Using MINUIT (in ROOT)

Road Map

- Using MINUIT Directly
- Fitting a TGraph
- Fitting a TH1F
Minimizing a Simple Function

**First Step: Minimize a function that we can understand analytically**

- \( f(x,y) = x^4 + x^2 + xy + y^2 \)
  - Not quadratic
  - Correlations between \( x \) and \( y \)
  - Take this to be a \( \chi^2 \) so \( \Delta \chi^2 = 1 \) defines errors

**Minimize the “usual” way**

- Solution is at \( x = 0 \) and \( y = 0 \)

**Calculate the error matrix**

- Error is the can be estimated from second derivative
  - Error on one parameter if other one is fixed!
  - Correlation \( \sigma_{xy} = \frac{1}{2} \frac{\partial^2 f(x,y)}{\partial x \partial y} \bigg|_{0} = 1 \)
  - This means we can't ignore the correlations!
  - Also, \( f(x,y) \) isn't quadratic, so we should solve \( f(x,y) - f(0,0) = 1 \) to find errors

\[
\frac{\partial}{\partial x} f(x,y) = 4x^3 + 2x + y = 0 \\
\frac{\partial}{\partial y} f(x,y) = x + 2y = 0
\]
Do the Minimization with Minuit

- Create the working file
  - Use a named macro!

```c
// Main function in minimizerExample.C
void minimizerExample() {
}
```

- Define the function to be minimized.

```c
// The function I want to minimize. Remember C++ rules,
// about function order (put it before minimizerExample)
double myFunction(double x, double y) {
    return x*x*x*x + x*x + x*y + y*y;
}
```

- This function will need to be “wrapped” before MINUIT can use it.

- If you need to read data from a file
  - Read the data in your main program
  - Use a global variable to pass the data to your function
    - This is UGLY, but that's how root works
Set up for the Minimization

- Create the minimization object
  - Tell it about the function to be minimized.
  - Tell it about the function parameters

```c
// Main function in minimizerExample.C
void minimizerExample() {
    TFitter* minimizer = new TFitter(2);

    // MAKE IT QUIET!!
    {
        double p1 = -1;
        minimizer->ExecuteCommand("SET PRINTOUT", &p1, 1);
    }

    // Tell the minimizer about the function to be minimized
    minimizer->SetFCN(minuitFunction);

    // Define the parameters
    //   arg1 - parameter number
    //   arg2 - parameter name
    //   arg3 - first guess at parameter value
    //   arg4 - estimated distance to minimum
    //   arg5, arg6 - ignore for now
    minimizer->SetParameter(0, "X", 2, 1, 0, 0);
    minimizer->SetParameter(1, "Y", 2, 1, 0, 0);
}
```
Wrapping Your Function for MINUIT

MINUIT calls a function that's defined as

```c
void (*fcn)(int& nDim, double* gout, double& result, double par[], int flag)
```

- Not the most convenient definition
- You should ignore everything except “result”, and “par”

```c
// The function I want to minimize. Remember C++ rules,
// about function order (put it before minimizerExample)
double myFunction(double x, double y) {
  return x*x*x*x + x*x + x*y + y*y;
}

void minuitFunction(int& nDim, double* gout, double& result, double par[], int flg) {
  result = myFunction(par[0], par[1]);
}
```
Do the Minimization

- **MINUIT** provides several possible minimizer algorithms
  - Stick to “SIMPLEX”, and “MIGRAD”
    - **Simplex:**
      - Robustly finds the nearest local minimum, but precision isn't very high
    - **MIGRAD:**
      - Precisely find the minimum, but you need to be careful since it assumes the inputs are close to the minimum

```c
// Main function in minimizerExample.C
void minimizerExample() {
    ...
    // Run the simplex minimizer to get close to the minimum
    minimizer->ExecuteCommand("SIMPLEX",0,0);

    // Run the migrad minimizer (an extended Powell's method) to improve the fit.
    minimizer->ExecuteCommand("MIGRAD",0,0);

    // Get the best fit values
    double bestX = minimizer->GetParameter(0);
    double bestY = minimizer->GetParameter(1);

    // Get the function value at the best fit.
    double minimum = myFunc(bestX, bestY);
}
```
Find the Uncertainty

- MINUIT provides several ways to estimate the error
  - DO NOT BE TEMPTED!
  - They can give you the right answer if you know exactly what you are doing
    - For reference, I never assume I know exactly what I'm doing
    - You shouldn't either!
    - I'm recommending paranoia
  - Estimate your error using $\Delta \chi^2 = 1$
    - As you get more advanced, you should also check the coverage of your error estimate, but that is beyond the scope of this class
Find the Uncertainty

- Before you start, estimate the expected uncertainty
- You'll want to quote the uncertainty to about two significant figures.
- Do a scan of the function with one variable fixed and the other being minimized at each step to find $\Delta \chi^2 = 1$
- Pick step so you need about 500 steps to find uncertainty value

```cpp
// Main function in minimizerExample.C
void minimizerExample() {
    ...
    ...
    // Scan the X parameter to find it's uncertainty
double errX;
for (errX=0; errX<10; errX = errX + 0.001) {
    minimizer->SetParameter(0,"X",errX,0,0,0);
    minimizer->SetParameter(1,"Y",bestY,1,0,0);
    minimizer->ExecuteCommand("MIGRAD",0,0);
    double t = myFunc(minimizer->GetParameter(0),
                      minimizer->GetParameter(1));
    if (t-minimum > 1.0) break;
}
    ...
    ...
}
```

- Repeat for other parameters
Handling a Complicated Function

- If you can, plot the function and estimate the minimum by hand
  - Use estimated minimum as starting point for MINUIT
- Interesting functions are usually multi-dimensional
  - Can't plot in 10 dimensions

![Diagram of a function](image)

- Only solution I've ever heard
  - Try lots of random starting points.
  - If you have a “best guess” at the minimum, take “near-by” points.
  - Otherwise, test the entire function range.
Example: The Mess on the Previous Slide

Define the function to be minimized.

```cpp
// The function I want to minimize. Remember C++ rules, 
// about function order (put it before minimizerExample)
double myFunction(double x, double y) {
    return cos(x+y*y)*exp((-y*y+2*y)/10)*exp(-x*x/10);
}
```

The rest of the minimizer is the same, except

```cpp
double bestX = 10;
double bestY = 10;
double minimum = myFunction(bestX,bestY);
for (int i=0; i<20; ++i) {
    double tryX = gRandom->Uniform(-6,6);
    double tryY = gRandom->Uniform(-6,6);
    minimizer->SetParameter(0,"X",tryX,1,0,0);
    minimizer->SetParameter(1,"Y",tryY,1,0,0);
    minimizer->ExecuteCommand("SIMPLEX",0,0);
    minimizer->ExecuteCommand("MIGRAD",0,0);

    // Get the best fit values
    double newX = minimizer->GetParameter(0);
    double newY = minimizer->GetParameter(1);

    // Get the function value at the best fit.
    double newMin = myFunction(newX, newY);
    if (newMin < minimum) {
        minimum = newMin;
        bestX = newX;
        bestY = newY;
    }
}
```
\( \chi^2 \) Fits to Graphs

- ROOT provides methods to directly fit TGraphError objects.
  - For most applications, the automatic fits (and reported errors) are adequate
  - Remember, in data analysis **paranoia** is a virtue
    - Plot the data with the fit, does the result look “right”
    - Look at the resulting errors, do they look reasonable
    - Test the fit
      - Simulate from your model
      - Fit the simulated data with your model

```c++
// Macro to generate a plot of current vs voltage.
TGraphErrors* g = new TGraphErrors("ohmsLaw.dat");
g->SetTitle("Current vs Voltage");
g->GetHistogram()->SetXTitle("Voltage (V)");
g->GetHistogram()->SetYTitle("Current (mA)");

// Make sure the fit parameters are printed (a magic incantation)
gStyle->SetOptFit(1112);

// Define the function
TF1* f = new TF1("ohm","x/[0]+[1]"");
f->SetParNames("Resistance", "Offset");
f->SetParameters(1000,0.0);

// Fit the data to a line.
g->Fit(f);
```
\( \chi^2 \) Fits to Histograms

- ROOT provides methods to directly fit THistograms
  - Uses MINUIT internally

```c
// Define the fit parameters.
TF1* life = new TF1("life","[0]+[1]*exp(-x/abs([2]))");
life->SetParNames("Background","Normalization","Lifetime");
life->SetParameters(0,9000,2500);

// Make sure the fit results are plotted on the graph
gStyle->SetOptFit(1112);

// Fit the function to the histogram
//   arg1 – the fit function
//   arg2 – An option to make sure errors are calculated
//   arg3 – Ignore this option
//   arg4 – The low bound of the fit range
//   arg5 – the high bound of the fit range
h->Fit(life,"E","",1500,10000);

// Draw the result with error bars on the points
h->Draw("E");
```

- The “E” fit option shows extra error information on the terminal
  - Check that “parabolic” and “minos” errors are similar
Finally

Numeric function minimization is a very important tool in data analysis
- Used to determine the “best fit” parameters for a set of data
- Can help determine the confidence interval for each parameter
  - Intervals can only be trusted after looking at the “goodness of fit” (see next lecture)

Numeric minimization will always give you “an answer”. It's your job to decide if the minimizer has given you “the answer”
- Worry about:
  - local minima
  - long flat valleys
  - numeric instabilities
The End