17 – Fitting Data to Non-Linear Models

Physics 281 – Class 17

Grant Wilson

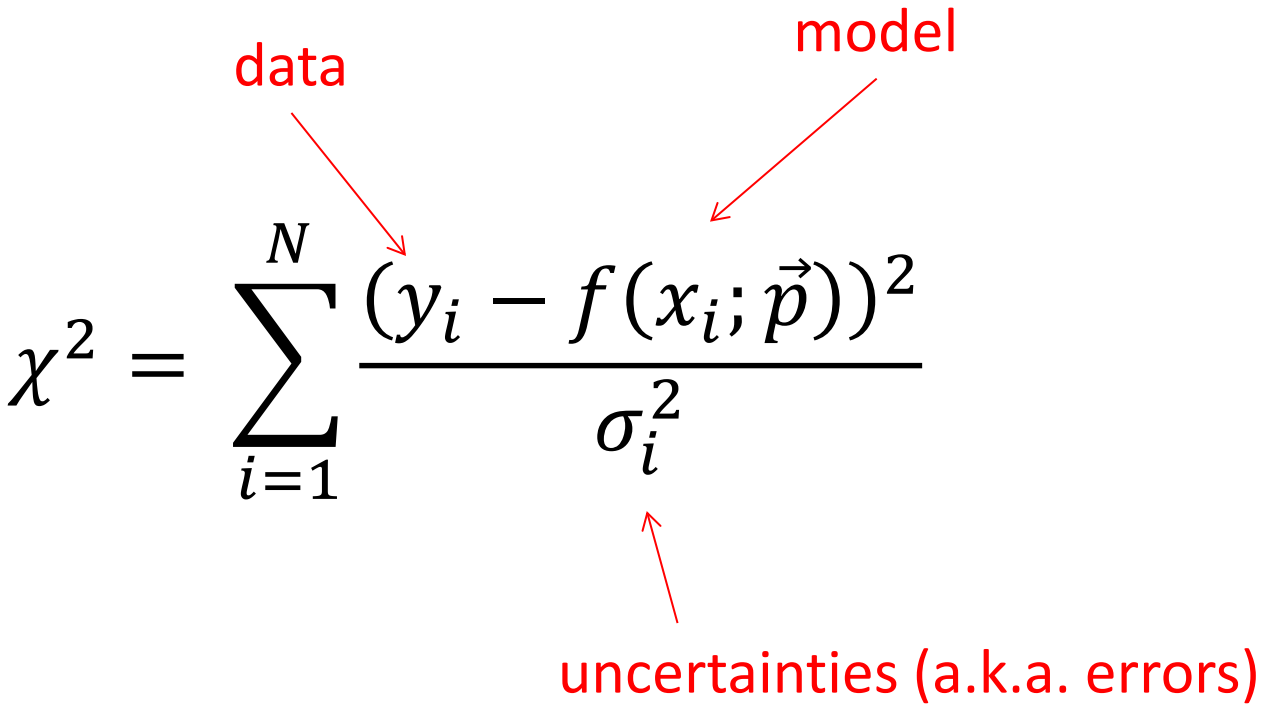
Remember

- All of our fitting is centered on the χ^2 statistic.

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i; \vec{p}))^2}{\sigma_i^2}$$

Remember

- All of our fitting is centered on the χ^2 statistic.



Previously ….

- We looked at an approach to fit models that were *linear* in their parameters. That is, models of the form

$$f(\vec{x};\vec{p}) = p_0 g(\vec{x}) + p_1 g_1(\vec{x}) + \cdots + p_{M-1} g_{M-1}(\vec{x})$$

- Our approach followed the recipe:
  1. write down expression for χ^2
  2. find equations for minimizing χ^2 for each of our "free parameters"
  3. solve the system of equations from step #2

- And this lead us to the matrix formalism you've all now owned.

The Approach for Models That Are Non-linear in Their Parameters is Different.

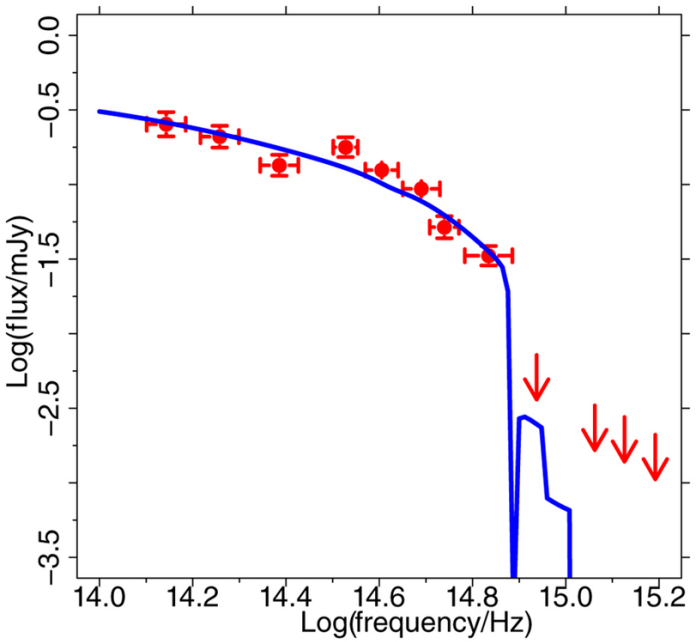
- We can still write down an expression for χ^2
- But there is no simple solution to minimizing this analytically.
- There is also no Matrix formalism

  – consider:

$$f(\vec{x};\vec{p}) = p_0 \cos(p_1 x)$$

However ….

- Our goal is the same, we want to minimize the mean square distance between our data and our model.



- Since we can write down an expression for χ^2, we should be able to find its minimum.

- Our goal is then: find the value of the parameters that minimizes $\chi^2(\vec{p})$

Take a moment to think about this

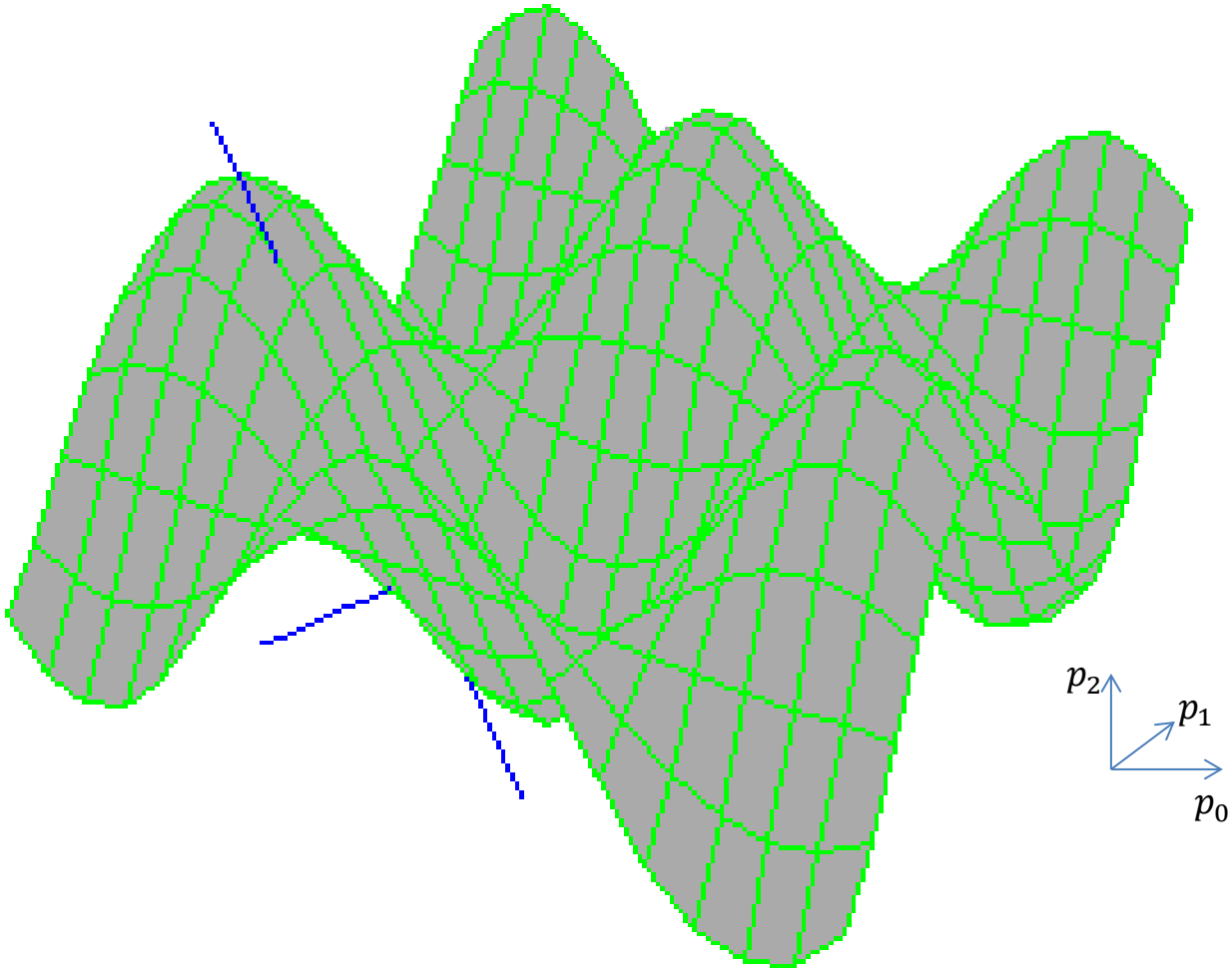
- What's so special about χ^2?

Take a moment to think about this

- What's so special about χ^2?

Nothing!

For a given data set and model $\{\vec{x}, \vec{y}, \vec{\sigma}, f(\vec{x}; \vec{p})\}$, $\chi^2(\vec{p})$ is just a function of the parameters $\vec{p}$.

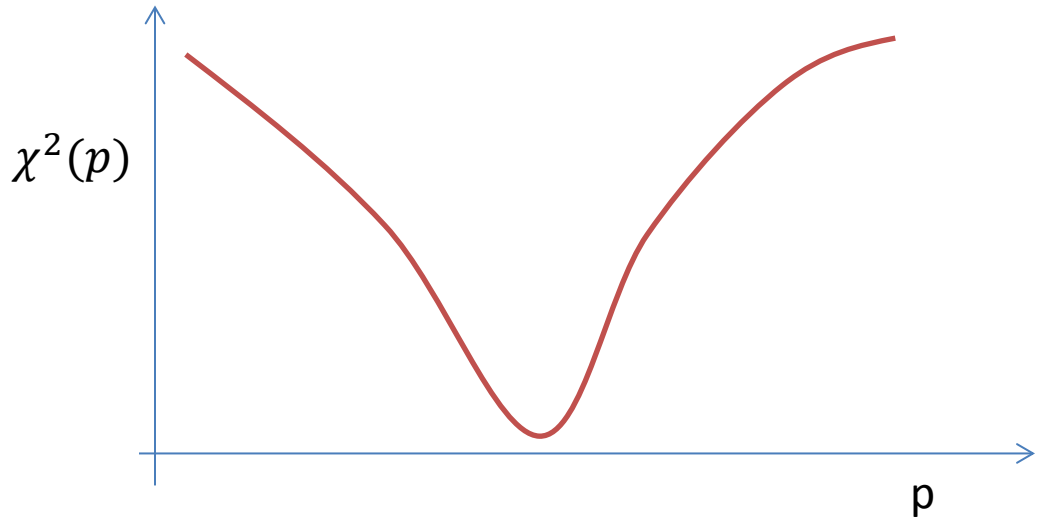

p_2
p_1
p_0

The New Question:

- How do you find the minimum of a function? (start in 1-dimension)

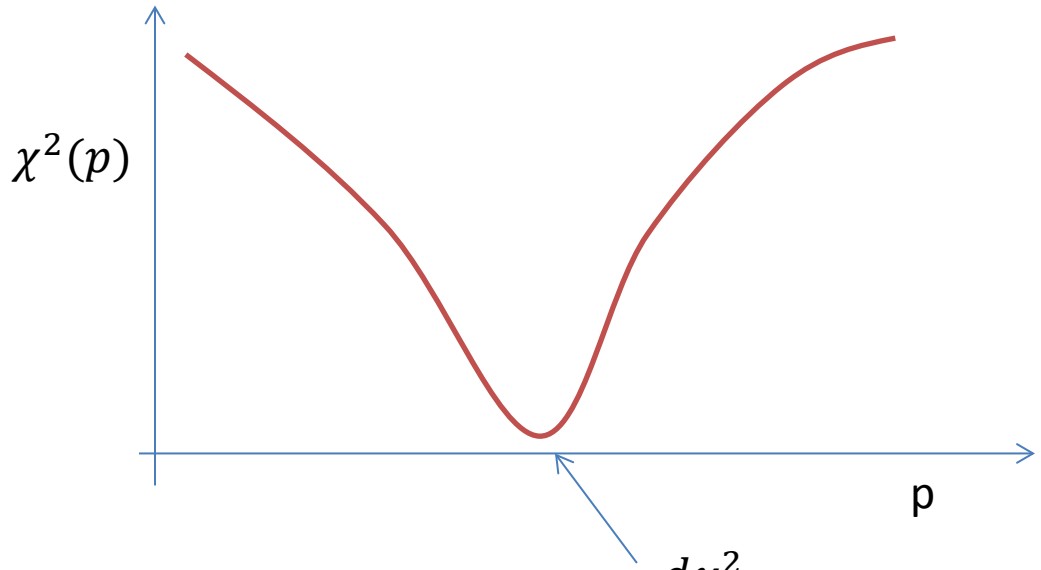
The New Question:

- How do you find the minimum of a function?



The New Question:

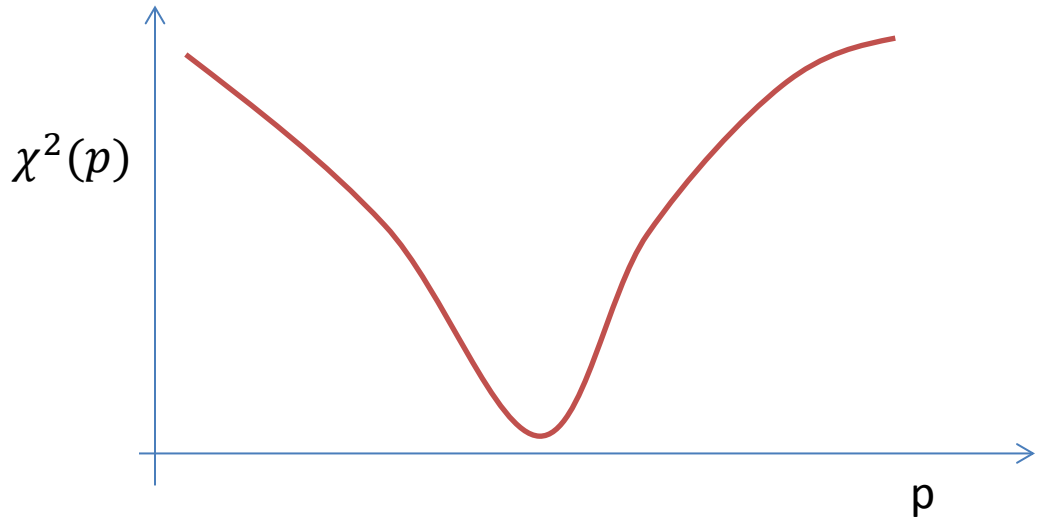
- How do you find the minimum of a function?



If you can take the derivative and solve it, great!

The New Question:

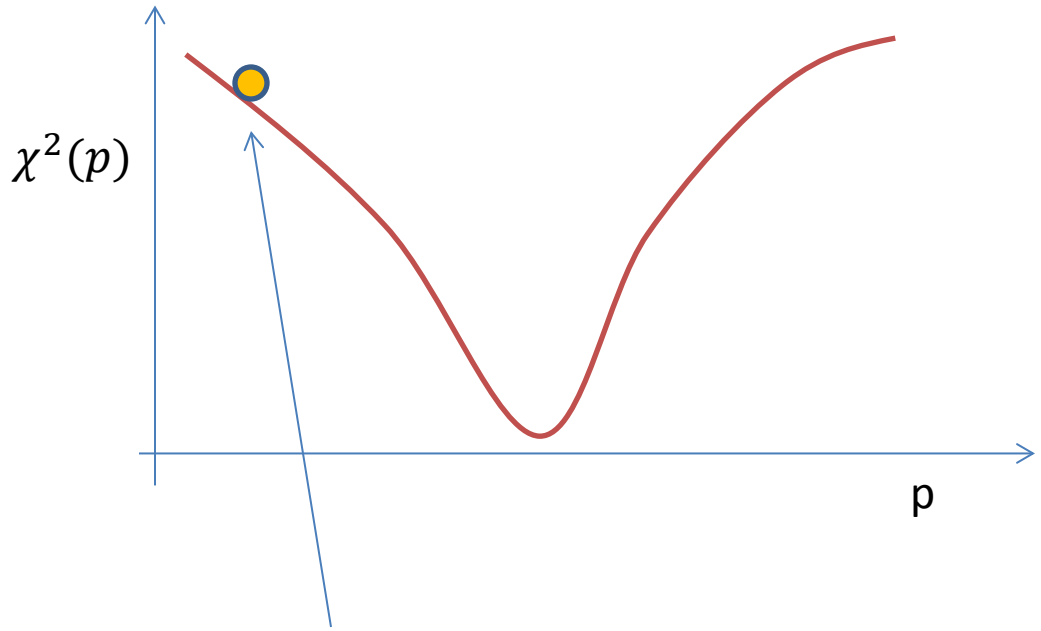
- How do you find the minimum of a function?



Suppose you can't take the derivative (or you have multiple dimensions).

The New Question:

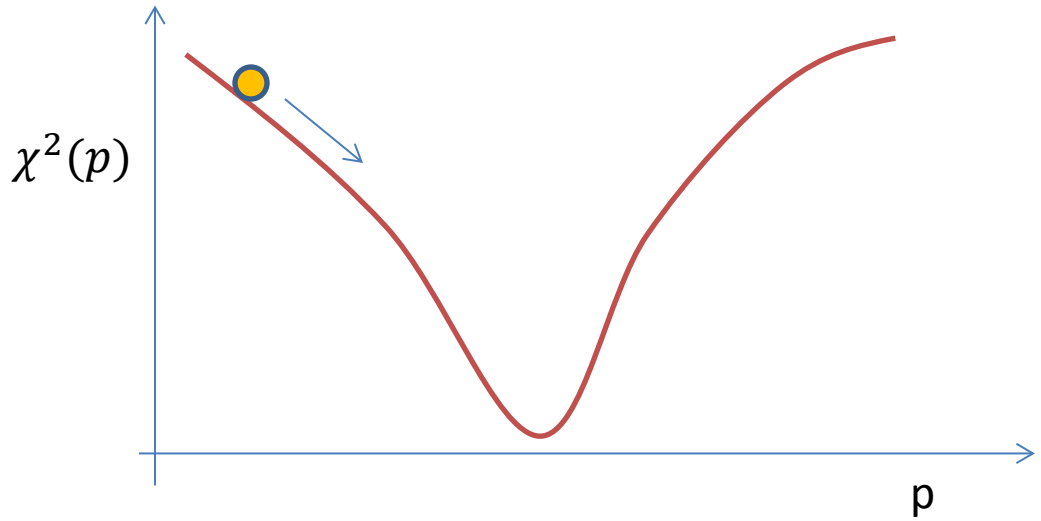
- How do you find the minimum of a function?



Step 1: Take a guess as to where you think the minimum might be.

The New Question:

- How do you find the minimum of a function?



Step 2: Head downhill.

So far so good but …

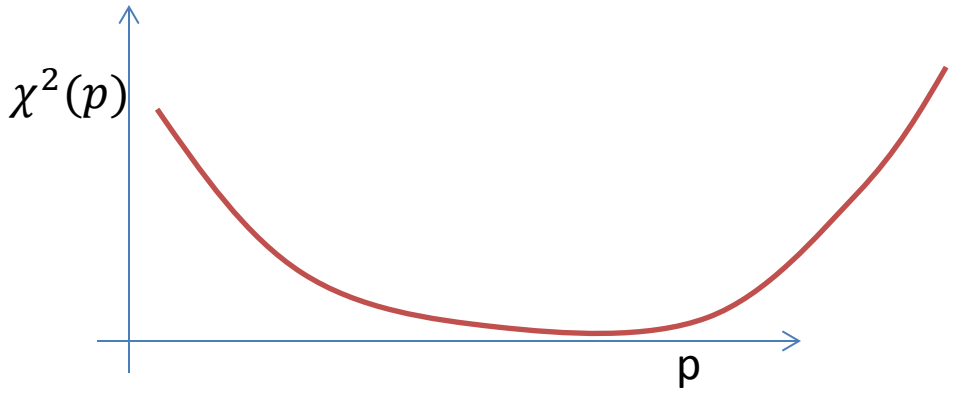
- What direction is downhill?
- How do you know when to stop heading downhill?
- How big should your steps be in the downhill direction?

What direction is downhill?

- The slope of the hill is given by the gradient (the vector of first partial derivatives).
- So the downhill direction is $-\nabla\chi^2(\vec{p})$
- In one-dimension (one free parameter) this is just $-\frac{d\chi^2}{dp}$
- So if our current position is p_i then our next step would be to $p_i + cnst \times \left(\frac{-d\chi^2}{dp}\right)$
- Heading downhill in this way is called the “steepest descent” method.

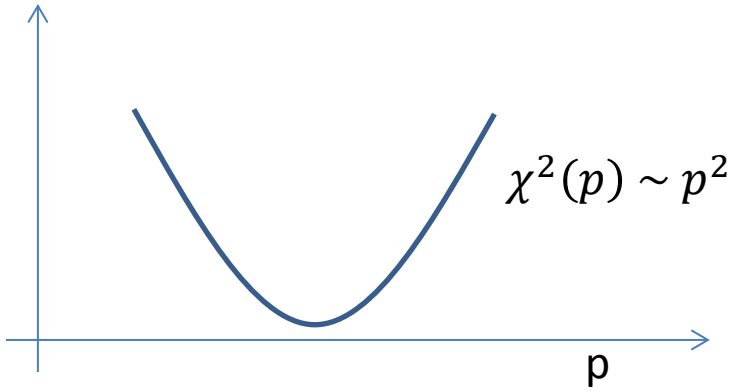
How do you know when to stop going downhill?

- Near the minimum of a function you can wander forever – bouncing back and forth across the true minimum.
- You must have a criteria for stopping – some sort of tolerance.
- The “steepest decent” method, following the gradient, is particularly poorly behaved near the minimum.



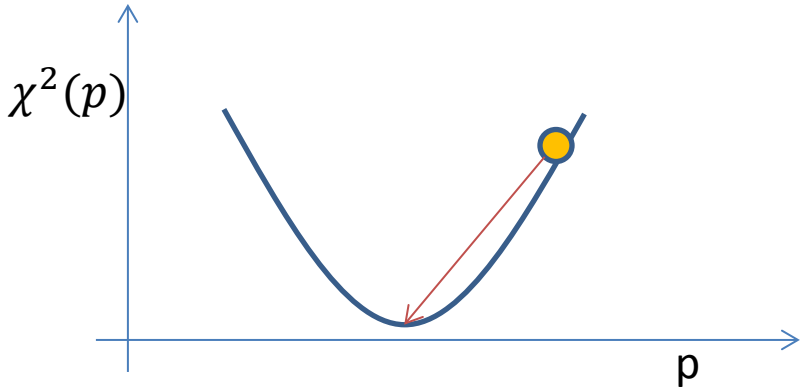
The Inverse-Hessian Method

- Since the region around the minimum looks something like a bowl (a parabola in one dimension, a multi-dimensional quadratic in more than one dimension), approximate it as such.



The Inverse-Hessian Method

- Since the region around the minimum looks something like a bowl (a parabola in one dimension, a multi-dimensional quadratic in more than one dimension), approximate it as such.



If it really is a parabola, we can take a single step directly to the minimum.

Combining Steepest Decent and Inverse-Hessian Methods

It turns out that the math behind both methods is so similar that we can write a common relation for the two.

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i; \vec{p}))^2}{\sigma_i^2}$$

Now define:

$$\beta_k = -\frac{1}{2}\frac{\partial \chi^2}{\partial p_k}, \qquad \alpha_{k,l} = \frac{1}{2}\frac{\partial^2 \chi^2}{\partial p_l \partial p_k}$$

Then our inverse-Hessian method can be written as:

$$\sum_{l=0}^{M} \alpha_{kl} \delta p_l = \beta_k$$

And our steepest decent method can be written as:

$$\delta p_l = cnst \times \beta_l$$

The Levenberg-Marquardt Method

- Levenberg and Marquardt independently found a way to seamlessly transition from the steepest descents method to the inverse Hessian method. Here's their approach:
- Choose the constant in the previous equation to be $1/\alpha_{ll}$- it's the only thing we've got with the right units anyway.
- Introduce a fudge factor, λ, that controls the step size. Steepest descents then becomes:

$$\delta p_l = \frac{1}{\lambda \alpha_{ll}} \beta_l$$

- Define a new matrix, α', such that

$$\alpha'_{jj} = \alpha_{jj}(1 + \lambda)$$
$$\alpha'_{jk} = \alpha_{jk} \; (j \neq k)$$

- Then both equations become:

$$\sum_{l=1}^{M} \alpha'_{kl} \delta p_l = \beta_k$$

The Levenberg-Marquardt Method

- Levenberg and Marquard independently found a way to seemlessly transition from the steepest descents method to the inverse Hessian method. Here's their approach:
- Choose the constant in the previous equation to be $1/\alpha_{ll}$- it's the only thing we've got with the right units anyway.
- Introduce a fudge factor, λ, that controls the step size. Steepest decents then becomes:

$$\delta p_l = \frac{1}{\lambda \alpha_{ll}} \beta_l$$

- Define a new matrix, α', such that

$$\alpha'_{jj} = \alpha_{jj}(1 + \lambda)$$
$$\alpha'_{jk} = \alpha_{jk} \; (j \neq k)$$

Why?

- Then both equations become:

$$\sum_{l=1}^{M} \alpha'_{kl} \delta p_l = \beta_k \quad (1)$$

The Levenberg-Marquardt Method

- Levenberg and Marquardt independently found a way to seamlessly transition from the steepest descents method to the inverse Hessian method. Here's their approach:
- Choose the constant in the previous equation to be $1/\alpha_{ll}$- it's the only thing we've got with the right units anyway.
- Introduce a fudge factor, λ, that controls the step size. Steepest descents then becomes:

$$\delta p_l = \frac{1}{\lambda \alpha_{ll}} \beta_l$$

- Define a new matrix, α', such that

$$\alpha'_{jj} = \alpha_{jj}(1+\lambda)$$
$$\alpha'_{jk} = \alpha_{jk} \; (j \neq k)$$

- Then both equations become:

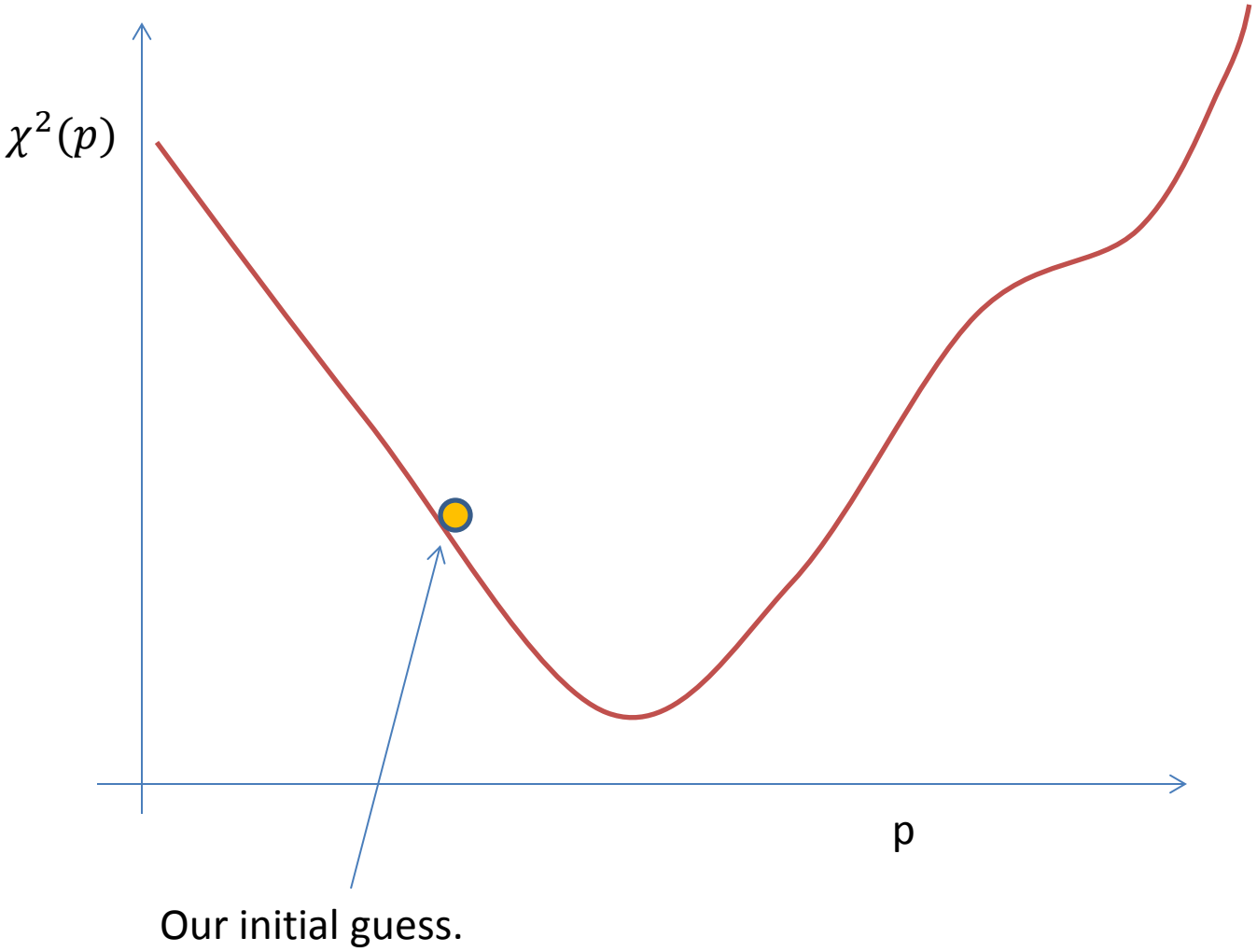
$$\sum_{l=1}^{M} \alpha'_{kl} \delta p_l = \beta_k \qquad (1)$$

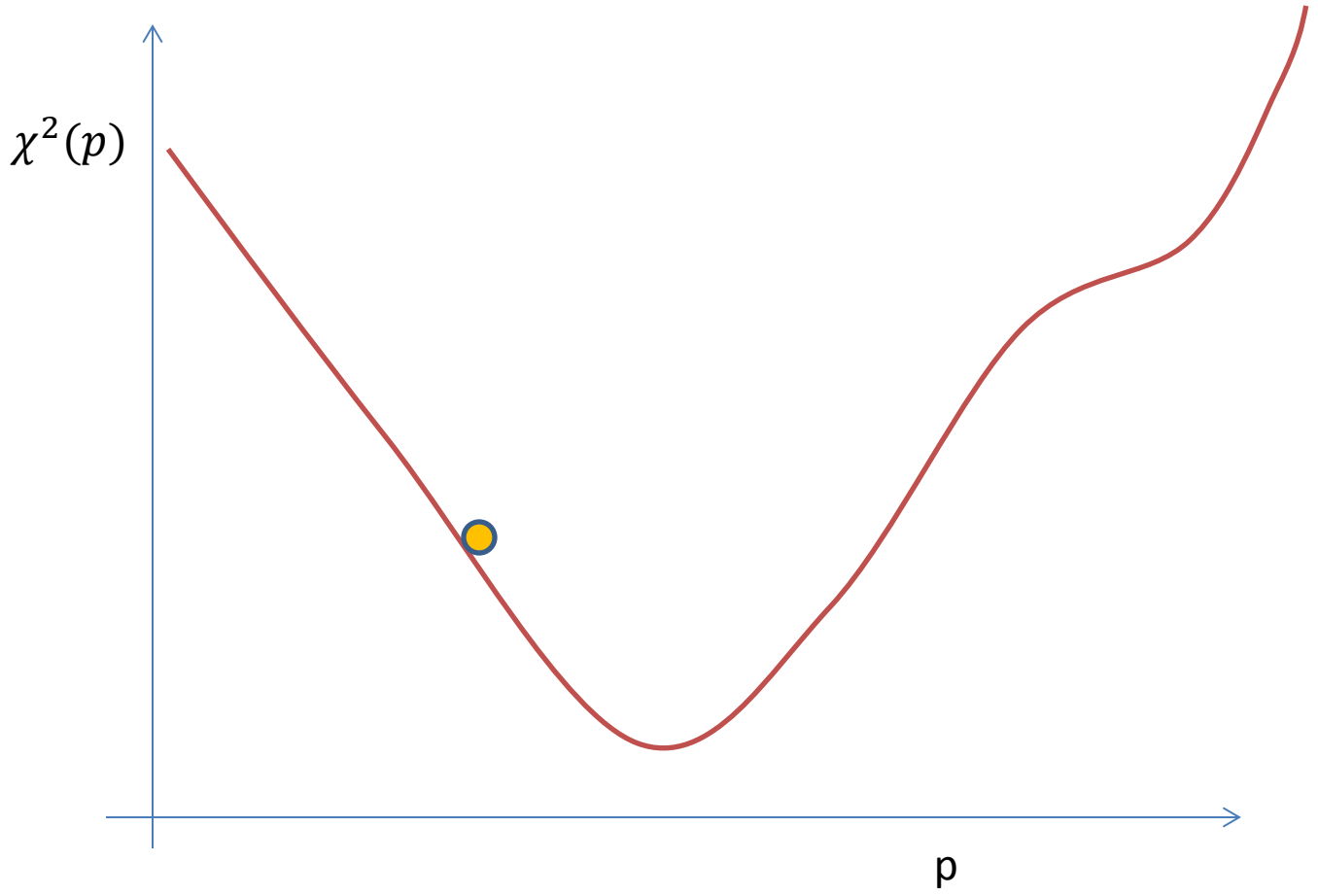
Why?

When λ is large, we have the steepest descent method. When λ is small, we have the inverse Hessian method.

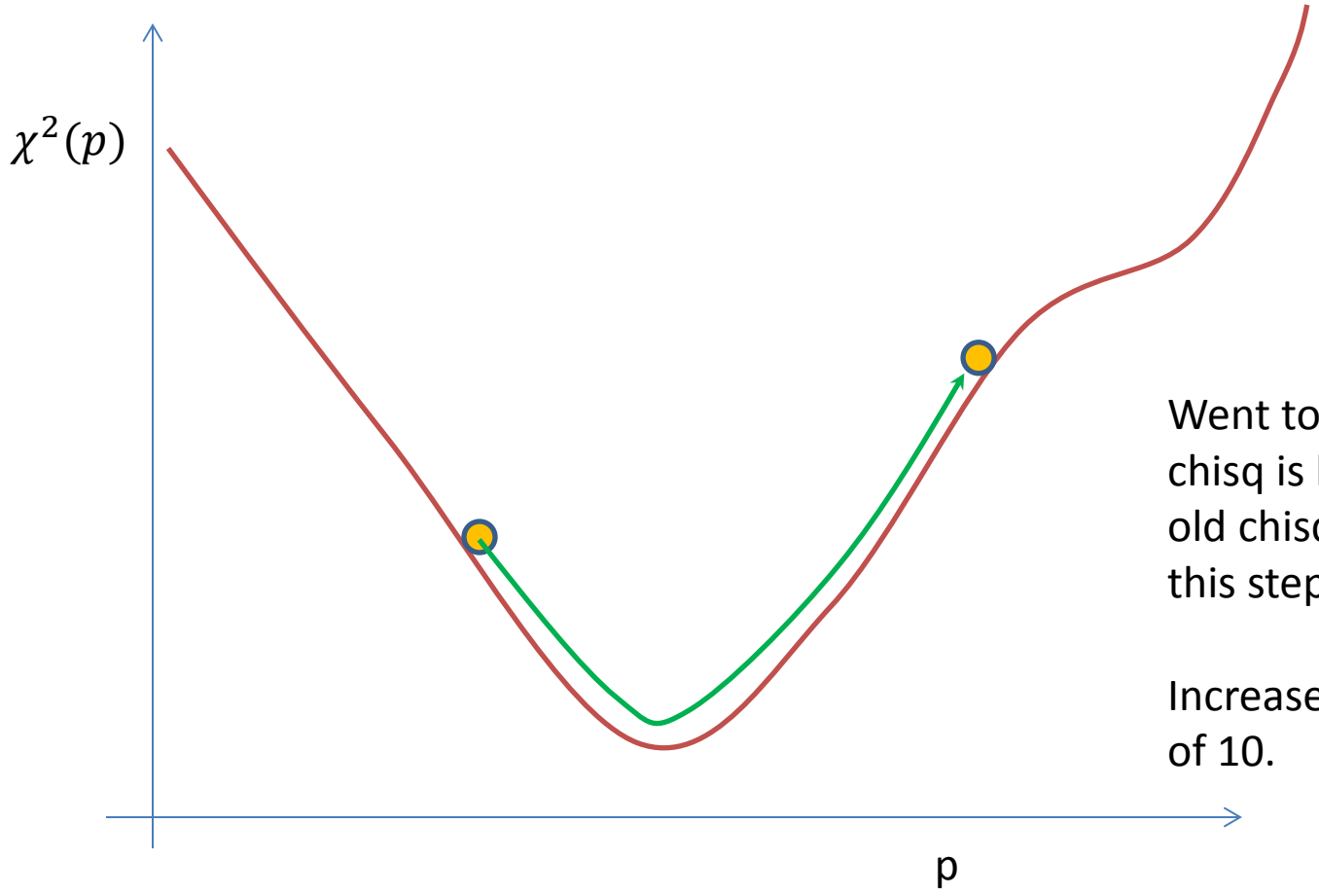
Our Levenberg-Marquardt Recipe is Then:

1. Guess a set of starting parameters, $\vec{p}$.
2. Compute $\chi^2(\vec{p})$ for your data and model.
3. Choose $\lambda = 0.001$
4. Solve equation (1) for $\delta\vec{p}$ and calculate the value for $\chi^2(\vec{p} + \delta\vec{p})$.
5. If $\chi^2(\vec{p} + \delta\vec{p}) \geq \chi^2(\vec{p})$, then ***increase*** λ by a factor of 10 and go back to step #4.
6. If $\chi^2(\vec{p} + \delta\vec{p}) < \chi^2(\vec{p})$, then
   1. ***decrease*** λ by a factor of 10
   2. update $\vec{p} \rightarrow \vec{p} + \delta\vec{p}$
   3. go back to step #4
7. Stop when $\Delta\chi^2 < 0.001$.
8. The parameter-parameter covariance matrix is α^{-1}


$\chi^2(p)$
p
Our initial guess.



Steepest Descents: $\delta p_l = \frac{1}{\lambda \alpha_{ll}} \beta_l$

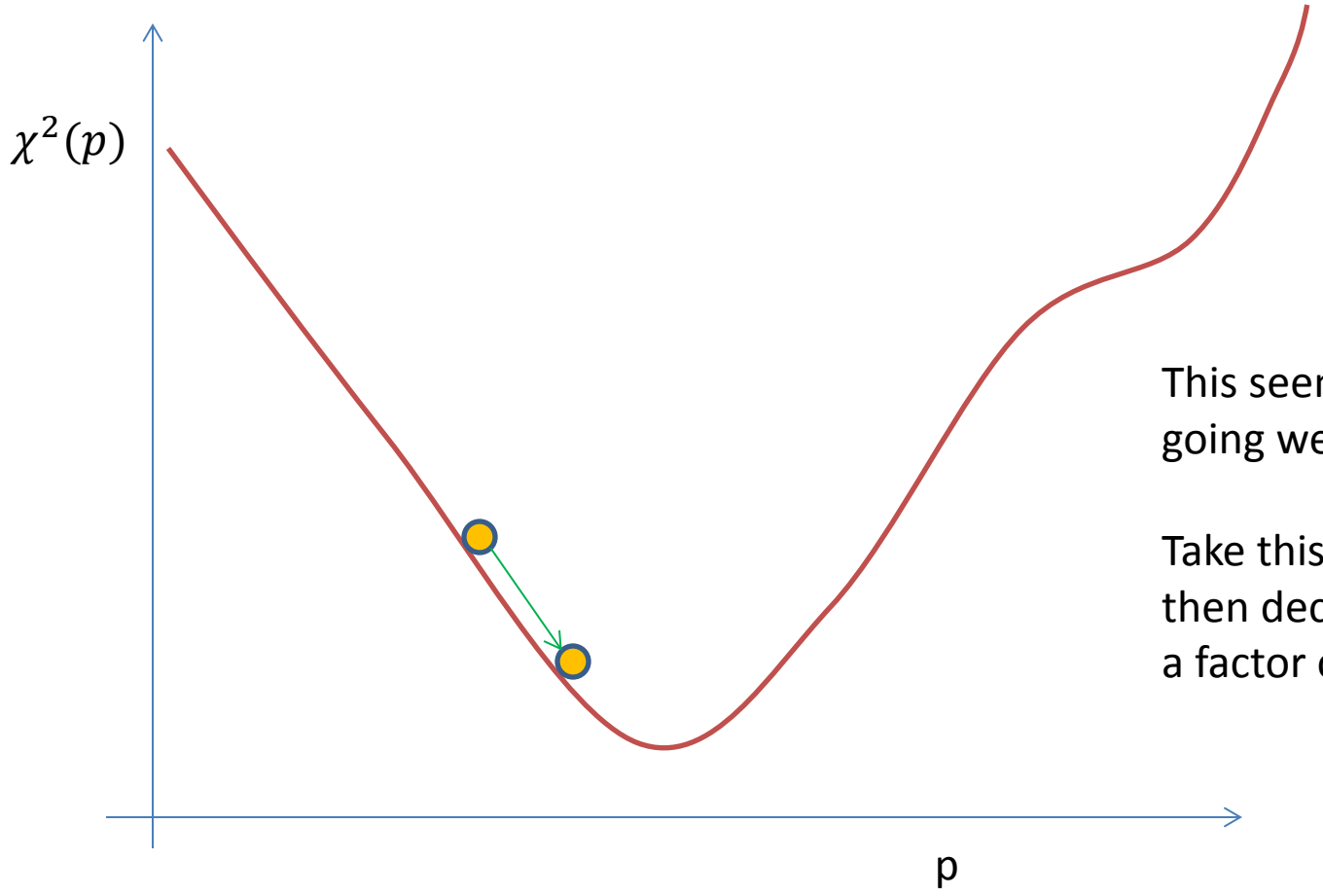


Went too far, new chisq is bigger than old chisq. Don't take this step!

Increase λ by a factor of 10.

Steepest Descents: $\delta p_l = \frac{1}{\lambda \alpha_{ll}} \beta_l$

Bigger λ means smaller step.

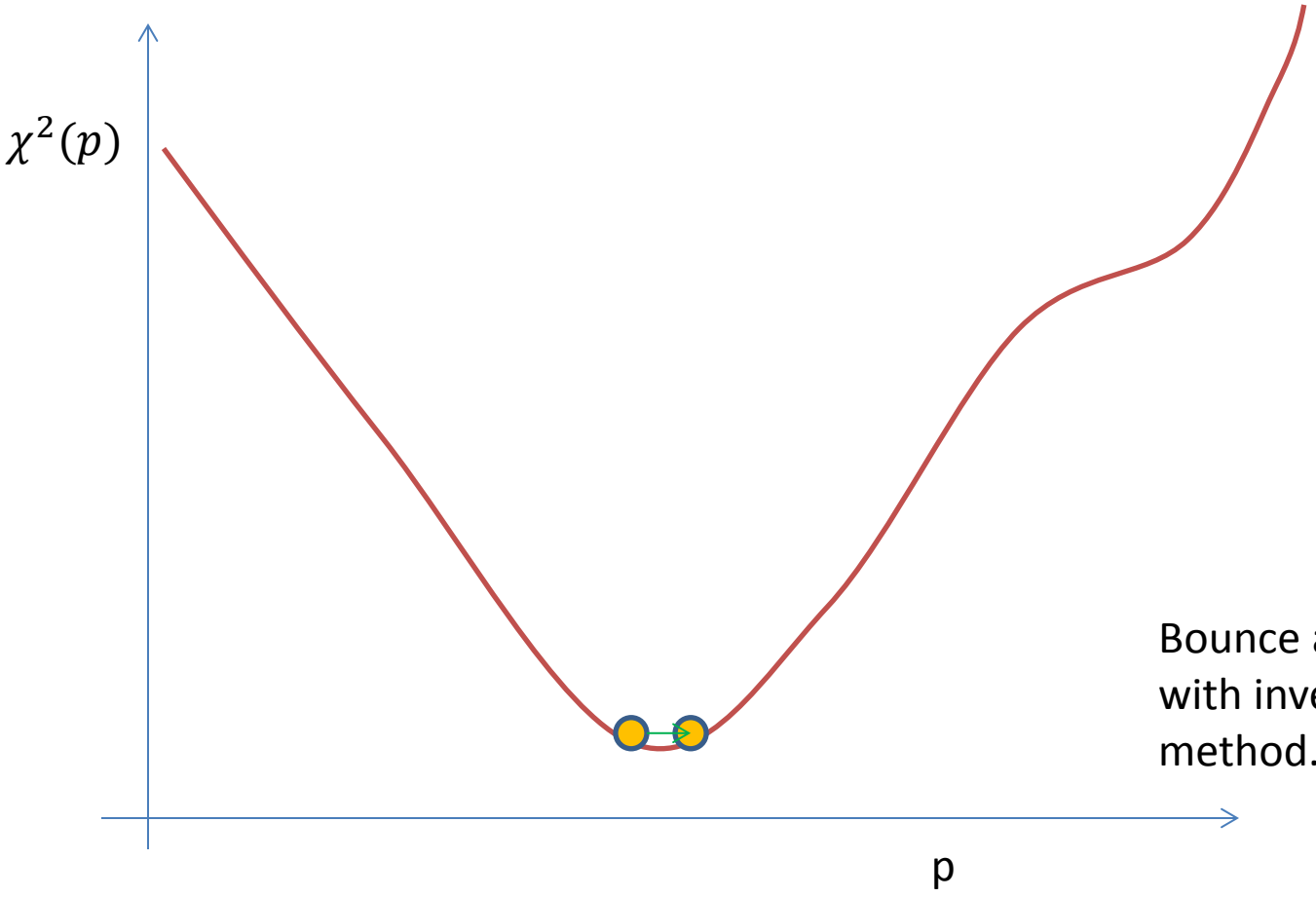


This seems to be going well.

Take this step and then decrease λ by a factor of 10.

Steepest Descents: $\delta p_l = \frac{1}{\lambda \alpha_{ll}} \beta_l$

Smaller λ means closer to inverse-Hessian method.



Bounce around here with inverse-Hessian method.

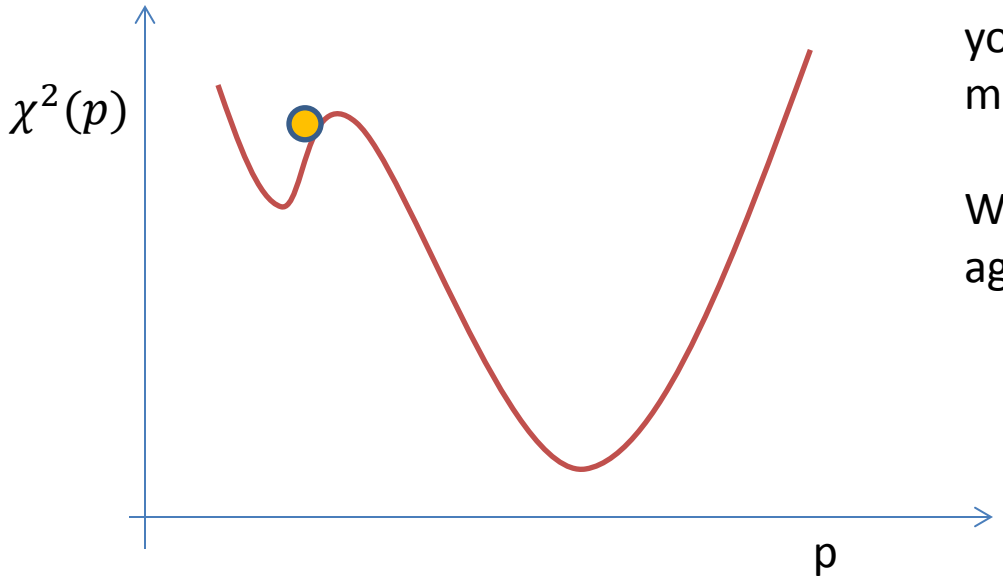
Stop when $\Delta\chi^2 < 0.001$.

Pitfalls

- Beware local minima

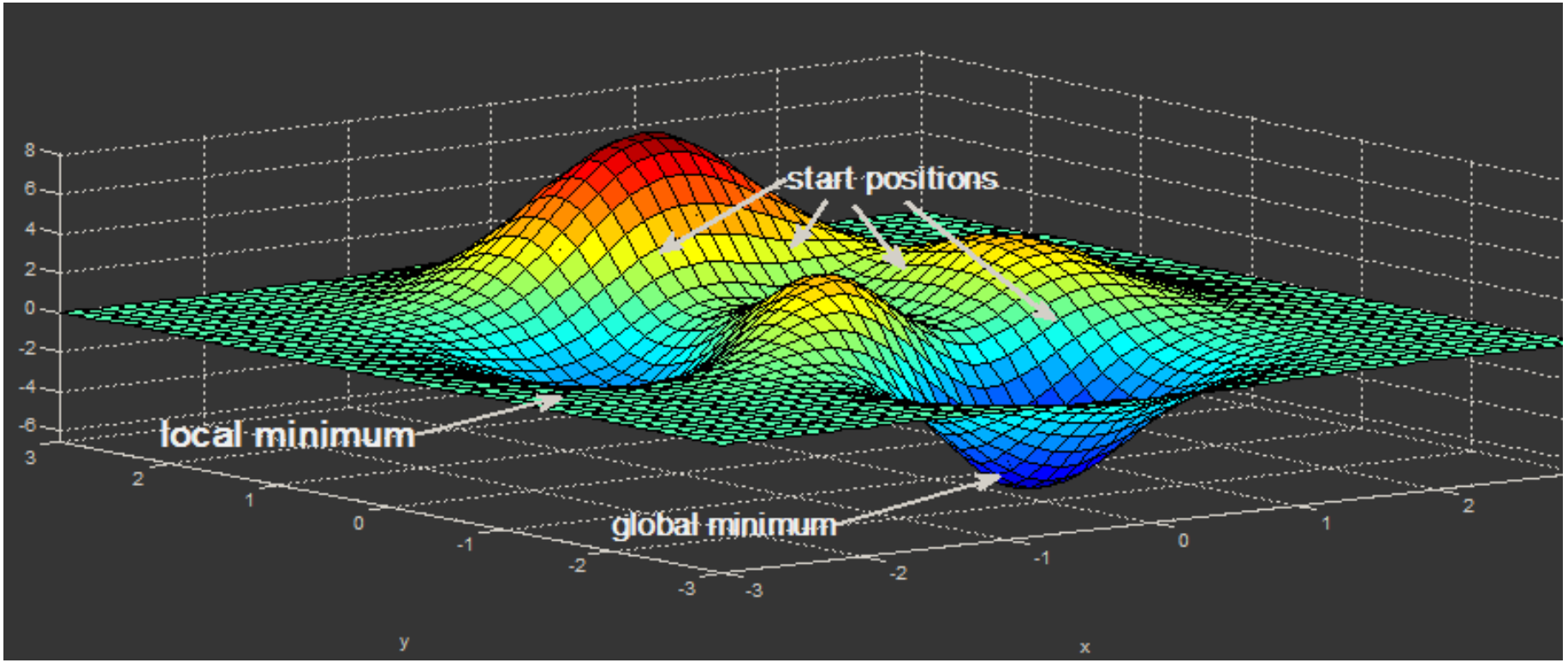
Pitfalls

- Beware local minima



If your initial guess is here, you will not find the global minimum.

What can you do to defend against this?


start positions
local minimum
global minimum
8
6
4
2
0
-2
-4
-6
3
2
1
0
-1
-2
-3
y
-3
-2
-1
0
1
2
x

I can’t stress enough how important this technique is.

Levenberg-Marquardt is a general-purpose tool for fitting data to models that are non-linear in their parameters.

If you are going to be a scientist, you must have this in your arsenal.

Implementation

- We are going to use the “lmfit” python package.

- Head to: http://lmfit.github.io/lmfit-py/intro.html and start reading the documentation.

Exercise

Today's exercises have you practicing Levenberg-Marquardt fitting in three different problems. In all three cases I have provided data in .npz files with the following variables:

- x - this is an array of independent values
- y - this is an array of dependent data (the measurements)
- sigma - this is the measurement error - the error in the y-values. The error in the y values is the same for all values of y.
- npts - this is the number of points in the x and y arrays

1. Fit the data found in data_e1.npz (available on the class moodle page) to the polynomial: y = p[0] + p[1]*x + p[2]*x**2 + p[3]*x**3 both using your existing linear fitting tools and with the lmfit package. Make sure you quote errors on all four parameters and show plots of the fits and the residuals. Did you get identical answers? Explain.

2. Fit the data found in data_e2.npz to the gaussian model described in the lmfit documentation and using the built-in python model for a gaussian. Again, your output should be the best-fit parameters and their errors, a plot of the fit and the data, and a plot of the residuals.

3. Finally, write your own python function to fit the data in data_e3.npz to the model: y = p[0]*np.cos(p[1]*x)**p[2]. The output of your code should be the same products as for the other problems.