9
x = numpy.linspace(0, 9, n)
# Compute Y values as the square of X values
y = x**2

# Create a TGraph object using the defined X and Y values
graph = ROOT.TGraph(n, x, y)

# Get the current number of points in the graph
n = graph.GetN()
# Add a new point to the graph at the specified index
graph.SetPoint(n, 10, 100)
```

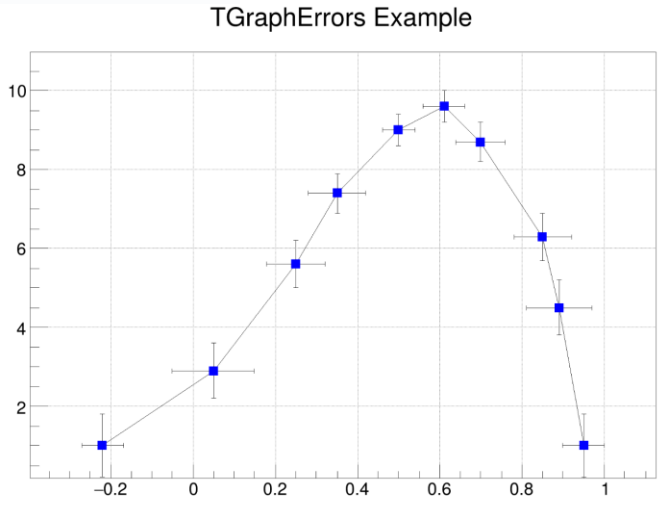
```python
# Retrieve the coordinates of the 3rd point (index 2)
x_val = graph.GetPointX(2)
y_val = graph.GetPointY(2)
print(f"Point {2}: ({x_val}, {y_val})")

# Customize graph appearance
graph.SetTitle("Example Graph;X-axis Title;Y-axis Title")
graph.SetMarkerStyle(21)
graph.SetMarkerSize(2)
graph.SetMarkerColor(4)
graph.SetLineColor(2)
graph.SetLineStyle(1)
graph.SetLineWidth(2)

# Draw the graph with axes ("A"), lines ("L"), and points ("P")
graph.Draw("ALP")
```

```
void gr001_simple() {
   TCanvas *c1 = new TCanvas("c1","Two simple graphs",200,10,700,500);

   c1->Divide(2,1); //Dividing the canvas in subpads for distinguishing the two examples, [See documentation]
       (https://root.cern/doc/master/classTCanvas.html)

   //FIRST EXAMPLE (Available data)
   c1->cd(1);

   const Int_t n = 20; //Fill the arrays x and y with the data points
   Double_t x[n], y[n];
   for (Int_t i=0;i<n;i++) {
     x[i] = i*0.1;
     y[i] = 10*sin(x[i]+0.2);
     printf(" i %i %f %f \n",i,x[i],y[i]);
   }

   TGraph *gr1 = new TGraph(n,x,y); //Create a TGraph object, storing the number of data n and the x, y variable

   //Set the color, width and style for the markers and line
   gr1->SetLineColor(2);
   gr1->SetLineWidth(4);
   gr1->SetMarkerColor(4);
   gr1->SetMarkerStyle(21);
   gr1->SetTitle("Graph from available data"); //Choose title for the graph
   gr1->GetXaxis()->SetTitle("X title"); //Choose title for the axis
   gr1->GetYaxis()->SetTitle("Y title");

   //Uncomment the following line to set a custom range for the x-axis (respectively for the y-axis):
   //gr1->GetXaxis()->SetRangeUser(0, 1.8);

   gr1->Draw("ACP"); //"A" draw axes, "C" = draw a smooth line through the markers (optional) and "P" = draw
       markers for data points
   //Optional customization can be done on a ROOT interactive session
```



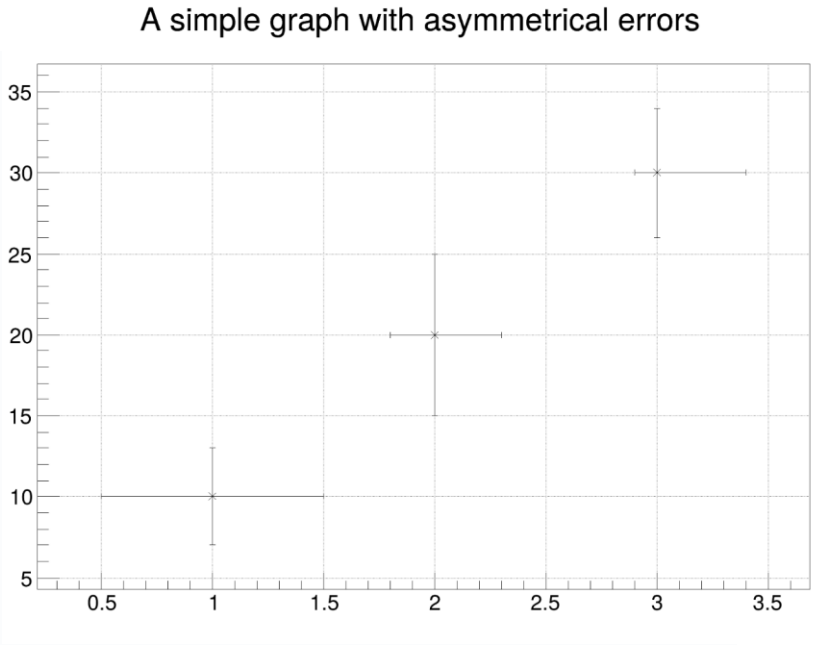
Create and draw a graph with error bars.

```
void gr002_errors() {
   TCanvas *c1 = new TCanvas("c1","A Simple Graph with error bars",200,10,700,500);

   c1->SetGrid();
   c1->GetFrame()->SetBorderSize(12);

   // We will use the constructor requiring: the number of points, arrays containing the x-and y-axis values, and
       arrays with the x- andy-axis errors
   const Int_t n = 10;
   Float_t x[n]  = {-0.22, 0.05, 0.25, 0.35, 0.5, 0.61,0.7,0.85,0.89,0.95};
   Float_t y[n]  = {1,2.9,5.6,7.4,9,9.6,8.7,6.3,4.5,1};
   Float_t ex[n] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};
   Float_t ey[n] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};

   // If all x-axis errors should zero, just provide a single 0 in place of ex
   TGraphErrors *gr = new TGraphErrors(n,x,y,ex,ey);

   gr->SetTitle("TGraphErrors Example");
   gr->SetMarkerColor(4);
   gr->SetMarkerStyle(21);

   // To draw in a new/empty canvas or pad, include the option "A" so that the axes are drawn (leave it out if the
       graph is to be drawn on top of an existing plot
   gr->Draw("ALP");

   c1->Update();
}
```



plot a graph with asymmetrical errors on both the x and y axes.

```
void gr004_errors_asym() {
   TCanvas *c2 = new TCanvas("c2","", 700, 500);

   c2->SetGrid();
   const Int_t npoints=3;
   Double_t xaxis[npoints] = {1.,2.,3.};
   Double_t yaxis[npoints] = {10.,20.,30.};

   Double_t exl[npoints] = {0.5,0.2,0.1}; //Lower x errors
   Double_t exh[npoints] = {0.5,0.3,0.4}; //Higher x errors
   Double_t eyl[npoints] = {3.,5.,4.}; //Lower y errors
   Double_t eyh[npoints] = {3.,5.,4.}; //Higher y errors

   TGraphAsymmErrors *gr = new TGraphAsymmErrors(npoints,xaxis,yaxis,exl,exh,eyl,eyh); //Create the
       TGraphAsymmErrors object with data and asymmetrical errors

   gr->SetTitle("A simple graph with asymmetrical errors");
   gr->Draw("A*"); //"A" = draw axes and "*" = draw markers at the points with error bars
}
```



Reading and Writing Data

- In ROOT, objects are written in files*, represented by **TFile** instances
- TFiles are *binary* and can be compressed (transparently for the user)

```
TFile f("myfile.root", "RECREATE");
```

Option	Description
NEW or CREATE	Create a new file and open it for writing, if the file already exists the file is not opened.
RECREATE	Create a new file, if the file already exists it will be overwritten.
UPDATE	Open an existing file for writing. If no file exists, it is created.
READ	Open an existing file for reading (default).

C++

```
TFile f("file.root", "RECREATE");
TH1F h("h", "h", 64, 0, 8);
h.Write("h");
f.Close();
```

- Write to a file
- Close the file and make sure the operation succeeded

C++

```
TFile f("file.root");
TH1F* h = f.Get<TH1F>("h");
h->Draw();
```

Python

```
import ROOT
f = ROOT.TFile("file.root")
f.h.Draw()
```

Get the histogram by name!
Possible only in Python

Listing TFile Content

- **TBrowser**: interactive tool

```
root [0] TBrowser tb
```

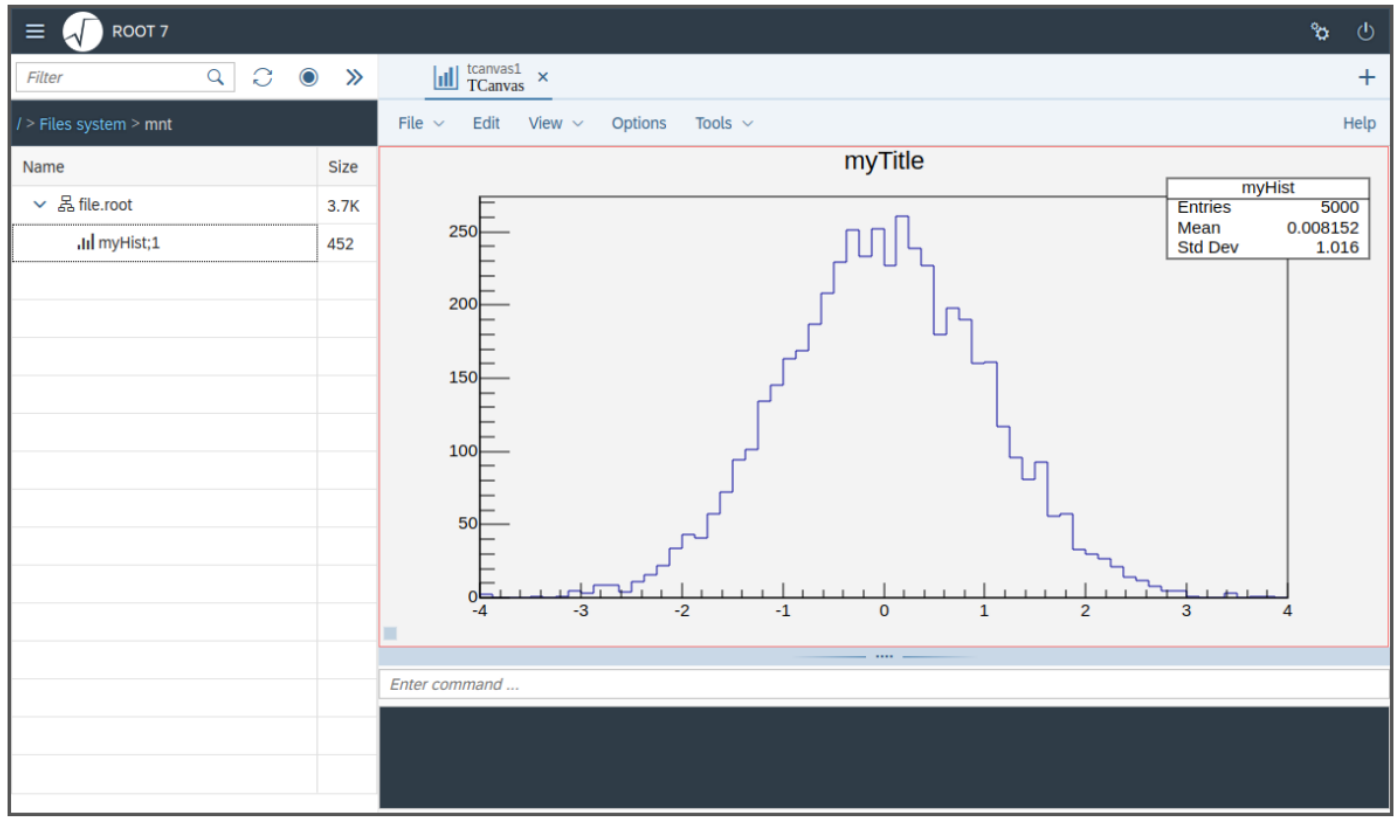
- **rootls** tool: list content

```
> rootls file.root
```

- **TFile::ls()**: prints content

```
> root file.root
root [0]
Attaching file /mnt/file.root as _file0...
(TFile *) 0x562cbf485d50
root [1] _file0->ls()
TFile**         /mnt/file.root
 TFile*         /mnt/file.root
  KEY: TH1D     myHist;1        myTitle
root [2] .q
```

  - great for interactive usage



The new TBrowser in ROOT 6.26.

Trees

https://root.cern/doc/master/group__tutorial__tree.html

TTree represents a columnar dataset. Trees are optimized for reduced disk space and selecting high-throughput columnar access with reduced memory usage.

Read data from an ascii file and create a root file with an histogram and an ntuple.

```
#include <iostream>

void tree101_basic()
{
   // read file $ROOTSYS/tutorials/io/tree/basic.dat
   // this file has 3 columns of float data
   TString dir = gROOT->GetTutorialDir();
   dir.Append("/io/tree/");
   dir.ReplaceAll("/./", "/");
   ifstream in;
   in.open(TString::Format("%sbasic.dat", dir.Data()));

   Float_t x, y, z;
   Int_t nlines = 0;
   auto f = TFile::Open("tree101.root", "RECREATE");
   TH1F h1("h1", "x distribution", 100, -4, 4);
   TNtuple ntuple("ntuple","data from ascii file", "x:y:z");

   while (1) {
      in >> x >> y >> z;
      if (!in.good())
         break;
      if (nlines < 5)
         printf("x = %+8.6f, y = %+8.6f, z = %+8.6f\n", x, y, z);
      h1.Fill(x);
      ntuple.Fill(x, y, z);
      nlines++;
   }
   printf(" found %d points\n", nlines);
   in.close();
   f->Write();
}
```

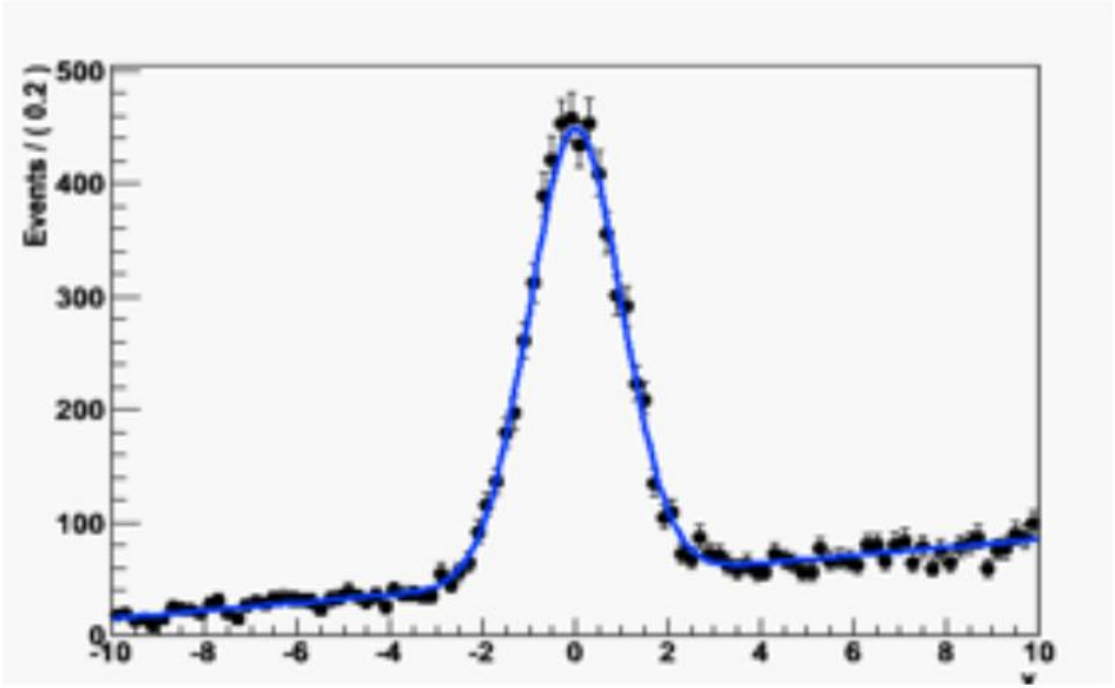
Copy a subset of a Tree to a new Tree, selecting entries.

```
void run()
{
   TFile oldfile("tree108.root");
   TTree *oldtree;
   oldfile.GetObject("t4", oldtree);

   // Deactivate all branches
   oldtree->SetBranchStatus("*", 0);

   // Activate only four of them
   for (auto activeBranchName : {"event_split", "fNtrack", "fNseg", "fH"})
      oldtree->SetBranchStatus(activeBranchName, 1);

   // Create a new file + a clone of old tree in new file
   TFile newfile("tree110.root", "recreate");
   auto newtree = oldtree->CloneTree();

   newtree->Print();
   newfile.Write();
}
```

```
void run()
{
   TFile oldfile("tree108.root");
   TTree *oldtree;
   oldfile.GetObject("t4", oldtree);

   const auto nentries = oldtree->GetEntries();

   Event *event = nullptr;
   oldtree->SetBranchAddress("event_split", &event);

   // Create a new file + a clone of old tree in new file
   TFile newfile("tree112.root", "recreate");
   auto newtree = oldtree->CloneTree(0);

   for (auto i : ROOT::TSeqI(nentries)) {
      oldtree->GetEntry(i);
      if (event->GetNtrack() > 605)
         newtree->Fill();
      event->Clear();
   }

   newtree->Print();
   newfile.Write();
}
```

RooFit: ROOT toolkit for complex fitting

- ROOT fitting can handle complicated functions but complex models require many lines of code
- **RooFit** provides functionality for building complex fitting models
- Fitting often requires **Normalization** of pdfs
  - not always trivial to perform → RooFit does it automatically
- RooFit also provides:
  - **MC data generation** from model
  - advance **visualization** of fitting results
  - **simultaneous fit** to different data samples
  - full model description for **reusability**
  - **built-in optimization** for optimal computational performances



For more info see the manual or the RooFit courses

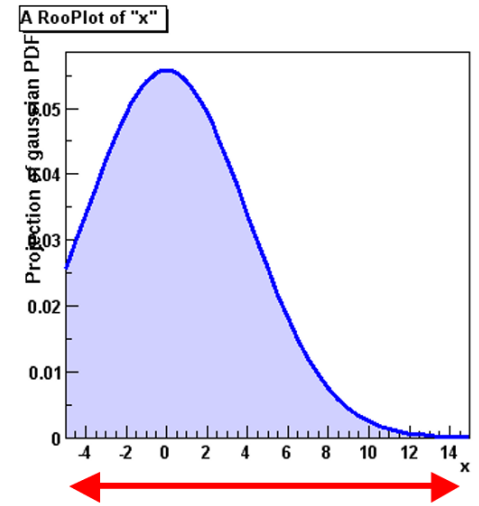
https://root.cern/manual/roofit/

Probability density functions

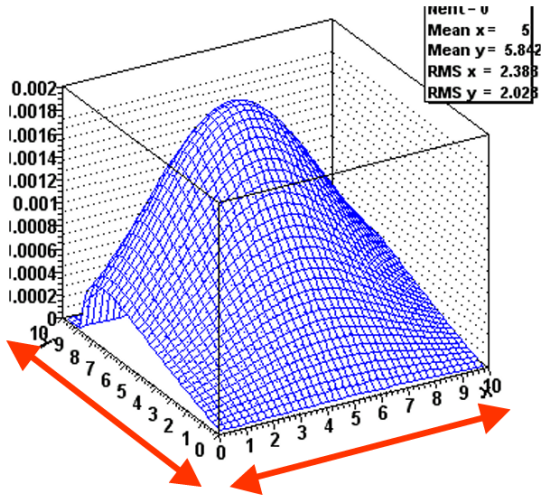
- Fundamental property of any probability density function g(x,p):

$$\int_{\vec{x}_{\min}}^{\vec{x}_{\max}} g(\vec{x}, \vec{p})\, d\vec{x} \equiv 1$$

  - Easy to construct for 1-dim. PDF much more effort for >1 dim.
  - **RooFit automatically takes care of this**
    - User supplied function need not be normalized



$\int Gdx(p) = 1$



$\int Gdxdy(p) = 1$

RooFit example

Define variables (observables and params.):

```
RooRealVar x{"x", "x", 0, 0, 10}; // observable

RooRealVar mu{"mu", "mu", 4, 0, 10};
RooRealVar sigma{"sigma", "sigma", 1, 0.01, 10};
RooRealVar c{"c", "c", -0.1, -10, -0.001};
```

Define pdf (here, Gaussian plus expo.):

```
RooGaussian gauss{"gauss", "gauss", x, mu, sigma};
RooExponential expo{"expo", "expo", x, c};
RooAddPdf model{"model", "0.2 * gauss + 0.8 * expo",
                {gauss, expo}, {RooFit::RooConst(0.2)}};
```

Sample toy dataset:

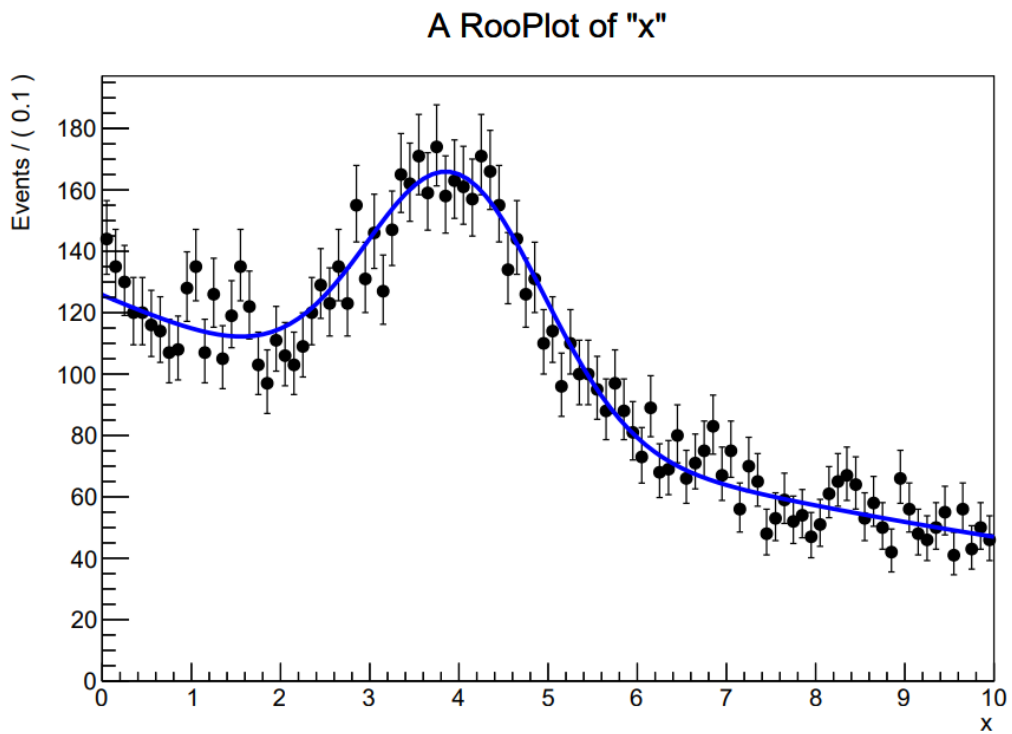
```
std::unique_ptr<RooDataSet> data{model.generate(x, 10000)};
```

Fit model to data with likelihood minim.:

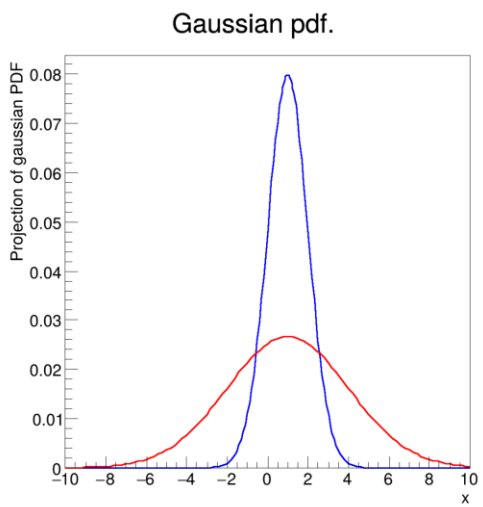
```
std::unique_ptr<RooFitResult> res{model.fitTo(*data)};
res->Print();
```

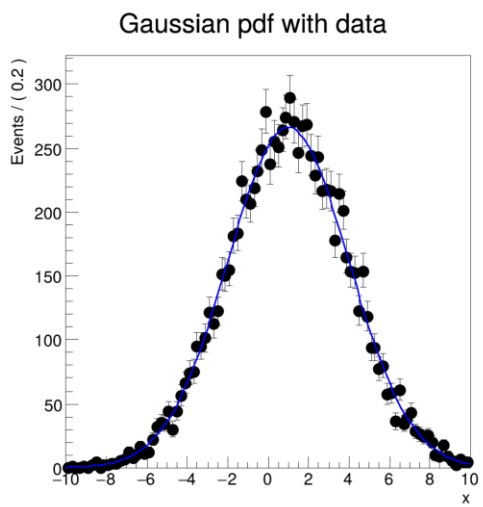
Plotting

```
RooPlot* frame = x.frame();
data->plotOn(frame);
model.plotOn(frame);
```



```
void rf101_basics()
{
   // S e t u p   m o d e l
   // ----------------------

   // Declare variables x,mean,sigma with associated name, title, initial value and allowed ra
   RooRealVar x("x", "x", -10, 10);
   RooRealVar mean("mean", "mean of gaussian", 1, -10, 10);
   RooRealVar sigma("sigma", "width of gaussian", 1, 0.1, 10);

   // Build gaussian pdf in terms of x,mean and sigma
   RooGaussian gauss("gauss", "gaussian PDF", x, mean, sigma);

   // Construct plot frame in 'x'
   RooPlot *xframe = x.frame(Title("Gaussian pdf."));

   // P l o t   m o d e l   a n d   c h a n g e   p a r a m e t e r   v a l u e s
   // ---------------------------------------------------------------------------

   // Plot gauss in frame (i.e. in x)
   gauss.plotOn(xframe);

   // Change the value of sigma to 3
   sigma.setVal(3);

   // Plot gauss in frame (i.e. in x) and draw frame on canvas
   gauss.plotOn(xframe, LineColor(kRed));
```






```
   // G e n e r a t e   e v e n t s
   // -----------------------------

   // Generate a dataset of 1000 events in x from gauss
   std::unique_ptr<RooDataSet> data{gauss.generate(x, 10000)};

   // Make a second plot frame in x and draw both the
   // data and the pdf in the frame
   RooPlot *xframe2 = x.frame(Title("Gaussian pdf with data"));
   data->plotOn(xframe2);
   gauss.plotOn(xframe2);

   // F i t   m o d e l   t o   d a t a
   // -----------------------------

   // Fit pdf to data
   gauss.fitTo(*data, PrintLevel(-1));

   // Print values of mean and sigma (that now reflect fitted values and errors)
   mean.Print();
   sigma.Print();

   // Draw all frames on a canvas
   TCanvas *c = new TCanvas("rf101_basics", "rf101_basics", 800, 400); c->Divide(2);
   c->cd(1);
   gPad->SetLeftMargin(0.15);
   xframe->GetYaxis()->SetTitleOffset(1.6);
   xframe->Draw();
   c->cd(2);
   gPad->SetLeftMargin(0.15);
   xframe2->GetYaxis()->SetTitleOffset(1.6);
   xframe2->Draw();
}
```

- RooFitModels provides a collection of 'building block' PDFs

RooArgusBG	–	Argus background shape
RooBCPEffDecay	–	B0 decay with CP violation
RooBMixDecay	–	B0 decay with mixing
RooBifurGauss	–	Bifurcated Gaussian
RooBreitWigner	–	Breit-Wigner shape
RooCBShape	–	Crystal Ball function
RooChebychev	–	Chebychev polynomial
RooDecay	–	Simple decay function
RooDircPdf	–	DIRC resolution description
RooDstD0BG	–	D* background description
RooExponential	–	Exponential function
RooGaussian	–	Gaussian function
RooKeysPdf	–	Non-parametric data description
Roo2DKeysPdf	–	Non-parametric data description
RooPolynomial	–	Generic polynomial PDF
RooVoigtian	–	Breit-Wigner (X) Gaussian

- More will PDFs will follow
  - Easy to for users to write/contribute new PDFs

Generic expression-based PDFs

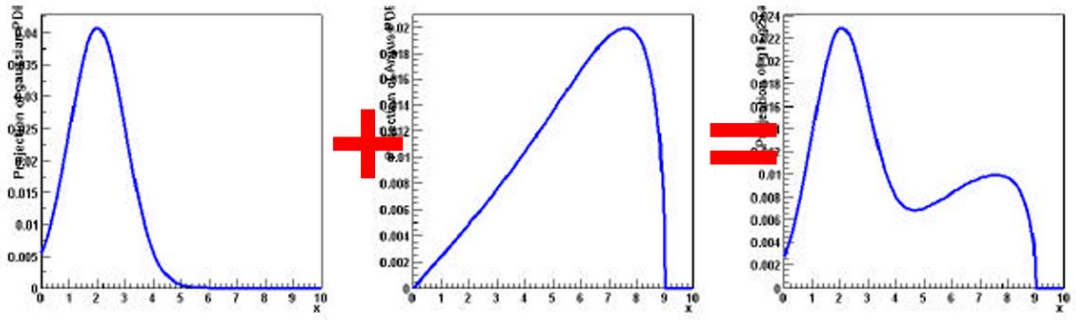
- If your favorite PDF isn't there
  and you don't want to code a PDF class right away
  → use RooGenericPdf
- Just write down the PDFs expression as a C++ formula

```
// PDF variables
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",0,5) ;
RooRealVar a("a","a",3.0) ;
RooRealVar b("b","b",-2.0) ;

// Generic PDF
RooGenericPdf gp("gp","Generic PDF","exp(x*y+a)-b*x",
                 RooArgSet(x,y,a,b)) ;
```

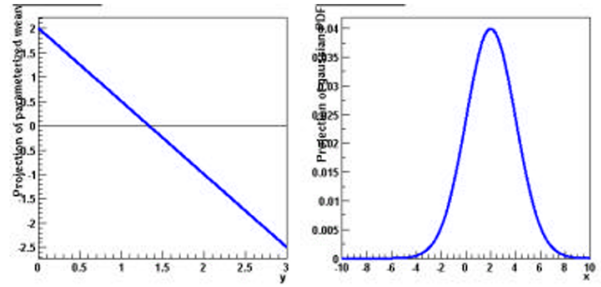


- Automatic normalization
  - Expression divided by numerical integral of expression

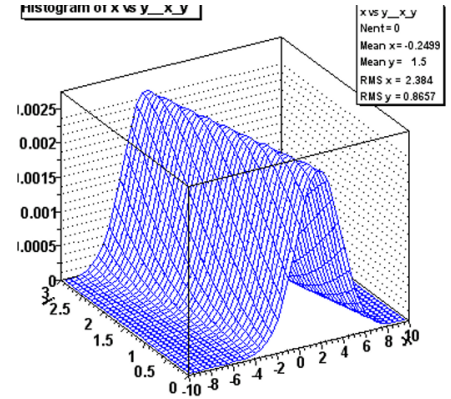
- Complex PDFs be can be trivially composed using operator classes
  - Addition



  - Composition ('plug & play')



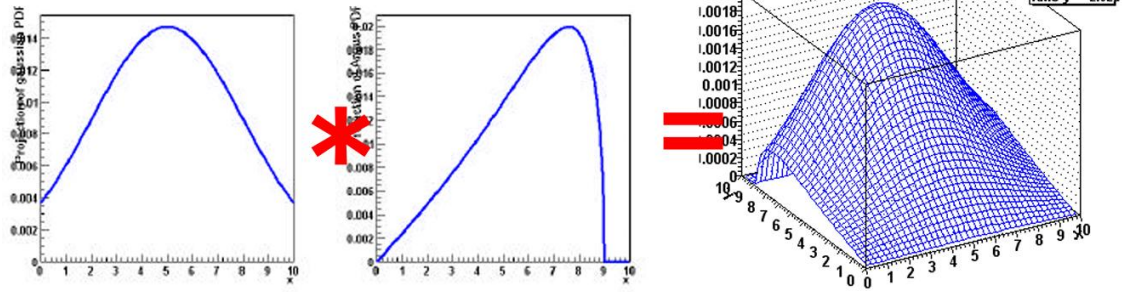
$m(y;a_0,a_1)$ $g(x;m,s)$



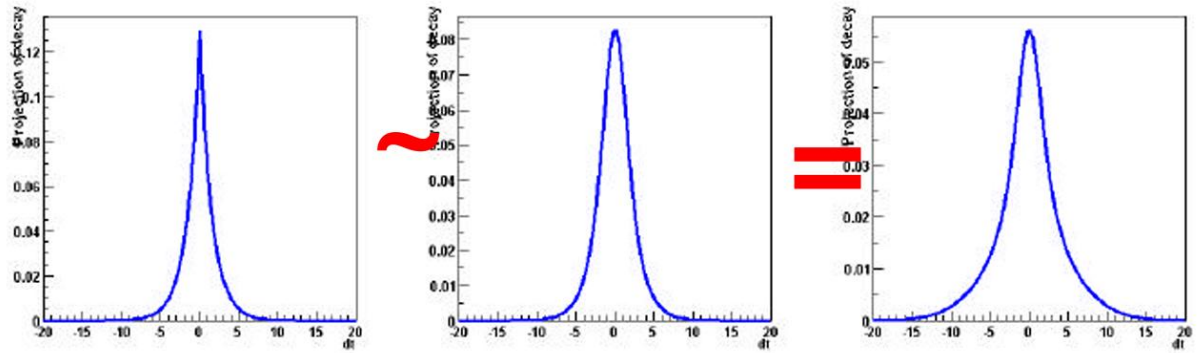
$g(x,y;a0,a1,s)$

Possible in *any* PDF
No explicit support in PDF code needed

  - Multiplication



  - Convolution



Adding PDF components

demo2.cc

RooAddPdf constructs the sum of N PDFs with N-1 coefficients

$$S = c_0 P_0 + c_1 P_1 + c_2 P_2 + \ldots + c_{n-1} P_{n-1} + \left(1 - \sum_{i=0,n-1} c_i\right) P_n$$

Build 2 Gaussian PDFs

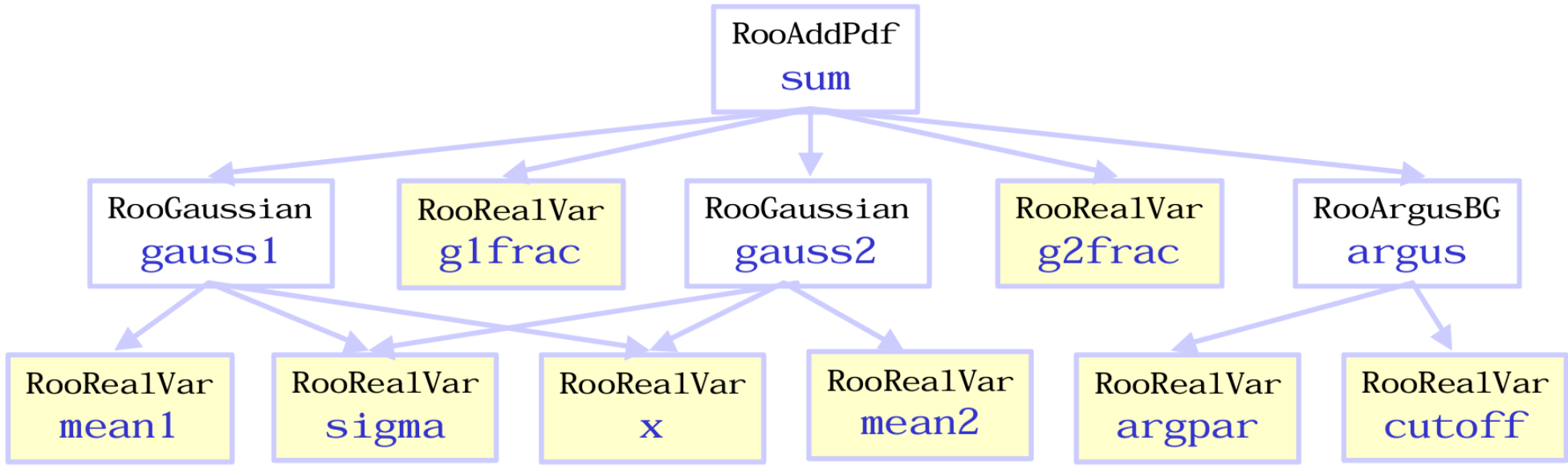
Build ArgusBG PDF

```
// Build two Gaussian PDFs
RooRealVar x("x","x",0,10) ;
RooRealVar mean1("mean1","mean of gaussian 1",2) ;
RooRealVar mean2("mean2","mean of gaussian 2",3) ;
RooRealVar sigma("sigma","width of gaussians",1) ;
RooGaussian gauss1("gauss1","gaussian PDF",x,mean1,sigma) ;
RooGaussian gauss2("gauss2","gaussian PDF",x,mean2,sigma) ;

// Build Argus background PDF
RooRealVar argpar("argpar","argus shape parameter",-1.0) ;
RooRealVar cutoff("cutoff","argus cutoff",9.0) ;
RooArgusBG argus("argus","Argus PDF",x,cutoff,argpar) ;

// Add the components
RooRealVar g1frac("g1frac","fraction of gauss1",0.5) ;
RooRealVar g2frac("g2frac","fraction of gauss2",0.1) ;
RooAddPdf  sum("sum","g1+g2+a",RooArgList(gauss1,gauss2,argus),
                               RooArgList(g1frac,g2frac)) ;
```

List of PDFs

List of coefficients

Parameters of composite PDF objects




```
RooArgSet *paramList = sum.getParameters(data) ;
paramList->Print("v") ;
RooArgSet::parameters:
  1) RooRealVar::argpar : -1.00000 C
  2) RooRealVar::cutoff :  9.0000 C
  3) RooRealVar::g1frac :  0.50000 C
  4) RooRealVar::g2frac :  0.10000 C
  5) RooRealVar::mean1  :  2.0000 C
  6) RooRealVar::mean2  :  3.0000 C
  7) RooRealVar::sigma  :  1.0000 C
```

The parameters of sum are the combined parameters of its components

Multiplying PDF components

demo3.cc

RooProdPdf constructs the product of N PDFs:

$$P = P_0(x_1, x_2) \cdot P_1(y_1, y_2, ..) \cdot P_2(z_1, z_2, ...) \cdot ... P_n(w_1, w_2, ...)$$

Build 2 Gaussian PDFs

```
// Build two Gaussian PDFs
RooRealVar x("x","x",-5,5) ;
RooRealVar y("y","y",-5,5) ;
RooGaussian gaussx("gaussx","gaussx",x,meanx,sigmax);
RooGaussian gaussy("gaussy","gaussx",y,meany,sigmay);

// Multiply the components
RooProdPdf prod("gaussxy","gaussx*gaussy",
                RooArgList(gaussx,gaussy)) ;
```

Component PDFs may not share dependents
e.g. $pdf_1(\mathbf{x},y)*pdf_2(\mathbf{x},z)$ not allowed

Such forms are not very common,
but can be performed with RooGenericPdf
Shared parameters no problem

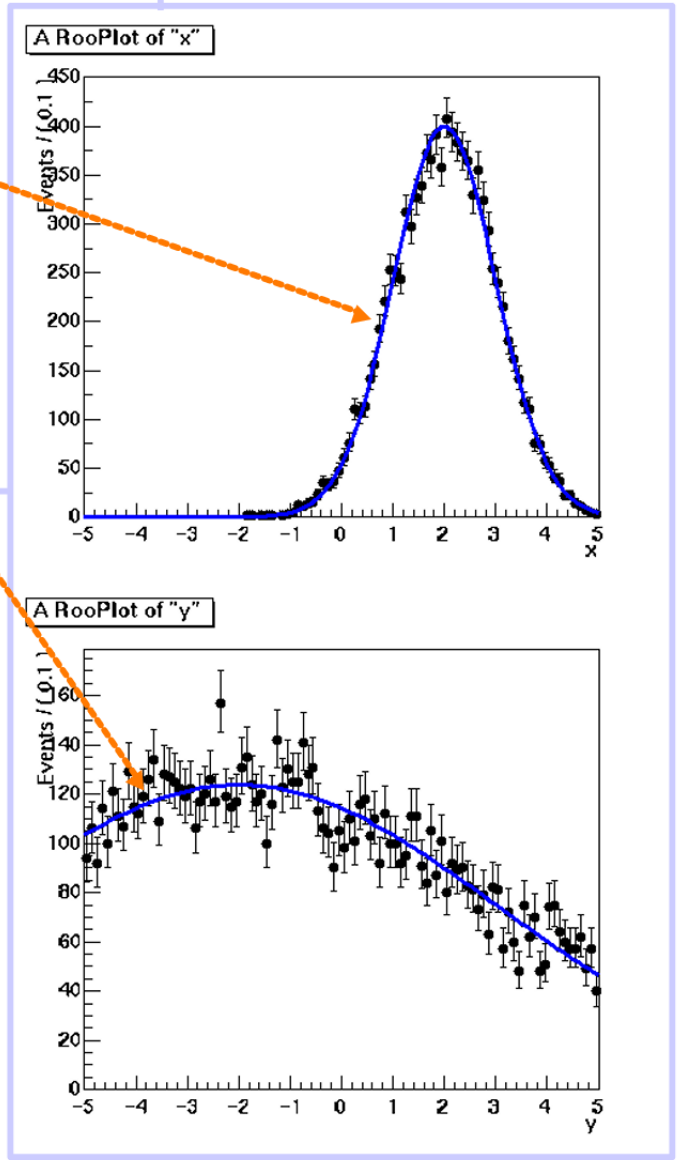
Normalization more complicated

Plotting multi-dimensional PDFs

```
RooPlot* xframe = x.frame() ;
data->plotOn(xframe) ;
prod->plotOn(xframe) ;
xframe->Draw() ;

c->cd(2) ;
RooPlot* yframe = y.frame() ;
data->plotOn(yframe) ;
prod->plotOn(yframe) ;
yframe->Draw() ;
```

$$f(x) = \int pdf(x, y)\, dy$$

$$f(y) = \int pdf(x, y)\, dx$$



-Plotting a dataset D(x,y) versus x represents a *projection over y*

-To overlay PDF(x,y), you must plot *Int(dy)PDF(x,y)*

-RooFit automatically takes care of this!

- RooPlot remembers dimensions of plotted datasets

Tailoring PDFs via composition

demo4.cc

Suppose you want to build a PDF like this

$$PDF(x,y) = gauss(x,m(y),s)$$

$$m(y) = m_0 + m_1 \cdot sqrt(y)$$

How do you do that? Just like that:

```
RooRealVar x("x","x",-10,10) ;
RooRealVar y("y","y",0,3) ;

// Build a parameterized mean variable for gauss
RooRealVar mean0("mean0","mean offset",0.5) ;
RooRealVar mean1("mean1","mean slope",3.0) ;
RooFormulaVar mean("mean","mean0+mean1*y",
                   RooArgList(mean0,mean1,y)) ;

RooRealVar sigma("sigma","width of gaussian",3) ;
RooGaussian gauss("gauss","gaussian",x,mean,sigma);
```

Build a function object $m(y)=m_0+m_1*sqrt(y)$

Simply plug in function mean(y) where mean value is expected!

PDF expects a real-valued object as input, not necessarily a variable

Plug-and-play parameters!

Convoluted PDFs

demo6.cc

- Convoluted PDFs that can be written if the following form can be used in a very modular way in RooFit

$$P(dt,\ldots) = \sum_k c_k(\ldots)\left(f_k(dt,\ldots) \otimes R(dt,\ldots)\right)$$

coefficient

'basis function'

resolution function

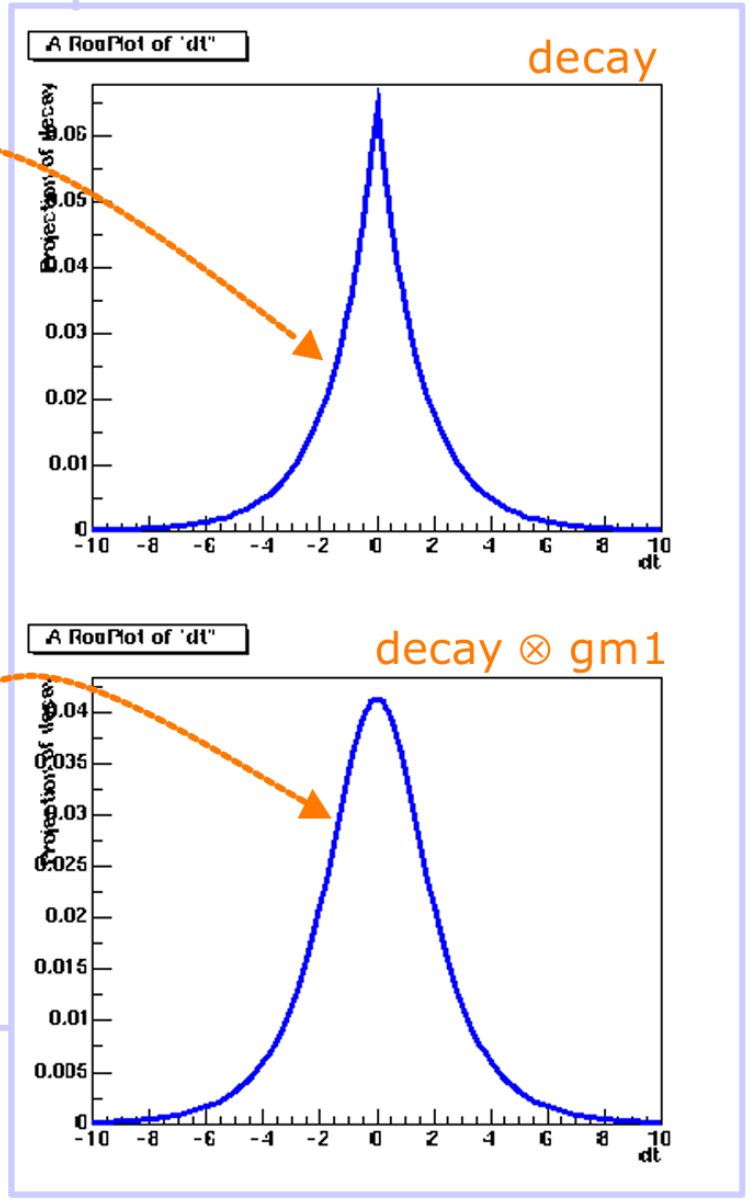
Example: B^0 decay with mixing

$$c_0 = 1 \pm \Delta w, \qquad f_0 = e^{-|t|/t}$$

$$c_1 = \pm(1-2w), \qquad f_1 = e^{-|t|/t}\cos(\Delta m \cdot t)$$

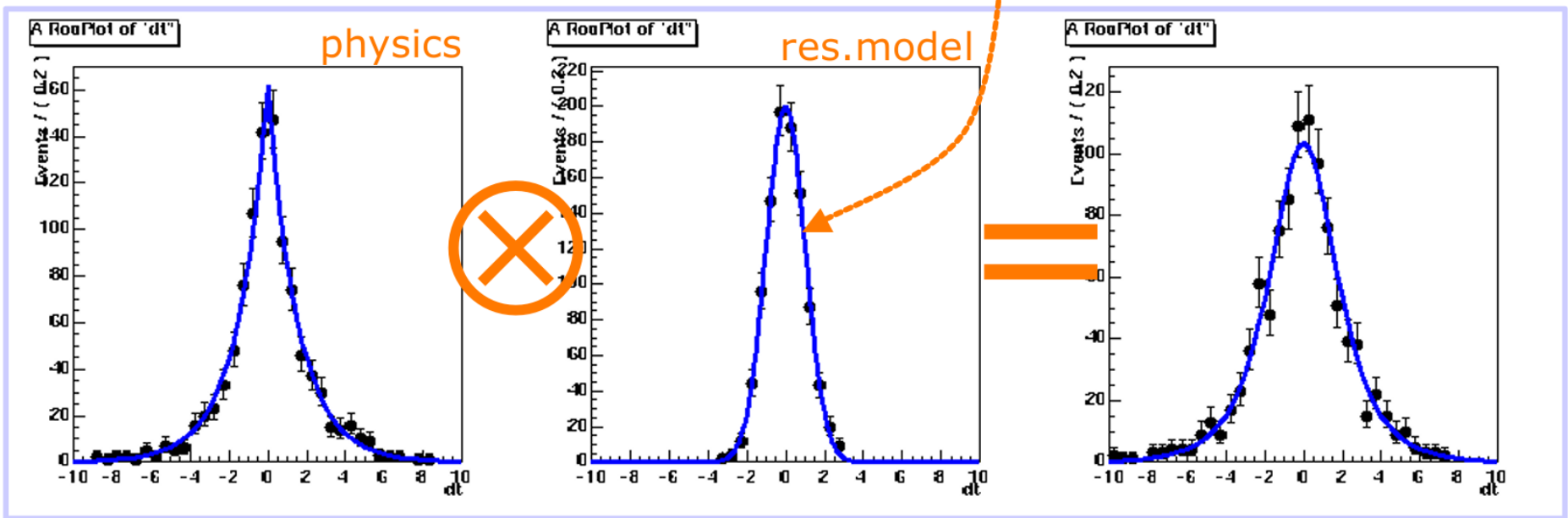
Convoluted PDFs

```
RooRealVar dt("dt","dt",-10,10) ;
RooRealVar tau("tau","tau",1.548) ;

// Truth resolution model
RooTruthModel tm("tm","truth model",dt)

// Unsmeared decay PDF
RooDecay decay_tm("decay_tm","decay",
    dt,tau,tm,RooDecay::DoubleSided) ;

// Gaussian resolution model
RooRealVar bias1("bias1","bias1",0) ;
RooRealVar sigma1("sigma1","sigma1",1) ;
RooGaussModel gm1("gm1","gauss model",
                  dt,bias1,sigma1) ;

// Construct a decay (x) gauss PDF
RooDecay decay_gm1("decay_gm1","decay",
     dt,tau,gm1,RooDecay::DoubleSided) ;
```



Resolution models

- Currently available resolution models
  - `RooGaussModel` – Gaussian with bias and sigma
  - `RooGExpModel` – Gaussian (X) Exp with sigma and lifetime
  - `RooTruthModel` – Delta function
- A `RooResolutionModel` is also a PDF
  - You can use the same resolution model
    you use to convolve your physics PDFs to fit to MC residuals



Managing data subsets / RooSimultaneous

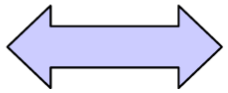
- Simultaneous fit to multiple data samples
  - E.g. to fit PDF_A to dataset D_A and PDF_B to dataset D_B simultaneously, the NLL is

$$NLL = \sum_{i=1,n} -\log(PDF_A(D_A^i)) + \sum_{i=1,m} -\log(PDF_B(D_B^i))$$

- Use categories to split a master dataset D into subsets D_A, D_B etc

Dataset A
5.0
3.7
1.2
4.3

Dataset B
5.0
3.7
1.2



Dataset A+B	
5.0	A
3.7	A
1.2	A
4.3	A
5.0	B
3.7	B
1.2	B

Using categories: RooSimultaneous

RooSimultaneous implements 'switch' PDF:

```
case (indexCat) {
  A: return pdfA ;
  B: return pdfB ;
}
```


Effectively fitting
pdfA to dataA
pdfB to dataB

Create dataset indexing category

Associate created PDFs with appropriate index category state

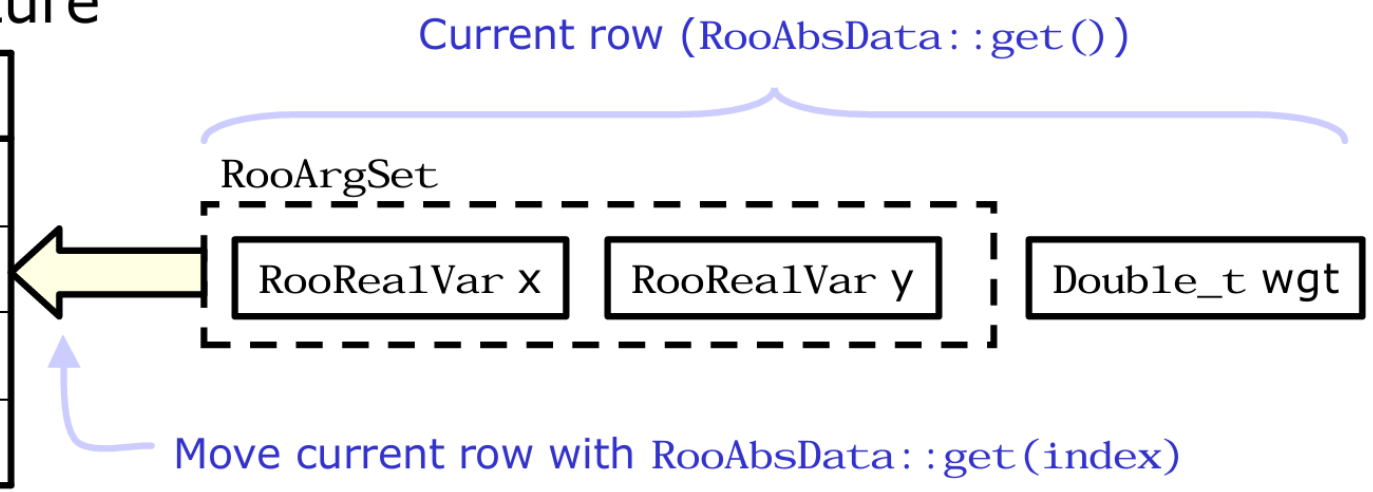
```
// Define a category with labels only
RooCategory tagCat("tagCat","Tagging category") ;
tagCat.defineType("Lepton") ;
tagCat.defineType("Kaon") ;

// Build PDFs for Lepton and Kaon data subsets

// Construct simultaneous PDF for lep and kao
RooSimultaneous simPdf("simPdf","simPdf",tagCat) ;
simPdf.addPdf(pdfLep,"Lepton") ;
simPdf.addPdf(pdfKao,"Kaon") ;
```

An introduction to datasets

demo10.cc

- A dataset is a collection of points in N-dimensional space
  - Dimensions can be either real or discrete
  - Two dataset implementations:
    - `RooDataSet` - unbinned (weighted & unweighted)
    - `RooDataHist` - binned
  - Common abstract base class `RooAbsData`
  - Nearly all RooFit classes/functions (*including fitting*) take `RooAbsData` objects
    - Operations universally supported for binned and unbinned data
- Dataset structure

x	Y	wgt
1.0	6.6	1
3.5	11.1	1
2.7	2.2	1
5.2	1.1	1



Importing data

- Unbinned datasets (RooDataSet) can be constructed from
  - ROOT TTree objects
    - RooRealVar dataset rows are taken /D /F /I tree branches with equal names
    - RooCategory dataset rows are taken from /I /b tree branches with equal names

```
Ttree* tree = <someTFile>.Get("<someTTree>") ;
RooDataSet data("data","data",tree,RooArgSet(x,c));
```

  - ASCII data data files
    - ASCII file fields are interpreted in order of supplied RooArgList

```
RooDataSet* data =
   RooDataSet::read("ascii.file",RooArgList(x,c)) ;
```

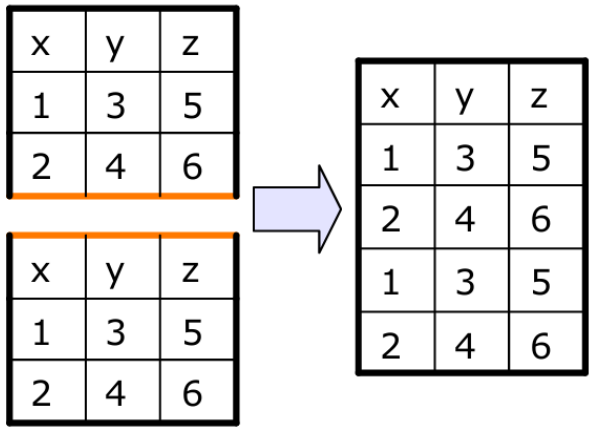
- Binned dataset (RooDataHist) can be constructed from
  - ROOT TH1/2/3 objects
    - TH dimensions are matched in order to supplied list of RooFit value holders

```
TH2* histo = <yourTHistogram> ;
RooDataHist bdata("bdata","bdata",RooArgList(x,y),histo);
```

Extending and reducing unbinned datasets

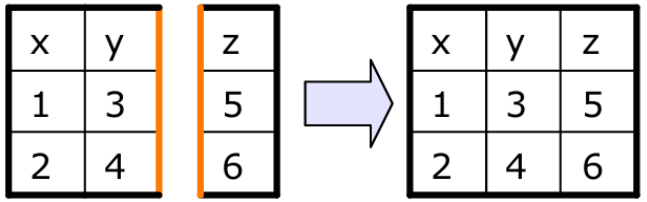
- Appending

```
RooDataSet d1("d1","d1",RooArgSet(x,y,z));
RooDataSet d2("d2","d2",RooArgSet(x,y,z));

d1.append(d2) ;
```



- Merging

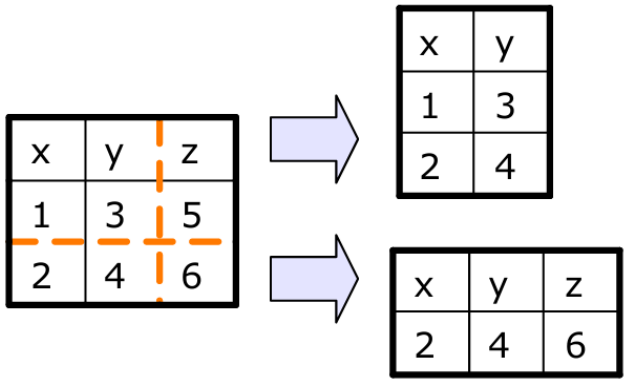
```
RooDataSet d1("d1","d1",RooArgSet(x,y) ;
RooDataSet d2("d2","d2",RooArgSet(z)) ;

d1.merge(d2) ;
```



Same for binned datasets

- Reducing

```
RooDataSet d1("d1","d1",RooArgSet(x,y,z) ;

RooDataSet* d2 = d1.reduce(RooArgSet(x,y));

RooDataSet* d3 = d1.reduce("x>1");
```



Fitting

```
RooAbsData* data ;
RooAbsPdf* pdf ;
RooFitResult* fitres = pdf->fitTo(*data,"<options>") ;
```

- Binned/unbinned fit performed depending on type of dataset (RooDataHist/RooDataSet)
- Fitting options:

MINUIT control options

- "m" = MIGRAD only, i.e. no MINOS
- "s" = estimate step size with HESSE before starting MIGRAD
- "h" = run HESSE after MIGRAD
- "e" = Perform extended MLL fit
- "0" = Run MIGRAD with strategy MINUIT 0 (faster, but no corr. matrix at end) Does not apply to HESSE or MINOS, if run afterwards.

output options

- "q" = Switch off verbose mode
- "l" = Save log file with parameter values at each MINUIT step
- "v" = Show changed parameters at each MINUIT step
- "t" = Time fit
- "r" = Save fit output in RooFitResult object (return value is object RFR pointer)

Browsing fit results with RooFitResult

- As fits grow in complexity (e.g. 45 floating parameters), number of output variables increases
  - Need better way to navigate output that MINUIT screen dump
- RooFitResult holds complete snapshot of fit results
  - Constant parameters
  - Initial and final values of floating parameters
  - Global correlations & full correlation matrix
  - Returned from RooAbsPdf::fitTo() when "r" option is supplied
- Compact & verbose printing mode

Compact Mode

Constant parameters omitted in compact mode

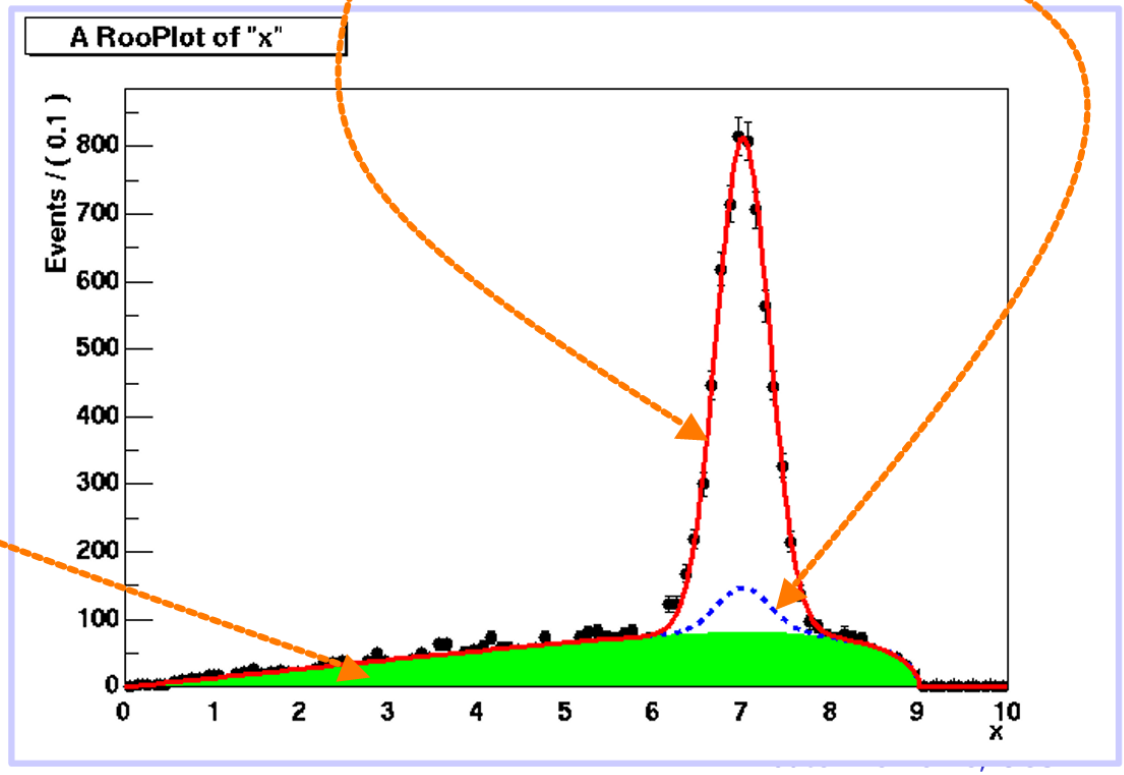
Alphabetical parameter listing

```
fitres->Print() ;

  RooFitResult: min. NLL value: 1.6e+04, est. distance to min: 1.2e-05

    Floating Parameter    FinalValue +/-  Error
  --------------------  --------------------------
                argpar   -4.6855e-01 +/-  7.11e-02
                g2frac    3.0652e-01 +/-  5.10e-03
                 mean1    7.0022e+00 +/-  7.11e-03
                 mean2    1.9971e+00 +/-  6.27e-03
                 sigma    2.9803e-01 +/-  4.00e-03
```

Customization

- Changing colors and styles of histograms and curves

```
sum->plotOn(xframe,Components(RooArgSet(argus)),
            DrawOption("F"), FillColor(kGreen)) ;

sum->plotOn(xframe,Components(RooArgSet(argus,gauss2)),
            LineStyle(kDashed)) ;

sum->plotOn(xframe, LineColor(kRed) ;
```



Addition and Convolution

Tutorial		Description
rf201_composite.C	rf201_composite.py	Composite pdf with signal and background component.
rf202_extendedmlfit.C	rf202_extendedmlfit.py	Setting up an extended maximum likelihood fit.
rf203_ranges.C	rf203_ranges.py	Fitting and plotting in sub ranges.
rf205_compplot.C	rf205_compplot.py	Options for plotting components of composite pdfs.
rf206_treevistools.C	rf206_treevistools.py	Tools for visualization of RooAbsArg expression trees.
rf207_comptools.C	rf207_comptools.py	Tools and utilities for manipulation of composite objects.
rf208_convolution.C	rf208_convolution.py	One-dimensional numeric convolution.
rf209_anaconv.C	rf209_anaconv.py	decay function pdfs with optional B physics effects (mixing and CP violation).
rf210_angularconv.C	rf210_angularconv.py	Convolution in cyclical angular observables theta.
rf211_paramconv.C	rf211_paramconv.py	Working with a pdf with a convolution operator in terms of a parameter.

Organisation and Simultaneous Fits

Tutorial		Description
rf501_simultaneouspdf.C	rf501_simultaneouspdf.py	Using simultaneous pdfs to describe simultaneous fits to multiple datasets.
rf502_wspacewrite.C	rf502_wspacewrite.py	Creating and writing a workspace.
rf503_wspaceread.C	rf503_wspaceread.py	Reading and using a workspace.
rf504_simwstool.C	rf504_simwstool.py	Using RooSimWSTool to construct a simultaneous pdf that is built of variations of an input pdf.
rf505_asciicfg.C	rf505_asciicfg.py	Reading and writing ASCII configuration files.
rf506_msgservice.C	rf506_msgservice.py	Tuning and customizing the RooFit message logging facility.

Multidimensional Models

Tutorial		Description
rf301_composition.C	rf301_composition.py	Multi-dimensional pdfs through composition, e.g. substituting a pdf parameter with a function that depends on other observables.
rf302_utilfuncs.C	rf302_utilfuncs.py	Utility functions classes available for use in tailoring of composite (multidimensional) pdfs.
rf303_conditional.C	rf303_conditional.py	Use of tailored pdf as conditional pdfs.s.
rf304_uncorrprod.C	rf304_uncorrprod.py	Simple uncorrelated multi-dimensional pdfs.
rf305_condcorrprod.C	rf305_condcorrprod.py	Multi-dimensional pdfs with conditional pdfs in product.
rf306_condpereventerrors.C	rf306_condpereventerrors.py	Conditional pdf with per-event errors.
rf307_fullpereventerrors.C	rf307_fullpereventerrors.py	Full pdf with per-event errors.
rf308_normintegration2d.C	rf308_normintegration2d.py	Normalization and integration of pdfs, construction of cumulative distribution functions from pdfs in two dimensions.

Data and Categories

Tutorial		Description
rf401_importttreethx.C	rf401_importttreethx.py	Advanced options for importing data from ROOT TTree and THx histograms.
rf402_datahandling.C	rf402_datahandling.py	Tools for manipulation of (un)binned datasets.
rf403_weightedevts.C	rf403_weightedevts.py	Using weights in unbinned datasets.
rf404_categories.C	rf404_categories.py	Working with RooCategory objects to describe discrete variables.

Likelihood and Minimization

Tutorial		Description
rf601_intminuit.C	rf601_intminuit.py	Interactive minimization with MINUIT.
rf602_chi2fit.C	rf602_chi2fit.py	Setting up a chi^2 fit to a binned dataset.
rf604_constraints.C	rf604_constraints.py	Fitting with constraints.
rf605_profilell.C	rf605_profilell.py	Working with the profile likelihood estimator.
rf606_nllerrorhandling.C	rf606_nllerrorhandling.py	Understanding and customizing error handling in likelihood evaluations.

Special Pdf's

Tutorial		Description
rf701_efficiencyfit.C	rf701_efficiencyfit.py	Unbinned maximum likelihood fit of an efficiency eff(x) function.
rf702_efficiencyfit_2D.C	rf702_efficiencyfit_2D.py	Unbinned maximum likelihood fit of an efficiency eff(x) function to a dataset D(x,cut), cut is a category encoding a selection whose efficiency as function of x should be described by eff(x).
rf703_effpdfprod.C	rf703_effpdfprod.py	Using a product of an (acceptance) efficiency and a pdf as pdf.
rf704_amplitudefit.C	rf704_amplitudefit.py	Using a pdf defined by a sum of real-valued amplitude components.