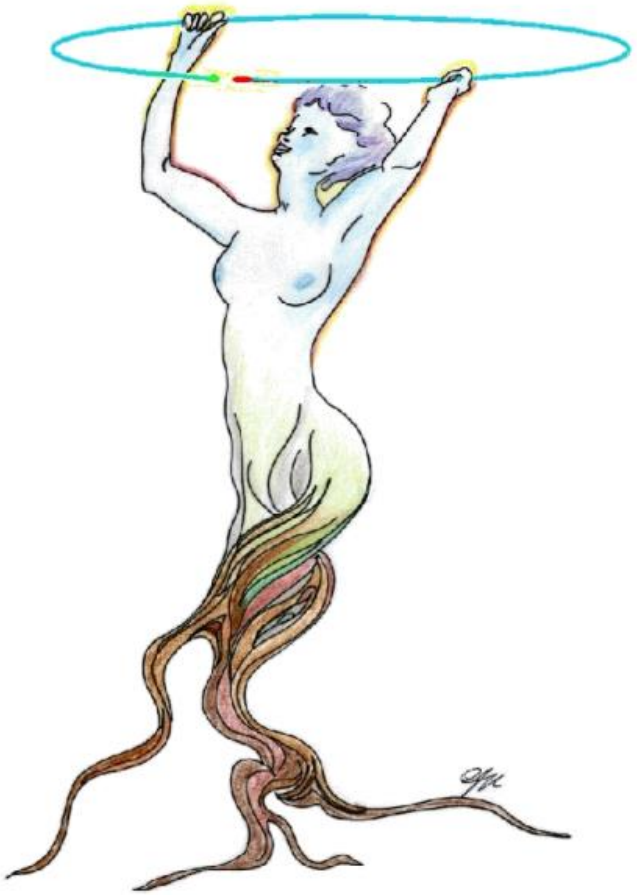
ROOT
Some Tips and Tricks



Manuel Calderon de la Barca Sanchez
UC Davis

ROOT: http://root.cern.ch/

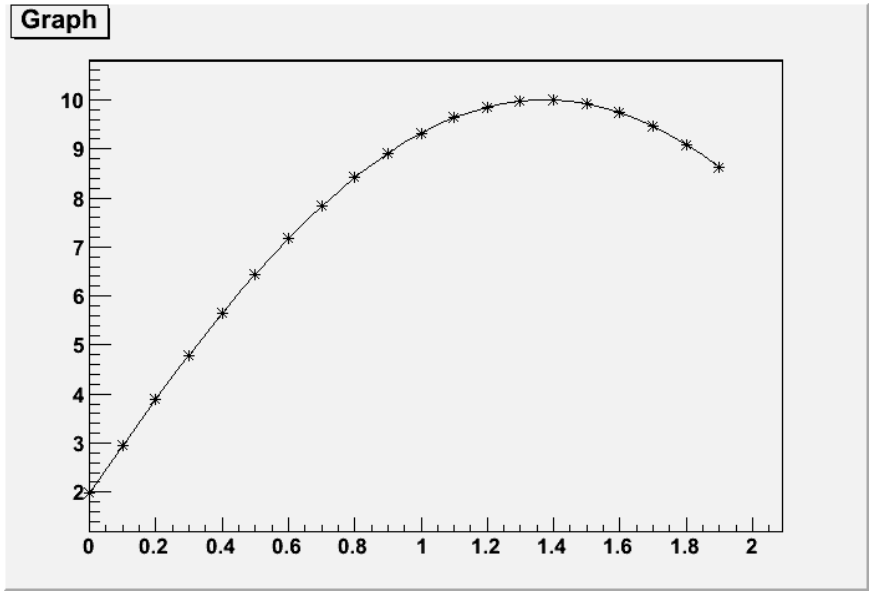
Resources for ROOT

- ROOT Web page:
  - http://root.cern.ch/
- User guides
  - http://root.cern.ch/root/doc/RootDoc.html
- Tutorials
  - $ROOTSYS/tutorials/

- This talk:
  - Use some examples from tutorials
  - Add some other “real world” examples

TGraph Example, from ROOT web

```
{
   TCanvas *c1 = new TCanvas("c1","A Simple Graph
Example",200,10,700,500);
   Double_t x[100], y[100];
   Int_t n = 20;
   for (Int_t i=0;i<n;i++) {
     x[i] = i*0.1;
     y[i] = 10*sin(x[i]+0.2);
   }
   gr = new TGraph(n,x,y);
   gr->Draw("AC*");
   return c1;
}
```



Graph Draw Options

The various draw options for a graph are explained in **TGraph**::PaintGraph. They are:

- "L" A simple poly-line between every points is drawn
- "F" A fill area is drawn
- "F1" Idem as "F" but fill area is no more repartee around X=0 or Y=0
- "F2" draw a fill area poly line connecting the center of bins
- "A" Axis are drawn around the graph
- "C" A smooth curve is drawn
- "*" A star is plotted at each point
- "P" The current marker of the graph is plotted at each point
- "B" A bar chart is drawn at each point
- "[]" Only the end vertical/horizontal lines of the error bars are drawn. This option only

applies to the **TGraphAsymmErrors**.

- "1" ylow = rwymin

The options are not case sensitive and they can be concatenated in most cases. Let us look at some examples

tutorials/hist/fillrandom.C

TCanvas *c1 = new TCanvas("c1","The FillRandom example",200,10,700,900); //last 4 arguments: top x-coord of window, top y-coord of window, x width, y width

c1->SetFillColor(18);

pad1 = new TPad("pad1","The pad with the function",0.05,0.50,0.95,0.95,21);

pad2 = new TPad("pad2","The pad with the histogram",0.05,0.05,0.95,0.45,21);

pad1->Draw();

pad2->Draw();

The Pad Constructor:

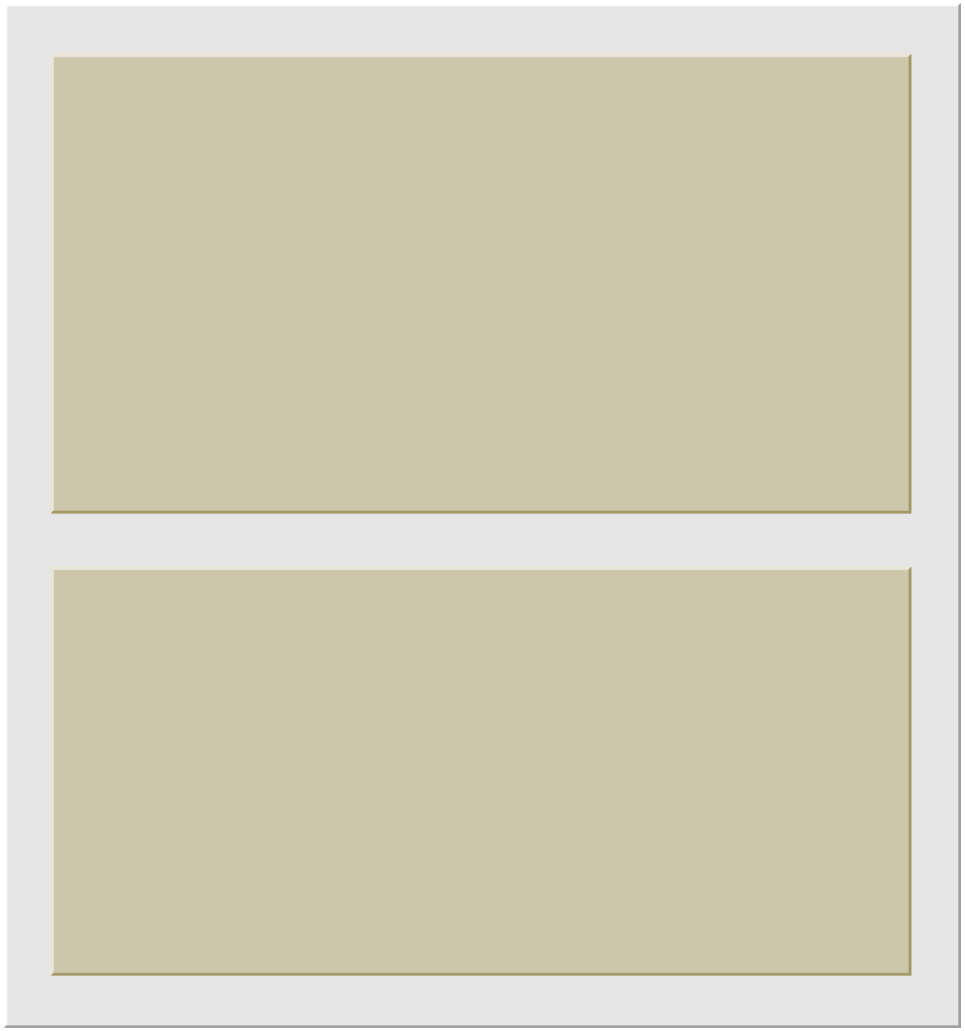
TPad(const char* name, const char* title, Double_t xlow, Double_t ylow, Double_t xup, Double_t yup, Color_t color = -1, Short_t bordersize = -1, Short_t bordermode = -2)

Result of Canvas and Pad creation

Canvas:
700 px wide, 900 px high

Pad 1:
Lower left corner:
5% of width from left edge
50% of height from low edge
Upper right corner:
95% of width from left edge
95% of height from low edge

Canvas Fill color : 18
Pad Fill color: 21



fillrandom.C : Drawing function

```
pad1->cd();
form1 = new TFormula("form1","abs(sin(x)/x)");
sqroot = new TF1("sqroot","x*gaus(0) + [3]*form1",0,10);
sqroot->SetParameters(10,4,1,20);
pad1->SetGridx();
pad1->SetGridy(); pad1->GetFrame()->SetFillColor(42);
pad1->GetFrame()->SetBorderMode(-1);
pad1->GetFrame()->SetBorderSize(5);
sqroot->SetLineColor(4);
sqroot->SetLineWidth(6);
sqroot->Draw();
lfunction = new TPaveLabel(5,39,9.8,46,"The sqroot
function");
lfunction->SetFillColor(41);
lfunction->Draw();
c1-Update();
```

Output after drawing function

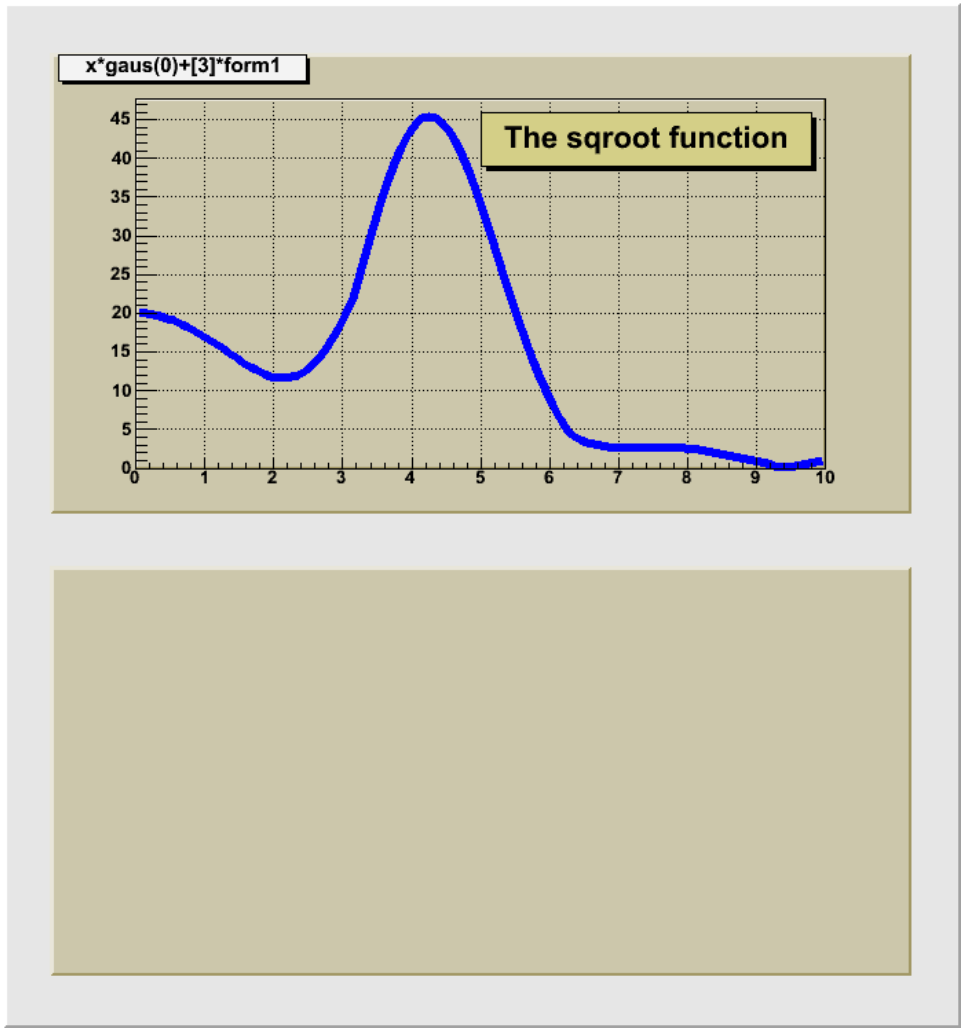
TFormula is drawn

Width of line is 2
Line Color 4 (blue)

Grids are drawn
both vertically and horizontally

TPaveLabel is drawn.
Fill Color is 41

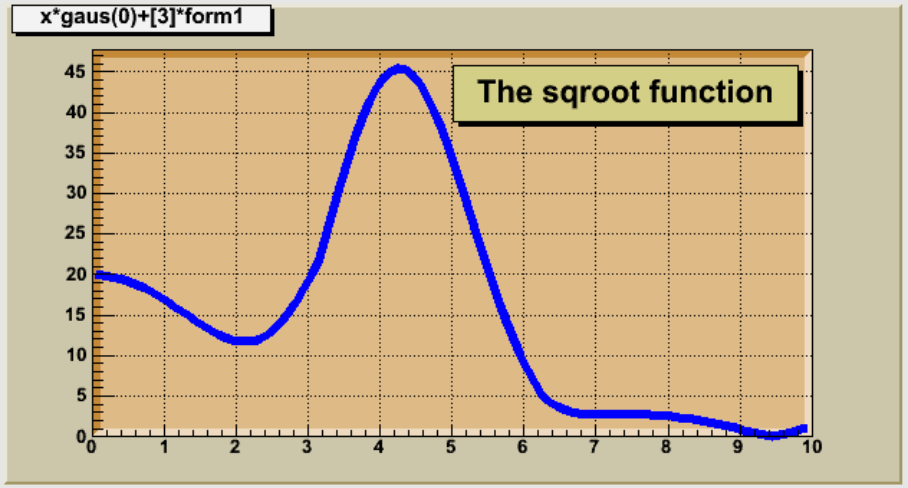
Question: does the Frame have a different color than the Pad? Should it?
What about the frame border?



What should have happened...

- After executing fillrandom, type the following lines at the command prompt:

```
pad1->GetFrame()->SetFillColor(42);
pad1->GetFrame()->SetBorderMode(-1);
pad1->GetFrame()->SetBorderSize(5);
```



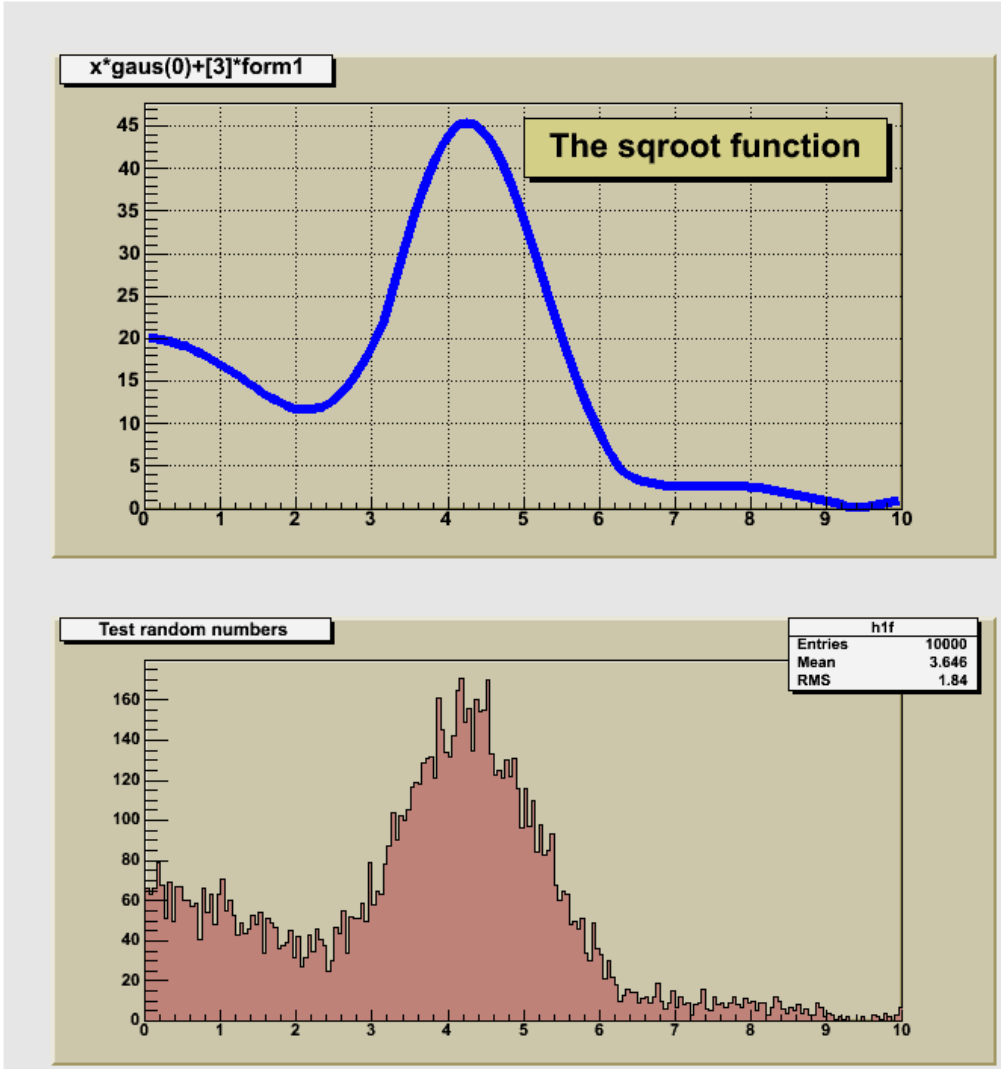
Fill a histogram randomly from TF1

```
pad2->cd();
pad2->GetFrame()->SetFillColor(42);
pad2->GetFrame()->SetBorderMode(-1);
pad2->GetFrame()->SetBorderSize(5);
h1f = new TH1F("h1f","Test random numbers",200,0,10);
h1f->SetFillColor(45);
h1f->FillRandom("sqroot",10000);
h1f->Draw();
c1->Update();
```

Canvas after filling TH1

- Histogram is filled with 10K entries
- Stat box displays
  - Entries, Mean, RMS
- Title is displayed
- TH1 Fill color : 45

- Note: Frame in pad2 did not change color, bordermode, bordersize



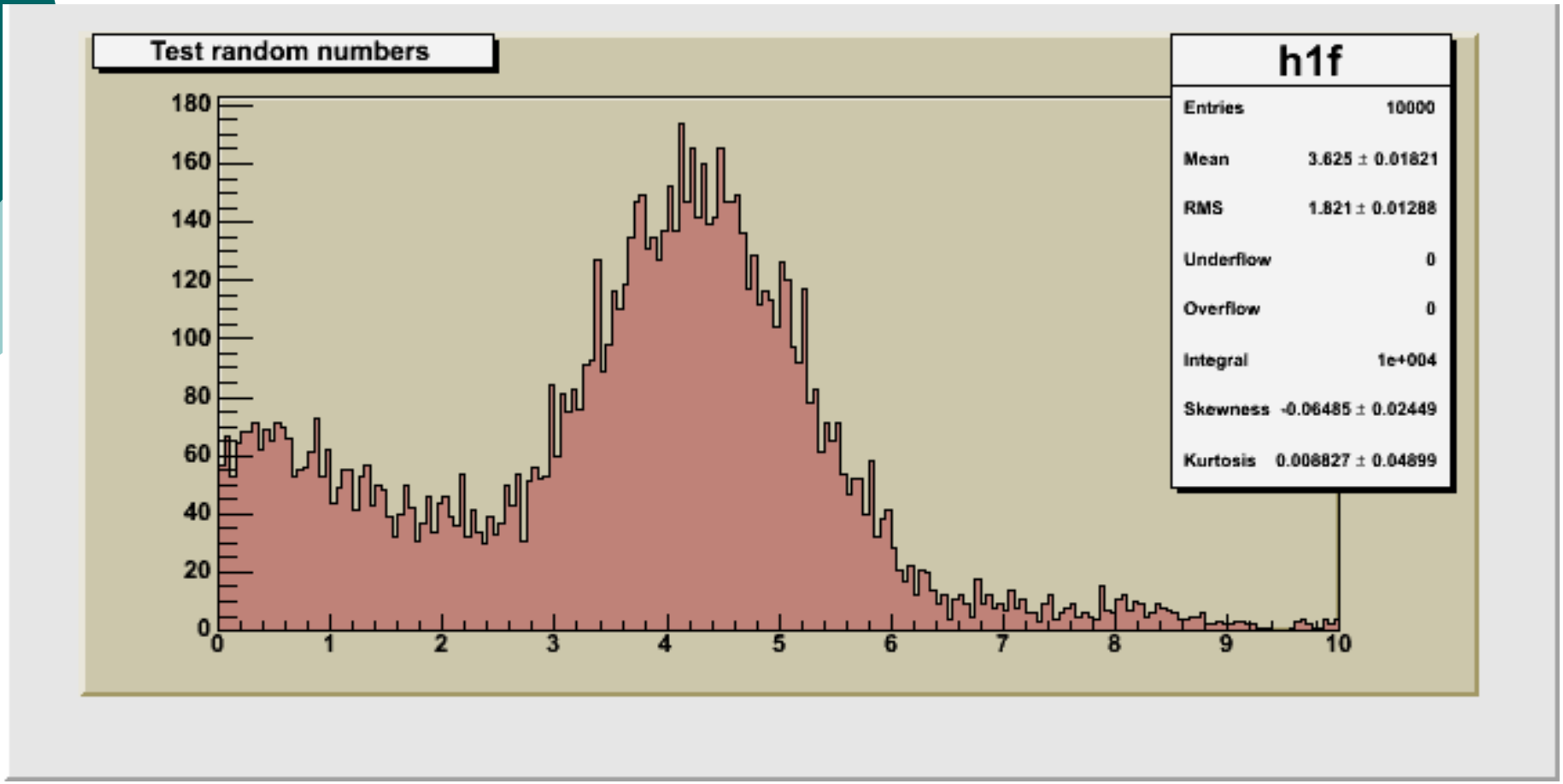
Changing Stat Box options

- Do not display the Stat Box
  - gStyle->SetOptStat(0);
- Things that can be displayed in Stat Box
  - Name, Entries, Mean, RMS, Underflow, Overflow, Integral, Skewness, Kurtosis.
- Traditional way of turning them on:
  - Each one is turned on by a bit, order as in previous bullet.
    - Name is LSB, Kurtosis is MSB.
  - Example: gStyle->SetOptStat(111110110)
    - Turns on all, except RMS and Name.
- But there is an updated way of turning them on ...

Changing StatBox options, updated

```
// The parameter mode can be any combination of
// kKsSiourRmMen
// k : kurtosis printed
// K : kurtosis and kurtosis error printed
// s : skewness printed
// S : skewness and skewness error printed
// i : integral of bins printed
// o : number of overflows printed
// u : number of underflows printed
// r : rms printed
// R : rms and rms error printed
// m : mean value printed
// M : mean value mean error values printed
// e : number of entries printed
// n : name of histogram is printed
```

Displaying all Stat Box Options



- gStyle->SetOptStat("kKsSiourRmMen");
- Rule of thumb: Don't use it if you don't have to.
  - Most useful stat box variables: entries, under-, overflows

Use gStyle and rootlogon.C

- gStyle can help you streamline your code
- Gives your plots a consistent look
- Use the rootlogon.C macro:
  - There are three levels of logon macros that will be executed: the system logon etc/system.rootlogon.C, the global user logon ~/.rootlogon.C and the local ./.rootlogon.C.
  - For backward compatibility also the logon macro as specified by the Rint.Logon environment setting, by default ./rootlogon.C, will be executed.
  - No logon macros will be executed when the system is started with the -n option.

My own rootlogon.C

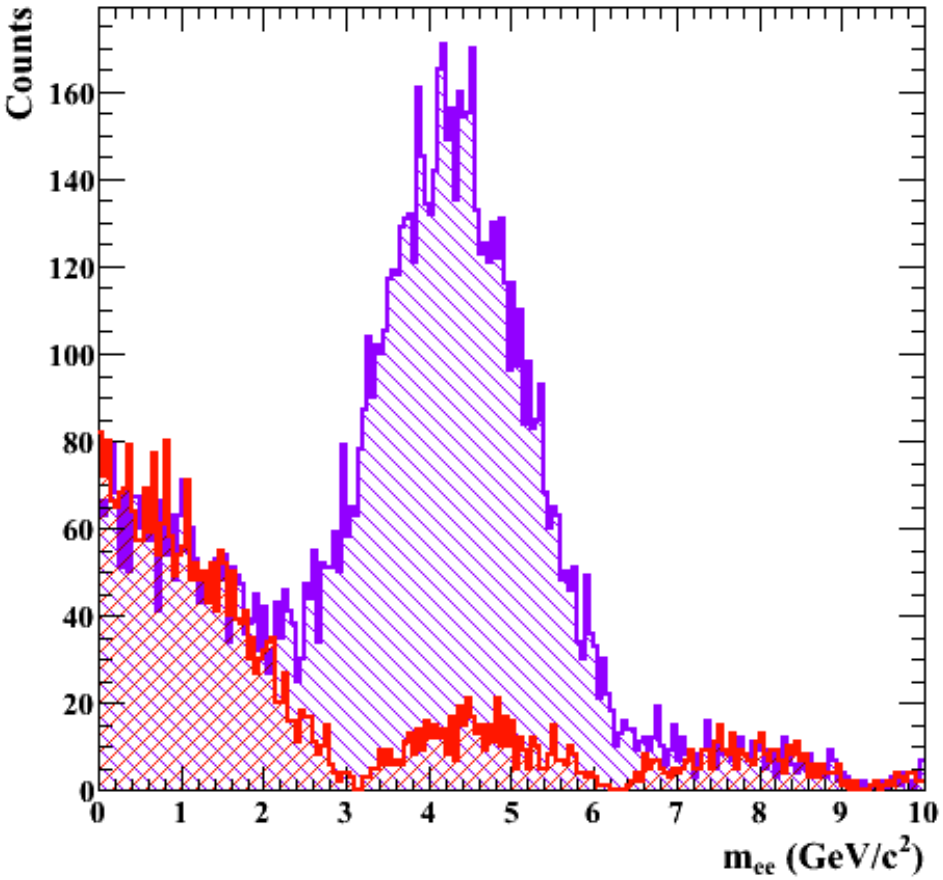
```
// rootlogon.C
// Manuel Calderon de la Barca
{// Add my own options here:
   TStyle* mcStyle = new TStyle("mcStyle","Manuel's Root
Styles");
   mcStyle->SetPalette(1,0); // avoid horrible default color scheme
   mcStyle->SetOptStat(0);
   mcStyle->SetOptTitle(0);
   mcStyle->SetOptDate(0);
   mcStyle->SetLabelSize(0.03,"xyz"); // size of axis value font
   mcStyle->SetTitleSize(0.035,”xyz”); // size of axis title font
   mcStyle->SetTitleFont(22,”xyz”); // font option
   mcStyle->SetLabelFont(22,"xyz");
 mcStyle->SetTitleOffset(1.2,"y");
// default canvas options
   mcStyle->SetCanvasDefW(500);
   mcStyle->SetCanvasDefH(500);
```

My rootlogon.C continued

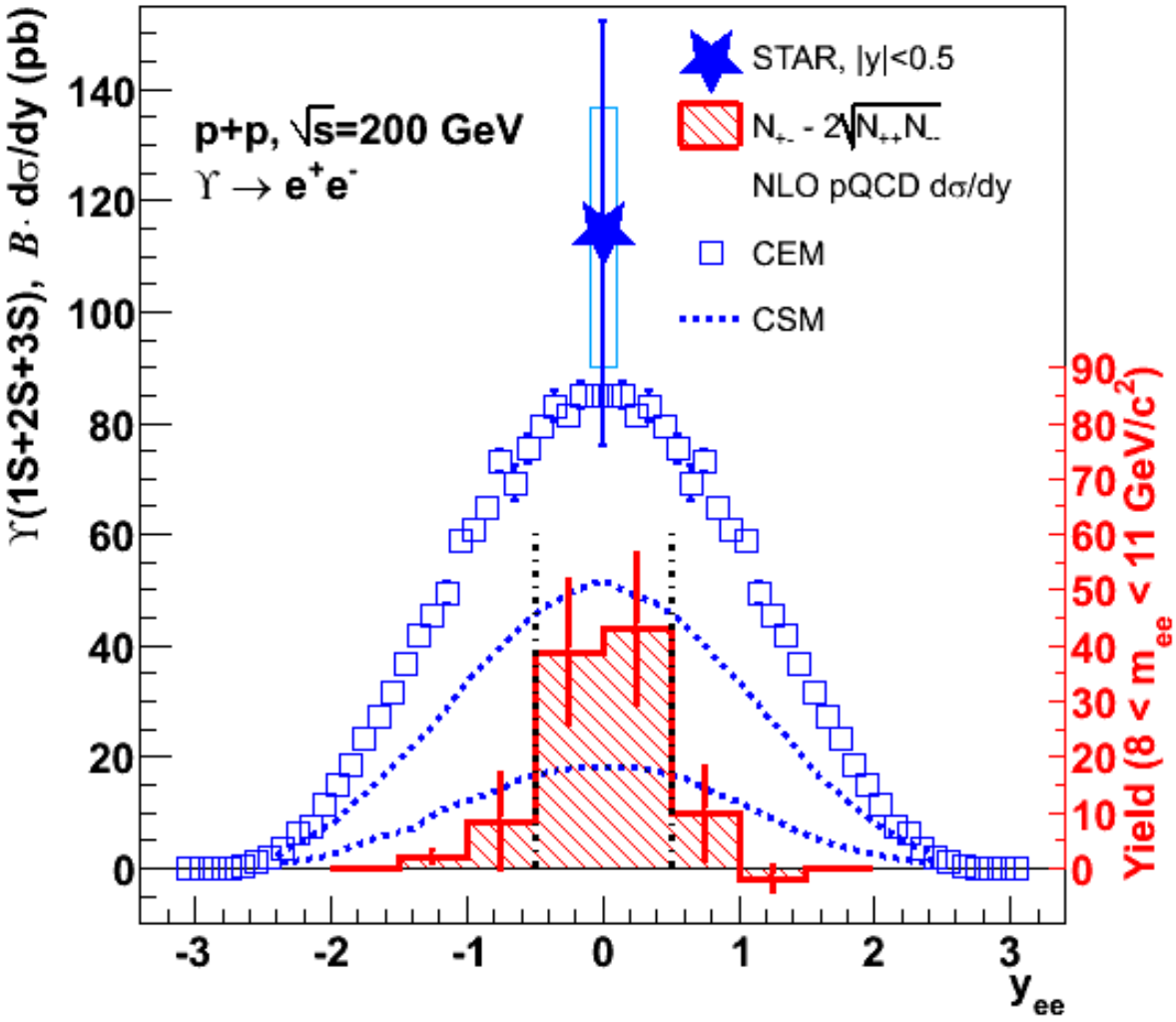
```
mcStyle->SetCanvasColor(0);  // canvas...
mcStyle->SetCanvasBorderMode(0);
mcStyle->SetCanvasBorderSize(0);
mcStyle->SetPadBottomMargin(0.1); //margins...
mcStyle->SetPadTopMargin(0.1);
mcStyle->SetPadLeftMargin(0.1);
mcStyle->SetPadRightMargin(0.1);
mcStyle->SetPadGridX(0); // grids, tickmarks
mcStyle->SetPadGridY(0);
mcStyle->SetPadTickX(1);
mcStyle->SetPadTickY(1);
mcStyle->SetFrameBorderMode(0);
mcStyle->SetPaperSize(20,24); // US letter size
gROOT->SetStyle("mcStyle");
cout << "Styles are Set!" << endl;
return;
}
```

Example Plot, fillrandom, With Style!

- Canvas color, bordersize, bordermode: all set to 0.
- Fonts set to 22
- Change font size.
  - titles, labels
- Change y-title offset
- Histograms:
  - change line color
  - change fill color
  - change fill style
  - add titles

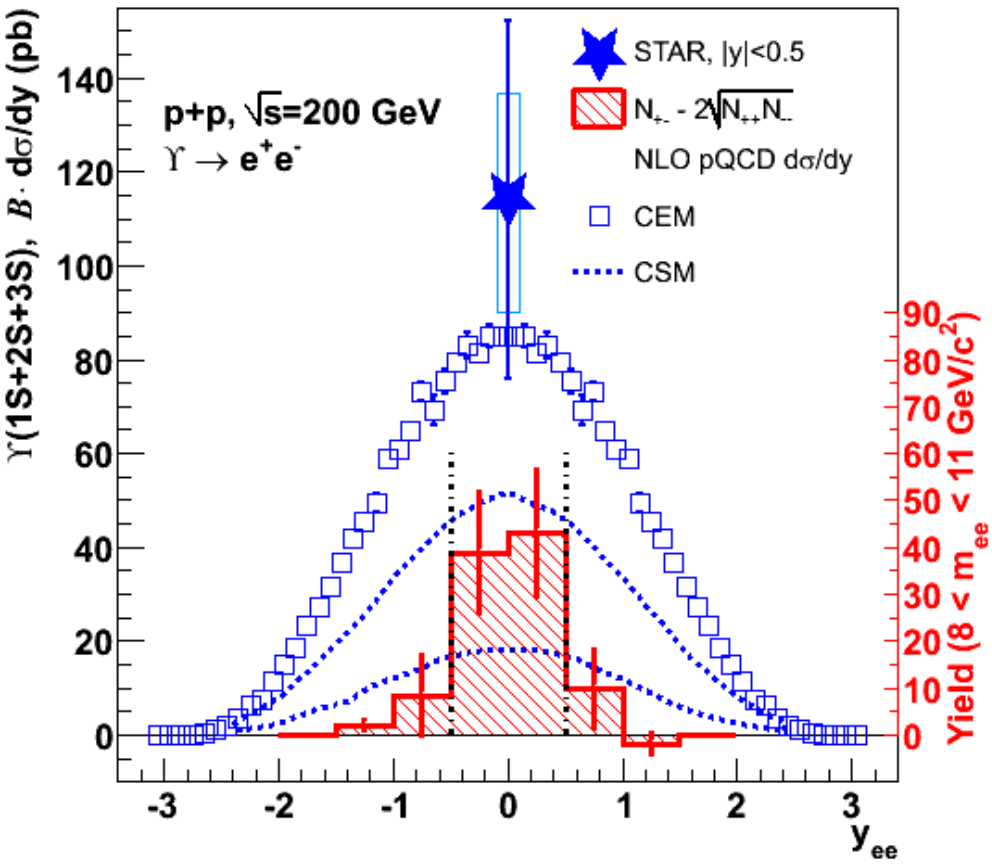


A real world example : Υ dσ/dy plot



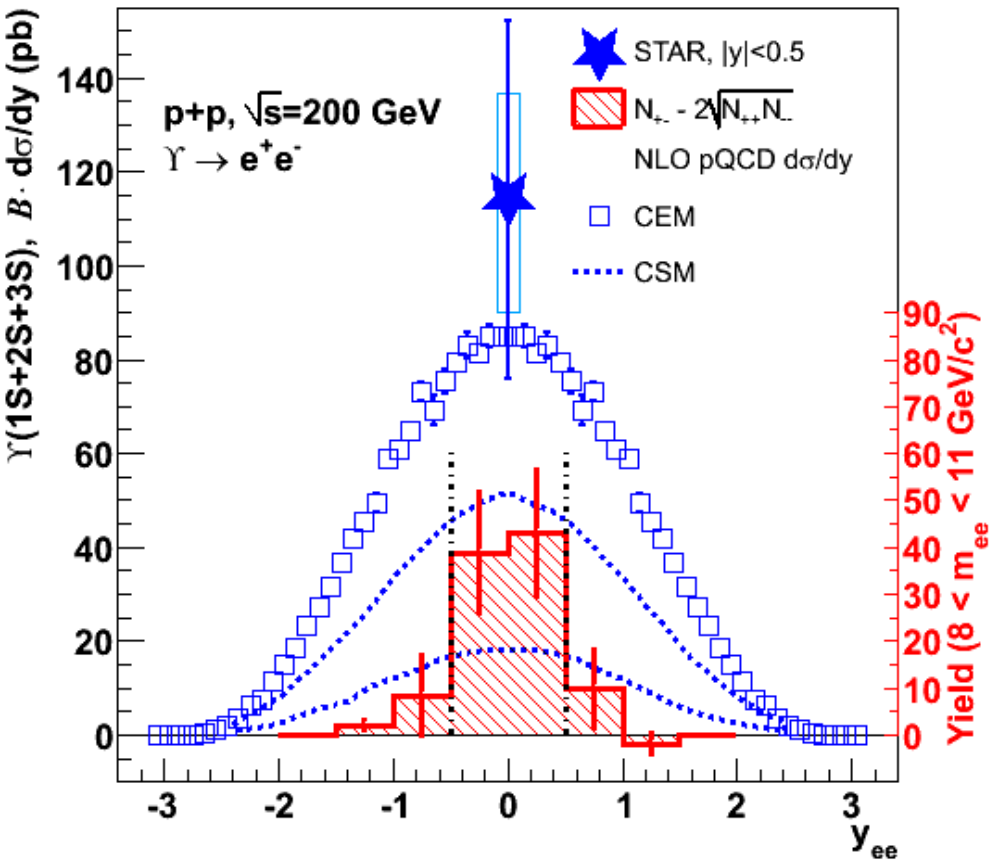
Theory calculations

- CEM model
  - TGraphErrors
  - MarkerStyle 25
  - MarkerColor 4
  - MarkerSize 1.3
  - Draw("P")
- CSM model
  - TGraph
  - LineColor 4
  - LineWidth 3
  - LineStyle 2

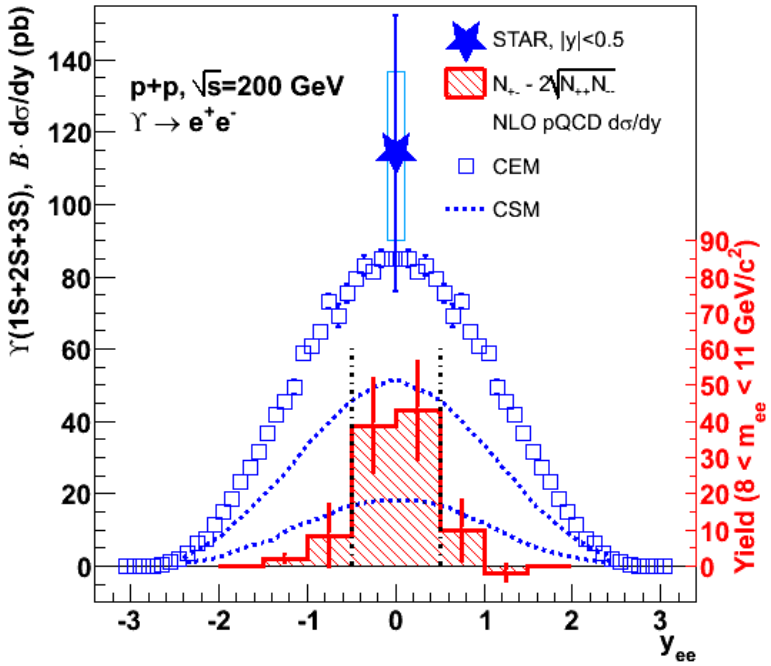


Drawing data

- Use dummy to:
  - Set axis titles
  - SetMaximum(155)
  - SetMinimum(-10)
  - y TitleOffset 1.5
    - via GetYaxis
- STAR data
  - TGraphErrors
  - MarkerStyle 29
    - STAR!
  - Marker, Line Color 4
  - MarkerSize 3.5



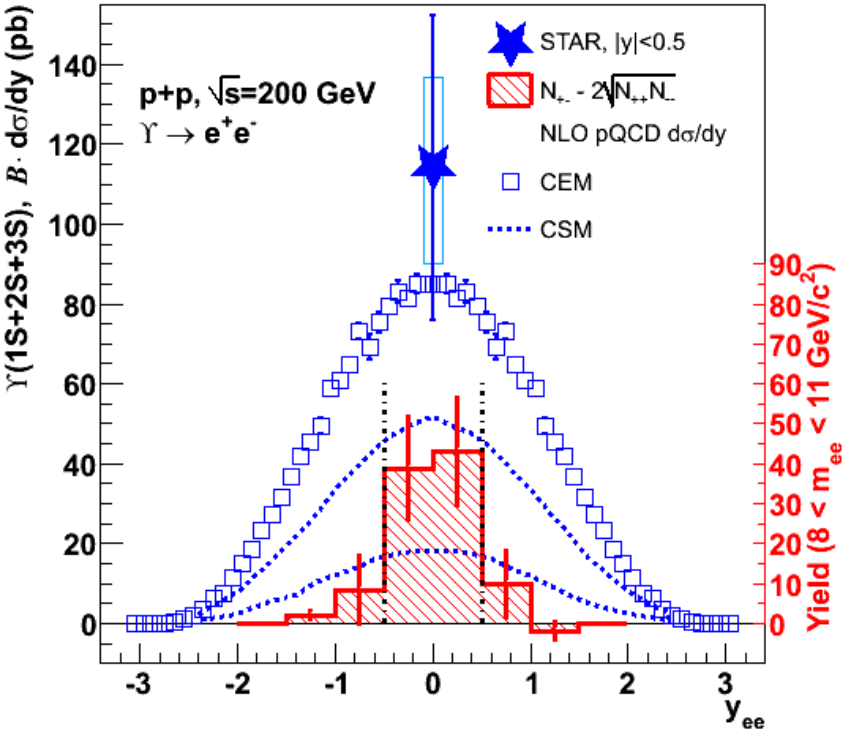
Systematic uncertainty



TPave* StarUpsSys = new TPave(-0.1, CrossSectionAverage-SystUncLo*CrossSectionAverage, 0.1, CrossSectionAverage+SystUncHi*CrossSectionAverage, 1,"tbrl"); // last two options: border size, “top bottom right left”

Histogram of raw yield

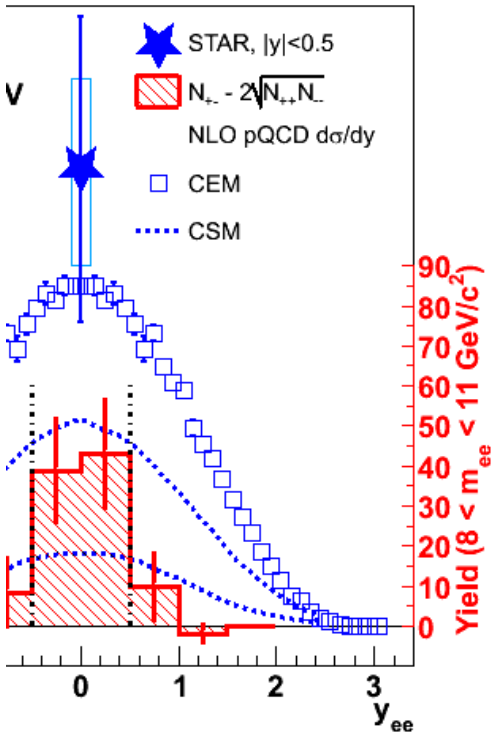
- Opened from a different file
- Line, Fill Color 2
- FillStyle 3005
- Draw("ehistsame")
  - error bars and histogram
- Add lines to indicate y integration region
- TLine: Color 1, Width 3, Style 4.



Additional Axis on Right side

```
TGaxis* RawYieldAxis = new
TGaxis(3.4,0,3.4,90,0,90,209,"+L");
//+ : draw on positive side
//L : left adjusted
RawYieldAxis->SetName("RawYieldAxis");
RawYieldAxis->SetLineColor(2);
RawYieldAxis->SetTextColor(2);
RawYieldAxis->SetTitle("Yield (8 < m_{ee} <
11 GeV/c^{2})");
RawYieldAxis->SetLabelColor(2);
RawYieldAxis->Draw();
```

http://root.cern.ch/root/html530/TGaxis.html#TGaxis:PaintAxis

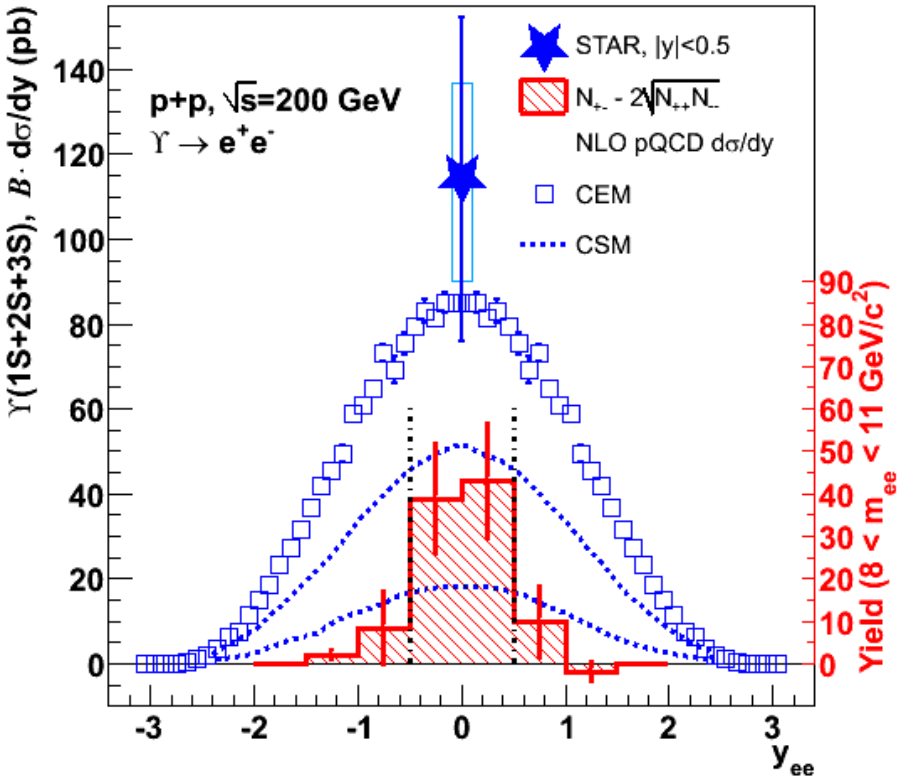


Use LaTeX syntax in titles and Legends

```
TLatex* ltx1 = new TLatex();
  ltx1->DrawLatex(-3,130,"p+p, #sqrt{s}=200 GeV");
  ltx1->DrawLatex(-3,120,"#varUpsilon #rightarrow e^{+}e^{-}");
```

From dummy title:

```
";y_{ee};#varUpsilon(1S+2S+3S
#font[32]{B}#upoint d#sigma/dy (pb)"
```



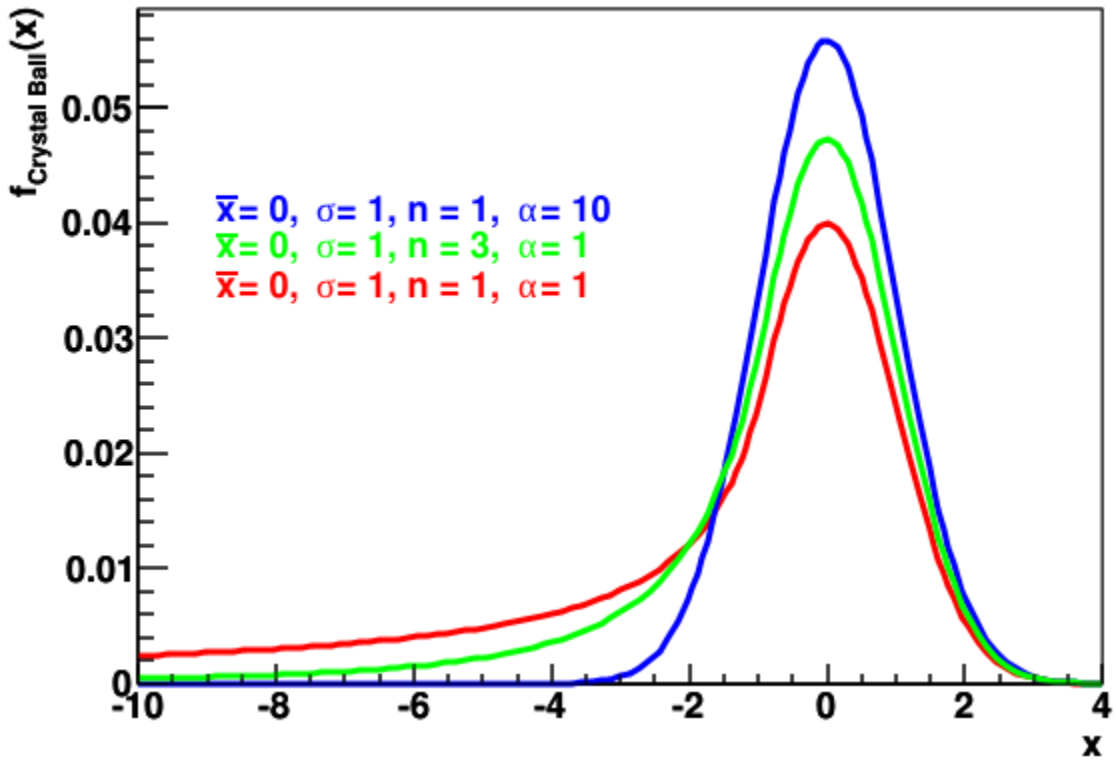
Plotting a user defined function in ROOT

```
double mysine(double* x, double* par) {
  double Amplitude = par[0];
  double wavelength = par[1];
  double phase = par[2];
  return Amplitude*sin(2*TMath::Pi()/wavelength*x[0]+phase);
}

void plotsine() {

  TCanvas* sineCanvas = new TCanvas("sineCanvas","A*sin(2pi/lambda*x +
     phi)",500,500);

  TF1* sineFunc = new TF1("sineFunc",&mysine,0,2*TMath::Pi(),3);
  sineFunc->SetParameters(2,TMath::Pi(),TMath::Pi()/2);
  sineFunc->Draw();
  return;
}
```

A more realistic example: Crystal Ball

$$f(x;\alpha,n,\bar{x},\sigma) = N \cdot \begin{cases} \exp(-\frac{(x-\bar{x})^2}{2\sigma^2}), & \text{for } \frac{x-\bar{x}}{\sigma} > -\alpha \\ A \cdot (B - \frac{x-\bar{x}}{\sigma})^{-n}, & \text{for } \frac{x-\bar{x}}{\sigma} \leqslant -\alpha \end{cases}$$

$$A = \left(\frac{n}{|\alpha|}\right)^n \cdot \exp\left(-\frac{|\alpha|^2}{2}\right) \qquad B = \frac{n}{|\alpha|} - |\alpha|$$



CrystalBall in Root

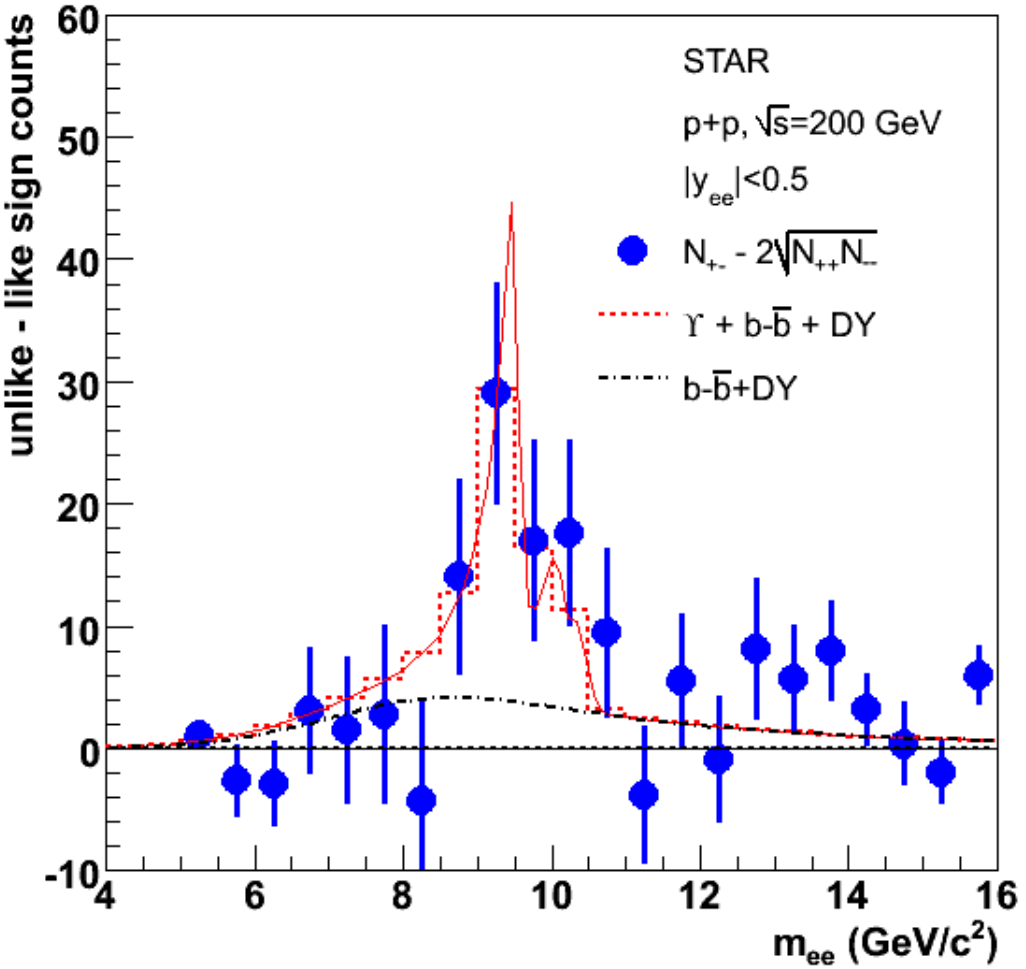
```
double CrystalBall(double* x, double* par){
  //http://en.wikipedia.org/wiki/Crystal_Ball_function
  double xcur = x[0];
  double alpha = par[0];
  double n = par[1];
  double mu = par[2];
  double sigma = par[3];
  double N = par[4];
  TF1* exp = new TF1("exp","exp(x)",1e-20,1e20);
  double A; double B;
  if (alpha < 0){
    A = pow((n/(-1*alpha)),n)*exp->Eval((-1)*alpha*alpha/2);
    B = n/(-1*alpha) + alpha;}
  else {
    A = pow((n/alpha),n)*exp->Eval((-1)*alpha*alpha/2);
    B = n/alpha - alpha;}
```

Crystall-Ball, Part 2

```
double f;
  if ((xcur-mu)/sigma > (-1)*alpha)
    f = N*exp->Eval((-1)*(xcur-mu)*(xcur-
mu)/(2*sigma*sigma));
  else
    f = N*A*pow((B- (xcur-mu)/sigma),(-1*n));
  delete exp;

  return f;
}
```

Three-Crystal Balls Fitting STAR data

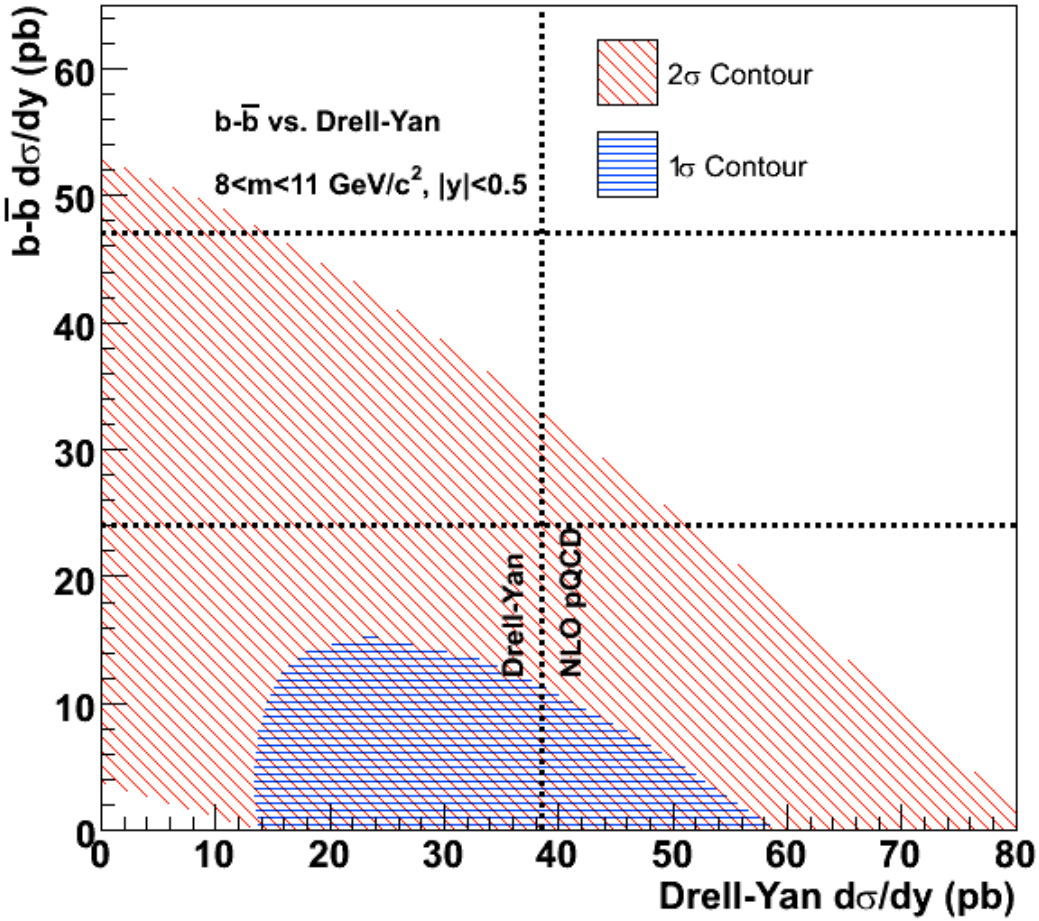
- Fit includes
  - 3 Crystal-Ball functions
  - Drell-Yan power law.
  - bottom quark power law.



Fit χ^2 contours : Real world example

For a tutorial see : $ROOTSYS/tutorials/fit/fitcont.C

- MINUIT can obtain the χ^2 contours from a multi parameter fit.
- Example
  - dielectron Invariant mass
  - Components
    - Upsilons
    - Drell-Yan
    - botttom-antibottom



Fit χ^2 contours :relevant code snippet

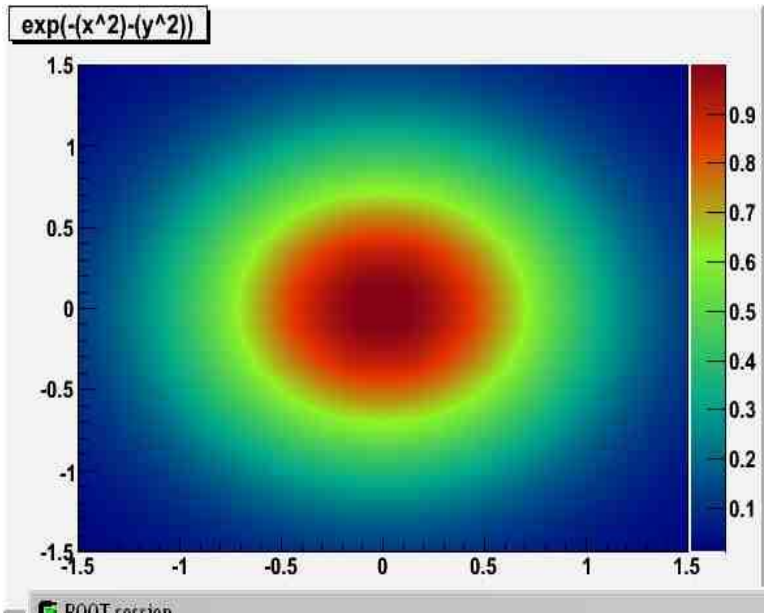
Somewhere in the macro, set:
TVirtualFitter::SetDefaultFitter("Minuit");

Fitting part:

```
 InvMass->Fit(FitFunc,"i","",5,16);
  gMinuit->SetErrorDef(5.99); //Use 5.99 for a 95% coverage probability contour with 2 parameters. See Table 7.1 of minuit manual. Argument assumes we are looking of hypercountour of chi^2 = chi^2_min + UP, where UP is the argument to SetErrorDef. See also Table 33.2 of PDG.
  cout << "Getting 95% coverage contour" << endl;
  TGraph* cont95 = (TGraph*) gMinuit->Contour(20,17,16);
  cont95->SetName("cont95");
gMinuit->SetErrorDef(2.3); //Use 2.3 for a 68.27% coverage;
  cout << "Getting 68.3% coverage contour" << endl;
  TGraph* cont68 = (TGraph*) gMinuit->Contour(20,17,16);
   cont68->SetName("cont68");
```

More control over colors

```
// Use of TColor::CreateGradientColorTable
void colorPalette() {
//example of new colors (greys) and definition of a new palette
   const Int_t NRGBs = 5;
   const Int_t NCont = 256;

   Double_t stops[NRGBs] = { 0.00, 0.30, 0.61, 0.84, 1.00 };
   Double_t red[NRGBs]   = { 0.00, 0.00, 0.57, 0.90, 0.51 };
   Double_t green[NRGBs] = { 0.00, 0.65, 0.95, 0.20, 0.00 };
   Double_t blue[NRGBs]  = { 0.51, 0.55, 0.15, 0.00, 0.10 };
   TColor::CreateGradientColorTable(NRGBs, stops, red, green, blue,
    NCont);
   gStyle->SetNumberContours(NCont);

  TF2 *f2 = new TF2("f2",
    "exp(-(x^2) - (y^2))",-1.5,1.5,-1.5,1.5);
  //f2->SetContour(colNum);
  f2->SetNpx(300);
  f2->SetNpy(300);
  f2->Draw("colz");
  return;
}
```



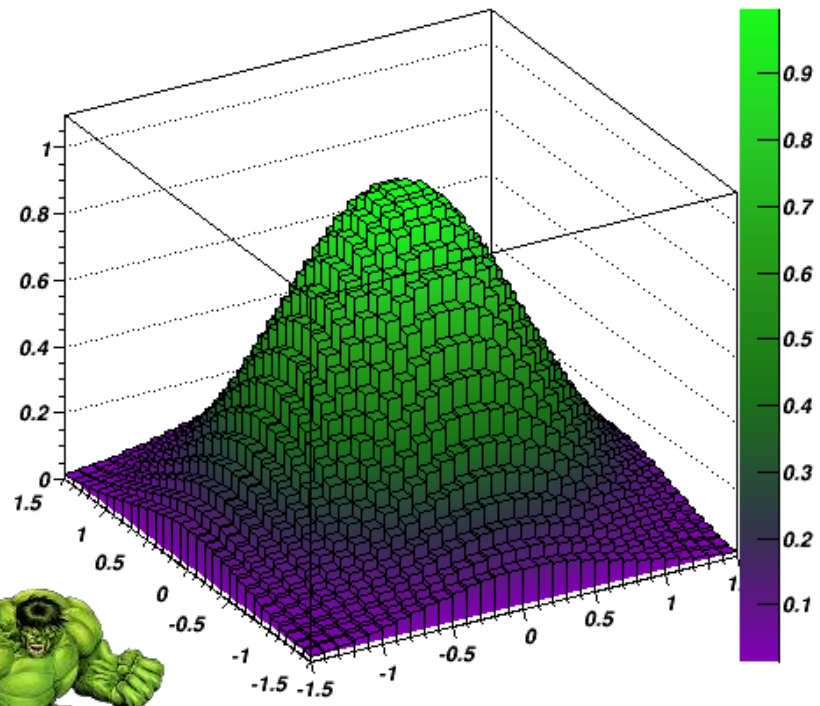
2 palettes in same canvas:

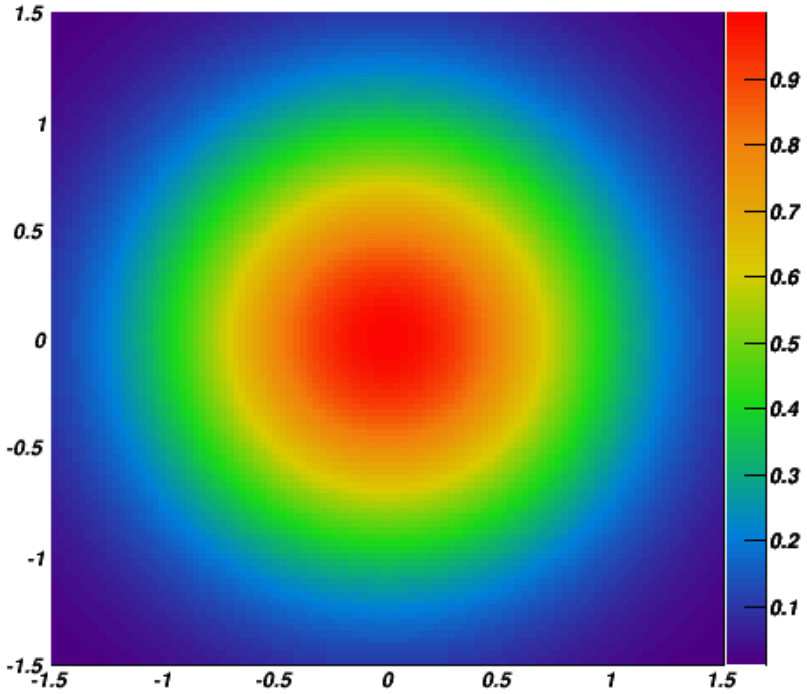
```
void Pal1(){
  const Int_t NRGBs = 6;
  const Int_t NCont = 99; // can only do up to 99 levels
  static Int_t  colors[NCont];
  static Bool_t initialized = kFALSE;

   Double_t stops[NRGBs] = { 0.00, 0.20, 0.40, 0.60,
0.80, 1.00 };
   Double_t red[NRGBs]   = { 0.30, 0.00, 0.10, 0.85,
0.95, 0.99 };
   Double_t green[NRGBs] = { 0.00, 0.50, 0.85, 0.80,
0.50, 0.00 };
   Double_t blue[NRGBs]  = { 0.51, 0.85, 0.10, 0.00,
0.05, 0.00 };

   if(!initialized){
     Int_t FI =
TColor::CreateGradientColorTable(NRGBs,stops,red,gree
n,blue,NCont);
     for (int i=0; i<NCont; i++) colors[i] = FI+i;
     initialized = kTRUE;
     return;
   }
  gStyle->SetPalette(NCont,colors);
}
// Same for Pal2...
```

```
TCanvas* colorCanvas = new
TCanvas("colorCanvas","Color Test",1000,500);
   colorCanvas->Divide(2,1);
   colorCanvas->cd(1);
   TF2 *f2 = new TF2("f2","exp(-(x^2) - (y^2))",-1.5,1.5,-
1.5,1.5);
   f2->SetContour(NCont);
   f2->SetNpx(30);
   f2->SetNpy(30);
   f2->Draw("lego2");

   TExec *ex1 = new TExec("ex1","Pal2();");
   ex1->Draw();
   f2->Draw("lego2z same");

   colorCanvas->cd(2);
   TF2 *f3 = new TF2("f3","exp(-(x^2) - (y^2))",-1.5,1.5,-
1.5,1.5);
   f3->SetContour(NCont);
   f3->SetNpx(100);
   f3->SetNpy(100);
   f3->Draw("colz");
   TExec *ex2 = new TExec("ex2","Pal1();");
   ex2->Draw();
   f3->Draw("colz same");
   return;
```

2 Palettes in 2 Pads, same TCanvas

- You can make your own Color Palettes.



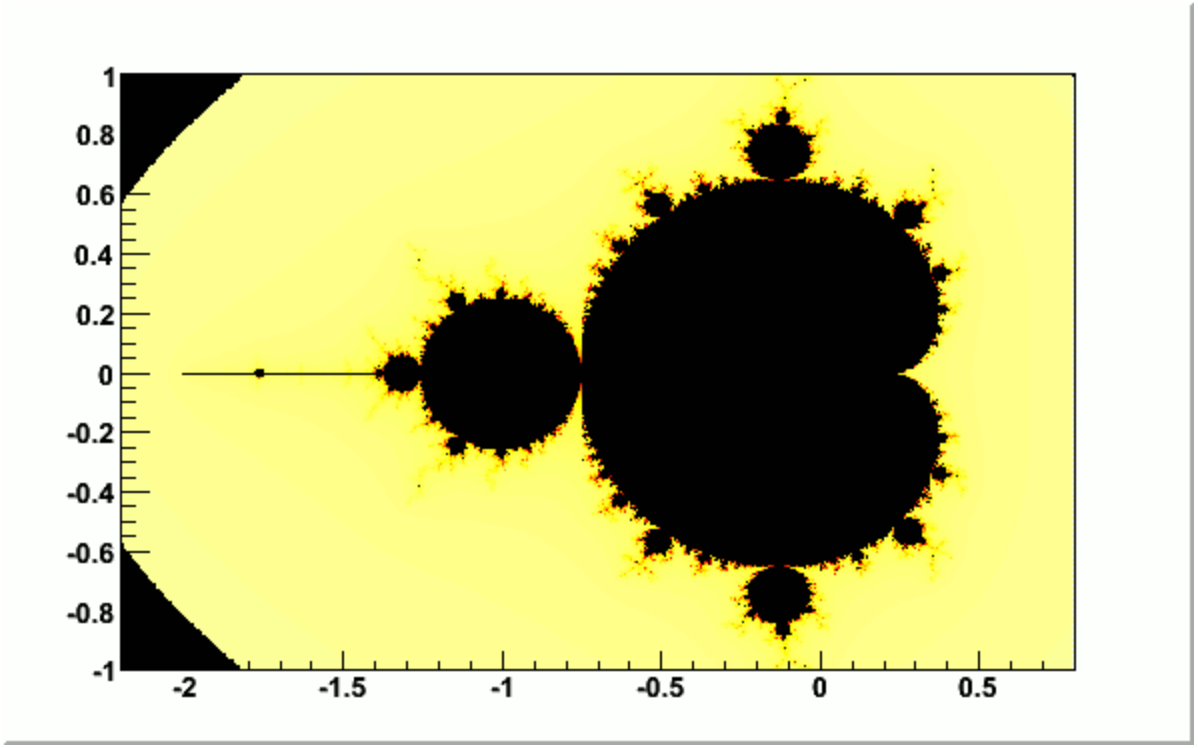


Last word: saving files, and animations

if filename is "", the file produced is padname.ps
if filename starts with a dot, the padname is added in front
if filename contains .eps, an Encapsulated Postscript file is produced
if filename contains .pdf, a PDF file is produced
if filename contains .svg, a SVG file is produced
if filename contains .gif, a GIF file is produced
if filename contains .gif+NN, an animated GIF file is produced
if filename contains .xpm, a XPM file is produced
if filename contains .png, a PNG file is produced
if filename contains .jpg, a JPEG file is produced
NOTE: JPEG's lossy compression will make all sharp edges fuzzy.
if filename contains .tiff, a TIFF file is produced
if filename contains .C or .cxx, a C++ macro file is produced
if filename contains .root, a Root file is produced
if filename contains .xml, a XML file is produced

Animated gifs

Rendering thousands canvases in a for loop
Use SaveAs("MSet.gif+10")
Obtain an animated gif after each iteration



Solving a Mechanics problem

Coffee cup and mass

- A coffee cup of mass M is connected to a mass m by a string. The coffee cup hangs over a frictionless pulley of negligible size, and the mass m is initially held with the string horizontal. The mass m is released from rest. Find the equations of motion for r (the length of the string between m and the pulley) and q (the angle that the string and m makes with the horizontal). Assume that m somehow doesn't run into the string holding the cup up.
- The coffee cup will initially fall, but it turns out that it will reach a lowest point and then rise back up. Write a program that numerically determines the ratio of the r at this lowest point to the r at the start, for a given value of m/M.

Taylor expansion: rederive Simple Euler method

- Simple Euler, Modified Euler, valid to O(h), can improve the accuracy.

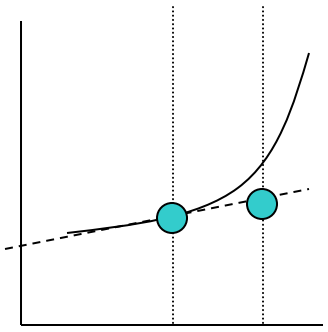
$$y_{n+1} \cong y(x_n + h) = y_n + hy'_n + \frac{h^2}{2} y''_n + O(h^3)$$

Simple Euler Method

- Method can be expressed as:

$$y'_n = f(x_n, y_n)$$

$$y_{n+1} = y_n + hf(x_n, y_n)$$



Where x_n and y_n are known, and the differential equation under consideration specifies $f(x_n,y_n)$

2nd Order Runge-Kutta: Rederive Midpoint Euler Method, Part I

- Assume $y_{n+1/2}$ is known.
- Next: Expand in a Taylor series:
  - First Use $\Delta x = -h/2$
  - Expand around $y_{n+1/2}$ to estimate y_n.
    - Keeping terms up to 2nd order.

$$y_n = y_{n+1/2} - \frac{h}{2} y'_{n+1/2} + \frac{(h/2)^2}{2} y''_{n+1/2} + O(h^3)$$

  - Now use $\Delta x = +h/2$, expand around $y_{n+1/2}$ to y_{n+1}.

$$y_{n+1} = y_{n+1/2} + \frac{h}{2} y'_{n+1/2} + \frac{(h/2)^2}{2} y''_{n+1/2} + O(h^3)$$

Rederive Midpoint Euler Method, Part II

- Subtract the two:

$$y_{n+1} = y_{n+1/2} + \frac{h}{2} y'_{n+1/2} + \frac{(h/2)^2}{2} y''_{n+1/2} + O(h^3)$$

$$y_n = y_{n+1/2} - \frac{h}{2} y'_{n+1/2} + \frac{(h/2)^2}{2} y''_{n+1/2} + O(h^3)$$

$$y_{n+1} - y_n = 0 + h y'_{n+1/2} + 0 + O(h^3)$$

- We get rid of the 2nd Order terms!
- But, hey! We don't know $y'_{n+1/2}$.
- Approximate it using simple Euler method.

$$y_{n+1/2} = y_n + \frac{h}{2} y'_n$$

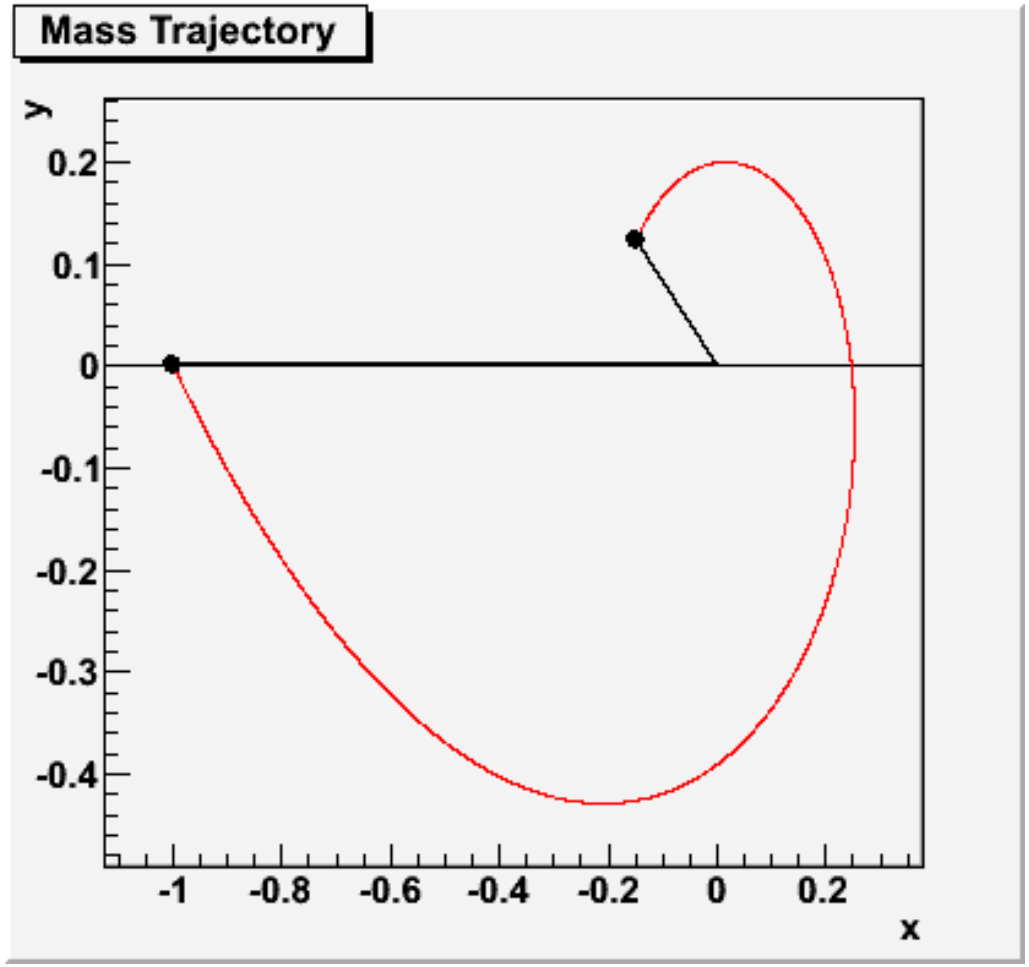
$$y'_{n+1/2} = f(x_{n+1/2}, y_{n+1/2})$$

or

$$k_1 \equiv y'_n = f(x_n, y_n)$$

$$k_2 \equiv y'_{n+1/2} = f(x_{n+1/2}, y_{n+1/2})$$

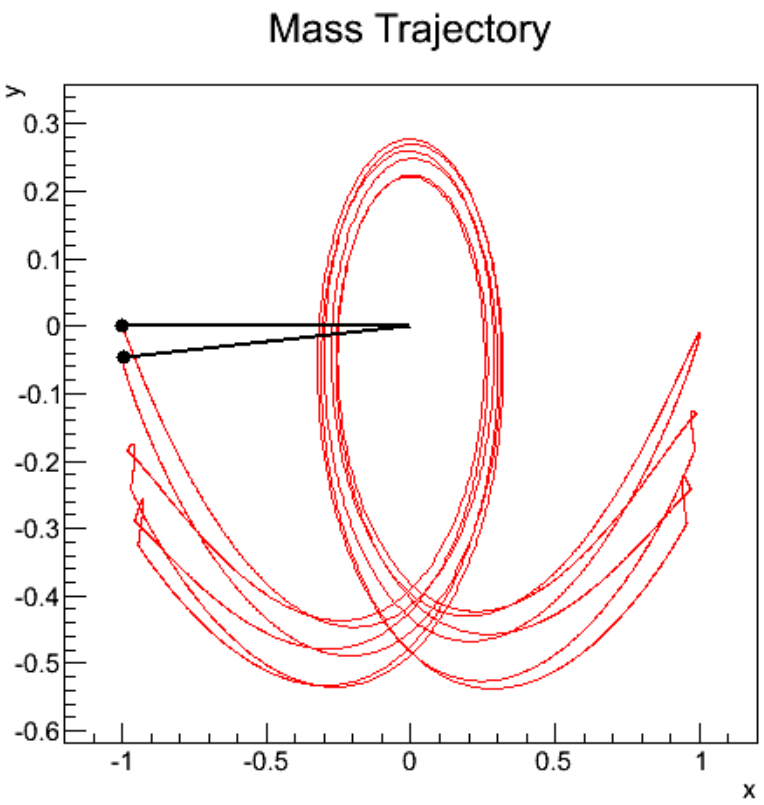
Trajectory of mass for m/M=1/10



- a

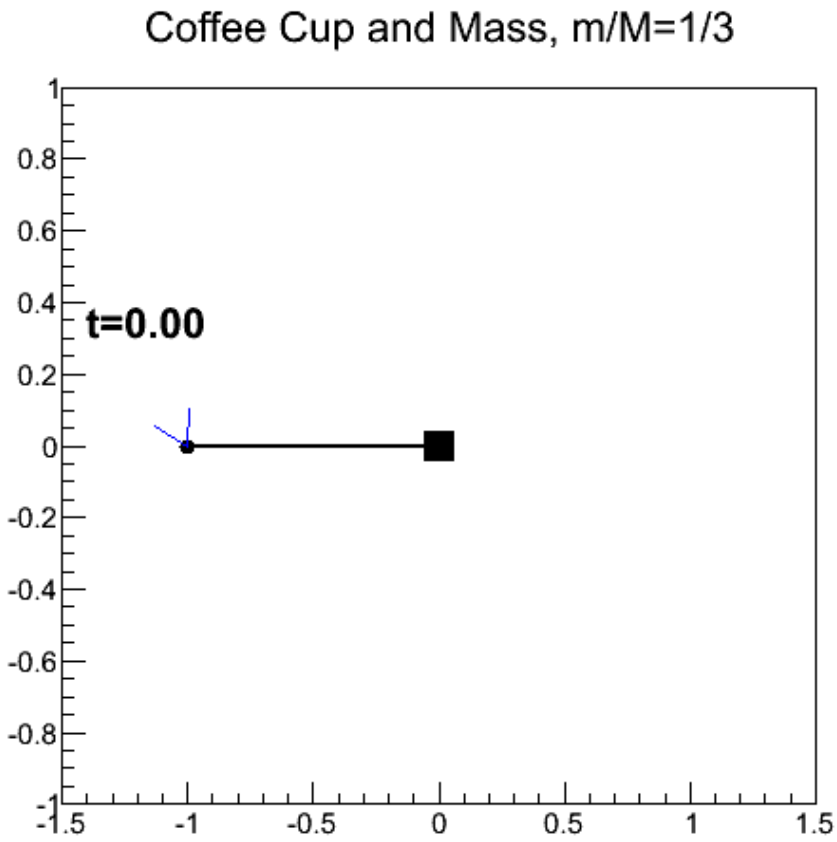
Plot the trajectory

- m/M=1/7



Animation

- Case m/M = 1/3
  - Arrow: velocity vector
  - square: coffee cup



Additional Material

ROOT commands

- Starting root, just type “root”
- At the root prompt:
  - .q = Exit from root
  - .ls = list the files loaded into root session
  - .! some-unix-command = execute some-unix-command in the shell
- Most c++ commands can also be interpreted.
- Executing a macro “myMacro.C”:
  - .x myMacro.C

ROOT Classes

- Since it is C++, everything is represented by classes:
  - Windows (or canvases) : TCanvas
    - A window where we can draw data, functions, etc.
  - Functions : TF1, TF2, TF3
    - Classes to manipulate mathematical functions, such as sin(x), in order to draw, evaluate, and integrate them.
  - Graphs : TGraph
    - Class used to plot data on a canvas
  - Histograms: TH1, TH2, TH3
    - Classes to manipulate histograms. Can draw them on a canvas, integrate them, obtain means and RMS values, evaluate bin contents.
  - Tutorials (lots of code to try out ROOT):
    - $ROOTSYS/tutorials/
    - For example: ./hsimple.C, and ./hist/h1draw.C

Graph Draw Options

The various draw options for a graph are explained in **TGraph**::PaintGraph. They are:

- "L" A simple poly-line between every points is drawn
- "F" A fill area is drawn
- "F1" Idem as "F" but fill area is no more repartee around X=0 or Y=0
- "F2" draw a fill area poly line connecting the center of bins
- "A" Axis are drawn around the graph
- "C" A smooth curve is drawn
- "*" A star is plotted at each point
- "P" The current marker of the graph is plotted at each point
- "B" A bar chart is drawn at each point
- "[]" Only the end vertical/horizontal lines of the error bars are drawn. This option only

applies to the **TGraphAsymmErrors**.

- "1" ylow = rwymin

The options are not case sensitive and they can be concatenated in most cases. Let us look at some examples

Text Fonts, Part 1

- http://root.cern.ch/root/html530/TAttText.html#T5
- Text font code = 10*fontnumber + precision
  - Font numbers must be between 1 and 14.
  - The precision can be:
    precision = 0 fast hardware fonts (steps in the size)
  - precision = 1 scalable and rotatable hardware fonts (see below)
  - precision = 2 scalable and rotatable hardware fonts
  - precision = 3 scalable and rotatable hardware fonts. Text size is given in pixels.

Text Fonts, part 2

- **List of the currently supported fonts**

Font number	X11 Names	Win32/TTF Names
1 :	times-medium-i-normal	"Times New Roman"
2 :	times-bold-r-normal	"Times New Roman"
3 :	times-bold-i-normal	"Times New Roman"
4 :	helvetica-medium-r-normal	"Arial"
5 :	helvetica-medium-o-normal	"Arial"
6 :	helvetica-bold-r-normal	"Arial"
7 :	helvetica-bold-o-normal	"Arial"
8 :	courier-medium-r-normal	"Courier New"
9 :	courier-medium-o-normal	"Courier New"
10 :	courier-bold-r-normal	"Courier New"
11 :	courier-bold-o-normal	"Courier New"
12 :	symbol-medium-r-normal	"Symbol"
13 :	times-medium-r-normal	"Times New Roman"
14 :		"Wingdings"
15 :	Symbol italic (derived from Symbol)	

Text Fonts, part 3

12 : *ABCDEFGH abcdefgh 0123456789 @#$*

22 : **ABCDEFGH abcdefgh 0123456789 @#$**

32 : ***ABCDEFGH abcdefgh 0123456789 @#$***

42 : ABCDEFGH abcdefgh 0123456789 @#$

52 : *ABCDEFGH abcdefgh 0123456789 @#$*

62 : **ABCDEFGH abcdefgh 0123456789 @#$**

72 : ***ABCDEFGH abcdefgh 0123456789 @#$***

82 : ABCDEFGH abcdefgh 0123456789 @#$

92 : *ABCDEFGH abcdefgh 0123456789 @#$*

102 : **ABCDEFGH abcdefgh 0123456789 @#$**

112 : ***ABCDEFGH abcdefgh 0123456789 @#$***

122 : ΑΒΧΔΕΦΓΗ αβχδεφγη 0123456789 ≅#∃

132 : ABCDEFGH abcdefgh 0123456789 @#$

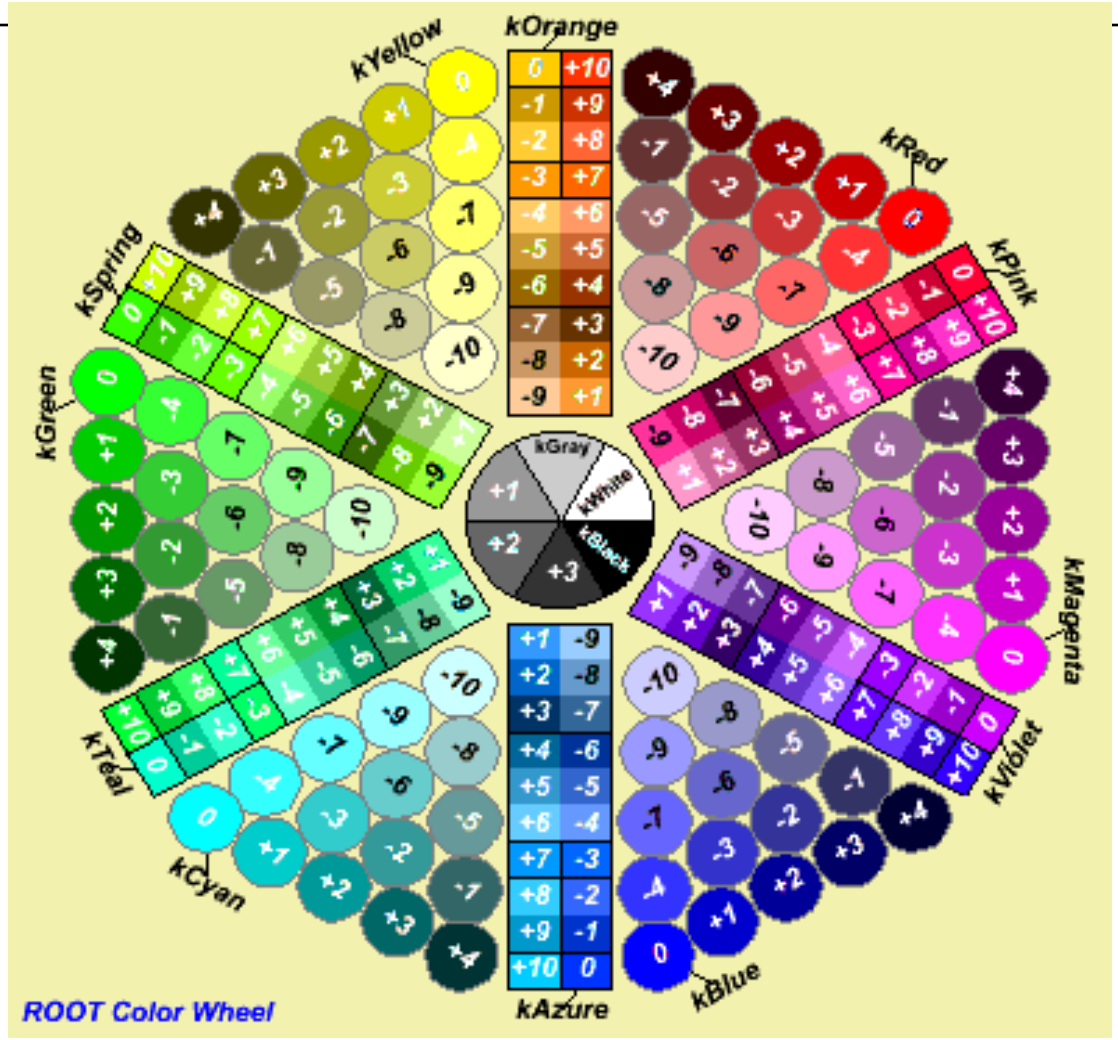
142 : ✌✍👌👎 ♏♐♑♒ 🗀🗁🗎🗐🗍 ✂✁✍

152 : *ΑΒΧΔΕΦΓΗ αβχδεφγη 0123456789 ≅#∃*

Colors

- See http://root.cern.ch/root/html530/TAttFill.html
- Default color palette

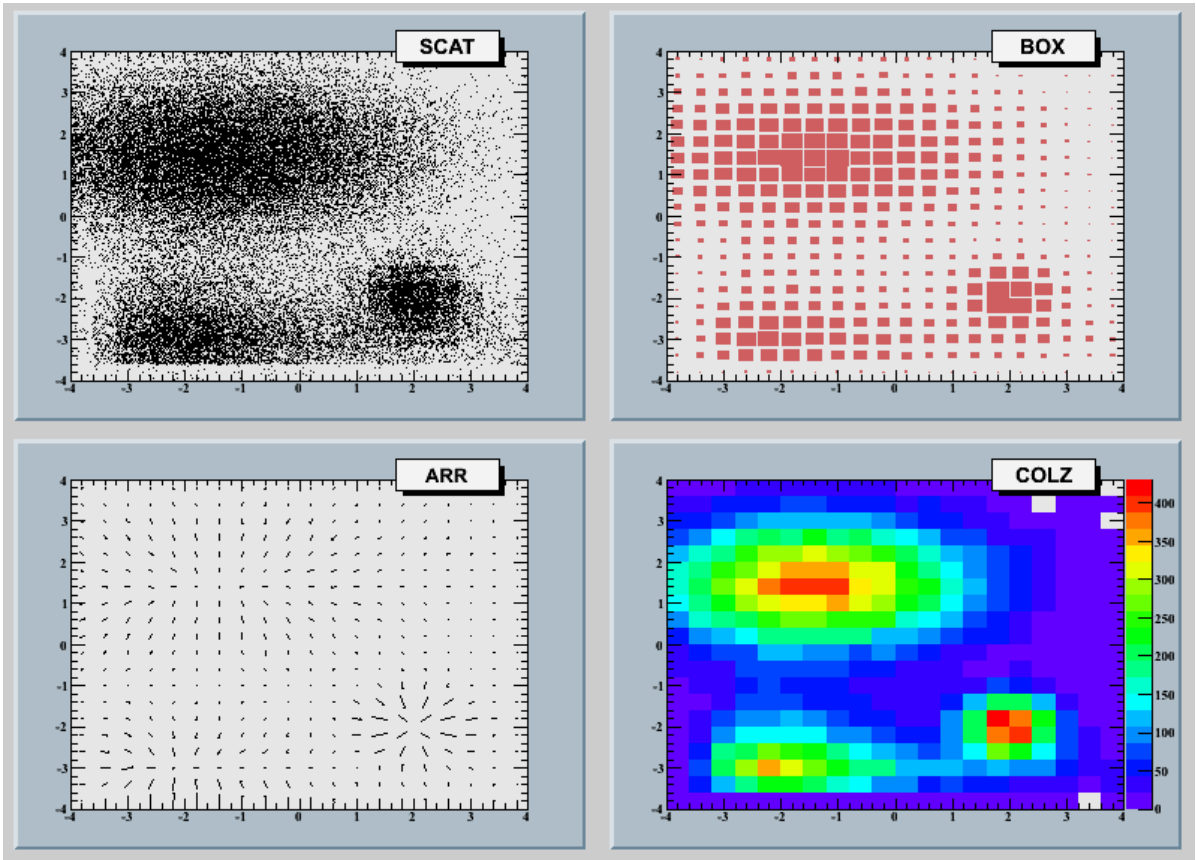
Using the Color Wheel



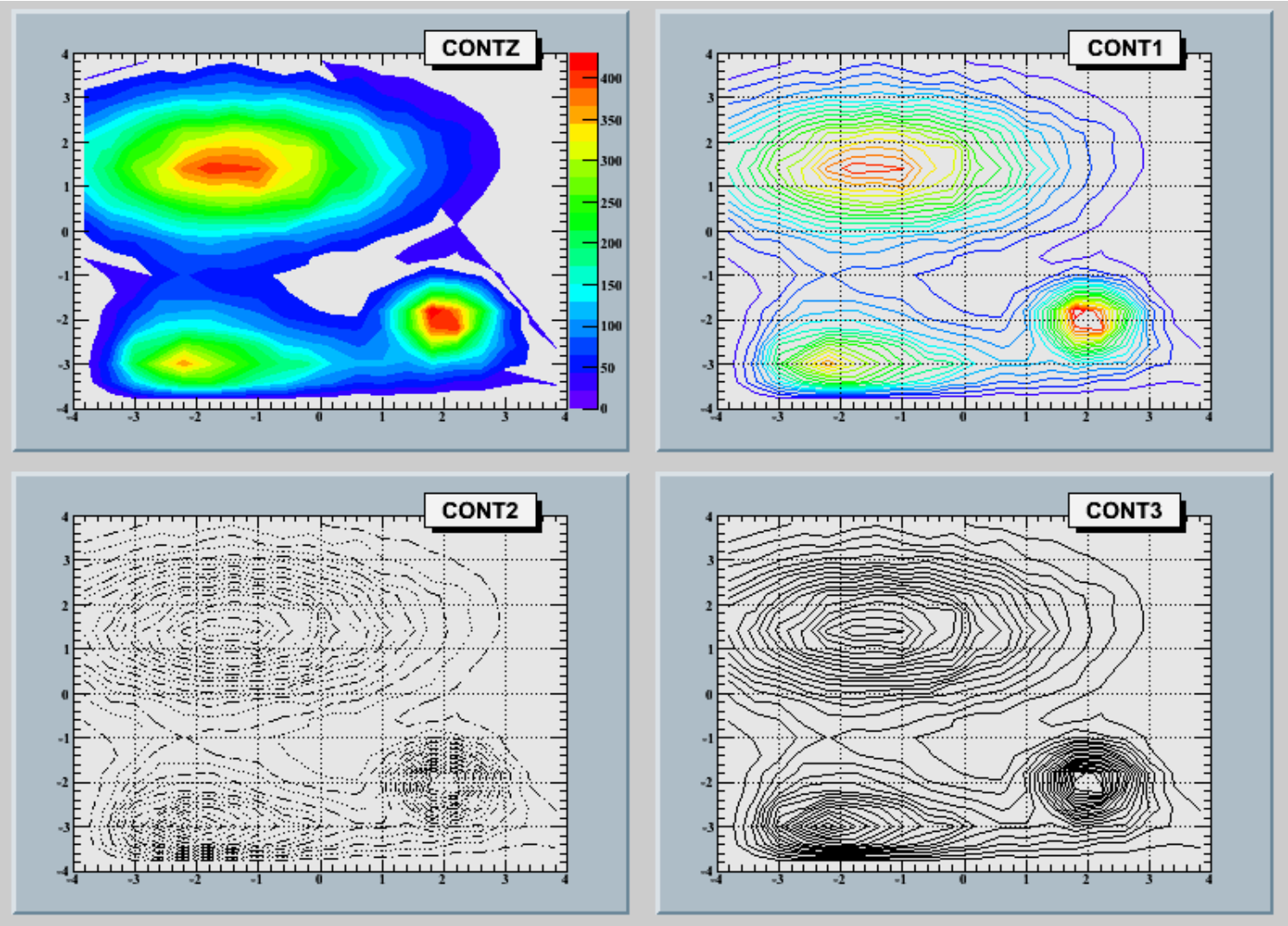

```
myObject.SetFillColor(kRed);
myObject.SetFillColor(kYellow-10);
myLine.SetLineColor(kMagenta+2);
```

2-D plot options : draw2dopt.C

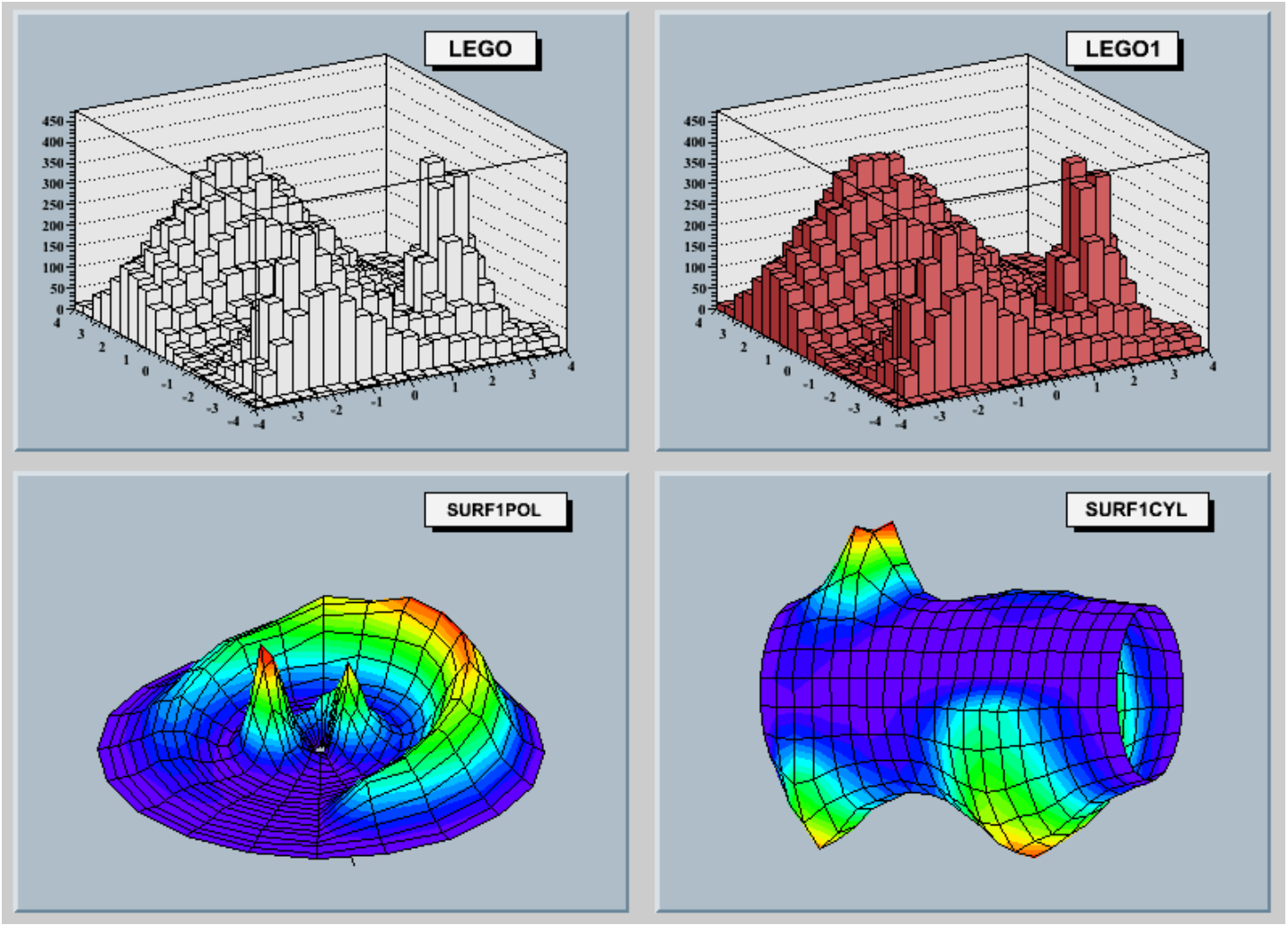
- $ROOTSYS/tutorials/hist/draw2dopt.C
- See THistPainter:Paint for drawing options
- Example uses: gStyle->SetPalette(1,0);



2-d options, contours

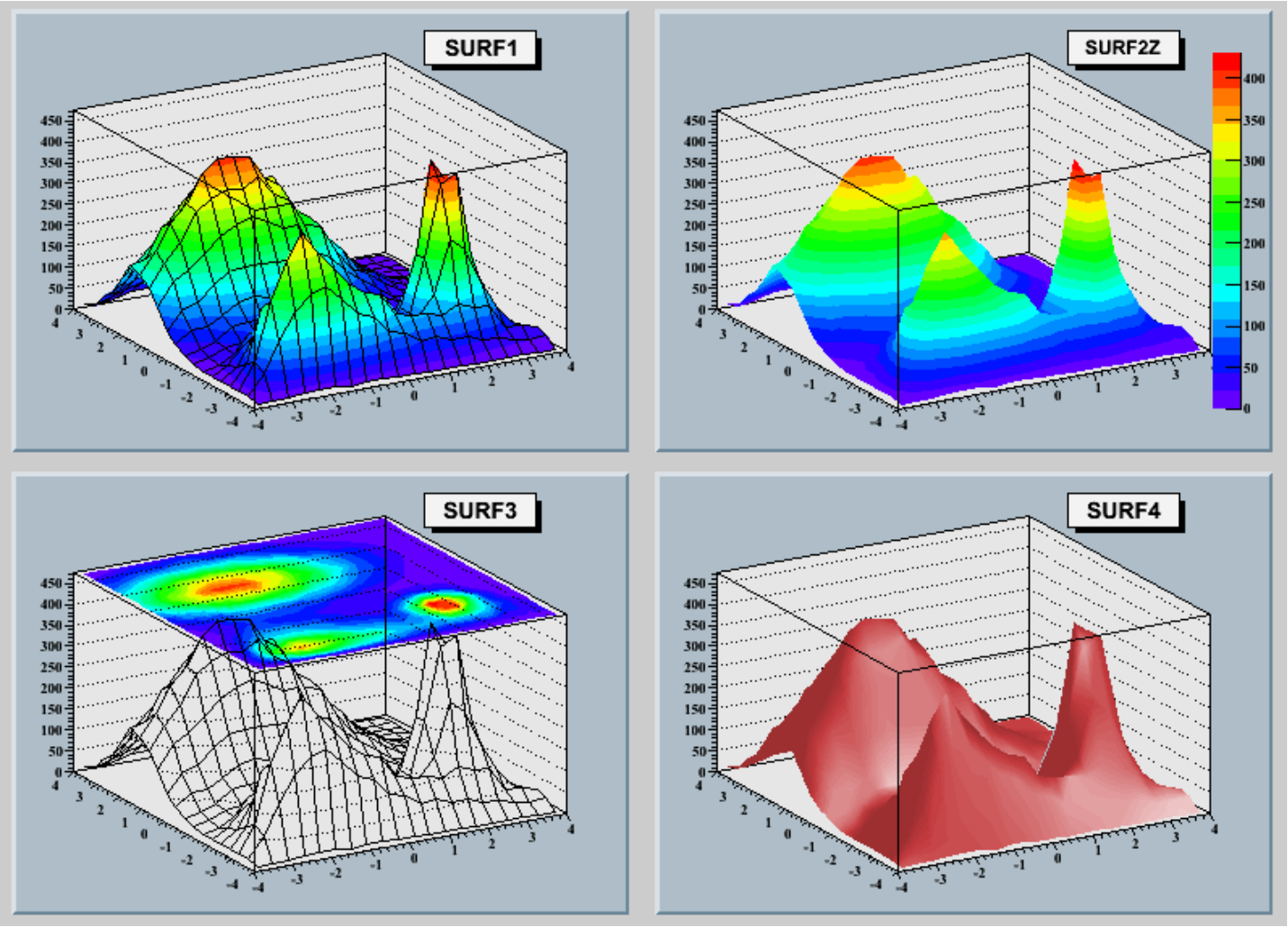


2-d options, lego, surfpol



Note: option lego2 not diplayed

2-d options, surface



TGaxis

- http://root.cern.ch/root/html530/TGaxis.html