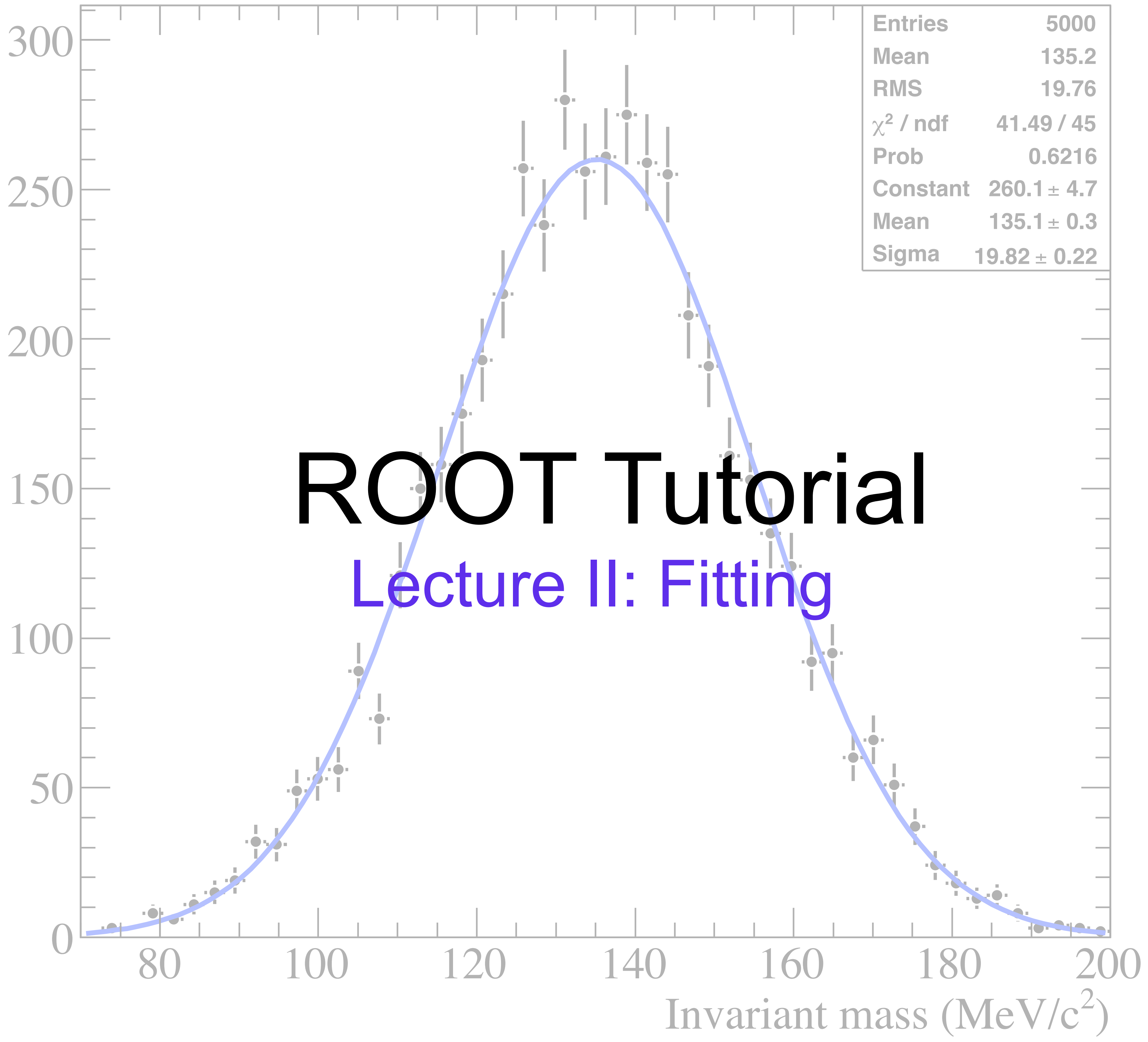
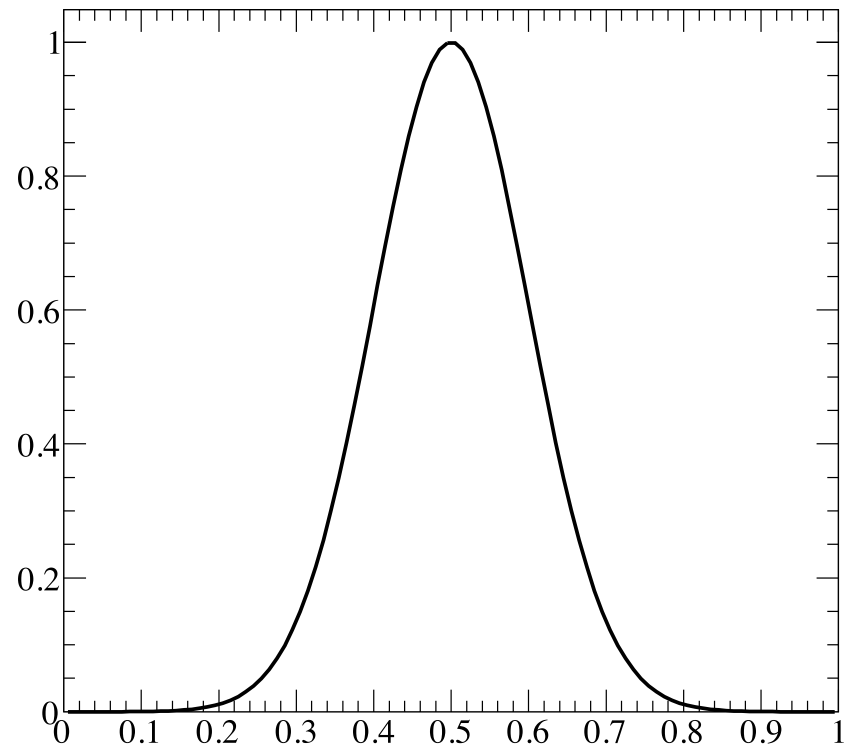


ROOT Tutorial

Lecture II: Fitting

Homework from last time...

Learning how to define a 1-dimensional function (TF1)

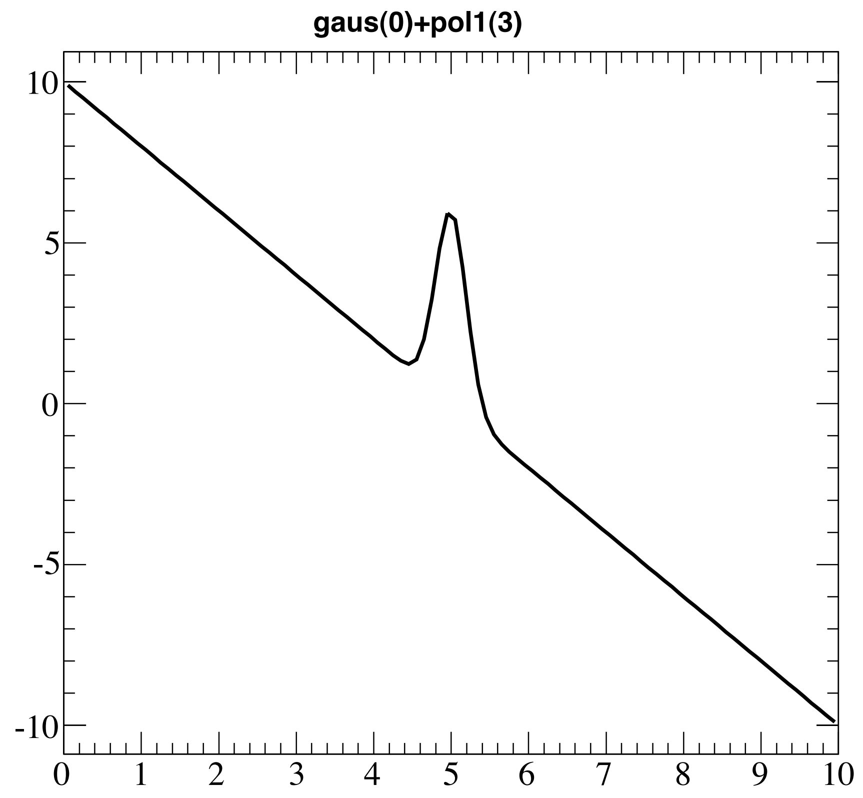


Pre-defined ROOT functions

```
TF1 *gs1 = new TF1("gs1", "gaus", 0.0, 1.0);
gs1->SetParameters(1.0, 0.5, 0.1);
gs1->Draw();
```

Optional arguments for range (defaults to 0.0, 1.0)

```
TF1 *gs1pol = new TF1("gs1pol", "gaus(0) + pol1(3)", 0.0, 10.0);
gs1pol->SetParameter(0, 6.0);
gs1pol->SetParameter(1, 5.0);
gs1pol->SetParameter(2, 0.2);
gs1pol->SetParameter(3, 10.0);
gs1pol->SetParameter(4, -2.0);
gs1pol->Draw();
```

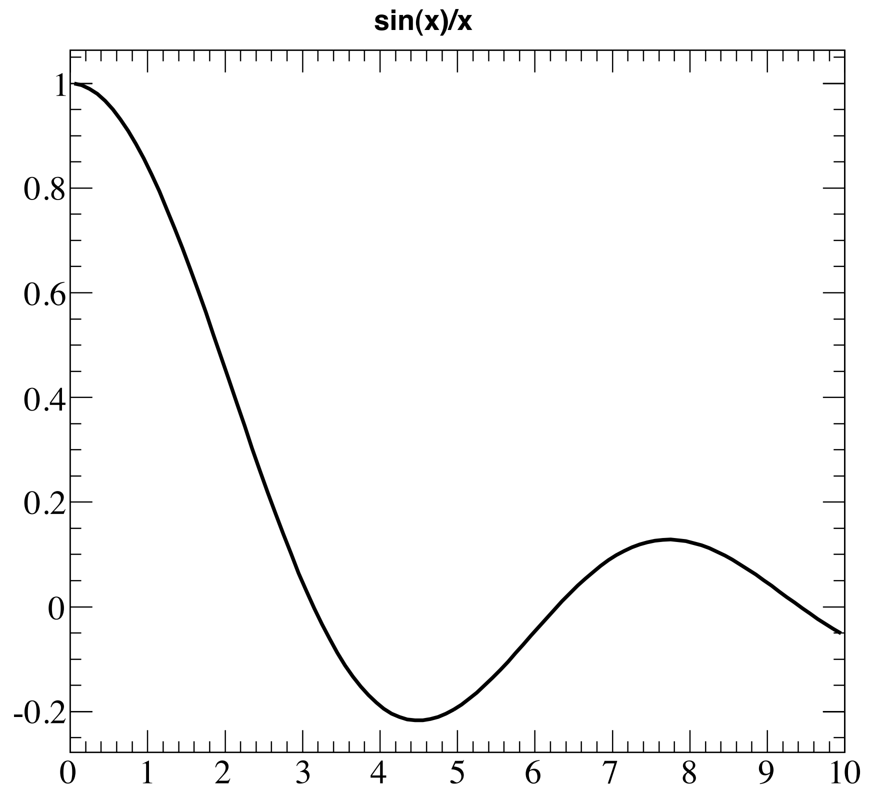


Start numbering parameters at 0 for gaussian, and at 3 for first-order polynomial

Other methods to define a function

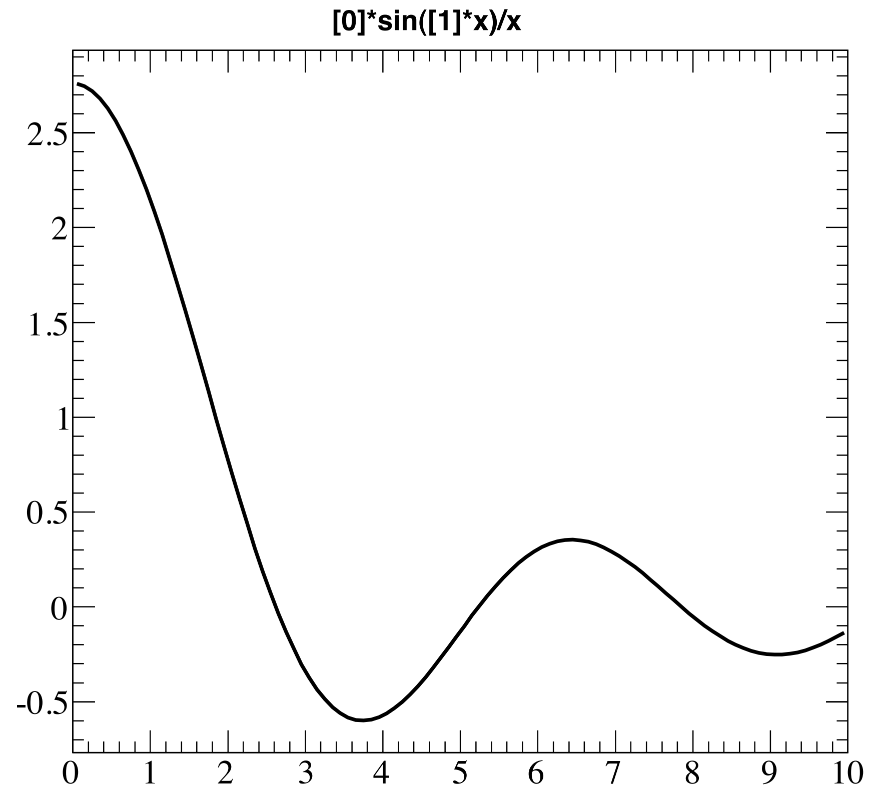
In-line expressions

```
TF1 *tfsine = new TF1("tfsine","sin(x)/x",0.0,10.0);
tfsine->Draw();
```



In-line expressions with parameters

```
TF1 *tfsine2 = new TF1("tfsine2", "[0]*sin([1]*x)/x", 0.0, 10.0);
tfsine2->SetParameters(2.3,1.2);
tfsine2->Draw();
```



Other methods to define a function (cont'd)

```
Double_t fitfunc(Double_t *x, Double_t *par){
  Double_t xx = x[0];
  Double_t fitval = par[0]*sin(par[1]*xx)/xx;
  return fitval;
}

void main() {
  TCanvas *c1 = new TCanvas("c1","c1",500,500);
  TF1 *myfunc = new TF1("bwahaha", fitfunc, 0,10,2);
  myfunc->SetParameters(3.1,5.3);
  myfunc->Draw();
}
```

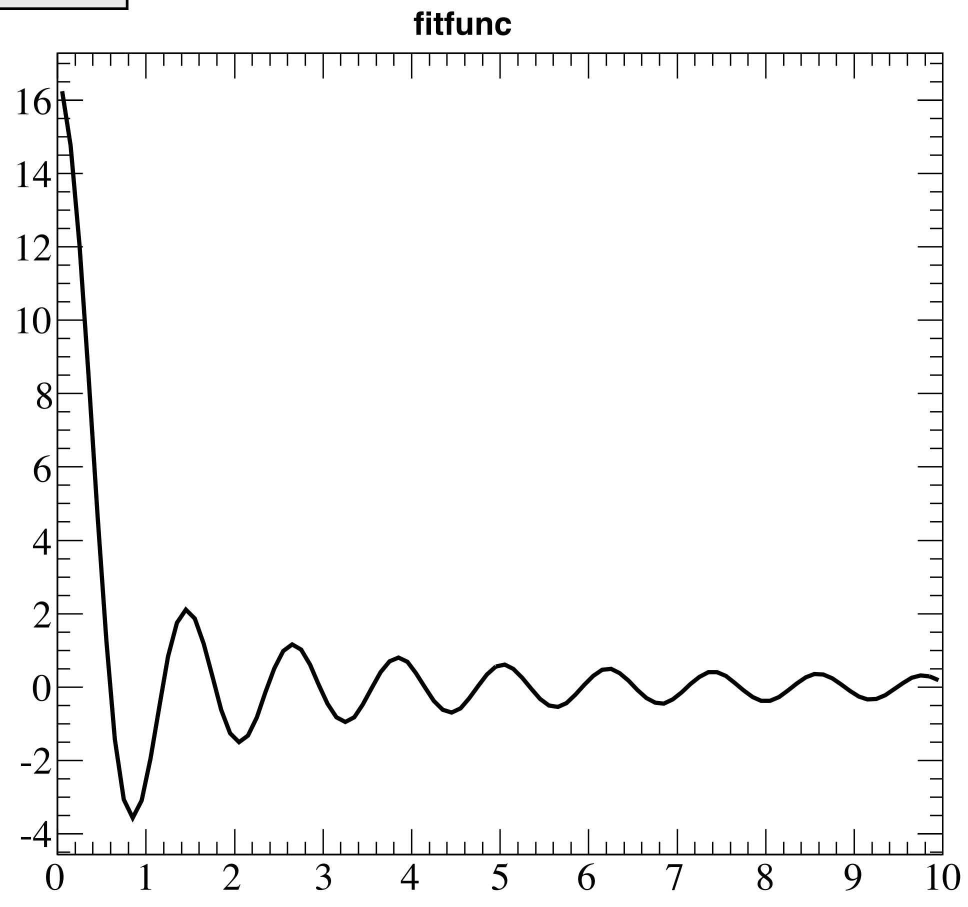
Arguments not optional!
xmin, xmax, number of parameters

User-defined functions

Named macro:
myfunction.C

Load the macro in ROOT and execute the main routine

```
root[0] .L myfunction.C
root[1] main()
```



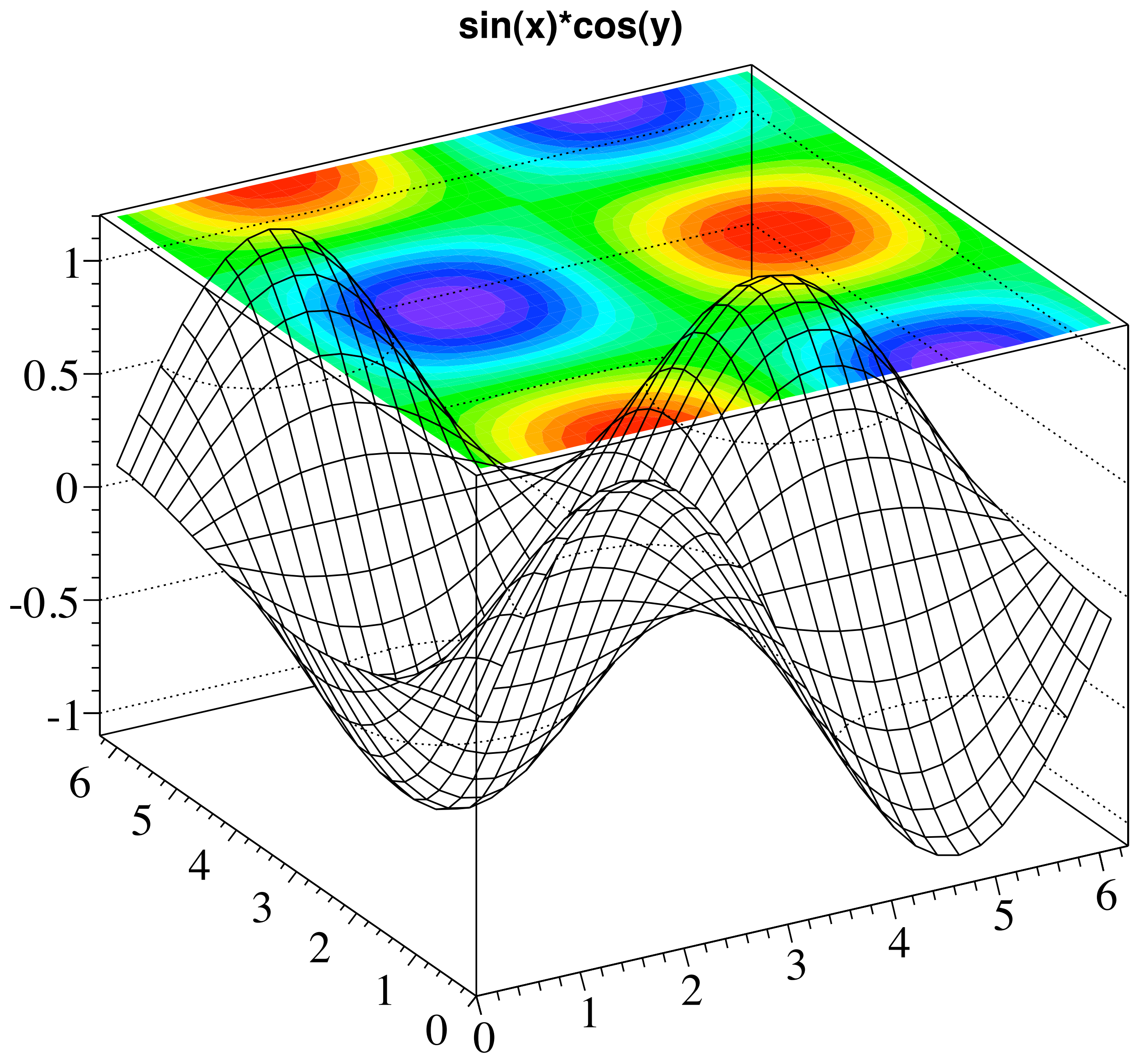
Beyond 1-dimensional functions...

Example:

x-range

```
TF2 *mytf2 = new TF2(“mytf2”, “sin(x)*cos(y)”, 0, 2*(TMath::Pi()),
                                                  0, 2*(TMath::Pi()));
```

y-range



```
mytf2->Draw(“surf3”);
```

http://root.cern.ch/root/html/TF2.html

http://root.cern.ch/root/html/TF3.html

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

What can you do with functions?

Draw	`f->Draw()`
Print	`f->Print()`
Evaluate	`f->Eval(2.3)`
Integrate	`f->Integral(0.3,1.5)`
Differentiate	`f->Derivative(4.8)`

and many more things... http://root.cern.ch/root/html/TF1.html

But most importantly (for this tutorial), you can use them for fitting!

A few words about fitting...

A mathematical procedure to find parameter values of a function, ***f***, that *best describe* your data distribution.

One common method for finding an optimum fit is to minimize a χ^2 function

adjustable parameters $a_0, a_1, \ldots, a_k$

measured value in bin i

value predicted by function ***f*** in bin i for parameters a_0, a_1, etc.

$$\chi^2(a_0 \ldots a_k) = \sum_{i=1}^{nbins} \left(\frac{y_i - f(x_i; a_0 \ldots a_k)}{\sigma_i} \right)^2$$

uncertainty on measured value in bin i

A rule of thumb for a “reasonable” fit: χ^2/NDF ~ 1

When you use ROOT to fit a distribution, its task is to find the set of parameters that give the lowest value of χ^2.

It reports the parameter values for the lowest χ^2 and its best estimate of the uncertainties on those parameters. You can tell a lot about the reliability of your fit results by looking at the fit itself and the parameter uncertainties.

A few (more) words about fitting...

Do not automatically trust the results of a fit!

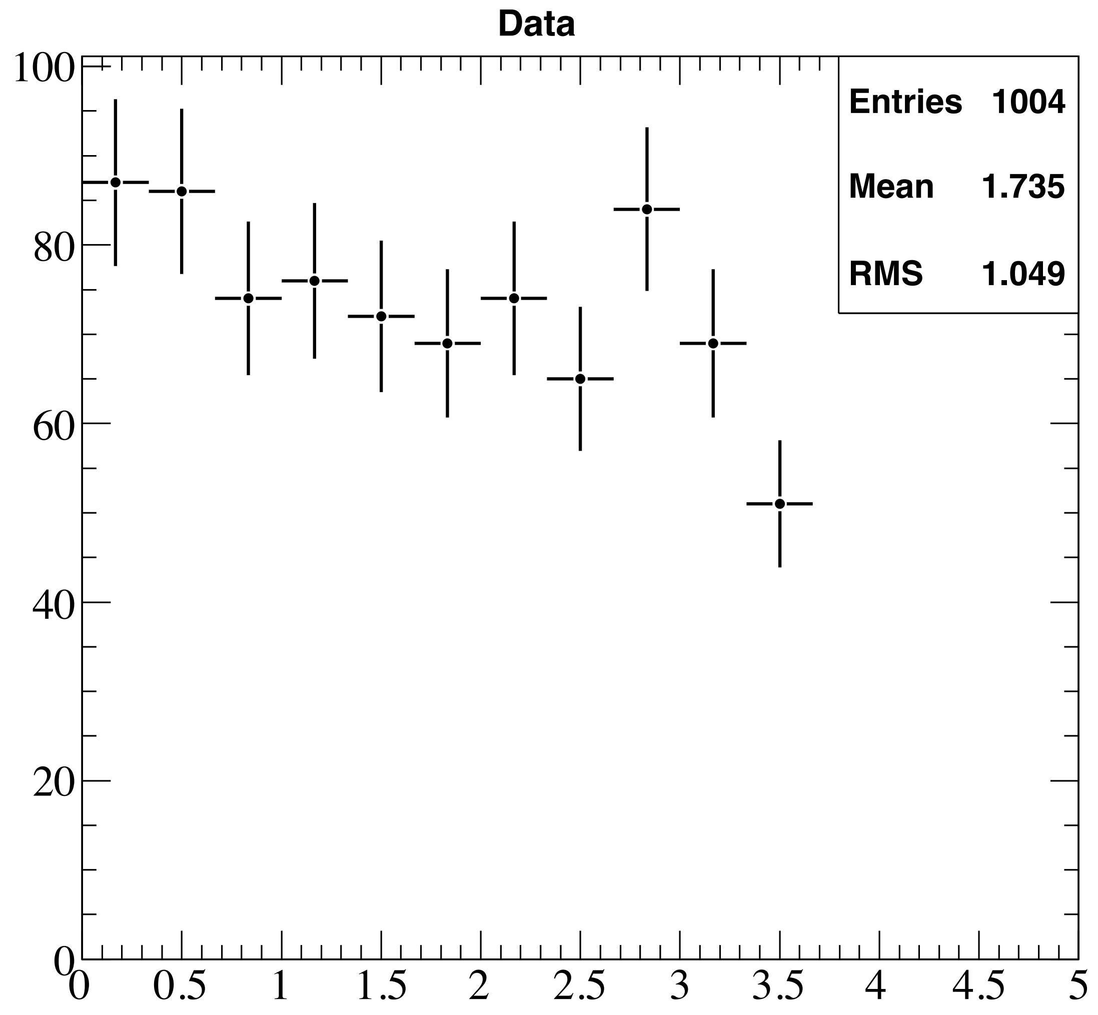
Think about the following things...

- Does the fit look good by eye?
- Are you using the right fit model to describe the data? (Crap in = crap out!)
- Are your error bars over- or under-estimated?
- Do the reported uncertainties on the fit parameters make sense?

Fitting can be done via the TH1::Fit method

Exercise 1: Straight line fit

Download the file
http://home.fnal.gov/~jlraaf/2011REU/lecture2_exercises/lines2.root



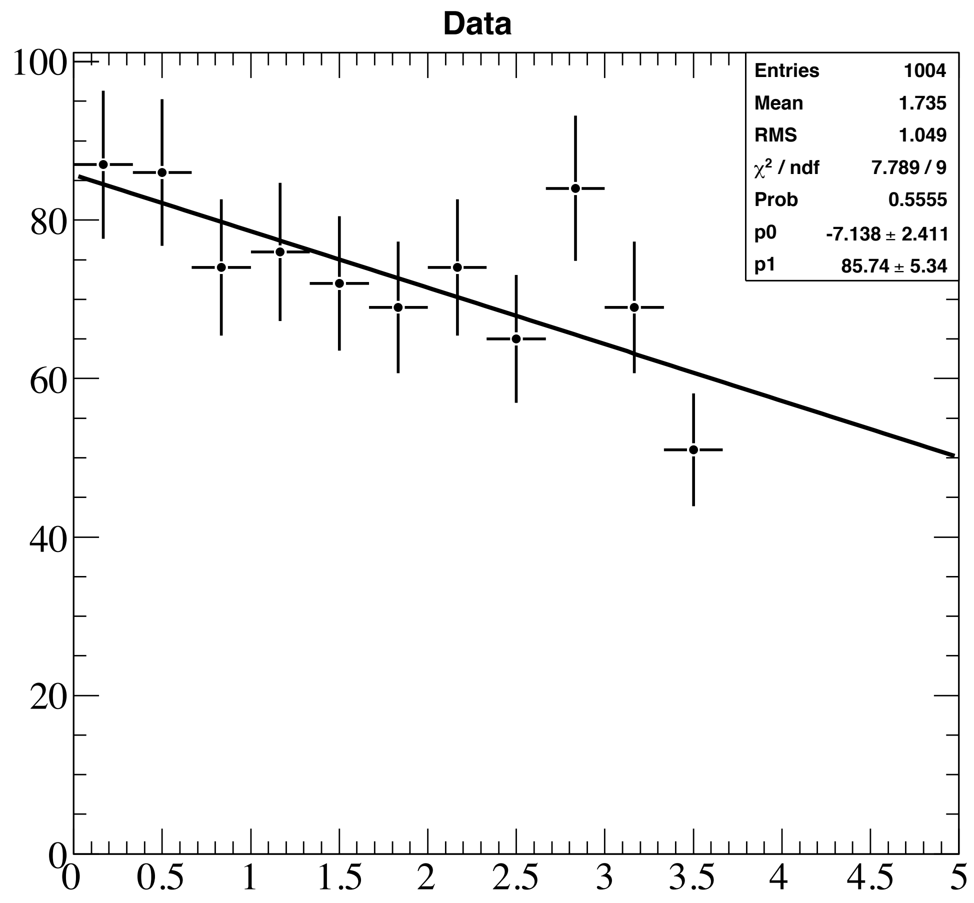
- Define a 1-dimensional function to fit a straight line to histogram "h0" and perform the fit.

Hint: Fitting can be done using the Fit method of TH1, of the form `histogram->Fit("myfunction")`

- What are the χ^2 and number of degrees of freedom? Is it a good fit?

- Do you think the uncertainties on the fit parameters are reasonable?
  Draw two extra TF1's on the plot:
  First, a new line with its parameters set to (best fit values + uncertainties) from part 1
  Second, a new line with parameters set to (best fit values - uncertainties) from part 1
  Why don't the lines look like they describe the data well?

Exercise 1: Straight line fit



```
root[0] TFile *infile = new TFile(“lines2.root”);

root[1] TF1 *line0 = new TF1("line0","[0]*x+[1]",0,5);
root[2] line0->SetParameters(-1.0,10.0);

root[3] h0->Fit("line0");
```

- What are the χ^2 and number of degrees of freedom? Is it a good fit?

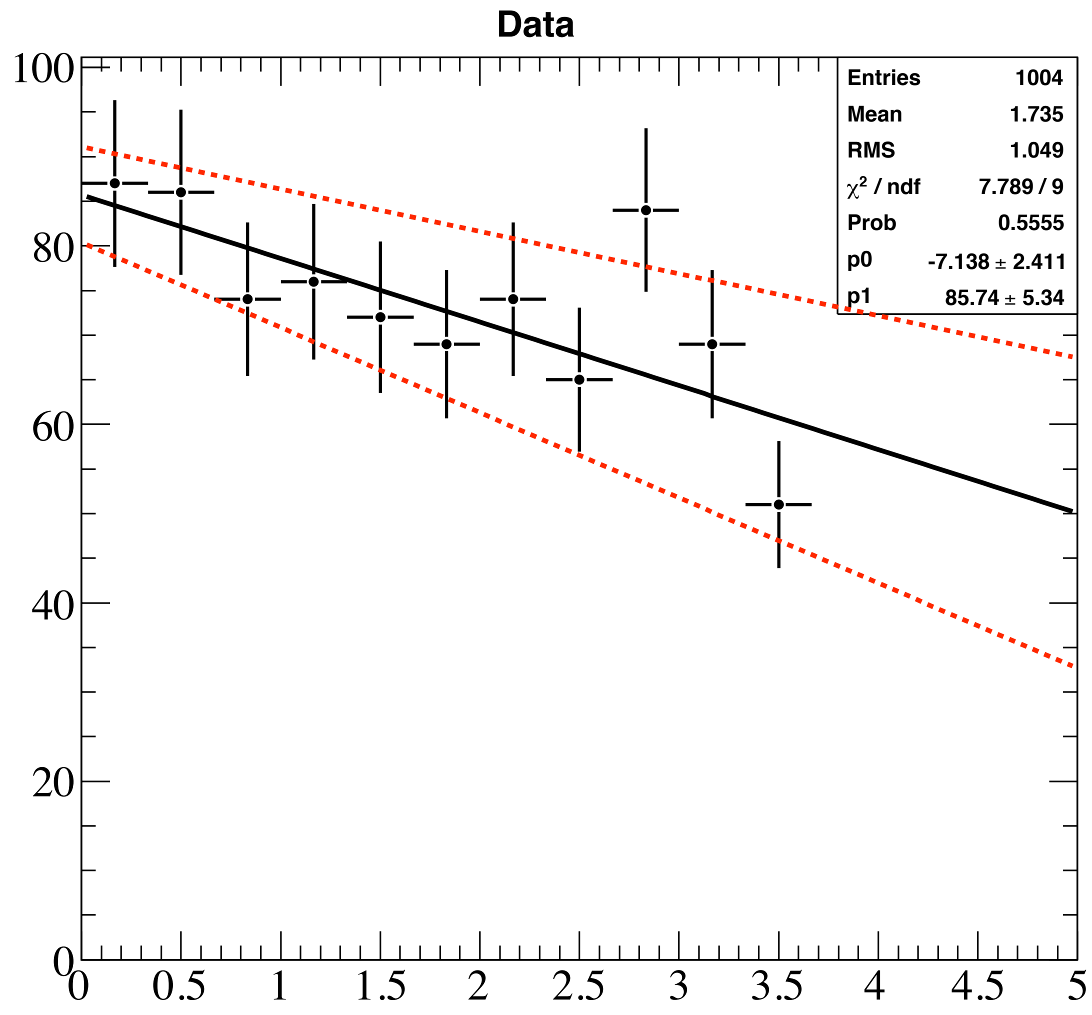
```
root[4] TF1 *fitresult = h0->GetFunction(“line0”);

root[5] fitresult->GetChisquare()
(const Double_t)7.78919022669480121e+00

root[6] fitresult->GetNDF()
(const Int_t)9

root[7] fitresult->GetProb()
(const Double_t)5.55522583770750478e-01
```

This probability is not the “probability that your fit is good.” If you did many fake experiments (draw many random samples of data points from the assumed distribution (your fit function)), this is the percentage of experiments that would give χ^2 values ≥ to the one you got in this experiment.

The probability is usually “reasonably large” or “unreasonably small.”

Exercise 1: Straight line fit

- Do you think the uncertainties on the fit parameters are reasonable?
  - Draw two extra TF1's on the plot:
    - First, a new line with its parameters set to (best fit values + uncertainties) from part 1
    - Second, a new line with parameters set to (best fit values - uncertainties) from part 1
  - Why don't the lines look like they describe the data well?



Blindly adjusting the fit parameters to the bounds of their 1-sigma errors gives unreasonable looking results.

Why?

Exercise 1: Straight line fit

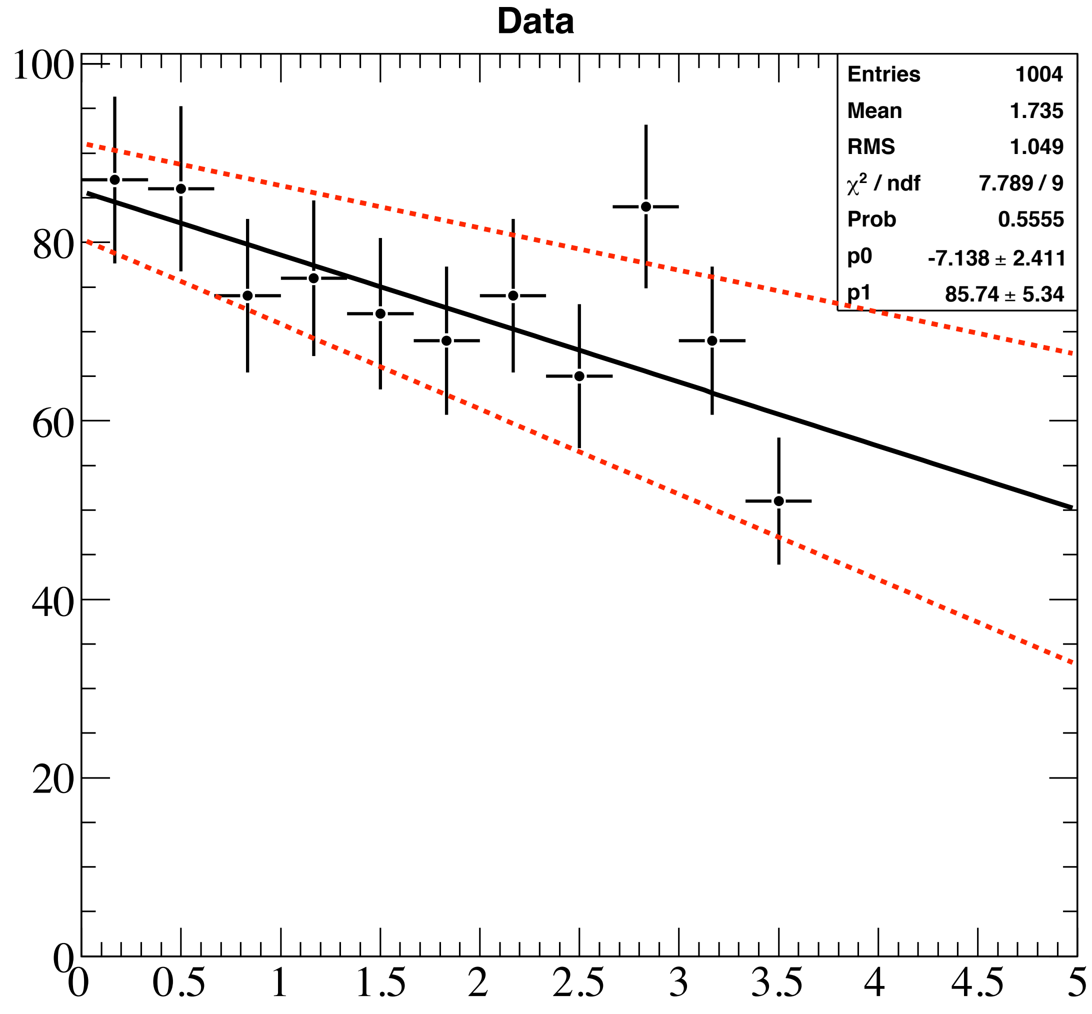
- Do you think the uncertainties on the fit parameters are reasonable?

  Draw two extra TF1's on the plot:

  First, a new line with its parameters set to (best fit values + uncertainties) from part 1

  Second, a new line with parameters set to (best fit values - uncertainties) from part 1

  Why don't the lines look like they describe the data well?



Blindly adjusting the fit parameters to the bounds of their 1-sigma errors gives unreasonable looking results.

Why?

The parameters are not independent -- they are correlated.

ROOT (by way of its underlying minimizer, called TMinuit) can print the covariance (error) matrix for you.

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

```
root[0] TMatrixD matrix0(2,2);

root[1] gMinuit->mnemat(matrix0.GetMatrixArray(),2);
root[2] matrix0.Print()
2x2 matrix is as follows

     |      0    |      1    |
-------------------------------
   0 |      5.812       -11.3
   1 |      -11.3       28.52
```

Diagonal elements: (parameter error)^2

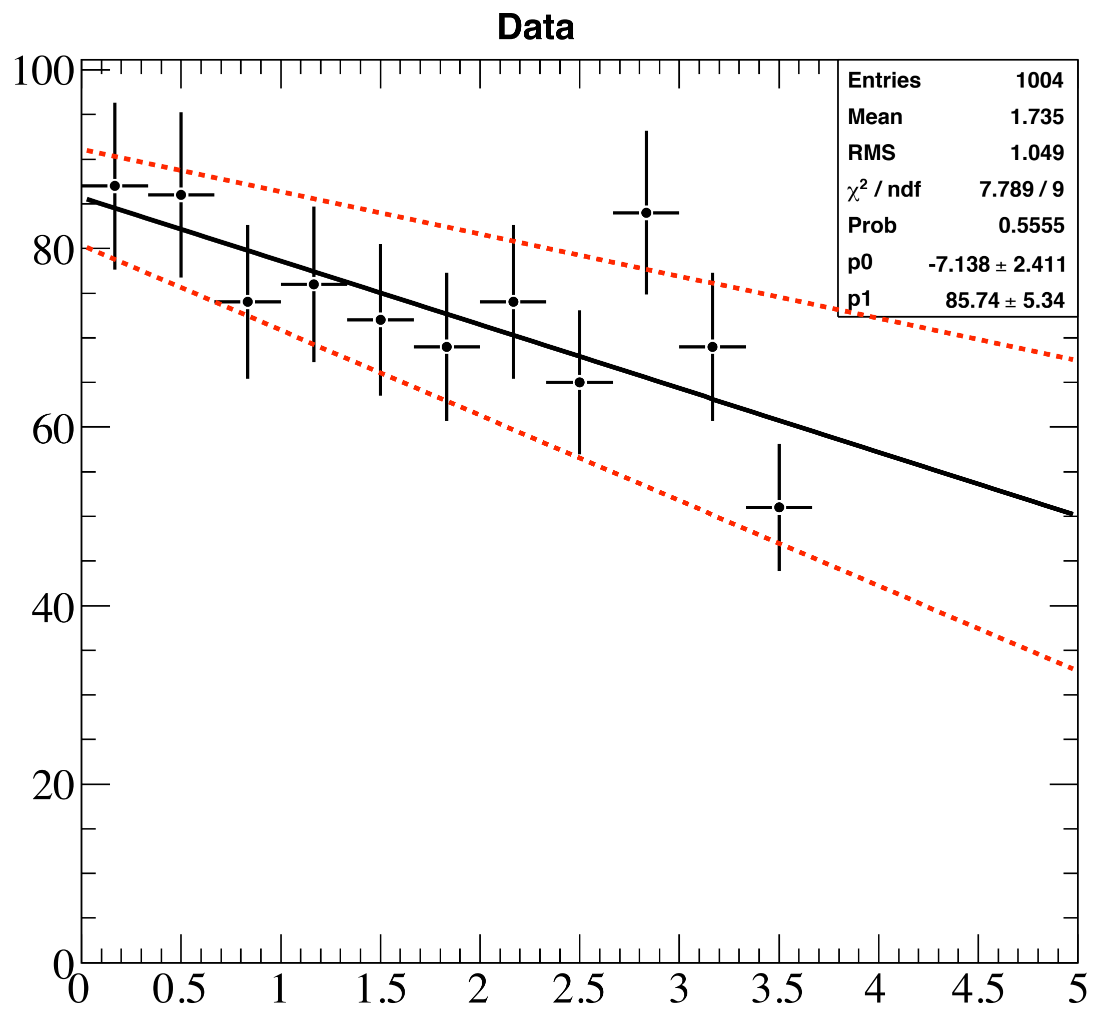
Off-diagonal elements: how parameters co-vary

(Negative number: parameters must change in opposite directions to stay within 1-sigma errors of fit)

Exercise 1: Straight line fit

• Do you think the uncertainties on the fit parameters are reasonable?

Make a new TF1 and set its parameters to the upper range of best fit parameters (best parameters + errors). Do the same for the lower range (best parameters - errors). Draw the two new functions on the same plot with the original fit. Why don't they look like they describe the data well?

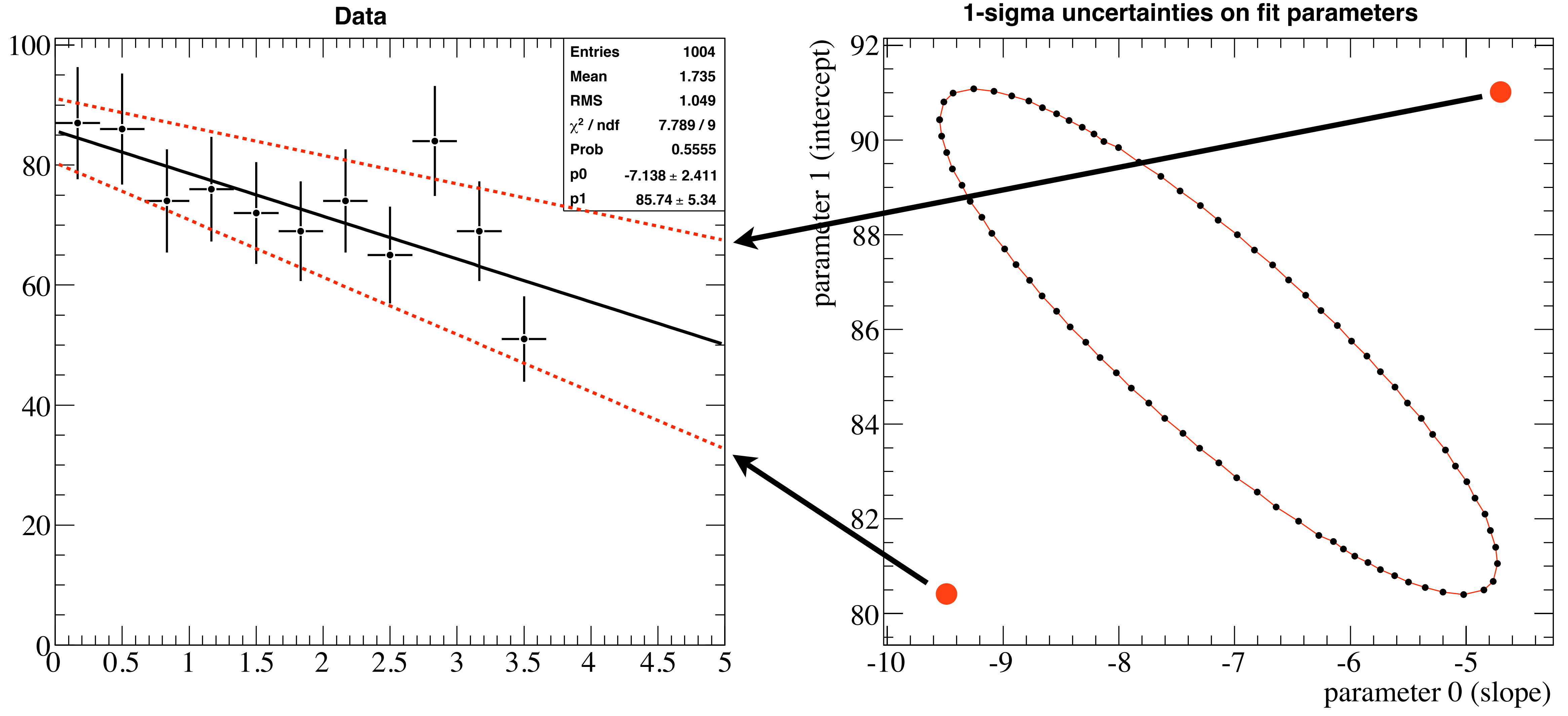


TMinuit can also calculate the correlation matrix from the covariance matrix...

```
root[3] gMinuit->mnmatu(1)
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2
ERR DEF=1
  5.812e+00 -1.130e+01
 -1.130e+01  2.852e+01
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2
        1  0.87803   1.000 -0.878
        2  0.87803  -0.878  1.000
```

-1	would indicate perfect anti-correlation
0	no correlation of these parameters
+1	perfect correlation

Parameter correlations

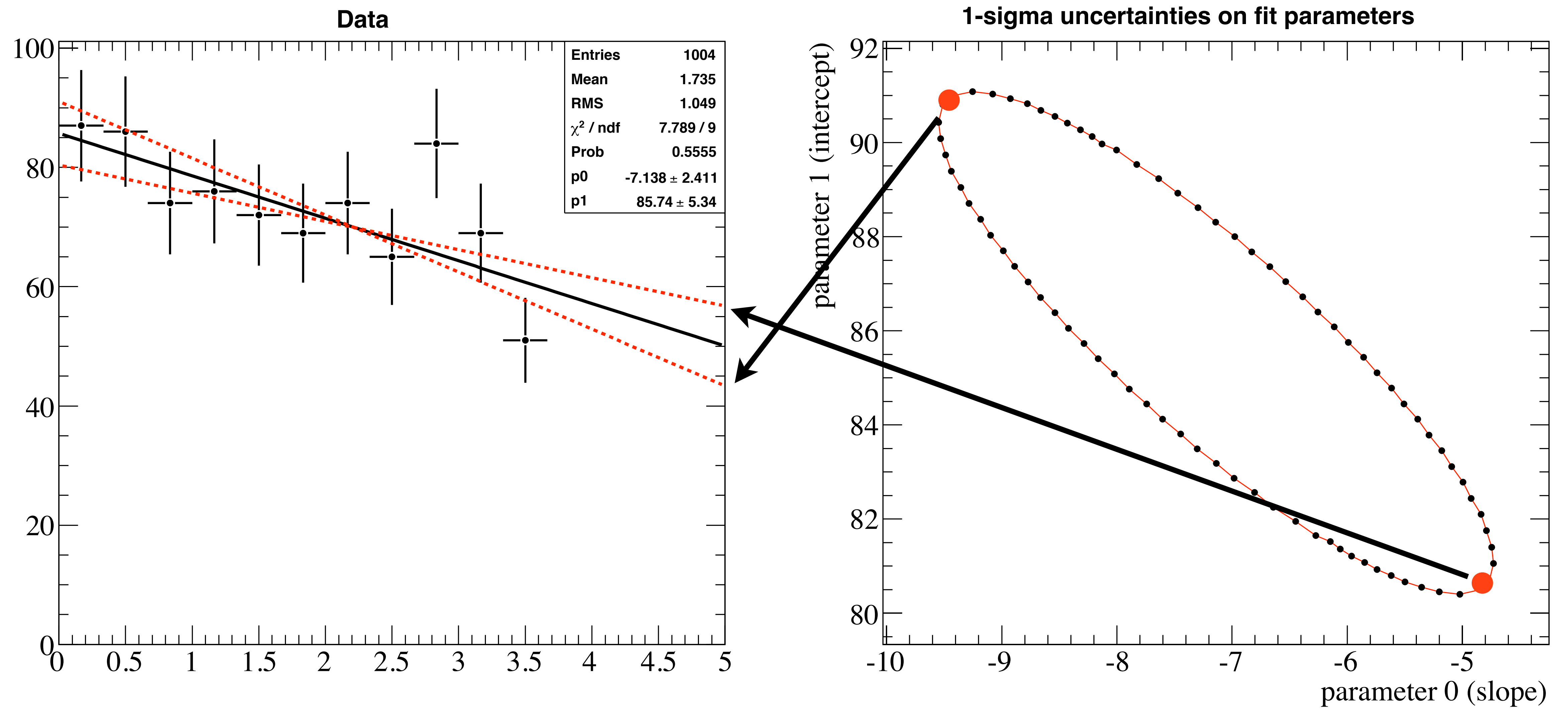


SetErrorDef(N^2) for N-sigma error

Show 80 points on contour, show correlation of param 0 with param 1

```
root[3] gMinuit->SetErrorDef(1);
root[4] TGraph *gr0 = (TGraph *)gMinuit->Contour(80,0,1);
root[5] gr0->SetLineColor(kRed);
root[6] gr0->Draw("alp");
root[7] gr0->GetXaxis()->SetTitle("parameter 0 (slope)");
root[8] gr0->GetYaxis()->SetTitle("parameter 1 (intercept)");
root[9] gr0->SetTitle("1-sigma uncertainties on fit parameters");
```

Parameter correlations

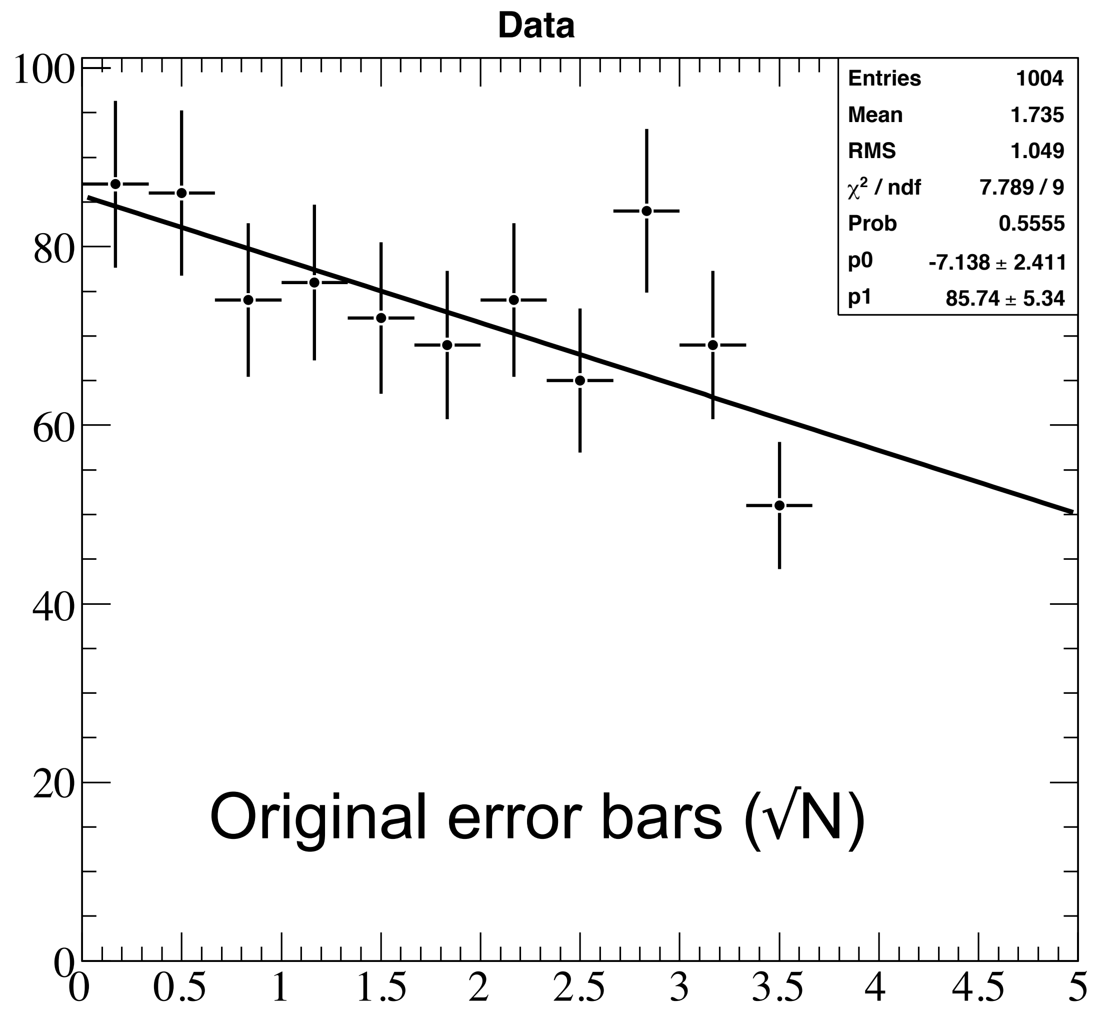


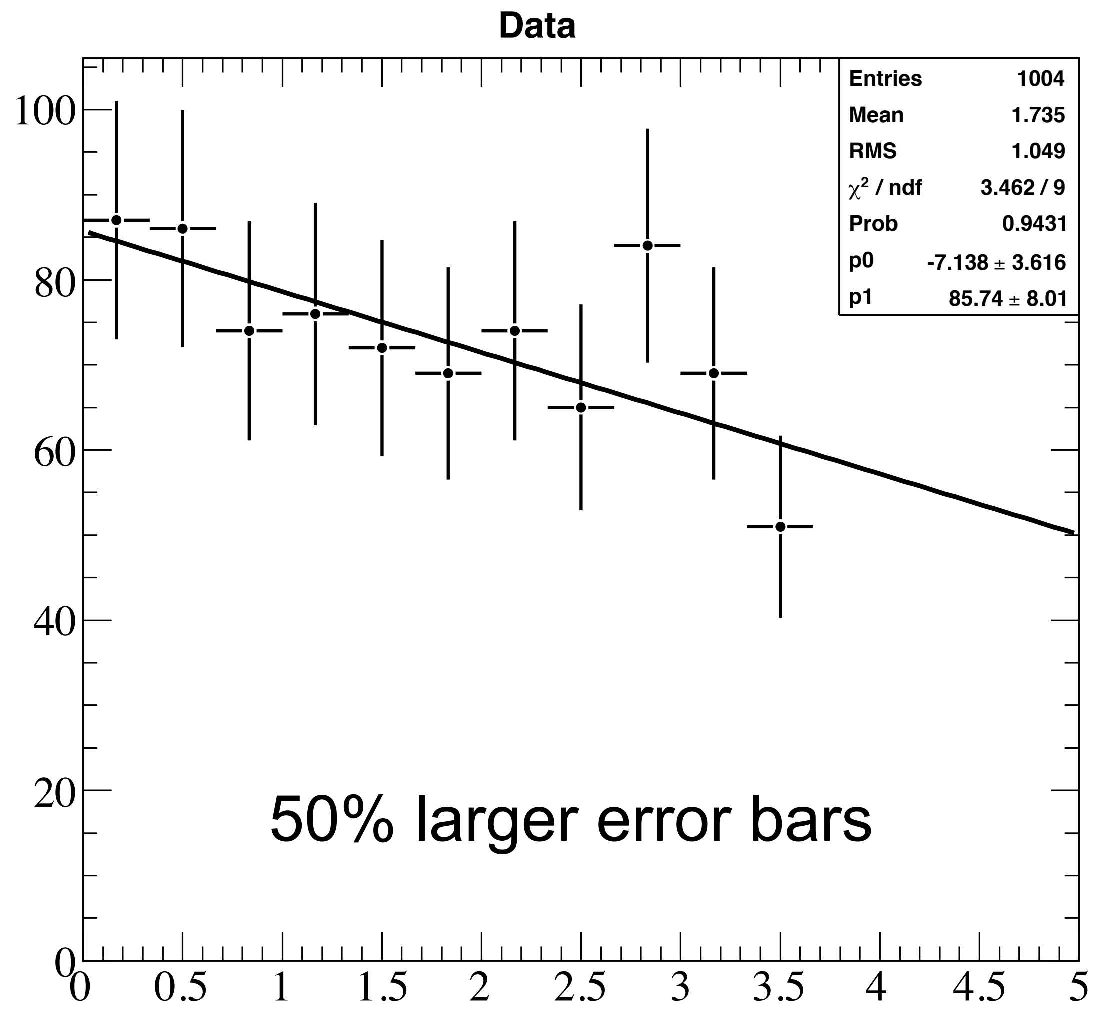
SetErrorDef(N^2) for N-sigma error

Show 80 points on contour, show correlation of param 0 with param 1

```
root[3] gMinuit->SetErrorDef(1);
root[4] TGraph *gr0 = (TGraph *)gMinuit->Contour(80,0,1);
root[5] gr0->SetLineColor(kRed);
root[6] gr0->Draw("alp");
root[7] gr0->GetXaxis()->SetTitle("parameter 0 (slope)");
root[8] gr0->GetYaxis()->SetTitle("parameter 1 (intercept)");
root[9] gr0->SetTitle("1-sigma uncertainties on fit parameters");
```

How do error bars affect the fit?





Entries	1004
Mean	1.735
RMS	1.049
χ^2 / ndf	7.789 / 9
Prob	0.5555
p0	-7.138 ± 2.411
p1	85.74 ± 5.34

Lower χ^2 value, higher probability

Best fit values remain the same, but larger uncertainty on parameter values.

Entries	1004
Mean	1.735
RMS	1.049
χ^2 / ndf	3.462 / 9
Prob	0.9431
p0	-7.138 ± 3.616
p1	85.74 ± 8.01

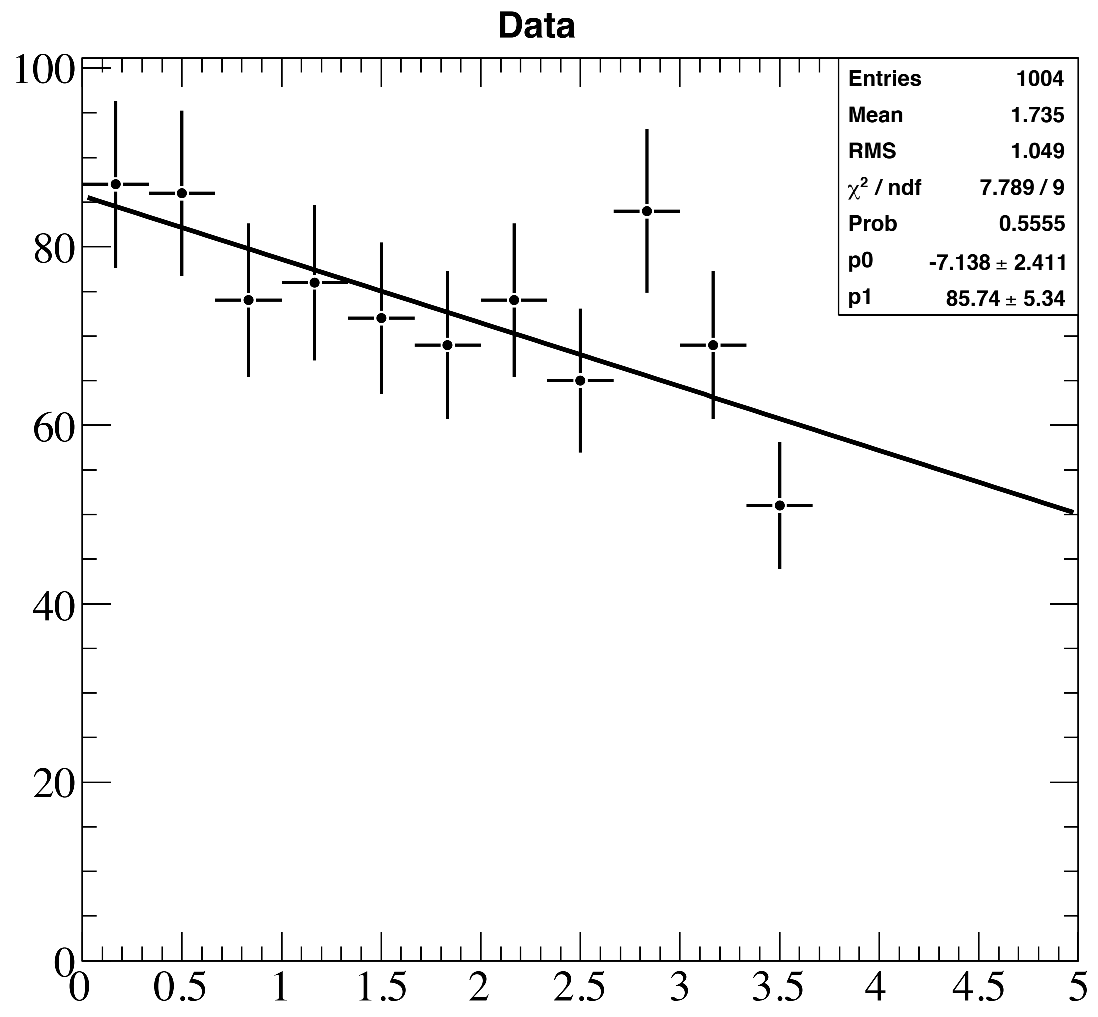
Very high probability, but very low χ^2 value:

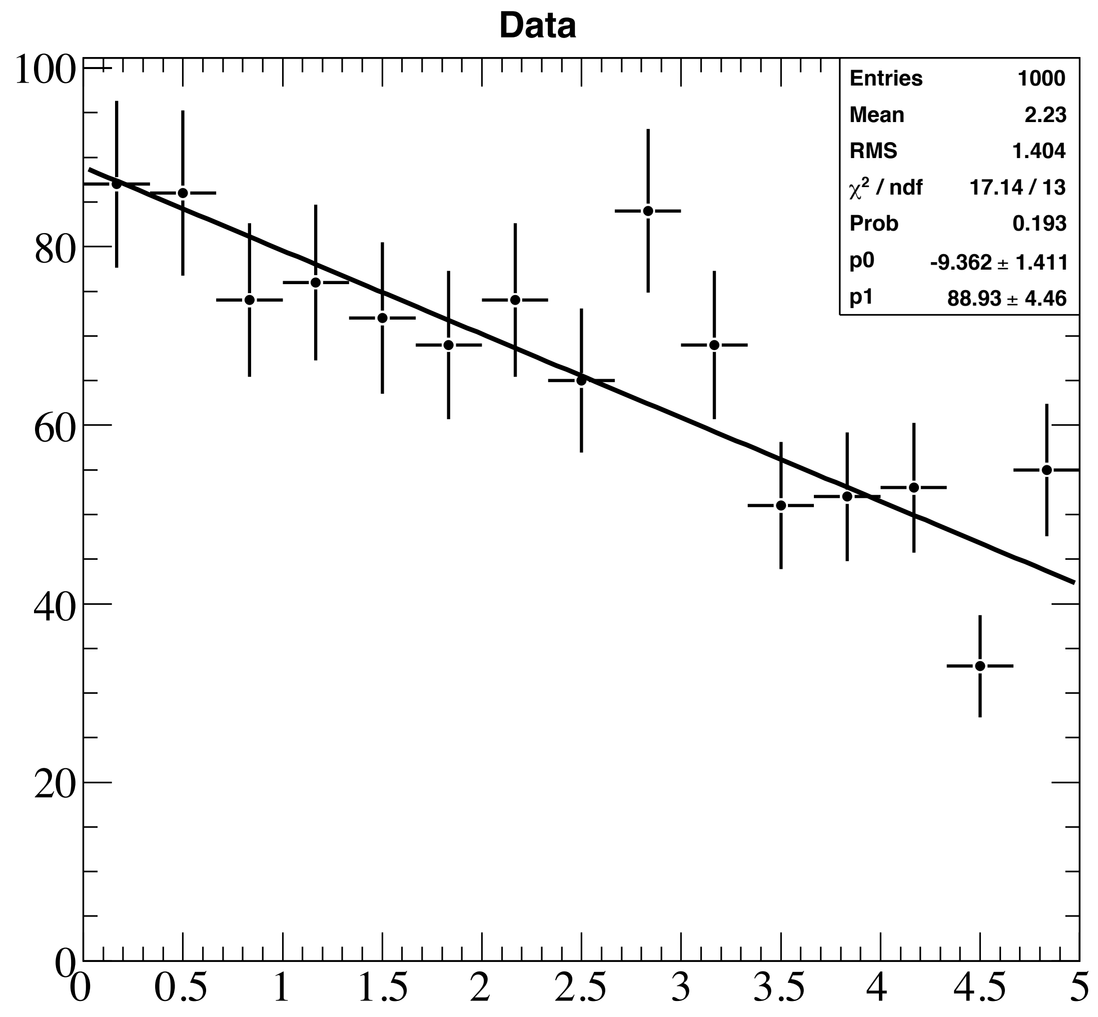
This does NOT necessarily mean that your fit is good!

A strong indication that something is not quite right...

maybe you've over-estimated your measurement errors.

What about adding more data points?



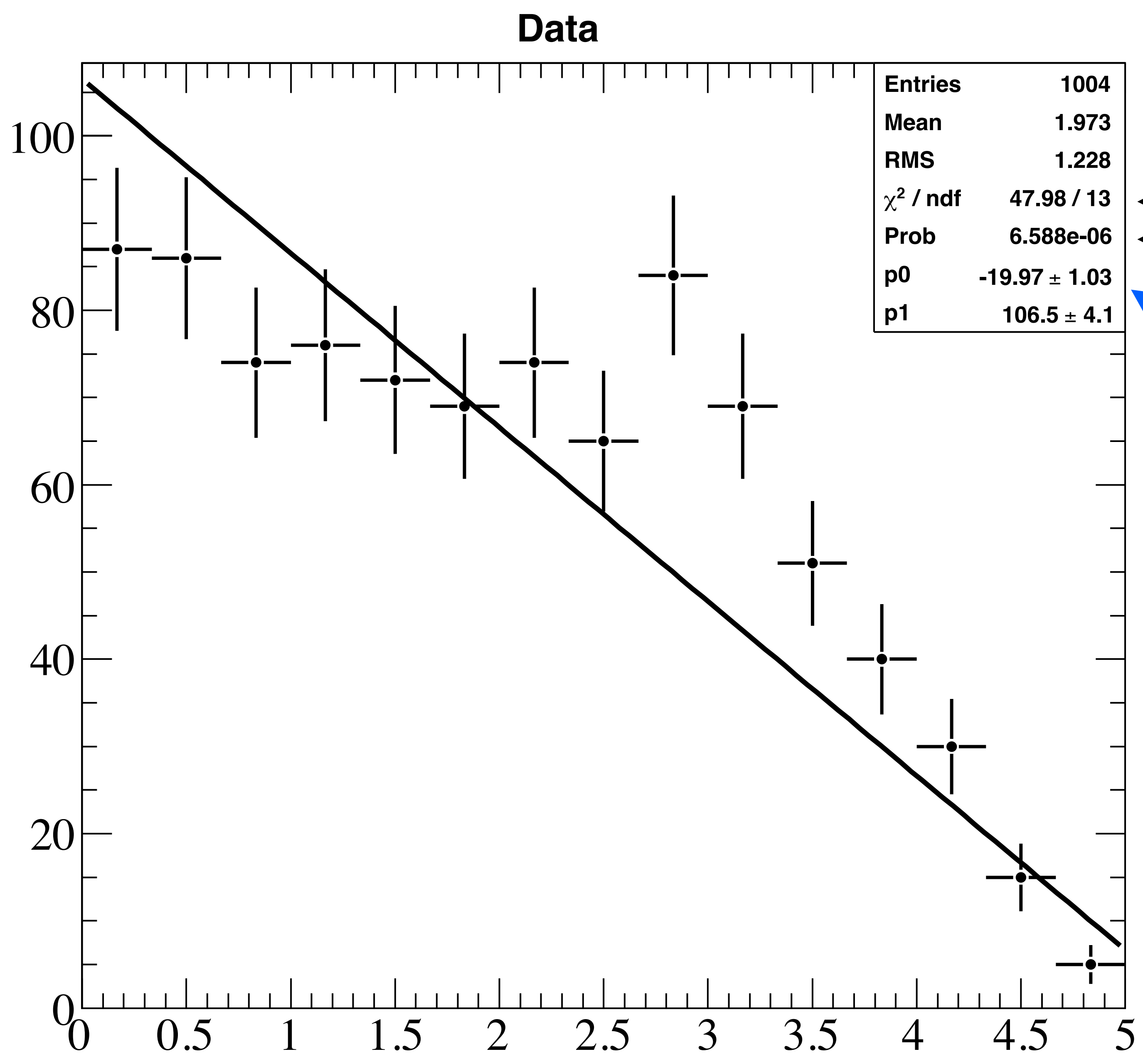


Both have reasonable χ^2/NDF, but the fit with more data has smaller uncertainties on the parameters.

Fit goodness (χ^2/NDF) and fit parameter uncertainties are affected by both the **number of data points** and by the **size of the data errors**.

Beware! It's easy to get a terrible fit that reports tiny fit uncertainties...

Terrible fit with tiny uncertainties



```
Entries        1004
Mean          1.973
RMS           1.228
χ² / ndf  47.98 / 13
Prob      6.588e-06
p0     -19.97 ± 1.03
p1       106.5 ± 4.1
```

Two indications of a bad fit...

χ^2/NDF >> 1
"Unreasonably small" probability

But the parameter uncertainties are smaller than in original fit with same number of data points!

Low probability and large χ^2 may indicate that the fit parameters and their uncertainties are meaningless.

This is not a good model to describe the data!
Don't just blindly accept the best fit parameters and their uncertainties from the fitter.
Think about whether or not they make sense!

Exercise 2: Signal + background

Download the file
http://home.fnal.gov/~jlraaf/2011REU/lecture2_exercises/signal+background.root

Inside you'll find a 1-dimensional histogram showing a Gaussian-distributed signal on top of a to-be-determined background.

- Use pre-defined ROOT functions to fit the spectrum on the command line
- Write a macro that fits this spectrum with a user-defined function

Fitting Options

TH1::Fit(const char* fname, Option_t* option, Option_t* goption, Axis_t xmin, Axis_t xmax)

fname	Function name
option	Fitting options: “W” Set all errors to 1 “I” Use integral of function in bin instead of value at bin center “L” Use log likelihood method (default is chi-square) “LL” Use log likelihood method and bin contents are not integers “U” Use a user-specified minimization algorithm (via SetFCN) “Q” Quiet mode (minimum printing) “V” Verbose mode (default is between Q and V) “E” Perform better error estimation using MINOS technique “M” More. Improve fit results “R” Use the range specified in the function range “N” Do not store the graphics function. Do not draw. “0” Do not plot the result of the fit “+” Add this new fitted function to the list of fitted functions
goption	Graphical options
xmin - xmax	Fitting range

So far, we have been fitting by the chi-square method.

Fitting binned datasets: Chi-square vs. Log Likelihood

Chi-square

- Assumes that the events within each bin of the histogram are gaussianly distributed
- Works well at high statistics, but doesn't always give reliable results when some of the histogram bins have few (or zero) entries.

Binned Log Likelihood

- Correct Poisson treatment of low statistics bins (≲10 events)
- Recommended when some histogram bins have few (or zero) entries

Given a set of parameters, what is the probability that your real dataset of "y_i's" could occur?

Gaussian case:

$$P(a_0...a_k) = \prod_{i=1}^{nbins} \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{1}{2}\sum\frac{(y_i - f(x_i;a_0...a_k))^2}{\sigma_i^2}}$$

$$\chi^2(a_0...a_k) = \sum_{i=1}^{nbins}\left(\frac{y_i - f(x_i;a_0...a_k)}{\sigma_i}\right)^2$$

Maximizing this probability is equivalent to minimizing the sum in the exponential... which we call χ^2

Poissonian case:

$$P(a_0...a_k) = \prod_{i=1}^{nbins} \frac{[f(x_i;a_0...a_k)]^{y_i}}{y_i!} e^{-f(x_i;a_0...a_k)}$$

$$\ln P(a_0...a_k) = \sum[y_i \ln f(x_i;a_0...a_k)] - \sum f(x_i;a_0...a_k) + const$$

It is easier (and equivalent) to maximize the natural log of the probability (or to minimize the negative log)

Fitting unbinned datasets: Unbinned Log Likelihood

Usually more powerful than binned chi-square or binned likelihood fits, and works well when you have multi-dimensional datasets to fit.

Minor annoyances:
- No “goodness of fit” parameter
- Can’t show fit with data (unless you bin it...)

In ROOT, you can do 1-, 2-, or 3-dimensional unbinned fits to TTree variables

```
myTree->UnbinnedFit(“function”,”treevariable”,”cuts”);
```

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

Other Fitting Techniques: `TFractionFitter`

Sometimes you may have a data distribution that you would like to fit, but its shape cannot be described by a standard mathematical function. If you know the expected shape from Monte Carlo, you can use that as the “fit function” instead of a gaussian, or exponential, etc.

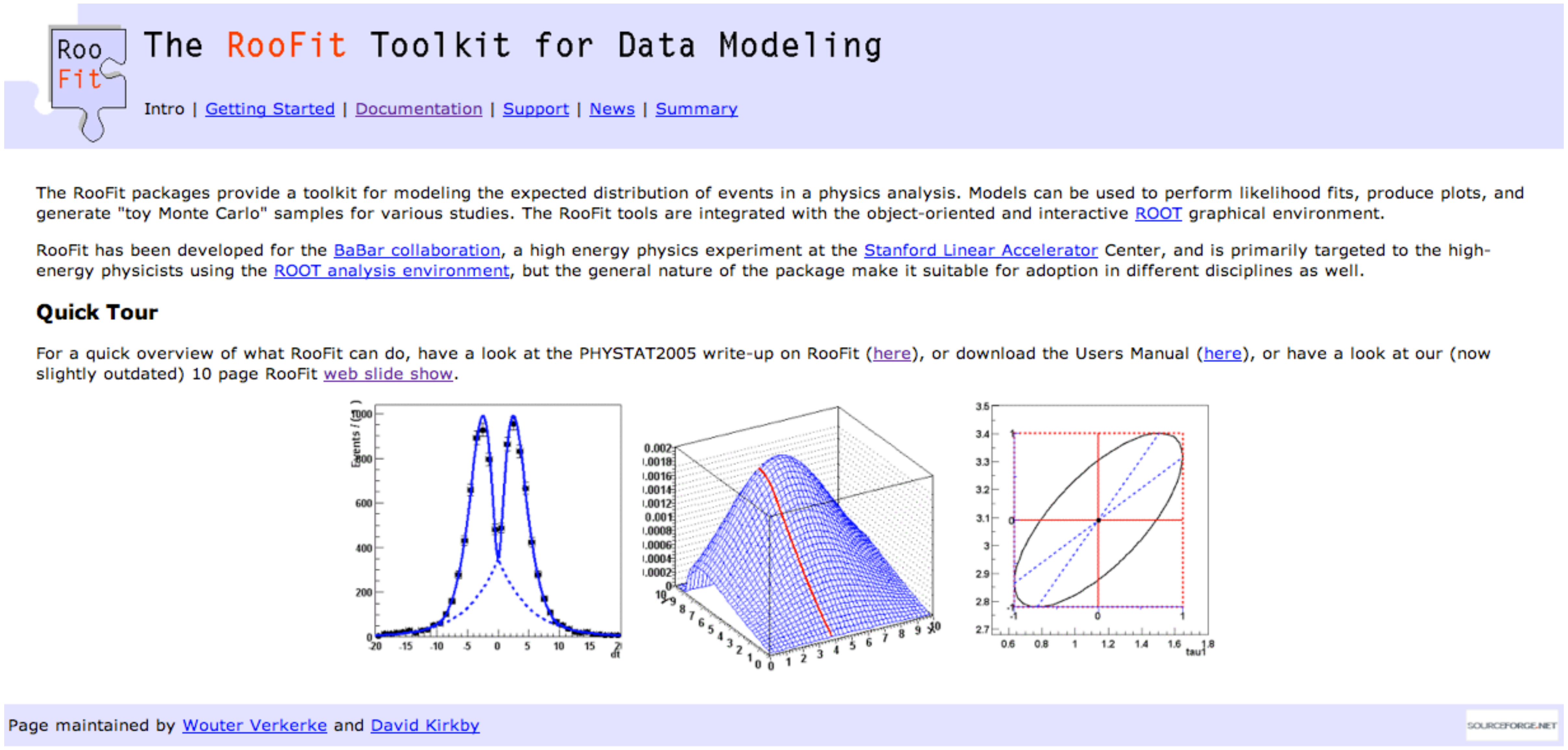
TFractionFitter allows you to fit a data histogram using several Monte Carlo histograms (instead of a defined function).

```
TH1F *data;                                 //data histogram
TH1F *mcsig;                                // MC signal histogram
TH1F *mcbg1;                                // MC background 1 histogram
TH1F *mcbg2;                                // MC background 2 histogram
 ...             // Fill your histograms here...

TObjArray *mctot = new TObjArray(3);        // MC histograms are put in this array
mctot->Add(mcsig);
mctot->Add(mcbg1);
mctot->Add(mcbg2);
TFractionFitter* myfit = new TFractionFitter(data, mctot);   // initialize

fit->SetRangeX(1,15);                         // use only the first 15 bins in the fit
Int_t status = myfit->Fit();                  // perform the fit

if (status == 0) {                            // check fit status and plot result if OK
  TH1F* result = (TH1F*) fit->GetPlot();
  data->Draw("Ep");
  result->Draw("same");
}
```

Other Fitting Techniques: `RooFit`

http://roofit.sourceforge.net/

Synopsis: A data modeling (not just fitting) package to facilitate complex likelihood fits. Extremely flexible and extensible.



The RooFit Toolkit for Data Modeling

Intro | Getting Started | Documentation | Support | News | Summary

The RooFit packages provide a toolkit for modeling the expected distribution of events in a physics analysis. Models can be used to perform likelihood fits, produce plots, and generate "toy Monte Carlo" samples for various studies. The RooFit tools are integrated with the object-oriented and interactive ROOT graphical environment.

RooFit has been developed for the BaBar collaboration, a high energy physics experiment at the Stanford Linear Accelerator Center, and is primarily targeted to the high-energy physicists using the ROOT analysis environment, but the general nature of the package make it suitable for adoption in different disciplines as well.

Quick Tour

For a quick overview of what RooFit can do, have a look at the PHYSTAT2005 write-up on RooFit (here), or download the Users Manual (here), or have a look at our (now slightly outdated) 10 page RooFit web slide show.

Page maintained by Wouter Verkerke and David Kirkby

Very nice set of tutorials available from ROOT website:

http://root.cern.ch/root/html/tutorials/roofit/index.html

Disclaimer: I am not an expert on statistics...

Some other sources of information which may be useful...

Philip R. Bevington “Data Reduction and Error Analysis for the Physical Sciences,” (1969)

Louis Lyons, “Statistics for Particle and Nuclear Physicists,” (1986)

CDF Statistics Committee: www-cdf.fnal.gov/physics/statistics/statistics_home.html

More details...

TMinuit Options: MIGRAD vs. HESSE vs. MINOS

MIGRAD (default)

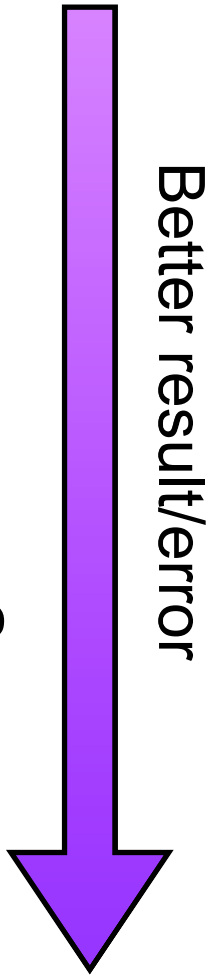
Local function minimization using a “variable-metric inexact line search” which is a modified version of Davidson-Fletcher-Powell switching method described in Fletcher, Comp J **13**, 317 (1970)

HESSE

Calculates full 2nd derivative matrix of user function FCN using a finite difference method.
Only works if given a starting point reasonably near minimum; often used to improve upon the result from MIGRAD

MINOS

Performs a better error analysis with asymmetric errors



It’s a good idea to use (at least) HESSE after MIGRAD to get reliable errors for your fit.
MINOS will give the best estimate of the errors for a given set of parameters.

Using TMinuit Directly

number of free parameters involved in minimization

computed gradient values (optional)

function value

array of parameters

to switch between several actions of FCN

```
extern void mychi2 (Int_t &npar, Double_t *gin, Double_t &f, Double_t *par, Int_t flag)
{
   double thechi2 = 0.0;

   /* Calculate your chi2... */
   f = thechi2;
```

← Equate variable f with the value of your chi2 before returning from the function

```
}

void main() {
   TMinuit *myMinuit = new TMinuit(2);
```

← Initialize Minuit with a max. of 2 parameters to minimize

```
   myMinuit->SetFCN(mychi2);
```

← Set the minimization function

```
   Double_t arglist[10];
   arglist[0] = 1;
   Int_t ierflg = 0;
   myMinuit->mnexcm("SET ERR", arglist, 1, iflag);
```

← Change the '1-sigma' error definition

```
   static Double_t vstart[2] = {0.04,    -0.03};
   static Double_t step[2]   = {0.0001, 0.0001};
   static Double_t minvals[2]= {0.0, 0.0};
   static Double_t maxvals[2]= {0.0, 0.0};
   gMinuit->mnparm(0,"a",vstart[0],step[0],minvals[0],maxvals[0],ierflg);
   gMinuit->mnparm(1,"b",vstart[1],step[1],minvals[1],maxvals[1],ierflg);
```

Set the parameter starting values, step sizes, min/max values

```
   //Call MIGRAD with 5000 iterations maximum
   arglist[0] = 5000;     //max number of iterations
   arglist[1] = 0.01;     //tolerance
  gMinuit->mnexcm("MIGRAD", arglist, 2, ierflg);
```

← Run the minimization using MIGRAD

```
}
```