Scientific Computing: Lectures 18

- Regression (Curve Fitting)
  - Linear regression (fitting data)
  - Non-linear functions that can be linearized
  - Polynomial regression [polyfit() in Pyplot]
  - Non-linear - Levenberg-Marquardt and scipy.curve_fit()

CLASS NOTES

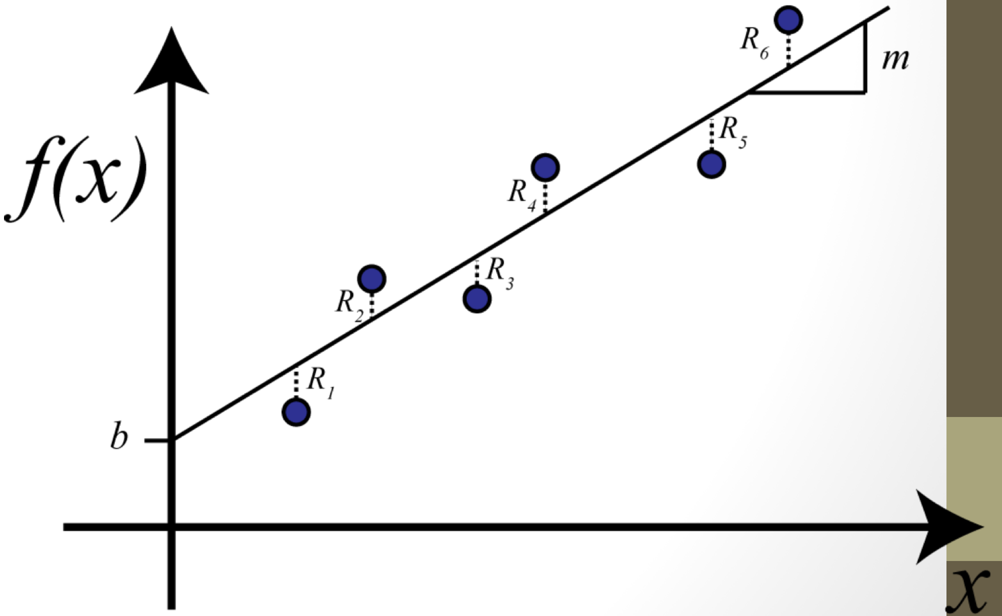
- HW#7 due Friday
- Plan for rest of semester (9 lectures): Regression, Root Finding, Linear Algebra, PDE methods, **plus optional components**
- Need to start thinking about Final Project. It will be due on the Thursday of exam week. Same format as mid-term, but should exhibit a higher degree of difficulty.
- Proposals due Thurs. April 18 in class.

Optional Component Options

- Components done in the past
  - Parallel Computing
  - GUI programming
- Some other possibilities
  - Time series analysis (such as Fourier analysis)
  - Interfacing with hardware
    - Data acquisition
    - Instrument Control
  - Deeper dive into modules: scipy, numpy, …
  - Deeper dive into advanced visualization (3D graphics,..)

Linear Regression

- A common task for scientists is to compare a set of measured data with a mathematical (theoretical) model.
- Simplest model is a line: $f(x) = mx + b$
- Problem is to determine the slope and intercept which 'best fits' the data.
- Criteria for 'best fit' is that which minimizes the sum of squares of the residuals (difference between data point and model).
- A type of 'optimization' problem.



Theory for linear fits

- A type of optimization problem: need to determine m and b which minimize the sum of the squares of residuals (R^2).
- Do this by taking derivative w.r.t. m and b and setting equal to zero. For N data points [(x,y) pairs]:

$$R^2 = \sum_{i=1}^{N} [y_i - mx_i - b]^2$$

$$\frac{\partial(R^2)}{\partial m} = 0$$

$$\frac{\partial(R^2)}{\partial b} = 0$$

- Now solve these equations to find optimal m and b:

$$\boxed{m = \frac{\sum_i y_i(x_i - \bar{x})}{\sum_i x_i(x_i - \bar{x})}}$$

$$\boxed{b = \bar{y} - m\bar{x}}$$

Quality of fit

- So how do we quantify how well the model fits the data?
- One option is the standard deviation (for 2 free parameters):

$$\sigma = \sqrt{\frac{R^2}{N-2}}$$

- More commonly see the correlation coefficient (r^2) in which 1.00 is a perfect fit.

$$r^2 = \frac{S_{xy}^2}{S_{xx}S_{yy}}$$

$$S_{xy} = \sum_i [x_i y_i - N\bar{x}\bar{y}]$$

$$S_{xx} = \sum_i [x_i^2 - N\bar{x}^2]$$

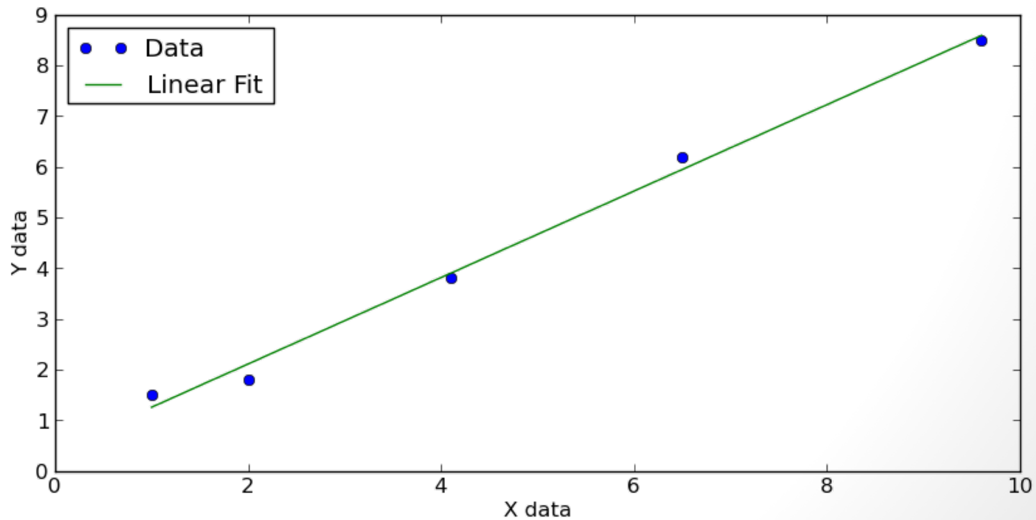
$$S_{yy} = \sum_i [y_i^2 - N\bar{y}^2]$$

Uncertainty in Parameters

- The quality of fit will determine the confidence (uncertainty) in the values for the fitted parameters.

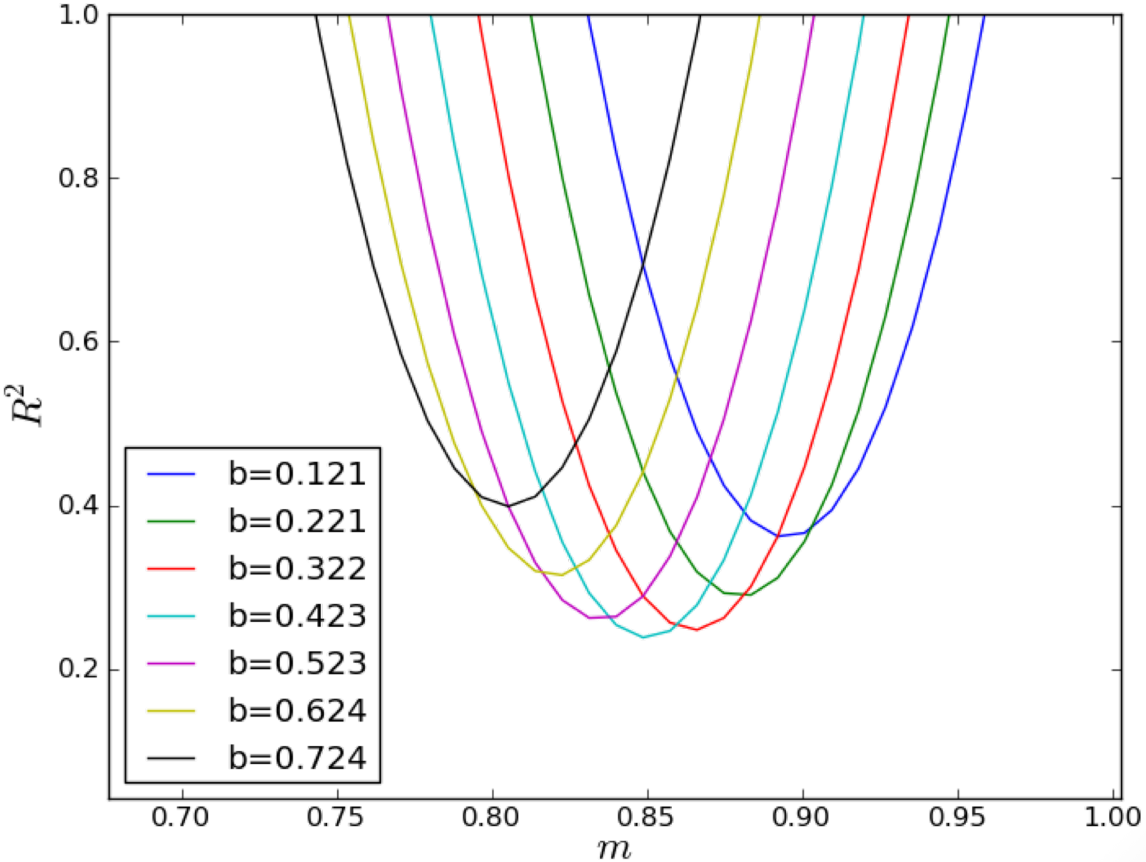
$$\delta m = \frac{\sigma}{\sqrt{S_{xx}}} \qquad \delta b = \sigma\sqrt{\frac{1}{N} + \frac{\bar{x}^2}{S_{xx}}}$$

- For data shown,
  m=0.853+/-0.04
  b=0.402+/-0.23
  r^2=0.993



Minimization of R^2

- Plot of the sum of squares of the residuals vs free parameters shows how fit values do in fact find the minimum.



Functions which can be linearized

- Some non-linear functions can be manipulated to take a linear shape.
- **Power Laws:** take log of both sides & exponent becomes slope, log-log plot is a line. Use

$$y = cx^b \rightarrow \ln(y) = b\ln(x) + \ln(c)$$

- **Exponentials:** take log of both sides, argument becomes r.h.s, log-linear plot is a line.

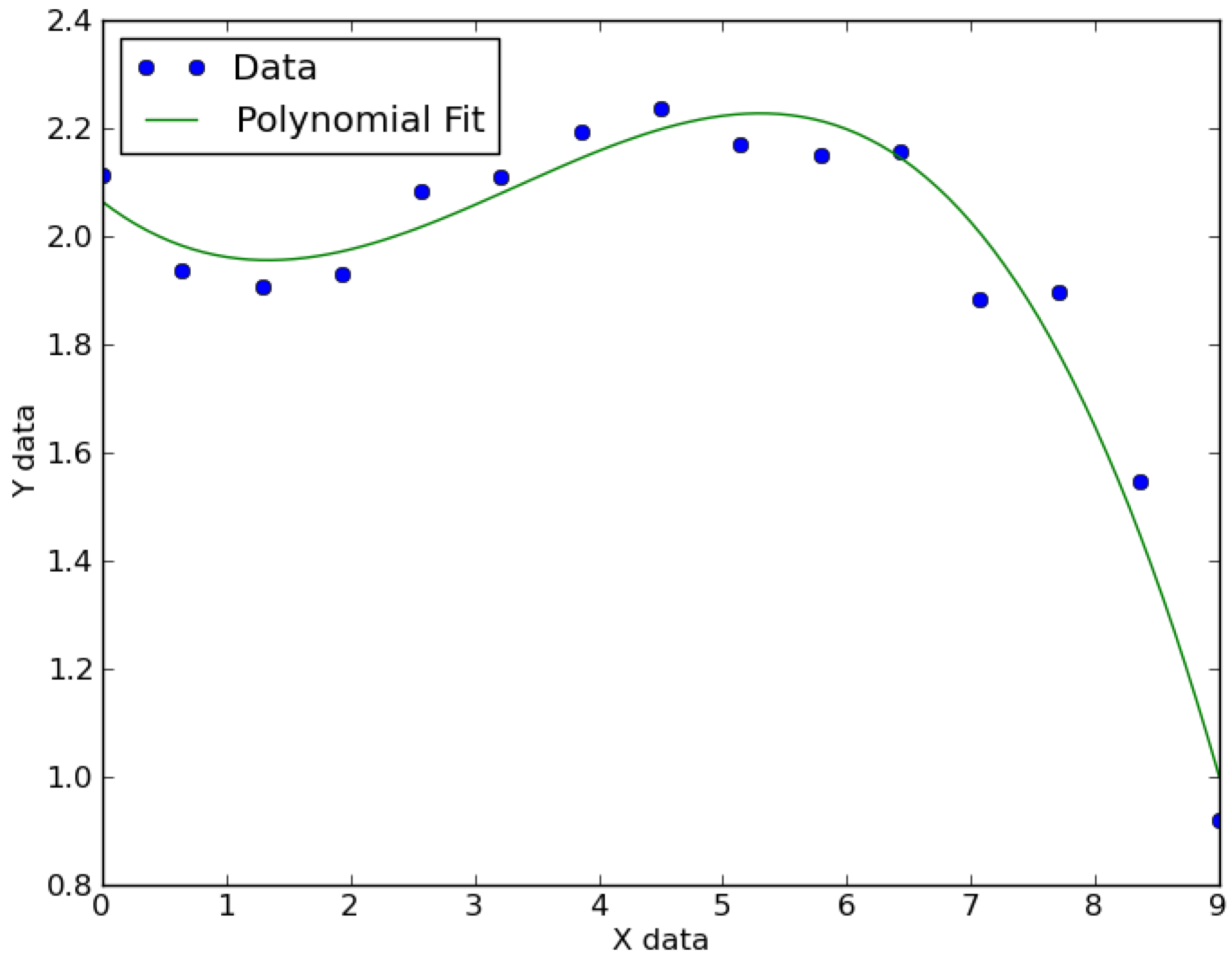
$$y = ae^{mx} \rightarrow \ln(y) = mx + \ln(a)$$

Polynomials

$$y = a + bx + cx^2 + dx^3 + ...$$

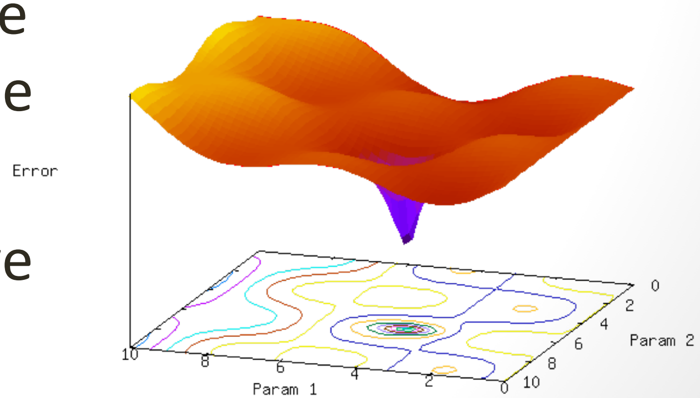
- Can be linearized, but math is trickier.
- Pylab has function called polyfit(xdata,ydata,order) which returns coefficients (a,b,c,..) which minimize the sum of squares of residuals.
- order = 1: line, order = 2: quadratic, order = 3: cubic,....
- Useful when taking derivatives of actual data which are sparse.
- For plotting, you can use the polyval() function to generate data for the curve.

Example: 4th order polynomial



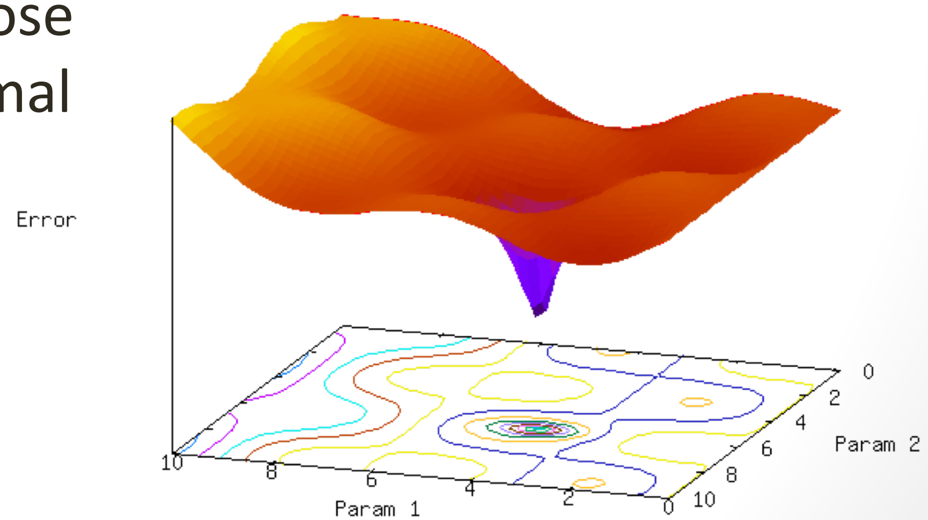
Non-linear Regression: General Ideas

- Most widely used algorithm used for non-linear fits is called the Levenberg-Marquardt method. It is iterative.
- Idea is compute gradient of error surface at starting point in parameter space (initial guess of parameter values). Then take a “step” (by adjusting parameters) “down hill” in direction of steepest descent.
- Compute gradient at new location and repeat.
- Once gradients are small, assume you are near minimum and shape is parabolic (like linear case).
- Compute minimum directly as we did with linear case.



Warnings about non-linear fits

- Your error surface is no longer monotonic about the global minimum – meaning there are LOCAL minima.
- Before settling on a final fit, try different starting points. You MAY end up in a lower minimum!
- Do everything you can to start the process with parameter values as close as possible to the optimal values.



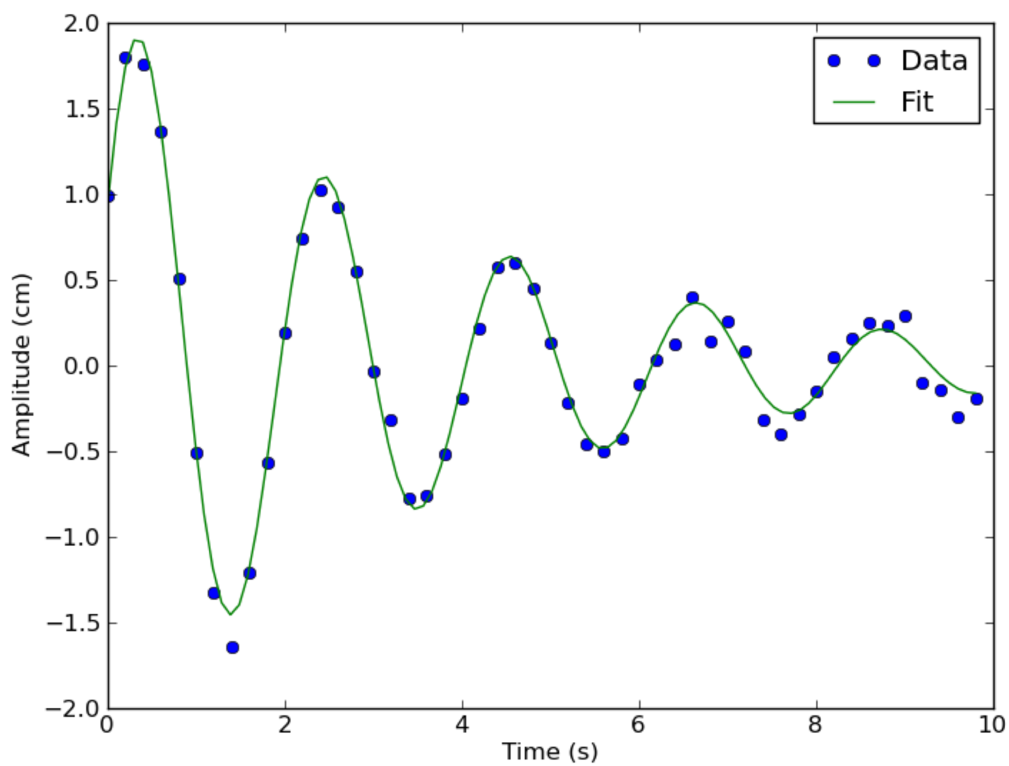
- Pay attention to the parameter values – do they make physical sense??
- Proceed with caution!

Python tools for non-linear fitting

- There are several approaches, some easier to use and some are more robust.
- curve_fit in scipy.optimize
  - use: fit=curve_fit (funct,xdata,ydata,p0=params0)
  - comments: fairly convenient and generally robust – almost always will converge.
  - returns: tuple of fitted parameters, variance-covariance (VC) matrix – more on this later.
- curve_fit() is a ‘wrapper’ function for scipy.optimize.functions.leastsq. Using leastsq() directly will provide MUCH more detail about the statistics of the fit.
- However, we can compute a ROUGH estimate of the quality of the fit from the curve_fit() output…

Example: Damped sine

- Model is a damped sine function: $y = Ae^{-\tau t} \sin(\omega t + \phi)$
- Four free parameters – need at least 12 data points to fit.



'Quality of Fit' metrics

- Linear fit has a correlation coefficient (r^2)
- Can compute a similar quantity with nonlinear fits as a ratio of the sum of squares of residuals (SSR) to total sum of squares (SST) :

$$SST = \sum_i [y_i - \bar{y}]^2$$

$$SSR = R^2 = \sum_{i=1}^{N} [y_i - F(x_i; \mathrm{params})]^2$$

- Then, r^2 can be computed by:

$$\boxed{r^2 = \sqrt{1 - \frac{SSR}{SST}}}$$

Confidence in parameters

- So how accurate are the fitted parameters?
- That is a complicated question. In an ideal world, you would run lots of fits, adjusting the data within error bars, and compute a standard deviation of the variance in the resulting parameters for each fit.
- A simpler (and less accurate) method is to multiply the diagonal elements of the variance-covariance (VC) matrix by the square root of the reduced sum of squares (or reduced chi-square).
- VC matrix returned by curve_fit is a nxn matrix for n free parameters. Diagonals give variance of each parameter, and off diagonals give covariance between variables – 'How much does a change 'A' effect the final value of B?')

Confidence in parameters

- If I have three free parameters (a,b,c):

$$\mathrm{cov} = \begin{pmatrix} aa & ab & ac \\ ab & bb & ab \\ ac & bc & cc \end{pmatrix}$$

- For N data points and m parameters, the reduced SSR is

$$\chi^2 = \frac{SSR}{N-m}$$

- Then an approximate error of the fitted parameters are:

$$\delta a = aa\sqrt{\chi^2} \quad ; \quad \delta b = bb\sqrt{\chi^2} \quad ; \quad \delta c = cc\sqrt{\chi^2}$$

Power of a Good Model

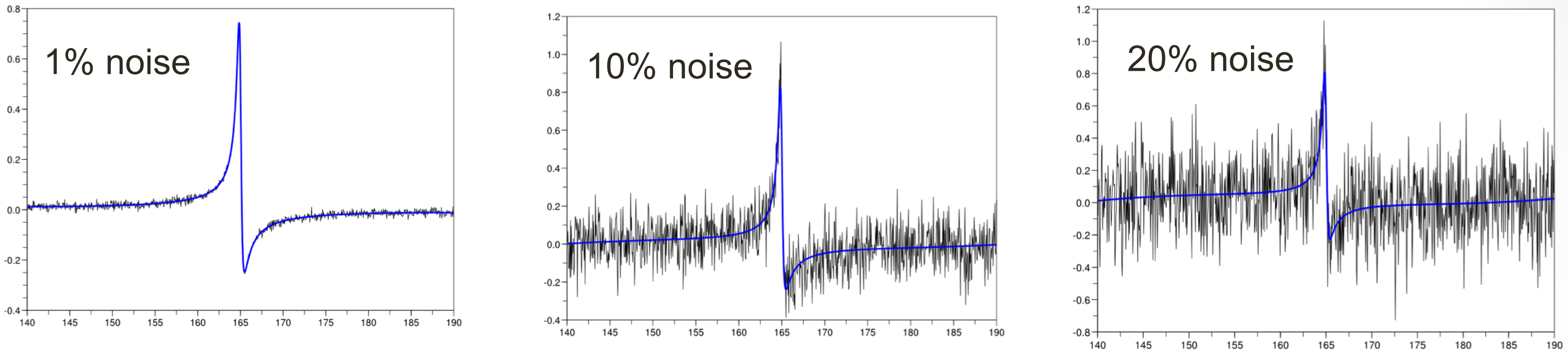
I gave a talk at a meeting several years ago in a Signal Processing and Noise session part of which showed the power of having a good model.

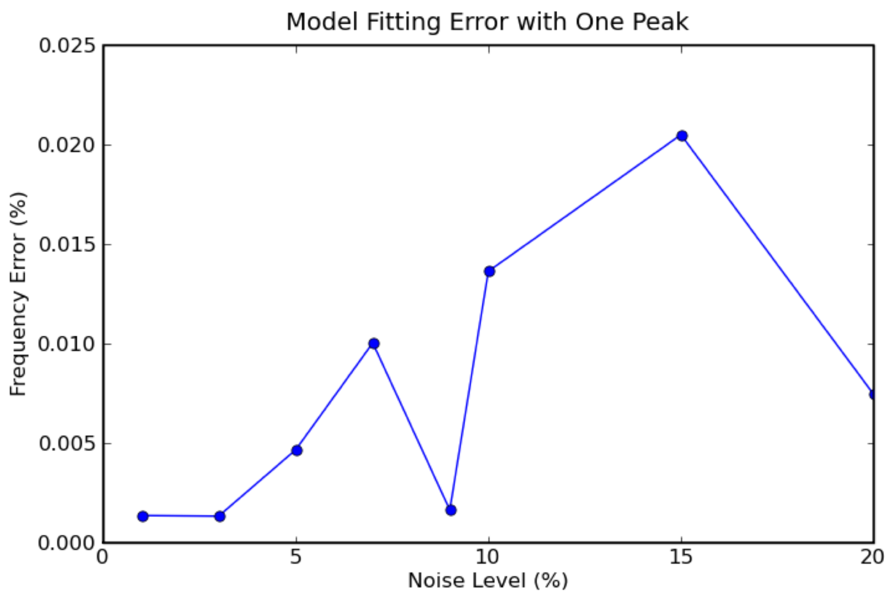
- Mechanical resonances follow a Lorentzian line shape.

$$A(f)=\frac{\frac{f}{f_0}\cos(\phi)+(1-\frac{f}{f_0}Q\sin(\phi))}{\left(\frac{f}{f_0}\right)^2+\left(1-\left(\frac{f}{f_0}\right)^2\right)^2 Q^2}+a_0+a_1 f+a_2 f^2$$

- Fit resonance data with model to determine fo and Q.
- Manually pick (click) best guess for fo as starting point (Python script).
- Test effect of fo on noise level by generating synthetic data with Gaussian noise of x% of peak amplitude.

Noise and frequency error: 1 peak





Noise and frequency error: 2 peaks