Non-linear Least-Squares Fitting with MPFIT

Some documentation:
http://www.physics.wisc.edu/~craigm/idl/fitting.html

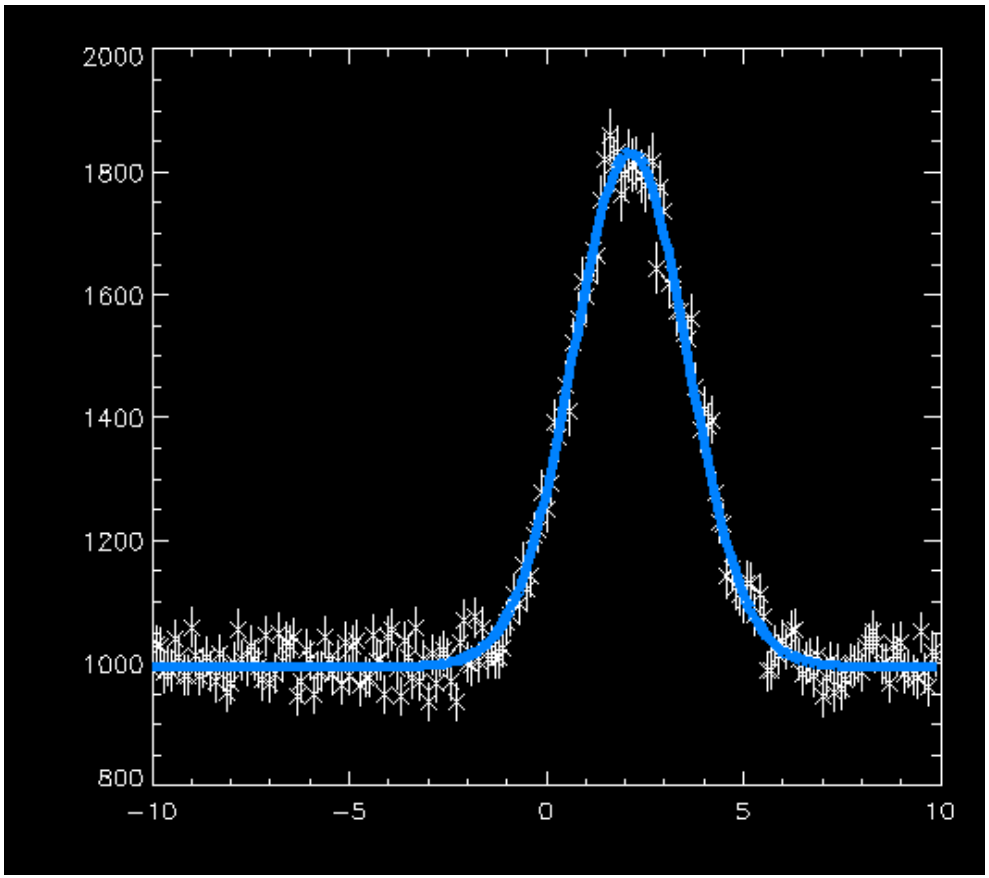
Tutorial:
http://www.physics.wisc.edu/~craigm/idl/mpfittut.html

Non-linear problems must be solved iteratively--unlike the problems you've used least-squares on before this lab.

Fitting a Gaussian function to data is a classic example.

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

Can't use linear techniques on this problem.



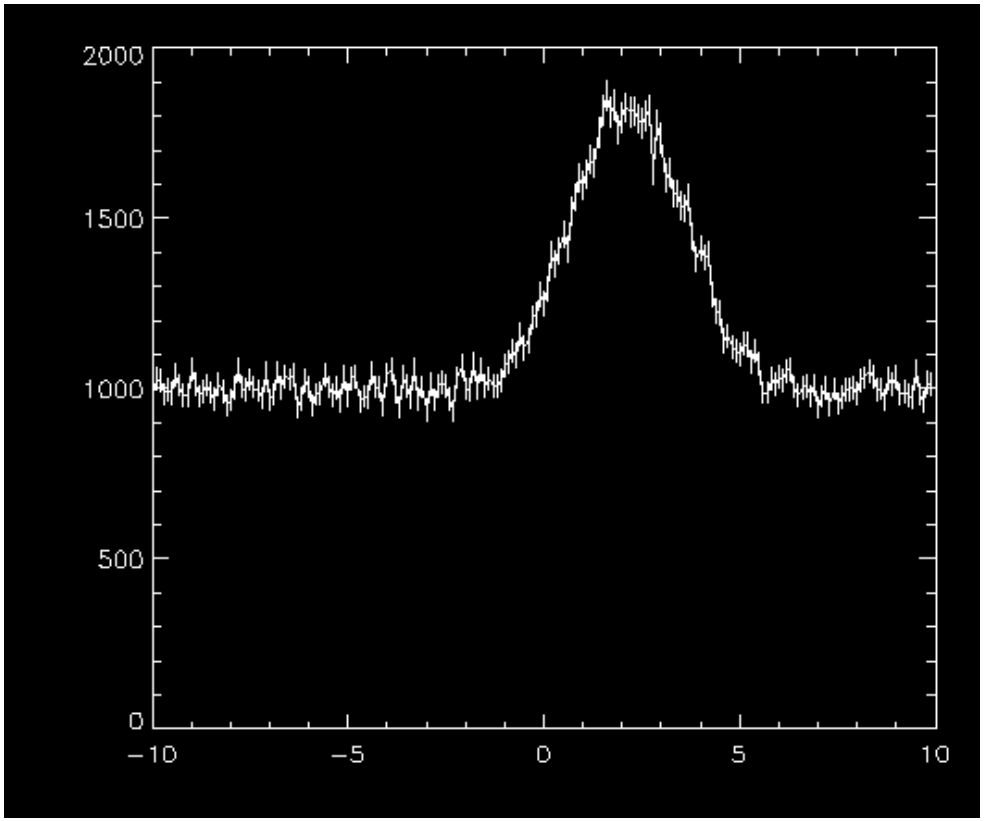
MPFIT example: Fitting a Gaussian

Step 1: Gather your data.

You have:

- X values
- Y values
- Y errors

(assume x values have no error in this example)



MPFIT example: Fitting a Gaussian

Step 2: Choose your model.

Ours will be a Gaussian function plus a constant offset.

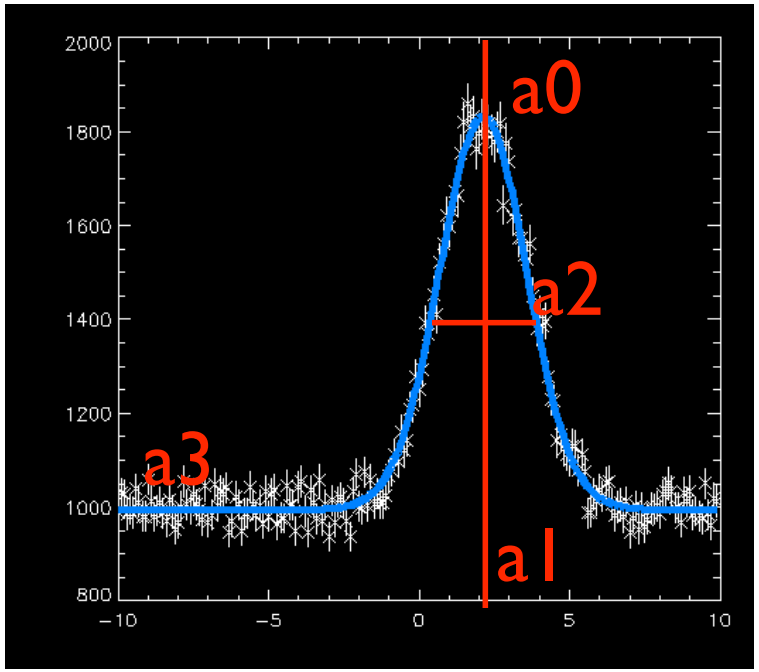
This has four parameters:

a0 = value at peak

a1 = central x value

a2 = sigma

a3 = offset



MPFIT example: Fitting a Gaussian

Step 3: Make a function that returns the difference between your data and your model, weighted by the errors on the data

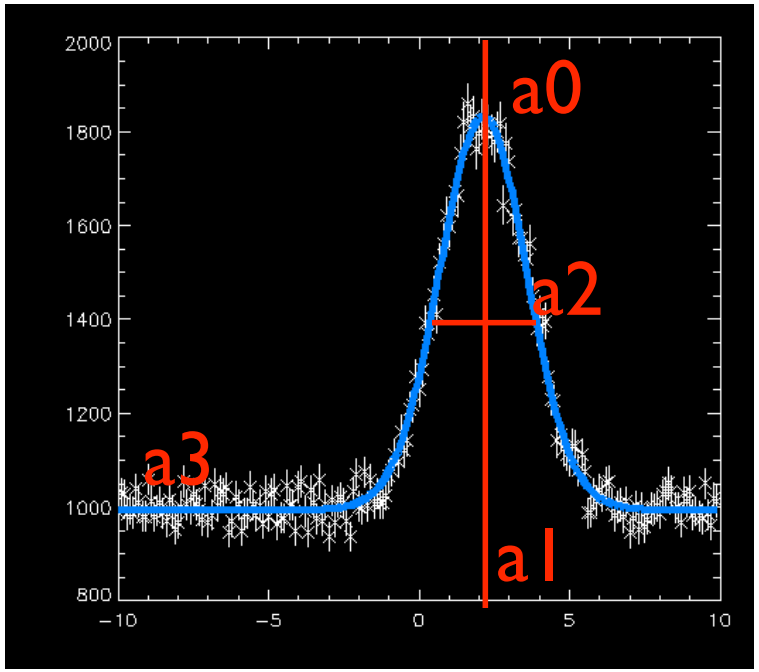
```
function mygaussian, p, x=x, y=y, err=err

  ; p = [a0, a1, a2, a3]

  model = p[3] + p[0] * exp(-0.5 * ((x-p[1])/p[2])^2.)

  return, (y-model)/err

end
```

MPFIT example: Fitting a Gaussian

Step 3: Pick an initial guess for the parameters p

```
guessp = [900., 2., 1., 1000.]
```

Try to make a good guess, some problems can be sensitive to the initial guess. (This can be automated).



MPFIT example: Fitting a Gaussian

Step 4: Use MPFIT

```
guessp = [950., 2.5, 1., 1000.]

fa = {X:x, Y:y, ERR:err}

p = mpfit('mygaussian', guessp, functargs=fa)
```

mpfit doesn't care about your data except in how it deviates from the model, so everything data-related is packaged up in "fa" the function arguments structure

```
IDL> p = mpfit('mygaussian', guessp, functargs=fa)
           Iter      1   CHI-SQUARE =       1462.3943
                          DOF = 196
              P(0) =              950.000
              P(1) =              2.50000
              P(2) =              1.00000
              P(3) =              1000.00
           Iter      2   CHI-SQUARE =       353.66009
                          DOF = 196
              P(0) =              749.300
              P(1) =              2.26459
              P(2) =              1.39788
              P(3) =              1007.73




            Iter      8   CHI-SQUARE =       187.48896
                          DOF = 196
              P(0) =              837.138
              P(1) =              2.15507
              P(2) =              1.44884
              P(3) =              997.619
```

Other things you can do with MPFIT

- adjust the weighting of each point
- limit parameters to be within a certain range
- hold some parameters constant
- adjust step sizes and convergence criteria
- lots of other stuff, very powerful!

Trying it out! (these programs are on the website)

- use fakedata.sav, mygaussian.pro and testgaussfit.pro to see how mpfit works in the case I've outlined in this presentation

- use makecircdata.pro, mycirc.pro and testcircfit.pro to see how mpfit works in the case where you have one independent variable (like time) and two dependent variables (like RA and Dec)