Fitting and Parameter Estimation in ROOT

ROOT Training at La Plata

G. Ganis, L. Moneta (CERN)
Outline

• Introduction to Fitting:
  – what is fitting,
  – how to fit a histogram in ROOT,
  – how to retrieve the fit result.
• Building complex fit functions in ROOT.
• Interface to Minimization.
• Common Fitting problems.
• Using the ROOT Fit GUI (Fit Panel).
What is Fitting?

- It is the process used to estimate parameters of an hypothetical distribution from the observed data distribution.

**Example**

Higgs search in CMS ($H \rightarrow \gamma\gamma$)

We fit for the expected number of Higgs events and for the Higgs mass.
What is Fitting (2)

• A histogram (or a graph) represents an estimate of an underlying distribution (or a function).
• The histogram or the graph can be used to infer the parameters describing the underlying distribution.
• Assume a relation between the observed variables y and x:
  \[ y = f( x | \theta ) \]
  • \( f( x | \theta ) \) is the fit (model) function
  • for an histogram y is the bin content
• Find the best estimate of the parameters \( \theta \) assuming the given function
Least Square ($\chi^2$) Fit

- Minimizes the deviations between the observed $y$ and the predicted function values:

- **Least square fit ($\chi^2$):**
  - minimize square deviation
  - weighted by the observed errors
  - $\sigma = \sqrt{N}$ for the histograms

\[
\chi^2 = \sum_i \frac{(Y_i - f(X_i, \theta))^2}{\sigma_i^2}
\]
Maximum Likelihood Fit

- Maximum Likelihood (ML) Fit:
  - The parameters are estimated by finding the maximum of the likelihood function (or minimum of the negative log-likelihood function).
  - Likelihood:
    \[ L(x|\theta) = \prod_i P(x_i|\theta) \]
    - find best value \( \theta \): max of
      \[ \log L = \sum_i \log(f(x_i, \theta)) \]

- The least-square fit and the maximum likelihood fit are equivalent when the distribution of observed events in each bin is normal.
  - \( f(x|\theta) \) is gaussian
The Likelihood for a histogram is obtained by assuming a Poisson distribution in every bin:

\[
\text{Poisson}( n_{\text{obs}} \mid n_{\text{exp}} )
\]

- \( n_{\text{obs}} \) is the observed bin content.
- \( n_{\text{exp}} \) is the expected bin content, obtained from the fit model function (the underlying distribution of the histogram).
- \( n_{\text{exp}} = f(x_c \mid \theta) \), where \( x_c \) is the bin center, assuming a linear function within the bin. Otherwise it is obtained from the integral of the function in the bin.

- For large histogram statistics (large bin contents) bin distribution can be considered normal
  - equivalent to least square fit
- For low histogram statistics the ML method is the correct one!
How do we do fit in ROOT:

- Create first a parametric function object, `TF1`, which represents our model, *i.e.* the fit function.
- Set the initial values of the function parameters.
- Fit the data object (Histogram or Graph):
  - call the `Fit` method on the Histogram or Graphs passing the function object as parameter
    - various options are possible (see the `TH1::Fit` documentation)
      » e.g. select type of fit: least-square (default) or likelihood (option “L”)
    - the resulting fit function is then drawn on top of the Histogram or the Graph.
- Examine result:
  - get parameter values;
  - get parameter errors (e.g. their confidence level);
  - get parameter correlation;
  - get fit quality.
• Suppose we have this histogram
  
  - we want to estimate the mean and sigma of the underlying gaussian distribution.

Example histogram

```
number of observed events/bin

<table>
<thead>
<tr>
<th>variable x</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Entries: 1000
Mean: 0.01102
RMS: 1.007
```
Creating the Fit Function

- To create a parametric function object (a TF1):
  - we can use the available functions in ROOT library
    
    ```
    TF1 * f1 = new TF1("f1","[0]*TMath::Gaus(x,[1],[2])");
    ```
  
  - or we can use pre-defined functions defined in TFormula (see TFormula documentation for the list of them):
    
    ```
    TF1 * f1 = new TF1("f1","gaus");
    ```

  - using pre-defined functions we have the parameter name automatically set to meaningful values.
  - initial parameter values are estimated whenever possible.

- We will see later in general how to build a more complex function objects
  - e.g. by using other functions
• How to fit the histogram:
  – after creating the function one needs to set the initial value of the parameters
  – then we can call the `Fit` method of the histogram class

```
root [] TF1 * f1 = new TF1("f1","gaus");
root [] f1->SetParameters(1,0,1);
root [] h1->Fit(f1);
```

FCN=27.2252 FROM MIGRAD STATUS=CONVERGED 60 CALLS 61 TOTAL
EDM=1.12393e-07 STRATEGY=

<table>
<thead>
<tr>
<th>NO.</th>
<th>NAME</th>
<th>VALUE</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constant</td>
<td>7.98760e+01</td>
<td>3.22882e+00</td>
</tr>
<tr>
<td>2</td>
<td>Mean</td>
<td>-1.12183e-02</td>
<td>3.16223e-02</td>
</tr>
<tr>
<td>3</td>
<td>Sigma</td>
<td>9.73840e-01</td>
<td>2.44738e-02</td>
</tr>
</tbody>
</table>

For displaying the fit parameters:
```
gStyle->SetOptFit(1111);
```
Retrieving The Fit Result

• The main results from the fit are stored in the fit function, which is attached to the histogram; it can be saved in a file (except for customized C/C++ functions).

• The fit function can be retrieved using its name:
  
  ```
  TF1 * fitFunc = h1->GetFunction("f1");
  ```

• The parameter values using their indices (or their names):
  
  ```
  fitFunc->GetParameter(par_index);
  ```

• The parameter errors:
  
  ```
  fitFunc->GetParError(par_index);
  ```

• It is also possible to access the TFitResult class which has all information about the fit, if we use the fit option “S”:
  
  ```
  TFitResultPtr r = h1->Fit(f1,"S");
  r->Print();
  TMatrixDSym C = r->GetCorrelationMatrix();
  ```

  C++ Note: the TFitResult class is accessed by using operator-> of TFitResultPtr
Retrieving The Fit Result

• The main results from the fit are stored in the fit function, which is attached to the histogram; it can be saved in a file (except for customized C/C++ functions).

• The fit function can be retrieved using its name:
  
  ```
  TF1 * fitFunc = h1->GetFunction("f1");
  ```

• The parameter values using their indices (or their names):
  
  ```
  fitFunc->GetParameter(par_index);
  ```

• The parameter errors:
  
  ```
  fitFunc->GetParError(par_index);
  ```

• It is also possible to access the TFitResult class which has all information about the fit, if we use the fit option “S”:
  
  ```
  TFitResultPtr r = h1->Fit(f1,"S");
  r->Print();
  TMatrixDSym C = r->GetCorrelationMatrix();
  ```

C++ Note: the TFitResult class is accessed by using operator-> of TFitResultPtr
Some Fitting Options

- Fitting in a Range
  - \( \text{h1->Fit("gaus","","",-1.5,1.5);} \)

- Fitting more functions to an histogram (option “+”)
  - \( \text{h1->Fit("expo","+","",2.,4);} \)

- Quite / Verbose:
  - option “Q”/”V”.

- Likelihood fit:
  - option “L” for count histograms;
  - option “LW” in case of weighted counts.

- Return a fit result class:
  - option “S”

- Plotting options for the histogram can be passed as well:
  - \( \text{h1->Fit("gaus","L","E");} \)
Building More Complex Functions

• It is possible to write some complex formulae and pass as string in the constructor of TF1
  – but difficult and prone to error
• Better to write directly the functions in C/C++
• A parametric TF1 can be constructed from
  – a general free function with parameters:
    ```cpp
double function(double *x, double *p){
    return p[0]*TMath::Gaus(x[0],p[0],p[1]);
}
TF1 * f1 = new TF1("f1",function,xmin,xmax,npar);
```
  – any C++ object implementing `double operator()(double *x, double *p)`
    ```cpp
struct Function {
    double operator()(double *x, double *p){
        return p[0]*TMath::Gaus(x[0],p[0],p[1]);
    }
};
Function func;
TF1 * f1 = new TF1("f1",&func,xmin,xmax,npar,"Function");
```
Minimization

• The fit is done by minimizing the least-square or likelihood function.
• A direct solution exists only in case of linear fitting
  – it is done automatically in such cases (e.g fitting polynomials).
• Otherwise an iterative algorithm is used:
  – Minuit is the minimization algorithm used by default
    • ROOT provides two implementations: Minuit and Minuit2
    • other algorithms exists: Fumili, or minimizers based on GSL, genetic and simulated annealing algorithms
  – To change the minimizer:
    ```cpp
    ROOT::Math::MinimizerOptions::SetDefaultMinimizer("Minuit2");
    ```
  – Other commands are also available to control the minimization:
    ```cpp
    ROOT::Math::MinimizerOptions::SetDefaultTolerance(1.E-6);
    ```
Minimization Techniques

- Methods like Minuit based on gradient can get stuck easily in local minima.
- Stochastic methods like simulated annealing or genetic algorithms can help to find the global minimum.

Example: Fitting 2 peaks in a spectrum
Interface to Minimization

- A common interface for all ROOT Minimizer algorithms exists: class `ROOT::Math::Minimizer`
- All minimizers in ROOT (Minuit, Minuit2, Fumili, GSL minimizers, simulated annealing, genetic) implement this interface
- Using the ROOT plug-in manager it is possible to change the implementation at run-time
- The interface can be used for fitting user defined likelihood or least-square functions
  - see ROOT tutorial fit/NumericalMinimization.C on how to use this interface
Comments on Minimization

• Sometimes fit converges to a wrong solution
  – Often is the case of a local minimum which is not the global one.
    • This is often solved with better initial parameter values. A minimizer like Minuit is able to find only the local best minimum using the function gradient.
    • Otherwise one needs to use a genetic or simulated annealing minimizer (but it can be quite inefficient, e.g. many function calls).

• Sometimes fit does not converge
  – can happen because the Hessian matrix is not positive defined
    • e.g. there are no minimum in that region → wrong initial parameters;
  – numerical precision problems in the function evaluation
    • need to check and re-think on how to implement better the fit model function;
  – highly correlated parameters in the fit. In case of 100% correlation the point solution becomes a line (or an hyper-surface) in parameter space. The minimization problem is no longer well defined.

PARAMETER | CORRELATION COEFFICIENTS
---|---
NO. | GLOBAL | 1 | 2
1 | 0.99835 | 1.000 | 0.998
2 | 0.99835 | 0.998 | 1.000

Warning in <Fit>: Abnormal termination of minimization.

Signs of trouble...
Mitigating fit stability problems

• When using a polynomial parametrization:
  – \( a_0 + a_1 x + a_2 x^2 + a_3 x^3 \) nearly always results in strong correlations between the coefficients.
  • problems in fit stability and inability to find the right solution at high order

• This can be solved using a better polynomial parametrization:
  – e.g. Chebychev polynomials

\[
\begin{align*}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1.
\end{align*}
\]
Parameter Errors

• Errors returned by the fit are computed from the second derivatives of the likelihood function
  – Asymptotically the parameter estimates are normally distributed. The estimated correlation matrix is then:

\[
\hat{V}(\hat{\theta}) = \left( -\frac{\partial^2 \ln L(x; \theta)}{\partial^2 \theta} \right)_{\theta=\hat{\theta}}^{-1} = H^{-1}
\]

• A better approximation to estimate the confidence level in the parameter is to use directly the log-likelihood function and look at the difference from the minimum.
  – Method of Minuit/Minos (Fit option “E”)
    – obtain a confidence interval which is in general not symmetric around the best parameter estimate

```cpp
TFitResultPtr r = h1->Fit(f1,"E S");
r->LowerError(par_number);
```

The Fit Panel

- The fitting in ROOT using the FitPanel GUI
  - GUI for fitting all ROOT data objects (histogram, graphs, trees)
- Using the GUI we can:
  - select data object to fit
  - choose (or create) fit model function
  - set initial parameters
  - choose:
    - fit method (likelihood, chi2)
    - fit options (e.g. Minos errors)
    - drawing options
  - change the fit range
The Fit Panel provides also extra functionality:

- Control the minimization
- Advanced drawing tools

**Contour plot**

**Scan plot of minimization function**
Put in practice the concepts to which you were just exposed: read the instructions and solve the fitting exercises

**Exercise 15:** Gaussian fit of an histogram

**Exercise 16:** Fit a peak histogram

**Exercise 17:** Using the Fit Panel GUI
Summary

• We have learned:
  – the concept of fitting,
  – how to fit a histogram in ROOT.

• We have also learned:
  – how to generate random numbers and distributions which can be used to test and validate the fitting procedure.

• We will see also how fitting can be facilitate by using a tool like RooFit.
  – How it can be extended in a statistical framework (RooStats).