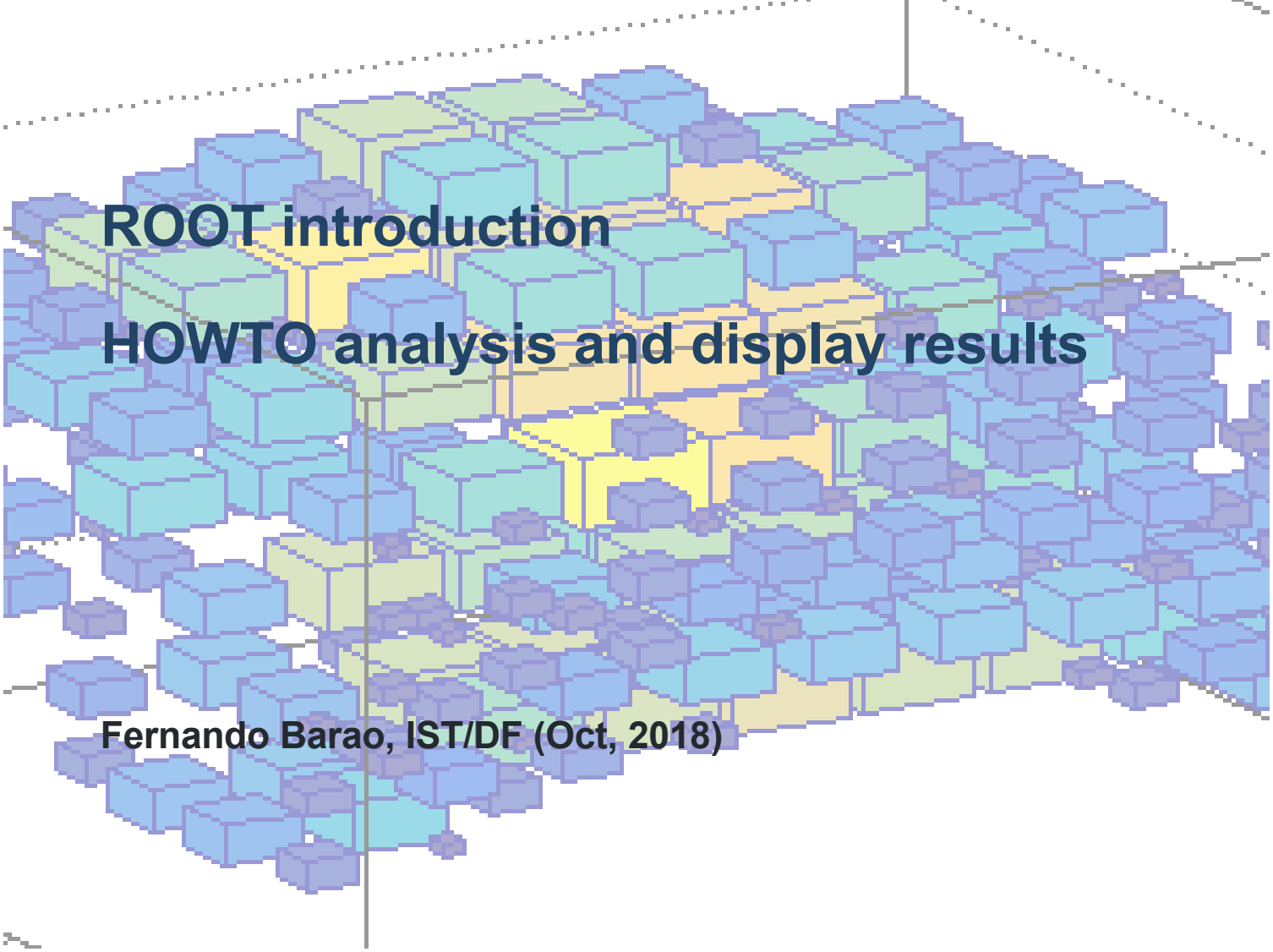

ROOT introduction

HOWTO analysis and display results

Fernando Barao, IST/DF (Oct, 2018)

ROOT instalation: easy way

Check if there exists a binary for your linux flavour

`in case of Debian based distributions (linux mint, ubuntu):`

```
sudo apt-cache search root-system
sudo apt-get -y install root-system
```

ROOT instalation: compiled way

- it's better because optimizes the binary to your hardware and it allows you to install very last version, but
  - it requires a bit more knowledge
  - it takes more time!
  - let's do it

ROOT instalation (cont.)

install auxiliary packages

```
sudo apt-get install git dpkg-dev cmake g++ gcc \
binutils libx11-dev libxpm-dev libxft-dev libxext-dev \
gfortran libssl-dev libpcre3-dev xlibmesa-glu-dev \
libglew1.5-dev libftgl-dev libmysqlclient-dev \
libfftw3-dev libcfitsio-dev graphviz-dev \
libavahi-compat-libdnssd-dev libldap2-dev python-dev \
libxml2-dev libkrb5-dev libgsl0-dev libqt4-dev
```

ROOT instalation (cont.)

define where you want to install it; for instance on below your login directory, on a softw/ created dir

```
mkdir -p ~/softw/root_src
cd ~/softw
```

checkout root source files for last version (6.14.04)

```
git clone http://github.com/root-project/root.git root_sr
cd root_src
git checkout -b v6-14-04 v6-14-04
```


ROOT instalation (cont.)

create root directory to install root

```
mkdir -p ~/softw/root.6_14_04
cd ~/softw/root.6_14_04
```

compile root (be patient!)

```
cmake ../root_src
cmake --build .
```

ROOT post-instalation

now assuming you have **bash shell** (confirm it!)

```
env | grep SHELL
```

you can edit the login file **.bashrc** that is runned every time you login, and place at its end,

```
source ~/softw/root.6_14_04/bin/thisroot.sh
```

and this ends the ROOT process instalation!

Starting and ending a ROOT session

to start a root session,

```
root -l
```

to exit a ROOT session

```
.q
```

display data

suppose we have data measurements on a file named **data.txt**,

```
# x   y    ey
1.    5.   0.15
2.    7.   0.05
3.   ...
```

To plot this data including error bars, we can use the **TGraphErrors** class.

To see marker styles and colors: link

display data with TGraph

example of drawing a graph with circles:

```
TGraphErrors g("data.txt","%lg %lg %lg");
g.SetTitle("main title ; X-axis title ; Y-axis title");
g.SetMarkerStyle(kCircle);
g.DrawClone("AL"); //A=draw axis, L=connect points with l
```

if you intend to introduce a legend box:

```
TLegend leg(.1,.8,.4,.9,"legend title");
leg.SetFillColor(0);
leg.AddEntry(&g,"values");
leg.DrawClone("Same");
```

a full C++ macro to save plot

```
TCanvas *c = new TCanvas();
c->SetGrid();
// graph
TGraphErrors g("data.txt","%lg %lg %lg");
g.SetTitle("main title ; X-axis title ; Y-axis title");
g.SetMarkerStyle(kCircle);
g.DrawClone("AL"); //A=draw axis, L=connect points with l
// legend
TLegend leg(.1,.8,.4,.9,"legend title");
leg.SetFillColor(0);
leg.AddEntry(&g,"values");
leg.DrawClone("SAME");
// save plot (para meter no famoso relatà¸£à¸“rio!)
c->SaveAs("graph.png");
c->SaveAs("graph.pdf");
```


save TGraph to a ROOT file

suppose you want to keep you ROOT object in a file to edit it later on,

```
// open file
TFile *f = TFile::Open("MyFile.root", "RECREATE");
// graph
TGraphErrors *g = new TGraphErrors("data.txt","%lg %lg %l
g->SetTitle("main title", 'X-axis title", "Y-axis title")
g->SetMarkerStyle(kCircle);
g->DrawClone("AL"); //A=draw axis, L=connect points with
// save to file
g->Write("graph1"); //graph1 is the object name (unique)
```

functions in ROOT

we can define n-dim functions in ROOT. Let's start with one dimensional function $f(x) = a\ sin(x)/x$

```
TF1 *f1 = new TF1("f1","[0]*sin(x)/x", -TMath::Pi(), +TMa
```

The parameter **a** can be defined before drawing the function,

```
f1->SetParameter(0, 2.234);
f1->Draw();
```


function fancy details

we can specify the line width, the line color, etc..

```
TF1 *f1 = new TF1("f1","[0]*sin(x)/x", -TMath::Pi(), +TMa
f1->SetParameter(0, 2.234);
f1->SetLineWidth(3);
f1->SetLineColor(kRed);
f1->SetLineStyle(2); //dashed
f1->Draw();
```

we can generate random numbers from function:

```
double x = f1->GetRandom();
```

User function

To plot a function defined by the user, start by defining the function, in this case **myFunction**,

```
Double_t myFunction(Double_t *x, Double_t *par) {
   return (par[0]*sin(par[1]*x[0])
}
```

And after use it to display it,

```
TF1 *f = new TF1("myfunc", myFunction, 0, 6.3, 2); // 2 p
f->SetParameter(0, 1.0);
f->SetParameter(1, 2.0);
f->SetLineColor(kRed);
f->SetLineStyle(1); //full line
f->Draw();
```


2-dim function

We will use class **TF2** to define function,

$$1000\ [(a\ sin(x)/x)\ (b\ sin(y)/y)] + 200$$

```
TF2 f2("f2","1000*(([0]*sin(x)/x)*([1]*sin(y)/y))+200", -
```

Set a, b parameters to 1, and draw function

```
f2.SetParameters(1,1);
```

2-dim function: drawing

Let's define line width, line color and finally draw function with "Surf1" option (see here list of options)

```
f2.SetLineWidth(1);
f2.SetLineColor(kBlue-5);
TF2 *f2c = (TF2*)f2.DrawClone("Surf1");
```

Note: The **TH1::DrawClone()** method allows to pass from a local variable (**f2**) to a **heap** variable (not deleted by the ROOT interpreter). It returns a pointer to a function ***clone***

Random numbers

To generate random numbers we use class **TRandom3**. Let's generate randoms:

```
TRandom3 r; //default seed
double x = r.Rndm(); //gen uniform random between 0 and 1
double x = r.Uniform(a,b); //uniform random between a and
double x = r.Gaus(mean,sigma);
double x = r.Exp(a);
```

2D graph

We can use **TGraph2DErrors** class to create 2D graph. Let's fill it with random numbers from TF2:

```
const int nd=100; // number of points
TF2 f2("f2","1000*(([0]*sin(x)/x)*([1]*sin(y)/y))+200", -
f2.SetParameters(1,1);
```

Choose (x, y) randomly,

```
TGraph2DErrors g2d(nd);
double rnd, x, y, z, ex, ey, ez;
for (Int_t i=0; i<nd; i++) {
  f2.GetRandom2(x,y);
```


2D graph (cont.)

```
  rnd = gRandom->Uniform(-e,e);
  z = f2.Eval(x,y)*(1+rnd);
  g2d.SetPoint(i,x,y,z);
  ex = 0.05*gRandom->Uniform();
  ey = 0.05*gRandom->Uniform();
  ez = fabs(z*rnd);
  g2d.SetPointError(i,ex,ey,ez);
}
```

Now plot it:

```
g2d.DrawClone("surf1");
```

histograms

Different classes available: **TH1D**, **TH2D**, **TH3D**
Let's start with 1-dim histogram filled with gaussian distribution (10 000 numbers gen) and draw it:

```
TH1F h("h","gaussian",100,-2,2); //100 bins(width=4/100)
h.FillRandom("gaus",1000); //fill with gaussian random
h.Fit("gaus"); // fit histogram wit gaussian function
h.SetMarkerStyle(kCircle);
h.DrawClone("hist EP");`
TF1 *f = h->GetFunction("gaus");
f->Draw("same");
```


histograms: fit results

retrieve now fit results:

```
Double_t chi2 = f->GetChisquare(); // value of the 1st pa
Double_t p1 = f->GetParameter(0); // error of the 1st par
Double_t e1 = f->GetParError(0);
```

access to covariance matrix,

```
TFitResultPtr r = f->Fit(“gaus”, “S”);
TMatrixD cov = r->GetCorrelationMatrix();
TMatrixD cor = r->GetCovarianceMatrix();
cov.Print();
cor.Print();
```

Very important: To display the Fit Panel right click on histogram to pop up the context menu, and then select the menu entry Fit Panel.

other tricks

Set log scale

```
gPad->SetLogx()
gPad->SetLogy()
```

Define axis labels,

```
f->GetXaxis()->SetTitle(“x-axis”)
f->GetYaxis()->SetTitle(“y-axis”)
```

X range

```
h->GetXaxis()->SetUserRange(xmin, xmax);
```


Integrate and derivative of a function

```
TF1 *f = new TF1("func","4.5*TMath::Power(x,0.7)/(1.+650.
f->SetNpx(10000);
f->Draw();
f->Eval(1.4);
f->Integral(0.2,1.7);
f->Derivative(0.4);
```