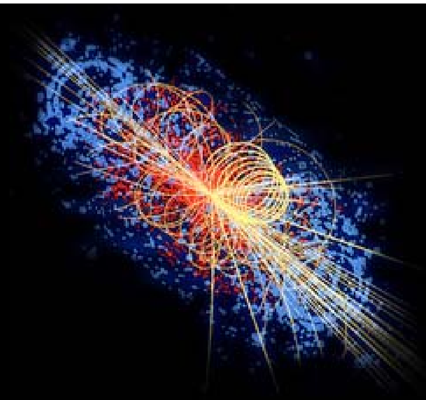


FITTING TUTORIAL

Using ROOT, TMinuit and RooFit for fitting.

Adrian Bevan

YETI January 2007

Uses ROOT 5.12.00

Queen Mary
University of London

OVERVIEW

- 3 tutorials over the next two days:
  - Introduction:
    - Introduction to ROOT.
  - Multi Variate Analysis:
    - Training Neural Networks
    - Tools to calculate fisher discriminants, train neutral networks and boosted decision trees.
  - Fitting:

    Fitting in ROOT (1): writing your own PDF to fit to.

    (2): Using TMinuit

    (3): RooFit

Aims of this Tutorial

- Lean how to define a user function for fitting within ROOT

- Learn how to set up a fit using TMinuit directly.

- Learn how to set up a fit using RooFit.
  - There is insufficient time for this tutorial to cover the use of RooFit, however there is self study material on this subject.

Fitting in ROOT

- You saw how to use the ROOT Fit Panel to fit a pre-defined function to a histogram.
- What if the function you need to fit is not pre-defined within ROOT?
  - Define a function to use in ROOT.
  - Write the function to minimise and use TMinuit directly to minimise the function.
  - Use a higher level fitting package to build a more complicated fit model (e.g. RooFit).

Defining a function to fit in ROOT

See Chapter 5 of the ROOT 5.12 User Guide for more details

- Several steps involved
  - Write the fit function.
  - Instantiate a TF1 object for a 1D fit.
    - Note you can also define 2D (3D) fit function using a TF2 (TF3). If you need to perform a higher dimensional fit, then you need to consider using TMinuit, RooFit or an equivalent fit method.
  - Set the initial parameter values, limits etc for the fit object
  - Fit the data.
    - You can choose between a χ^2 fit (default) or a $-\ln\mathcal{L}$ fit.

Example: Triple Gaussian

- Sometimes a Gaussian function is insufficient to model both the core and outlier parts of a peaking distribution. The sum of more than one Gaussian might give a better χ^2 than a single Gaussian model.

$$G3(x, N, \mu_1, \sigma_1, \mu_2, \sigma_2, \mu_3, \sigma_3) = N\left\{ f_1 e^{-(x-\mu_1)^2/2\sigma_1^2} + f_2 e^{-(x-\mu_2)^2/2\sigma_2^2} + \left(1 - f_1 - f_2\right) e^{-(x-\mu_3)^2/2\sigma_3^2} \right\}$$

- Variables to fit are:
  - N, the overall normalisation.
  - Three means and widths of the Gaussian functions (μ_i, σ_i).
  - Two fractions: f_1 and f_2.

Example: Triple Gaussian

- Write the fit function for

$$G3(x,N,\mu_1,\sigma_1,\mu_2,\sigma_2,\mu_3,\sigma_3) = N\left\{ f_1 e^{-(x-\mu_1)^2/2\sigma_1^2} + f_2 e^{-(x-\mu_2)^2/2\sigma_2^2} + (1-f_1-f_2)e^{-(x-\mu_3)^2/2\sigma_3^2} \right\}$$

```
/*
 * Define the fit function to determine the parameters of the
 * sum of three Gaussians
 *
 * parameter   name                  description
 * --------------------------------------------------------------
 * 0          norm              Normalisation.  The value of this parameter is dependent on the
 *                                               fit range and choice of histogram binning.
 * 1          mu1               mean of Gaussian 1
 * 2          sigma1            width of Gaussian 1
 * 3          mu2               mean of Gaussian 2
 * 4          sigma2            width of Gaussian 2
 * 5          mu3               mean of Gaussian 3
 * 6          sigma3            width of Gaussian 3
 * 7          frac1             fraction of Gaussian 1
 * 8          frac2             fraction of Gaussian 2
 */
Double_t fitFunc(Double_t * x, Double_t * par)
{
  Double_t PDF     = 0.0;
  Double_t g1     = 0.0;
  Double_t g2     = 0.0;
  Double_t g3     = 0.0;

  // Calculate the exponents of the Gaussians
  Double_t arg1 = (par[2] != 0.0) ? (x[0] - par[1])/(par[2]) : 0.0;
  Double_t arg2 = (par[4] != 0.0) ? (x[0] - par[3])/(par[4]) : 0.0;
  Double_t arg3 = (par[6] != 0.0) ? (x[0] - par[5])/(par[6]) : 0.0;

  // add each Gaussian contribution to the PDF
  g1 = exp(-0.5*arg1*arg1)/(par[2]*sqrt(2.0*TMath::Pi()));
  g2 = exp(-0.5*arg2*arg2)/(par[4]*sqrt(2.0*TMath::Pi()));
  g3 = exp(-0.5*arg3*arg3)/(par[6]*sqrt(2.0*TMath::Pi()));
  PDF = par[0]*(par[7]*g1 + par[8]*g2 + (1-par[7]-par[8])*g3);

  return PDF;
}
```

It's a good idea to keep track of the parameter Names when writing the fit function.

Calculate the value of the function G3

Return the calculated value

Example: Triple Gaussian

- Set up the TF1 object for fitting

```
TF1 * myFitFunc = new TF1("fitFunc", fitFunc, min_range, max_range, npar);
```

Fit function name to use when you want to fit with this function.

The name of the function to call to evaluate G3

```
Double_t fitFunc(Double_t * x, Double_t * par)
```

Variables to fit: i.e. x

Array of parameters

The fit range in x

The number of fit parameters (9 for this example)

Example: Triple Gaussian

- Set up the TF1 object for fitting

```
TF1 * myFitFunc = new TF1("fitFunc", fitFunc, min_range, max_range, npar);
```

- Define the parameters, and give initial values

```
myFitFunc->SetParName(0, "norm");
myFitFunc->SetParName(1, "mu1");
myFitFunc->SetParName(2, "sigma1");
myFitFunc->SetParName(3, "mu2");
myFitFunc->SetParName(4, "sigma2");
myFitFunc->SetParName(5, "mu3");
myFitFunc->SetParName(6, "sigma3");
myFitFunc->SetParName(7, "frac1");
myFitFunc->SetParName(8, "frac2");
```

parameter number

parameter name

```
myFitFunc->SetParameters(1500.0, 5, 0.5, 6, 1.5, 5, 5, 0.4, 0.3);
```

Ordered list if initial parameter values

Example: Triple Gaussian

- Set parameter limits

```
myFitFunc->SetParLimits(0, 0.0, 2500.0);
myFitFunc->SetParLimits(1, 0.01, 10.0);
myFitFunc->SetParLimits(2, 0.0,  5);
myFitFunc->SetParLimits(3, 0.01, 10.0);
myFitFunc->SetParLimits(4, 0.0,  5);
myFitFunc->SetParLimits(5, 0.01, 10.0);
myFitFunc->SetParLimits(6, 0.0,  10.);
myFitFunc->SetParLimits(7, 0.0,  1.0);
myFitFunc->SetParLimits(8, 0.0,  1.0);
```

parameter number

parameter limits (min, max)

- Fix any required parameters before fitting

```
myFitFunc->FixParameter(5, 5.0);
myFitFunc->FixParameter(6, 5.0);
```

parameter number

parameter value

Example: Triple Gaussian

- Generate a dataset & fit
  - A quick way to do this is to get ROOT to generate a data set by filling a histogram

```
TH1F data_hist("data_hist", "", 50, min_range, max_range);
data_hist.FillRandom(fitFunc, 1500);
```

Your fit function

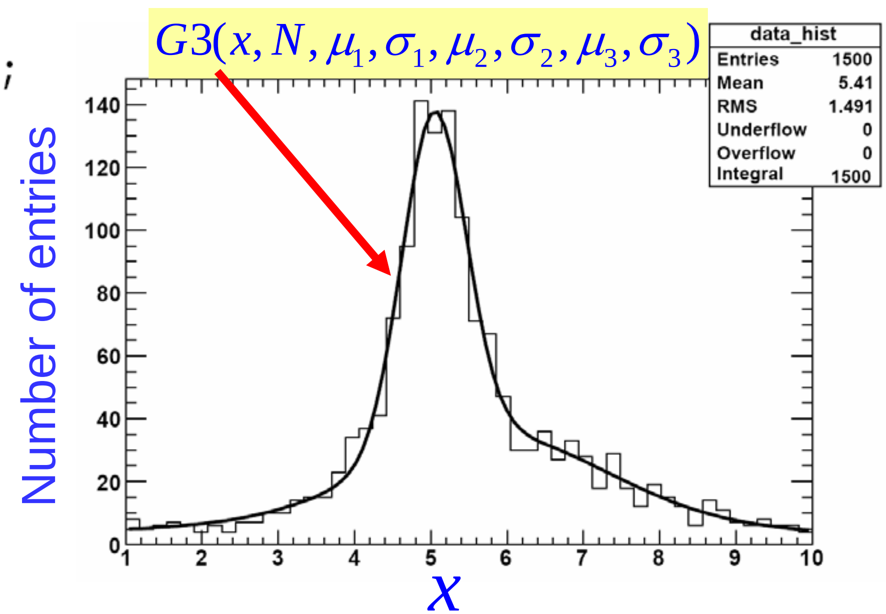
The number of entries to generate

  - Then you can do a likelihood fit to the data

```
data_hist.Fit("fitFunc", "L");
```

The name of your fit function

Fit options



  - You can also fit to a variable in a tree using this function.

Example: Triple Gaussian

- The full list of possible fit options is:

```
W  Set all errors to 1
I  Use integral of function in bin instead of value at bin center
L  Use Loglikelihood method (default is chisquare method)
LL Use Loglikelihood method and bin contents are not integers)
U  Use a User specified fitting algorithm (via SetFCN)
Q  Quiet mode (minimum printing)
V  Verbose mode (default is between Q and V)
E  Perform better Errors estimation using Minos technique
B  Use this option when you want to fix one or more parameters
   and the fitting function is like "gaus","expo","poln","landau".
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. By default the fitted function
   is drawn unless the option"N" above is specified.
+  Add this new fitted function to the list of fitted functions
   (by default, any previous function is deleted)
```

Exercises: Triple Gaussian

- Using the f_triple_gaussian.cc macro
  1. Run the macro and note the output parameter values.
  2. Change the fit option to be “” to perform a χ^2 fit. What happens to the parameters and convergence status, and why do you think this happens?
  3. Do the same using MINOS (E) and the Improve solution (M) options & note differences. What option gives the best fit χ^2/degree of freedom?
     - Hint:

χ2 can be calculated using `myFitFunc->GetChisquare();`
ν `(#degrees of freedom)` `myFitFunc->GetNDF();`
P(χ2, ν) can be calculated using `myFitFunc->GetProb();`

Exercises: Triple Gaussian

- Using the f_triple_gaussian.cc macro
  1. Run the macro and note the output parameter values.
  2. Change the fit option to be "" to perform a χ^2 fit. What happens to the parameters and convergence status, and why do you think this happens?
  3. Do the same using MINOS (E) and the Improve solution (M) options & note differences. What option gives the best fit χ^2/degree of freedom?
     - Hint:

χ2 can be calculated using	`myFitFunc->GetChisquare();`
ν `(#degrees of freedom)`	`myFitFunc->GetNDF();`
P(χ2, ν) can be calculated using	`myFitFunc->GetProb();`

  4. You can print the covariance and correlation matrices by adding the following line at the end of the macro:

     ```
     gMinuit->mnmatu(1);
     ```

     Make this alteration and re-run the fit. What does the correlation matrix tell you about the fit parameters?
  5. Is there a better way to write the PDF?

Using TMinuit

See Chapter 5 of the ROOT 5.12 User Guide for more details

- Many logical steps involved:
  - Write the function to minimise.
  - Define the global data to fit to.
  - Set up a TMinuit object for fitting.
    - Need to specify parameters (and ranges) to use in the fit.
      - Are parameters allowed to vary in the fit?
      - Are parameters fixed to a constant value in the fit?
    - Need to specify what type of fit to do.
      - MIGRAD?
      - HESSE?
      - MINOS?
  - Recall the results (and test statistic at the fit minimum) for interpretation. [can be read from file or screen]

Example: Unitarity Triangle

- Define the problem to solve.
  - Write down the equations to solve and calculate the function to minimise (e.g. a χ^2):

$$\chi^2 = \sum_{observables} \frac{(x_i - \hat{x})^2}{\sigma_i^2}$$

Measured value

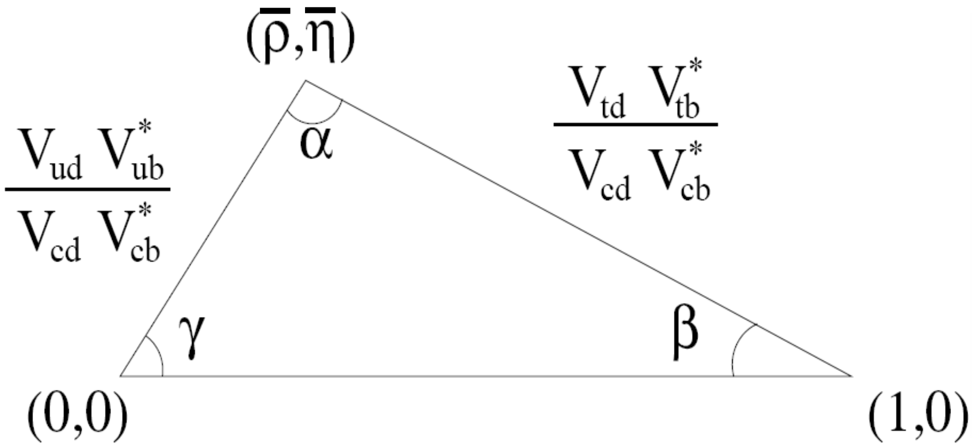
Calculated value (e.g. from model)

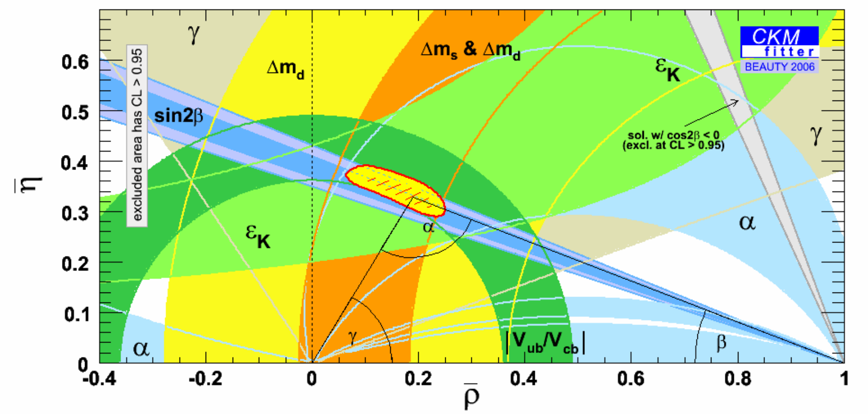
Error on measurement

  - Then code into a function with a specific prototype.
  - Measured values should be declared with global scope (i.e. outside any function & before they are used).

Example: Unitarity Triangle

- Two independent variables define the shape of the Unitarity Triangle: $\bar{\rho}, \bar{\eta}$





- And the B-factories have produced measurements of the angles (and sides of this triangle). e.g.

Measurements

$$\alpha = (92 \pm 7)^\circ$$

$$\beta = (21.2 \pm 1)^\circ$$

Relation to apex co-ordinate

$$\alpha = \mathrm{atan}\left(\frac{\bar{\rho}}{\bar{\eta}}\right) + \mathrm{atan}\left(\frac{1-\bar{\rho}}{\bar{\eta}}\right)$$

$$\beta = \mathrm{atan}\left(\frac{\bar{\eta}}{1-\bar{\rho}}\right)$$

Example: Unitarity Triangle

- So we can write the χ^2 to calculate for a given value of the apex co-ordinates:

$$\chi^2 = \frac{\left(\alpha_{meas} - \alpha(\bar{\rho},\bar{\eta})\right)^2}{\sigma^2_{\alpha_{meas}}} + \frac{\left(\beta_{meas} - \beta(\bar{\rho},\bar{\eta})\right)^2}{\sigma^2_{\beta_{meas}}}$$

- Need to write a function that calculates the χ^2 for a given set of values of the parameters $\bar{\rho}$ and $\bar{\eta}$.
- This fit is coded up in the file yeti07/TMinuit/FitUT.cc.
- To compile & run the program:

```
cd yeti07/TMinuit/
gmake
./FitUT
```

Example: Unitarity Triangle

- Declare the global data: measurements of β and α (with their uncertainties)

```
double deg2rad = M_PI/180.0;
double  beta_meas = 21.2*deg2rad;
double dbeta_meas = 1.0*deg2rad;

double  alpha_meas = 95.8*deg2rad;
double dalpha_meas = 7.0*deg2rad;
```

An array of the parameters defined in the fit $(\bar{\rho}, \bar{\eta})$

- Write the function to minimise

```
extern void chi2(Int_t &npar, Double_t *gin, Double_t &f, Double_t *par, Int_t iflag)
{
  double thechi2 = 0.0;

  // calculate the alpha chi2 contribution for a given set of rhobar and etabar
  double alpha_arg1(0), alpha_arg2(0);
   if(par[1] != 0.0) { alpha_arg1 = par[0]/par[1];
                       alpha_arg2 = (1-par[0])/par[1];}
   double alpha_chi2 = (alpha_meas  - ( atan(alpha_arg1) + atan(alpha_arg2) ))/dalpha_meas;

  double beta_arg = 0.0;
  if( (1-par[0]) != 0.0) beta_arg = par[1]/(1-par[0]);
  double beta_chi2 = (beta_meas - atan(beta_arg))/dbeta_meas;

  // add the chi2 contributions together and set the return value
  thechi2 += alpha_chi2*alpha_chi2;
  thechi2 += beta_chi2*beta_chi2;

  f= thechi2;
}
```

Calculate the χ^2

Equate the variable f with the value of the χ^2 before returning from the function

Example: Unitarity Triangle

- Set up a TMinuit object in the main function

```
TMinuit *gMinuit = new TMinuit(2);
```

Initialise Minuit with a maximum of 2 parameters to minimise

```
// set the function to minimise with minuit.
gMinuit->SetFCN(chi2);
```

Set the function to minimise

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

Interprets command

Set the ‘1σ’ tolerance for the change in FCN
That determines when a function has been minimised

Example: Unitarity Triangle

- Set up a TMinuit object in the main function

```
TMinuit *gMinuit = new TMinuit(2);
```

Initialise Minuit with a maximum of 2 parameters to minimise

```
// set the function to minimise with minuit.
gMinuit->SetFCN(chi2);
```

Set the function to minimise

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

Interprets command

Set the '1σ' tolerance for the change in FCN
That determines when a function has been minimised

```
gMinuit->mnparm(0, "rhobar",  0.2,     .1, 0.0,     0.5, iflag);
gMinuit->mnparm(1, "etabar",  0.3,     .1, 0.0,     0.5, iflag);
```

Set the parameters used in the fit

mnparam(Int_t iPar, TString name, Double_t startval, Double_t stepval, Double_t min, Double_t max, Int_t &errFlag)

- iPar: Parameter number (start counting from zero)
- name: Parameter name
- startval: Start value for the parameter
- stepval: Step size for parameter
- min, max: Allowed limits
- errFlag: OK if zero

Example: Unitarity Triangle

- Set up a TMinuit object in the main function

```
TMinuit *gMinuit = new TMinuit(2);
```

Initialise Minuit with a maximum of 2 parameters to minimise

```
// set the function to minimise with minuit.
gMinuit->SetFCN(chi2);
```

Set the function to minimise

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

Interprets command

Set the ‘1σ’ tolerance for the change in FCN
That determines when a function has been minimised

```
gMinuit->mnparm(0, "rhobar",  0.2,     .1, 0.0,    0.5, iflag);
gMinuit->mnparm(1, "etabar",  0.3,     .1, 0.0,    0.5, iflag);
```

Set the parameters used in the fit

```
gMinuit->mnexcm("CALL FCN", arglist, 1, iflag);
```

Call the user defined function, to calculate the value FCN, and print the result out to the screen.

Example: Unitarity Triangle

- Set up a TMinuit object in the main function

```
TMinuit *gMinuit = new TMinuit(2);
```

Initialise Minuit with a maximum of 2 parameters to minimise

```
// set the function to minimise with minuit.
gMinuit->SetFCN(chi2);
```

Set the function to minimise

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

Interprets command

Set the '1σ' tolerance for the change in FCN
That determines when a function has been minimised

```
gMinuit->mnparm(0, "rhobar",  0.2,     .1, 0.0,    0.5, iflag);
gMinuit->mnparm(1, "etabar",  0.3,     .1, 0.0,    0.5, iflag);
```

Set the parameters used in the fit

```
gMinuit->mnexcm("CALL FCN", arglist, 1, iflag);
```

```
gMinuit->mnexcm("MIGRAD", arglist,2, iflag);
```

Run the minimisation Using MIGRAD

MIGRAD vs. HESSE vs. MINOS

- MIGRAD
  - Performs a local function minimization using a modified version of the Davidson-Fletcher-Powell switching method described in Fletcher, Comp.J. **13**,317 (1970).
- HESSE
  - Calculates the full second-derivative matrix of the user function FCN using a finite difference method. This is often used to improve upon the result obtained by MIGRAD.
- MINOS
  - Performs a MINOS error analysis. This can result in different errors than obtained using MIGRAD or HESSE methods. In general one obtains asymmetry errors from MINOS.
- You should use (at least) HESSE after MIGRAD to obtain reliable errors for a given fit result. MINOS will give the best estimate of the errors of a given set of parameters.

Exercises: Unitarity Triangle

- Using the FitUT program
  1. Run the program and note the output values of the apex coordinates and errors.
  2. Run with the –hesse and –minuit options and note how the fitted values and errors change, and what the different options are doing.
  3. What is the significance of the correlation between the two fit parameters?
  4. What do you do when the fit fails to converge?

Exercises: Unitarity Triangle

- Using the FitUT program
  1. Run the program and note the output values of the apex coordinates and errors.
  2. Run with the –hesse and –minuit options and note how the fitted values and errors change, and what the different options are doing.
  3. What is the significance of the correlation between the two fit parameters?
  4. What do you do when the fit fails to converge?
- Try extending the program to include $\gamma=(82\pm20)^\circ$.

Exercises: Unitarity Triangle

- Using the FitUT program
  1. Run the program and note the output values of the apex coordinates and errors.
  2. Run with the –hesse and –minuit options and note how the fitted values and errors change, and what the different options are doing.
  3. What is the significance of the correlation between the two fit parameters?
  4. What do you do when the fit fails to converge?
- Try extending the program to include $\gamma=(82\pm20)^\circ$.
  - So add the χ^2 term:

$$\chi^2 = \frac{\left(\gamma_{meas} - \gamma(\bar{\rho},\bar{\eta})\right)^2}{\sigma^2_{\gamma_{meas}}}$$

where $\gamma = \mathrm{atan}\left(\frac{\bar{\eta}}{\bar{\rho}}\right)$

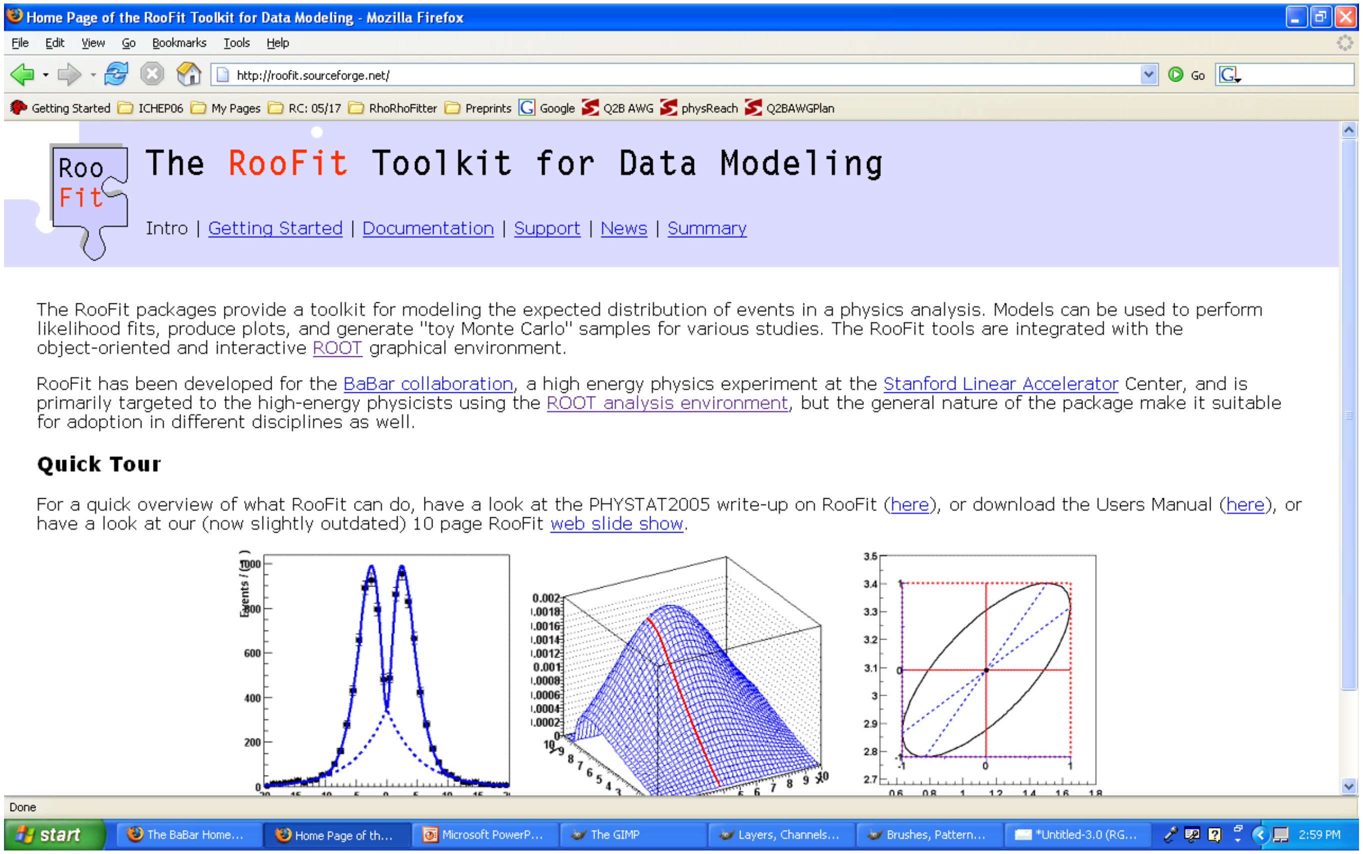
Possible commands for mnexcm

void mnexcm(const char *command, Double_t *plist, Int_t llist, Int_t &ierflg)

	mnexcm command	equivalent TMinuit function call
MIGrad	Call MIGRAD for minimisation	Migrad()
HESse	Call HESSE for minimisation	
MINOs	Call MINOS for minimisation	
MINImize	Call MIGRAD for minimisation	Migrad()
SEEk	Use a Monte Carlo method to minimise with a 3σ hypercube	mnseek()
SIMplex	Use a simplex method for minimisation	mnsimp()
SET xxx	Set the values of configurable options for Minuit	mnset()
SHOw xxx	Display covariance matrix, value of FCN etc.	mnset()
FIX	Fix the specified parameter in the fit	FixParameter(Int_t iPar)
REStore	Restore fixed parameters to variable status	
RELease	Allow the specified parameter to vary in the fit	Release(Int_t iPar)
SCAn	Scan the value of the user function for a parameter	mnscan()
CONtour	Trace contour lines between parameters	Contour(Int_t npoints=10, Int+t pa1=0, Int_t pa2=1)
SAVe	Write output to file	mnsave()
TOP of pag	Set output to top of file	
IMProve	Given a fit minimum, search for additional local minima	mnimpr()
CALI fcn	Call the specified user function	Eval(Int_t npar,Double_t *grad,Double_t &val,Double_t&fval,Double_t *par,Int_t flag)
STAndard	Do nothing	
END	Signals the end of a fit	
EXIt	Calculate final parameters and exit from minuit	
RETurn	Return from Minuit to the calling function	
CLEar	Reset all parameters and values to undefined	mncler()
HELP	Access the Minuit help interface	mnhelp(), mnhelp(TString command)
MNContour	Calculate the Nσ contour between two parameters	Contour(Int_t npoints=10, Int+t pa1=0, Int_t pa2=1)
STOp	Same as EXIT	
JUMp	Jump to a new point to continue minimisation	
COVARIANCE	Obsolete	
PRINTOUT	Obsolete, please use SET PRInt	
GRADIENT	Obsolete, please use SET GRAdient	
MATOUT	Obsolete, please use SHOW COVar	
ERROR DEF	Obsolete, please use SET ERRordef	
LIMITS	Obsolete, please use SET LIMits	
PUNCH	Obsolete please use SAVe	

RooFit

Synopsis: A fitting package to facilitate complex likelihood fits within the HEP community. This is an extremely flexible and an extendible set of tools.



More information is available at http://roofit.sourceforge.net

See the sourceforge website for examples, tutorials and a user guide.

Adrian Bevan (a.j.bevan@qmul.ac.uk)

Basic Types

- RooRealVar
  - Used for discriminating variables (x,y,…).
  - Used for shape parameters (Gaussian width and mean etc.).

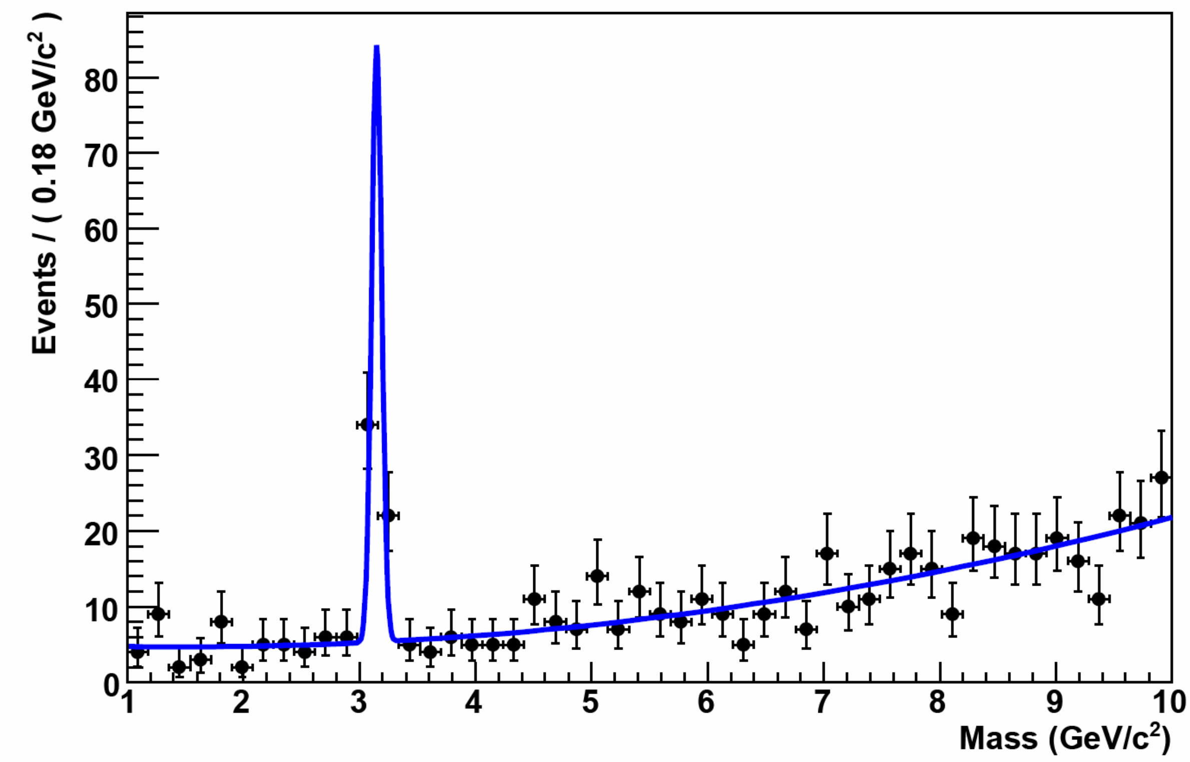
- Probability Density Function types (e.g. RooPolynomial and RooGaussian)
  - See the RooFit web site for examples and a user guide listing other available PDF types.
- RooDataSet
  - Data container that stores information on an entry by entry basis.
  - Can get a TTree from a RooDataSet.

Example: Peak finding

- Fit for a Gaussian peak of known mass on a polynomial background.

Need the following:

- Signal
  - Shape (PDF)
  - Yield
- Background
  - Shape (PDF)
  - Yield
- Data or Monte Carlo simulated data (to fit and plot result of)



The ROOT macro for this exercise is yeti07/fitting/rf_fit_for_peak.cc

Peak finding: Signal PDF

- Need to define the discriminating variable:

```
RooRealVar mass("mass", "Mass", 1, 10, "GeV/c^{2}");
```

Allowed range for the parameter mass

- Then define the signal PDF

Parameter fixed to 3.15

```
RooRealVar sig_mu("sig_mu", "mean", 3.15);
RooRealVar sig_sigma("sig_sigma", "width", 0.05, 0.0, 0.1);
```

Parameter set to 0.05

range

```
RooGaussian sig_pdf("sig_pdf", "signal pdf", mass, sig_mu, sig_sigma);
```

$$G(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

mass → x, sig_mu → μ, sig_sigma → σ

Peak finding: Background PDF

- Use the same discriminating variable as the signal

- Define the background PDF

```
RooRealVar bg_pol1("bg_pol1", "linear coefficient",  0.01, -10, 10.);
RooRealVar bg_pol2("bg_pol2", "quadratic coefficient", 0.05, -10, 10);
```

$$P = ax + bx^2 + C$$

```
RooArgList bgCoefList;
bgCoefList.add(bg_pol1);
bgCoefList.add(bg_pol2);
```

C is determined from the automatic PDF normalisation in RooFit

```
RooPolynomial bg_pdf("bg_pdf", "background pdf", mass, bgCoefList);
```

- RooFit defines a polynomial of order *n* (*Pn*) as

$$Pn = \sum_{i=1}^{n} a_i x^i + C$$

Peak finding: Total PDF

- Need to define signal and background yields

```
RooRealVar signalYield("signalYield", "", 50.0, -10.0, 150.0);
RooRealVar backgroundYield("backgroundYield", "", 500.0, 0.0, 15000.0);
```

- And make the total PDF that corresponds to the extended likelihood that we want to fit

```
RooArgList pdfList;
pdfList.add(sig_pdf);
pdfList.add(bg_pdf);

RooArgList coefList;
coefList.add(signalYield);
coefList.add(backgroundYield);
```

- Need to respect the order that you Add the PDFs and the yields.
- If the number of entries added to the pdfList match the number for the coefList, RooFit will automatically set up an extended likelihood fit.

```
RooAddPdf TotalPdf("TotalPdf", "Extended PDF", pdfList, coefList);
```

$$TotalPdf = N_{signal}\left[\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-\mu)^2/2\sigma^2}\right] + N_{background}\left[ax + bx^2 + C\right]$$

Peak finding: Generate & fit a toy data sample

- Use *TotalPdf* to generate a data set

```
Int_t nToGen = (Int_t)( signalYield.getVal() + backgroundYield.getVal() );
RooDataSet * data = (RooDataSet*)TotalPdf.generate(RooArgSet(mass), nToGen);
```

The set of discriminating variables to generate

- Then fit…

```
RooFitResult * result = TotalPdf.fitTo(*data, "etr");
```

AVAILABLE FIT OPTIONS:

"m" = MIGRAD only (no MINOS)
"s" = Estimate step size with HESSE
"h" = Run HESSE after MIGRAD
"e" = Perform extended –lnL fit
"0" = Run MIGRAD with strategy MINUIT 0 (don't calculate correlation matrix – not valid if running HESSE or MINOS)
"q" = Switch off verbose mode
"l" = Save log file with values at each MINUIT step
"v" = Show change in parameters at each step
"t" = Time the fit
"r" = Save the fit output in a RooFitResult object.

Peak finding: Looking at the data and fit result

- The fit result obtained is printed to the screen:

```
**********
**   23 **MINOS        2500
**********
FCN=-1817.09 FROM MINOS     STATUS=SUCCESSFUL    511 CALLS         682 TOTAL
                     EDM=3.55819e-06    STRATEGY= 1       ERROR MATRIX ACCURATE
 EXT PARAMETER                            PARABOLIC          MINOS ERRORS
 NO.   NAME              VALUE              ERROR          NEGATIVE      POSITIVE
  1  backgroundYield   5.02392e+02   2.25756e+01  -2.22056e+01   2.29511e+01
  2  bg_pol1          -1.37144e-01   1.00305e-01  -8.06255e-02   1.34265e-01
  3  bg_pol2           4.53396e-02   7.60813e-03  -7.33347e-03   8.40973e-03
  4  sig_sigma         4.32667e-02   5.39951e-03  -5.07501e-03   5.83116e-03
  5  signalYield       4.75675e+01   7.39504e+00  -7.07024e+00   7.75175e+00
```

Fit worked and errors are sensible

- Get a RooPlot
  - Add the data to the plot

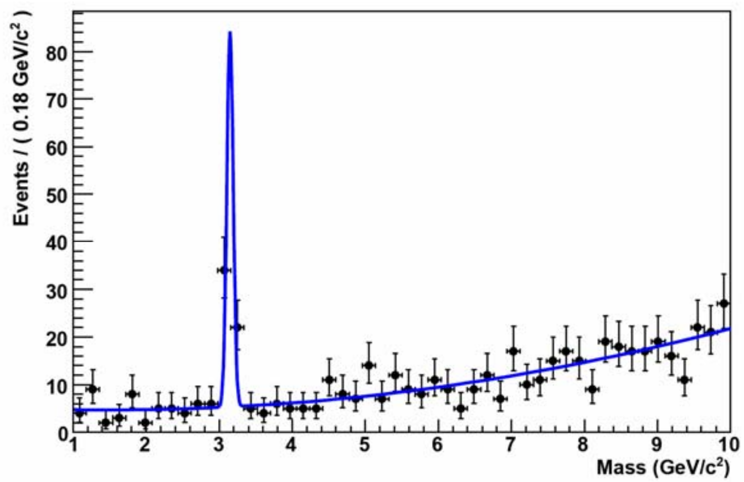
```
RooPlot * frame = mass.frame(50);
data->plotOn(frame);
```

  - Add the fitted PDF to the plot with correct normalisation

```
TotalPdf.plotOn(frame);
```

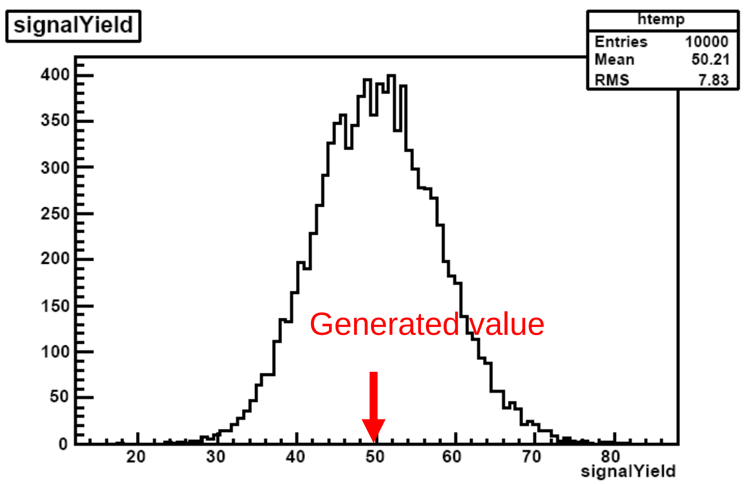
  - Display the result on-screen

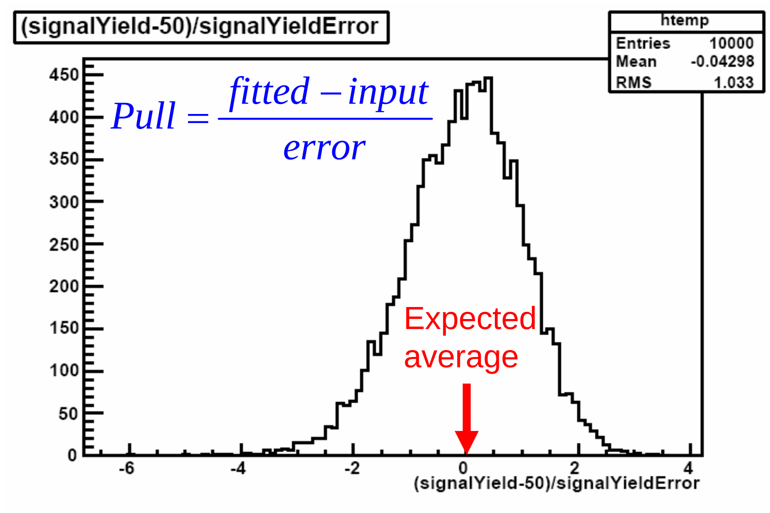
```
frame->Draw();
```



Peak finding: Validating the fit

- χ^2 and likelihood fits are intrinsically biased. So the fit needs to be validated before you can trust the result.
  - The first step in validating a fit is to run ensembles of toy MC simulations and compare the results with the fit to data.
  - Then check the pull distribution.

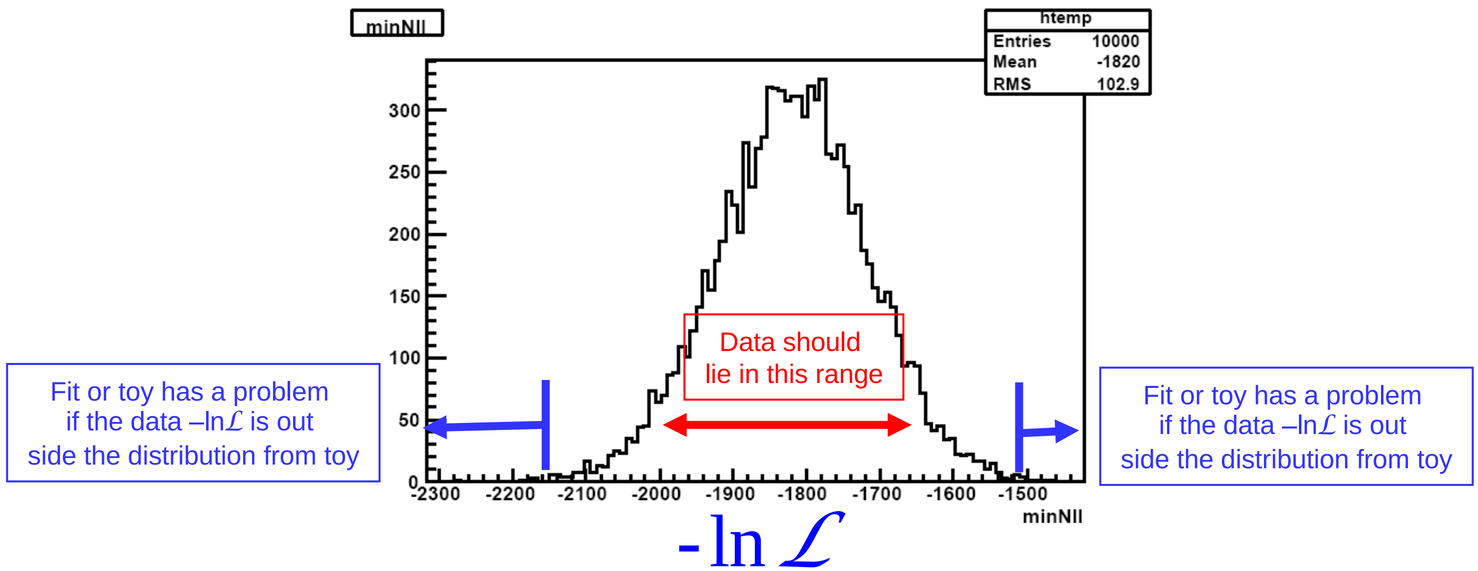




Peak finding: Validating the fit

- If the fit works then you can compare the value of $-\ln\mathcal{L}$ you obtain in data with that of the ensemble of toy MC experiments that you run.
  - $-\ln\mathcal{L}$ from data should be compatible with the toys or you have a problem!

$$\mathcal{L} = e^{-(N_{signal}+N_{background})}\prod_{i=1}^{n\ events} N_{signal}PDF_{signal} + N_{background}PDF_{background}$$



Exercises: Peak Finding

1. Run the macro yeti07/fitting/rf_fit_for_peak.cc and look at the resulting plot and fit result.
2. Change the signal yield from 50 events and see what happens.
3. Increase the background yield by a factor of 10, 20, 100 and see what happens to the error on the signal yield.

Exercises: Peak Finding

1. Run the macro yeti07/fitting/rf_fit_for_peak.cc and look at the resulting plot and fit result.
2. Change the signal yield from 50 events and see what happens.
3. Increase the background yield by a factor of 10, 20, 100 and see what happens to the error on the signal yield.
4. Run the macro yeti07/fitting/rf_toy_mc.cc
    1. What are the mean (μ) and width (σ) of the pull distribution when you fit a Gaussian to them?
    2. If you increase the number of toy MC studies run to 1000 toys what happens to μ and σ?

Summary

- You have seen
  - How to do simple fits using ROOT.
    - Fitting a user defined PDF to a histogram (or data from a TTree).
  - How to write a program that uses TMinuit to solve a problem.
    - Fitting for model parameters given the results of measurements from experiment.
  - How to write a fit using RooFit, and how to use ensembles of toy Monte Carlo simulated data to check that the fit is unbiased.
    - Hunting for a particle of known mass, sitting on a slowly varying background.
- More information on fitting using ROOT and RooFit is available on the web:

http://root.cern.ch
http://roofit.sourceforge.net