

Chapter 3

Least Squares and regression techniques, goodness of fit and tests, non-linear least squares techniques

3.1 Statistical Basis for Regression

Often in working with data, we must ask specific quantitative questions about how well the data reflects some underlying model of processes in nature. The question may arise, for example, as to whether some property (*e.g.*, dissolved oxygen in seawater) changes linearly with time. The actual rate of change (the slope of the relationship) may be of quantitative interest, or the initial value (the intercept of the relationship) may be important. Moreover, in addition to obtaining the most accurate estimate of one or more of these parameters, we need to know how precisely we know the values, or more explicitly, the confidence interval (range of probable values) of the parameters. Finally, we must decide if our linear model (or some other function) is appropriate or robust by checking the goodness-of-fit.

3.1.1 Underlying Distributions Sometimes Lie

Implicit in the Least Squares techniques is the belief that there is some underlying “truth” that we are attempting to model, however imperfectly, due to errors in our observations. A typical assumption is that the data are normally distributed, *i.e.*, the deviations between the actually measured data from the underlying, true values, if plotted as a histogram, form a Gaussian (bell shaped) curve. As experimentalists, or as consumers of experimental data, you must realize that although the bulk of experimental data does indeed adhere to this Gaussian ideal, there is a tendency for unknown, episodic interferences to produce outliers (also known as fliers) whose probability of occurrence approaches the astronomically small under the assumption of a normal distribution. We are taught in undergraduate laboratories that with a Gaussian distribution, only about 1/3 of the data lies more than one standard deviation from the mean (the standard deviation is the root mean square width,

aka RMS width, of the Gaussian curve). Further, you learn that only about 5% fall more than two standard deviations from the mean, about 1% more than 3 standard deviations, etc. What is the probability, then, of that point you just measured that is 10 standard deviations away from the others? Try roughly one in 10^{11} . To put this in perspective, having just one flier like that, you would be obliged to make of order 10^{11} more measurements to adequately unbiased the data set that has been contaminated by that one flier. Thus the take home lesson is that you must be very vigilant in determining whether you have outliers influencing your data. Often what experimentalists do, if they want an unbiased estimate of some specific parameter (*e.g.* a mean, slope or intercept) is to first fit all of the data to the model, eliminated the improbable data (*i.e.*, those data more than 3 or 4 standard deviations from the model fit), then refit the model to the reduced data set. If you find yourself throwing out large portions of your data, though, you had better think again.

3.1.2 The χ^2 Squared Defined (and Goodness of Fit)

How do you judge (aside from the eyeball test) the goodness of a fit? The most common goal is to reduce the “distance” between the observations and the model. The standard measure, which can be derived from the Gaussian nature of the underlying distributions, is the Chi-Squared, which is defined as

$$\chi^2 = \sum_{i=1}^N \frac{(y_f - y_i)^2}{\sigma_i^2} \quad (3.1)$$

where y_f is the model (the fit) estimate, y_i is the actual observation, and σ_i in the denominator is the uncertainty in the individual measurement (y_i). You could of course choose other measures of distance, *e.g.* the absolute values $|y_i - y_f|$. But chi-squared has the nice property that it results in relatively simple, analytical forms and it can be derived for a normal distribution from the method of Maximum Likelihood. You may arrive at the choice of σ_i in a number of ways :

- it could be the size of the smallest graduation on your measuring stick or analytical balance
- there may be some fundamental physical limitation to your measurements which you derive from basic principles
- there is some internal statistic associated with the measurement itself (*i.e.*, you may be using the mean of repeated measurements as your observation)
- you may derive it from repeated measurements of the same thing (statistically)
- you may read it in a manual, or from manufacturer’s specifications, probably obtained from one or more of the above

Note also, that this σ_i may/can/will vary from measurement to measurement and hence represents some kind of weighting factor that you might use when incorporating data into a larger set. That is, you would not want to value a poorly made or inaccurate measurement as much as a more carefully made, precise measurement.

Regardless of how you arrived at your choice of σ_i , you would tend to think that the Reduced Chi Squared, which is really just the root mean square deviation normalized to measurement errors,

would tend to be close to 1 if things are working correctly. Here we define the reduced chi squared (note the subscript ν , to distinguish it from its bigger brother) as:

$$\chi_\nu^2 = \frac{1}{\nu} \sum_{i=1}^N \frac{(y_f - y_i)^2}{\sigma_i^2} = \frac{1}{\nu} \chi^2 \quad (3.2)$$

where $\nu = N - n$ is the degrees of freedom, and n is the number of coefficients or parameters used in the regression fit. For example, computing a mean gives $N - 1$ as the degrees of freedom, and a regression to a straight line gives $N - 2$.

If your reduced chi squared is much larger than 1, say 10 or 100, it means that you are either doing a lousy job making measurements (*i.e.*, something is wrong with your apparatus or technique), or you have been overly optimistic about your measurement uncertainties. Another possibility is that you may have a bad model, that is an inappropriate fitting function. For example the data better fit a quadratic or exponential relationship rather than a straight line. If the reduced chi squared is too small, say .1 or .01, it may mean that you have been too pessimistic about measurement errors. It is very important to pay careful attention to estimation of your measurement errors.

3.1.3 Look at Your Residuals

Finally, a good reduced chi squared may not mean you have a good fit or model. It may be a conspiracy between the above factors, or that you have overlooked something. Look at your data, and particularly look for patterns or structure in the residuals of the data. That is, plot up the differences between your data and the values predicted by your regression model. If the deviations between your data and your model fit show a characteristic large scale structure, this is indicative of unresolved and unfit characteristics. Look, for example, at the two plots in Fig 3.1.

The left hand figure shows a straight line regression to a data set in which the fit looks rather good, yet when you subtract the fit values from the observations (figure on the right) you see a definite pattern. The reduced chi squared is just one of many diagnostics of model success, and like any single number indicator, does not tell the whole story.

If you do not have an independent determination of your measurement uncertainty (this, however, should be a most unusual and unphysical thing to happen) do not despair. You can use the reduced chi squared as a means of determining your measurement error, presupposing that you know you have a good “model” of your data, *i.e.*, that you are fitting it to the correct curve. Regardless of your situation, one thing is important to remember:

MINIMIZATION OF THE χ^2 IS THE FOUNDATION OF ALL THE LEAST SQUARES REGRESSION TECHNIQUES.

Hence the term “least squares”. All of the routines discussed below are aimed at finding the model parameters (coefficients) which minimize the chi squared of a given data population.

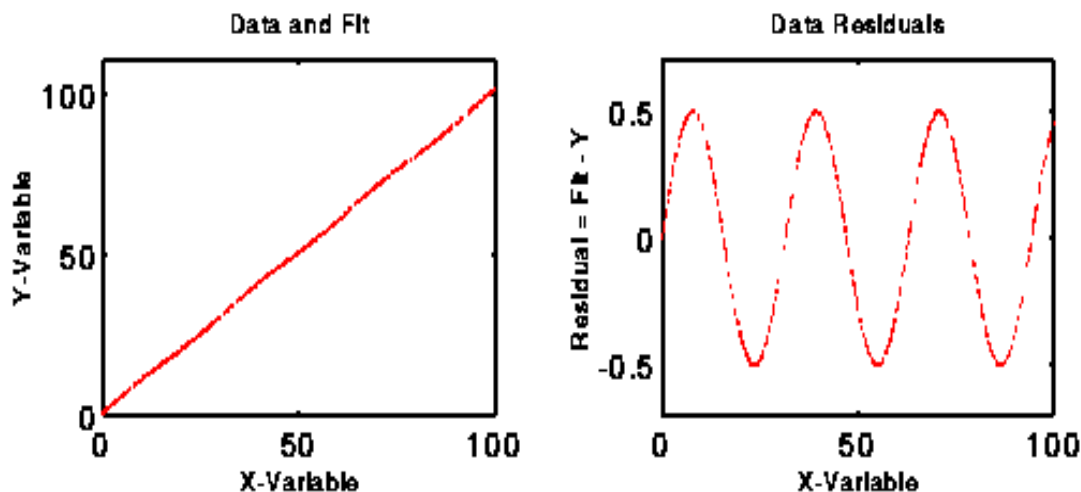


Figure 3.1: The data in the left panel appear to be a good fit to a straight-line. But when you look at the data residuals (right panel) from the least squares fit, you see that the residuals actually have structure (in this case a sine wave)!

3.2 Least Squares Fitting a Straight Line

Perhaps the most common model of data regression (aside from mean and standard deviation) is the fit to a straight line. It is also the easiest formulation to visualize and derive from basic principles (although we will try to convince you that there is an even more robust, succinct, general, and easy to understand approach in section 3.3). By substituting into the definition of χ^2 the formula for a least squares regression $y = a_1 + a_2x$ (where a_1 is the intercept, and a_2 is the slope), you have

$$\chi^2 = \sum_{i=1}^N \frac{(a_1 + a_2x_i - y_i)^2}{\sigma_i^2} \quad (3.3)$$

The name of the game is to choose values for the coefficients a_1 and a_2 to minimize the value of chi squared. What does that really mean? If you look at Fig 3.2, it is equivalent to minimizing the sum of the squares of the lengths of the green lines, which are the vertical distances between the observations (y_i) and the regression estimated values ($a_1 + a_2x_i$).

3.2.1 Doing Things the Hard Way (the Normal Equations)

Now how do we choose the coefficients a_1 and a_2 to minimize chi squared? Well, you need to recall back to your first year calculus course. The extrema (maxima or minima) of a function

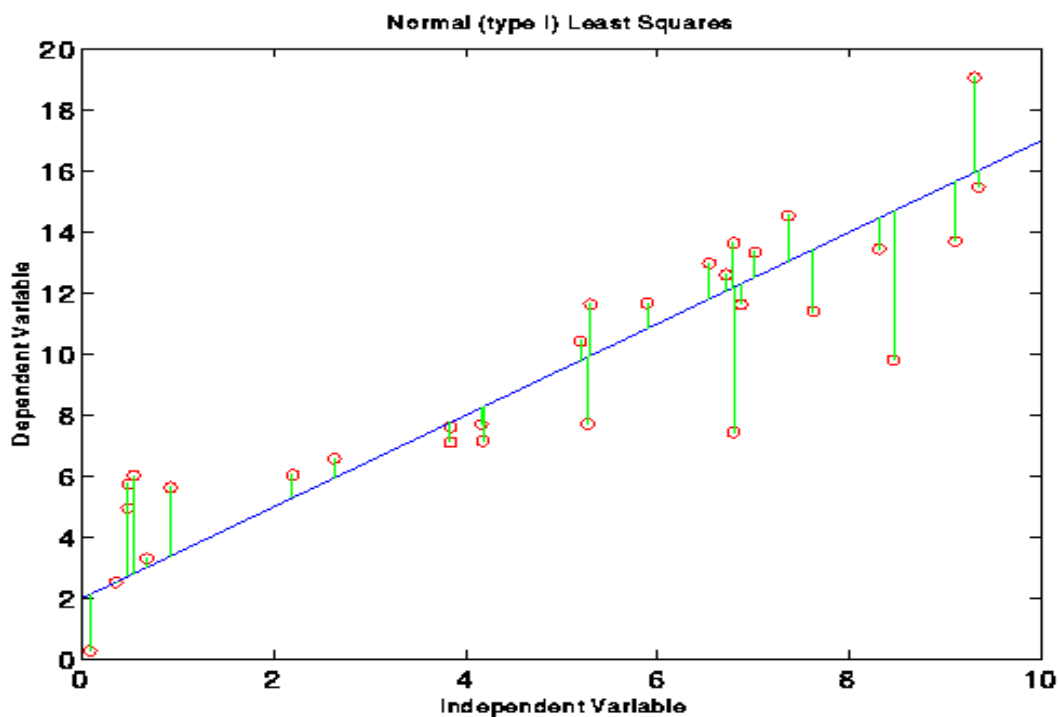


Figure 3.2: The basis of the least-squares method is to minimize the sum of the squared model data-misfit, for a type I regression taken as the vertical distance between the data points and the fit line.

is characterized by its first derivative going to zero. So if we differentiate the chi squared with respect to each of the coefficients and set the derivatives equal to zero, we have two equations in two unknowns. That is, we have

$$\frac{\partial \chi^2}{\partial a_1} = \sum_{i=1}^N \frac{2(a_1 + a_2 x_i - y_i)^2}{\sigma_i^2} = 0 \quad (3.4)$$

and

$$\frac{\partial \chi^2}{\partial a_2} = \sum_{i=1}^N \frac{2x_i (a_1 + a_2 x_i - y_i)^2}{\sigma_i^2} = 0 \quad (3.5)$$

This can be reduced to two equations in two unknowns (the coefficients),

$$S a_1 + S_x a_2 = S_y \quad (3.6)$$

$$S_x a_1 + S_{xx} a_2 = S_{xy} \quad (3.7)$$

where for short-hand notation, we have used the notation that:

$$S = \sum_{i=1}^N \frac{1}{\sigma_i^2}; \quad S_x = \sum_{i=1}^N \frac{x_i}{\sigma_i^2}; \quad S_y = \sum_{i=1}^N \frac{y_i}{\sigma_i^2}; \quad S_{xx} = \sum_{i=1}^N \frac{x_i^2}{\sigma_i^2}; \quad S_{xy} = \sum_{i=1}^N \frac{x_i y_i}{\sigma_i^2} \quad (3.8)$$

Keep in mind that the above are just numbers that you simply and mechanically compute from your data. You then have a pair of simultaneous equations that you can easily solve for your coefficients. You can solve this with MATLAB by first populating a matrix A with the sums (the S 's), such that:

$$A = \begin{pmatrix} S & S_x \\ S_x & S_{xx} \end{pmatrix} \quad (3.9)$$

and (for the right hand side of the equations),

$$b = \begin{pmatrix} S_y \\ S_{xy} \end{pmatrix} \quad (3.10)$$

and thus you solve the equations $Aa = b$ (here the unknowns to solve for are the coefficients and are in the column vector " a ") like we did in Chapter 1 with $a = A \setminus b$. Pretty simple, huh?

These equations are called the **normal equations** and can be generalized for higher order polynomial fits. For example, if you were to fit a quadratic equation, $y = a_1 + a_2 x + a_3 x^2$, the normal equations would be

$$\begin{aligned} S a_1 + S_x a_2 + S_{xx} a_3 &= S_y \\ S_x a_1 + S_{xx} a_2 + S_{xxx} a_3 &= S_{xy} \\ S_{xx} a_1 + S_{xxx} a_2 + S_{xxxx} a_3 &= S_{xxy} \end{aligned} \quad (3.11)$$

and so on. Do note one thing, however, that the size of the sums begin to mount very rapidly (for the quadratic, you are summing up the 4th power of x) and roundoff errors soon become a problem (see Chapter 2).

The solution to the normal equations for the straight line regression can be easily obtained in MATLAB using the $a = A \setminus b$ solution. Commonly, however, you'll find this solution solved by something called *Cramer's Rule* (we won't go into this here), which gives explicitly

$$a_1 = \frac{S_y S_{xx} - S_{xy} S_x}{\Delta} \quad (3.12)$$

$$a_2 = \frac{S S_{xy} - S_x S_y}{\Delta} \quad (3.13)$$

where we have defined the denominator (actually the determinant of A) as:

$$\Delta = SS_{xx} - S_x S_x \quad (3.14)$$

Remembering the concept of singular matrices, if A is singular, due to inadequate data, then the determinant will be zero, and the solutions to equations 3.12 and 3.13 will fail for obvious reasons. This is almost never a problem for straight-line fits but can be problematic for more complex situations, where the normal equations become “almost singular” (we will talk about this more later).

3.2.2 Uncertainties in Coefficients:

But we also need to know the uncertainties in the coefficients. This is a relatively straightforward thing to do, if a little tedious to calculate. But if you assume there are no systematic errors in your measurements (systematic errors induce covariances, not random errors), then you can derive that the uncertainty in any function $f(x)$ is given by:

$$\sigma_f = \sqrt{\sum_{i=1}^N \sigma_{x_i}^2 \left(\frac{\partial f}{\partial x}\right)^2} \quad (3.15)$$

But here, the function $f(x)$ that we are interested in is a_1 (or a_2), and the variable is y_i since that is the one that is uncertain. We differentiate equations 3.6 and 3.7 with respect to the observations y_i (*i.e.*, the contribution of each data point to the fit parameters) and substitute into the equation above to obtain

$$\sigma_{a_1}^2 = \frac{S_{xx}}{\Delta} \quad (3.16)$$

$$\sigma_{a_2}^2 = \frac{S}{\Delta} \quad (3.17)$$

Note something quite profound in the structure of equations 3.16 and 3.17; the size of the uncertainties in the coefficients depend not on the data you measured (the y_i 's) but on where you made the measurements (the x_i 's) and the uncertainties in the measurements (the σ 's). No summation incorporating y_i is involved! This says something about experiment design; obviously maximizing the Δ (the A determinant) in relation to the S_{xx} and S is a good thing. Maximizing the determinant is a question of maximizing the difference between S_{xx} and $(S_x)^2$, which is equivalent to maximizing the spread (range of x values) of your measurements. The larger the range in your data, the better you know the slope and intercept. That makes sense. Look again at the upper equation (3.16). The numerator (S_{xx}) says that the larger the 2nd moment of the x distribution (*i.e.*, the distance between the centroid of your data and the $x = 0$ axis, where the intercept is) the larger your intercept error. This also makes sense.

But where does the *number* of measurements come into this? You would expect that increasing the number of measurements (all other things being equal) would improve our knowledge of the

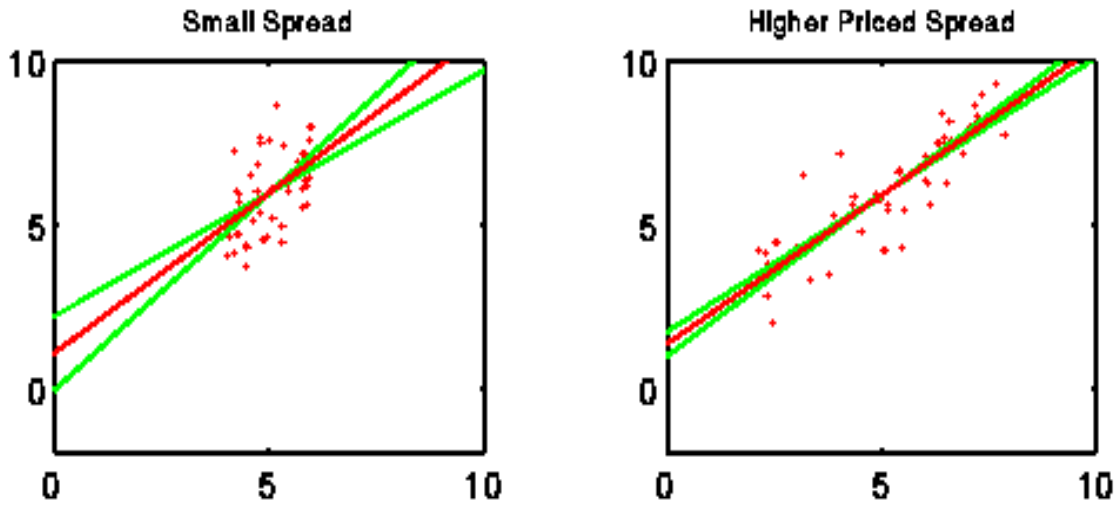


Figure 3.3: The error in the slope estimate depends on the x sampling locations; closely clumped samples in x will have a larger error.

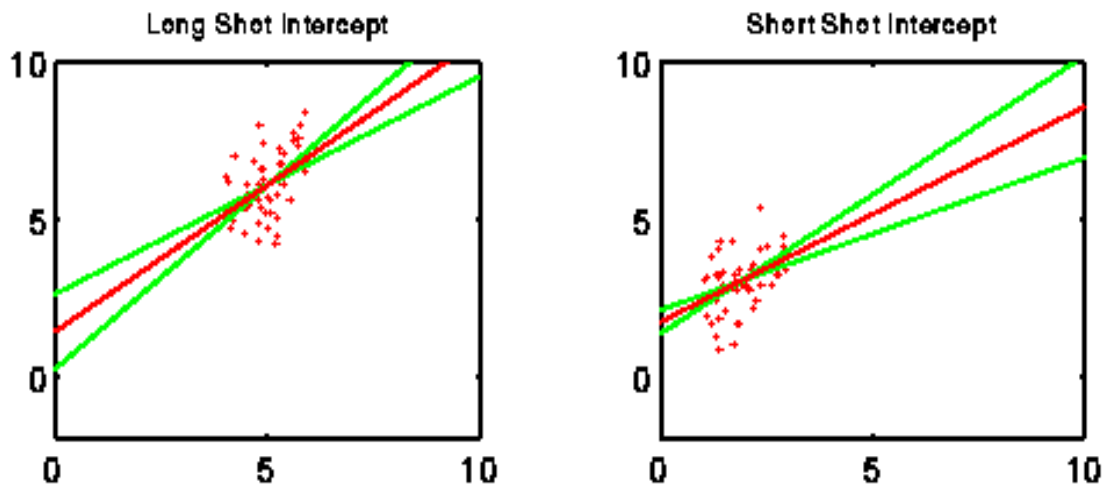


Figure 3.4: The error estimate for the intercept depends on how far the x sampling locations are from $x = 0$.

coefficients. It only seems fair. Well, N **does** come into the coefficients, but through the determinant Δ . The reason is not too subtle to argue here. Consider a random distribution of measurements (*i.e.*, various x_i 's) centered around zero (this argument works regardless of this stipulation, but it makes it easier to understand it this way). Note the terms in the equation that defines the determinant. The second term, is the square of S_x , which is the sum of x_i 's. This sum will be close to zero, since about half of the x_i 's will be negative, and the other half positive. The first term, S_{xx} which is the sum of the *squares* of the x_i 's will always be positive, and will continue to grow as the number of measurements increases. Thus Δ will always increase for increasing numbers of measurements. Figure 3.5 shows the uncertainties in the intercept (upper curve) and slope (lower curve) for a random group of measurements of varying numbers (horizontal axis).

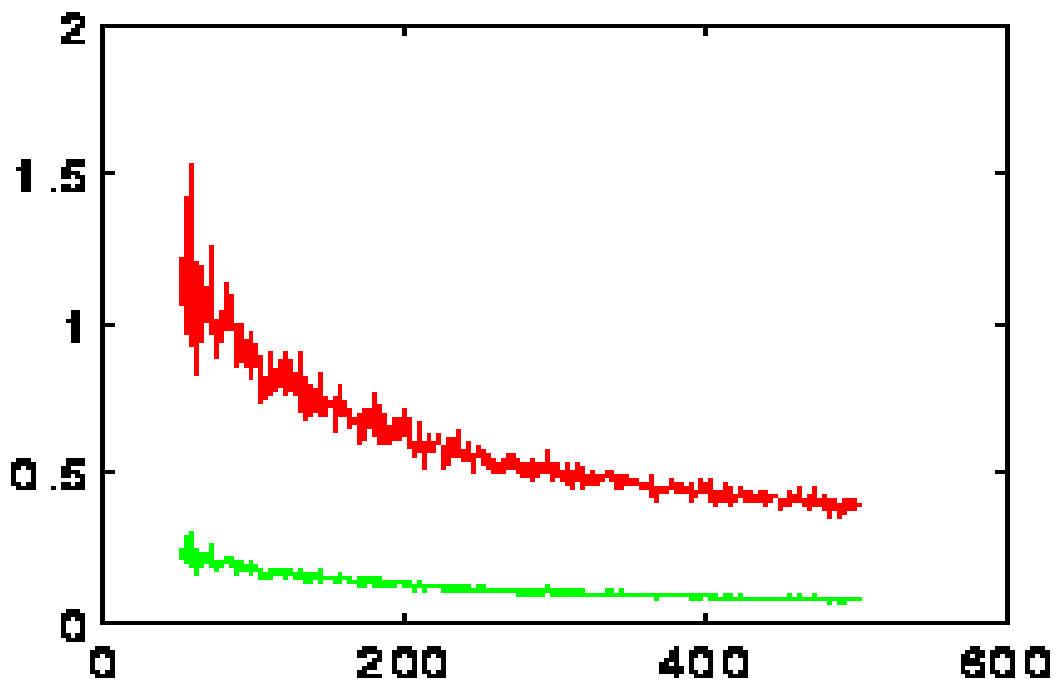


Figure 3.5: The uncertainty estimates (*y*-axis) for the intercept (upper, red line) and slope (lower, green line) parameters from a least-squares fit decrease as the number of data points increases (*x*-axis), shown here for randomly generated data.

You could easily do this yourself because we have a MATLAB routine called [linfit.m](#), which you can download and use as a general straight line fitting routine. It is very simple to use, requiring three arguments (x , y , and s_y) that you need to supply. It returns the coefficients, uncertainties in the coefficients, the covariance of intercept on slope, and the linear correlation

coefficient.

3.2.3 Uncertainties in an Estimated Y-value

Now that you have determined the coefficients of your straight-line fit, and the uncertainties in those coefficients, you might want to go ahead and calculate some best fit value of y_e at some desired location x_e . That's easy, $y_e = a_1 + a_2x_e$ does the trick. But how well do you know this new estimate? Your first instinct is to use the error propagation equation, but that estimate:

$$\sigma_e^2 = \sigma_{a_1}^2 + \sigma_{a_2}^2 x_e^2 \quad (3.18)$$

is **fundamentally wrong!** You have left out the fact that the uncertainties in the intercept (a_1) and the slope (a_2) are *correlated* and a third term enters into the above equation involving the covariance of the two uncertainties:

$$\sigma_e^2 = \sigma_{a_1}^2 + \sigma_{a_2}^2 x_e^2 + 2\sigma_{a_1 a_2} x_e \quad (3.19)$$

This is a specific case of *error propagation*, which gives the uncertainty in a function f due to uncertainties in variables x_i as:

$$\sigma_f^2 = \sum_{i=1}^m \sum_{j=1}^m \sigma_{ij}^2 \left(\frac{\partial f}{\partial x_i} \right) \left(\frac{\partial f}{\partial x_j} \right) \quad (3.20)$$

where m is the number of variables and where i and j vary over all possible combinations of the variables. The factor of two in the last term of the explicit equation above is because the covariance of $a_1 a_2$ is the same as the covariance of $a_2 a_1$ (the covariance matrix is always symmetric) and thus enters twice into the summation.

The $\sigma_{a_1 a_2}^2$ in the very last term of eqn 3.19 is the *covariance* term that is returned from [linfit.m](#), and is defined by:

$$\sigma_{a_1 a_2}^2 = -\frac{S_x}{\Delta} \quad (3.21)$$

Look again at Figs 3.3 and 3.4 above, and note how the intercept changes with the slope. The only time that the two uncertainties are uncorrelated is when the x -variable is normalized so that its mean is zero. Of course, this is *only* applicable if all the errors in the y -variable are equal. You can picture the slope pivoting around the centroid of the data cloud, and since the intercept now resides at the centroid, the two coefficients are independent. If you do this, then $S_x = 0$, and equations 3.12, 3.13, and 3.14 reduce to:

$$\begin{aligned} \Delta &= SS_{xx} \\ a_1 &= S_y \\ a_2 &= \frac{SS_{xy}}{S_{xx}} \end{aligned} \quad (3.22)$$

And the uncertainties reduce to: .

$$\begin{aligned}\sigma_{a_1}^2 &= \frac{1}{S} \\ \sigma_{a_2}^2 &= \frac{1}{S_{xx}}\end{aligned}\quad (3.23)$$

which makes life a little simpler. So all you need to do is to subtract the mean of x from your x -values before you do the regression and add it back in before you calculate your results. It pays to normalize! In the last chapter we normalized by dividing each x by the square root of the sum of the squares of all x , but subtracting the mean also works too. It all depends on *units*, if the x and the y are in the same units this trick will work, but if they are in different units it is better to *standardize*.

3.2.4 Type II Regressions (Two Dependent Variables)

The regression performed above presumes that you know x infinitely well. That is one way of defining the independent variable. What happens when both y and x have uncertainties? Do you regress y against x , or x against y ? It turns out that *neither* is correct. You can prove this experimentally by taking the same data set and doing both. In a perfect world, the slope of y regressed against x should be the inverse of the slope of x regressed on y . When you try this for a synthetic data set that has some noise in it, however, you get a difference (as seen in Fig. 3.6).

Then the process of minimizing the vertical distances between your data and the fit line is not correct, and you should be minimizing the **perpendicular distance** between your data points and the regression line. This sounds simple, and it is conceptually. But the math becomes a little tangled because the equations corresponding to those for a *type I regression* above are now non-linear and require an iterative solution. There is a simple approach, however, that works well. You do the following for a data set called x and y :

1. Regress y on x and obtain a slope, which we'll call m_{yx} ;
2. Regress x on y and obtain an inverse slope, call it m_{xy} ;
3. Calculate the new slope as the geometric mean of the combined fits:

$$a_2 = \sqrt{\frac{m_{yx}}{m_{xy}}}\quad (3.24)$$

4. From this new slope, calculate the new intercept from:

$$a_1 = \bar{y} - \bar{x}a_2\quad (3.25)$$

There are other, more general techniques, which we can use (see for example Press *et al.*, 1992, *Numerical Recipes*). The reader is also directed toward a MATLAB m-file on our WWW site that was developed by Ed Peltzer at WHOI called [lsqfitma.m](#), which uses one of these techniques.

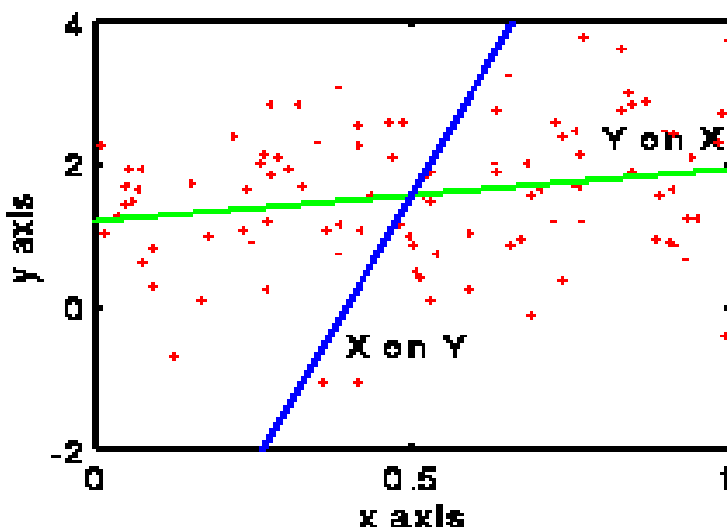


Figure 3.6: *Type I regressions (assume one independent variable and one dependent variable) can give very different slope estimates depending on whether y is regressed on x or x is regressed on y . For many real world applications, both variables have error and a Type II regression should be used.*

For the full meal deal (which deals with the nasty math), we also have a more advanced least squares routine known as the “*least squares cubic*” (York, 1966). However, this routine requires that you have error estimates of both the x and y for each point; you can find this routine at lsqcubic.m.

3.3 General Linear Least Squares Technique

3.3.1 Choose your model functions wisely

It is possible to set up the *normal equations* for any arbitrary set of functions (just as we did for the straight-line formula above). The difficulty is that the more complicated the functions, the more difficult the formulation, and the more the risk that the solution to the normal equations becomes *ill-behaved*. This happens when the functions chosen are not very orthogonal (that is they are more similar than not), or when the measurements actually made are not *well posed* to distinguish between the differing functions. The latter happens more often than people realize. Finally, the choice of functions may not really do a good job of matching the actual observations. Regardless of the underlying reason, when this happens the solutions end up being a delicate balance between very large numbers. This may yield the minimum χ^2 for the actual measurements,

but outside of the range of the measurements, or sometimes in between data points (if there are gaps) the solution does rather strange things. A classic example (and often the worst offender) is a polynomial regression to an abruptly changing data set, which is otherwise very quiescent. Consider, for example a step function in data, which you try desperately to fit with increasing order of polynomial.

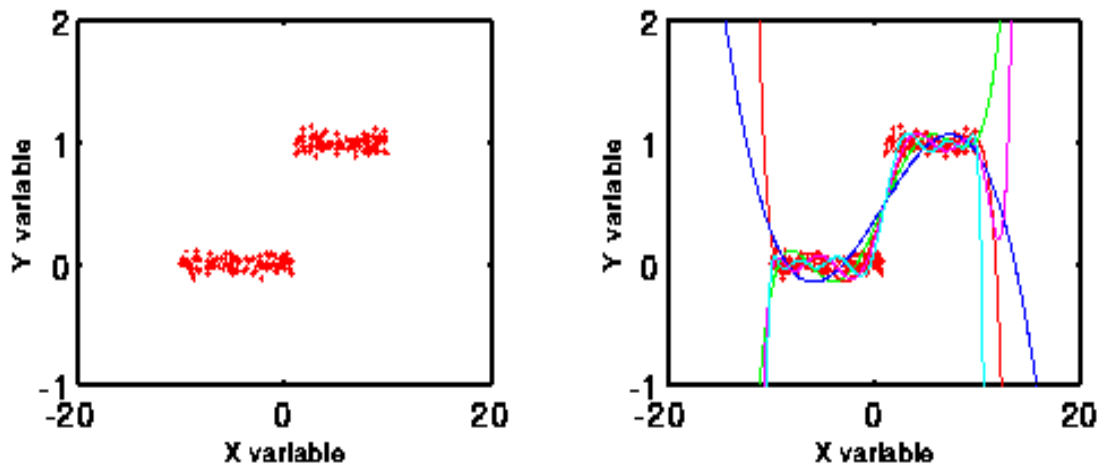


Figure 3.7: Discontinuities in data set such as the step function in the left panel can lead to all sorts of problems for fitting routines. Also note that once you get beyond the range of data either to the left or the right, the polynomial fits go haywire. Don't use polynomial fits for extrapolation!

The left hand figure (in Fig. 3.7) shows the data, which is 0 for x less than 0, and 1 for x greater than 0 (with a little noise added on). The right hand plot shows the efforts of various order polynomial fits ranging from order 3 (cubic) up to 13th order. Note that you have to go to very high order (the cyan curve is 13th order) to begin to approximate the step function's sharpness, but the price you pay is tight little oscillations where you have data (kind of an induced "ringing") plus wildly aberrant behavior outside of the data range. The latter may be particularly troublesome if you wish to try extrapolating your fits beyond your measurement range.

3.3.2 There Is an Easier Way: the Design Matrix Approach

The solution by means of the normal equations can be unstable, and it is rather tedious to have to build the analytical forms for them when the functions are more complicated. Now we'll show you an easier, numerical way. By the way, the term "*Linear Least Squares*" means not that the functions you are fitting are linear, but rather that the formulation is linear in the coefficients. Thus you could have a function to fit that looks like, for example:

$$y = a_1 + a_2x^2 + a_3 \exp\left(\frac{x}{\pi}\right) + a_4 \sin(0.17x) \quad (3.26)$$

which can be fit with linear least squares, as long as there are no unknown coefficients inside the parenthetically closed arguments to the non-linear functions. For example, the following is definitely a nonlinear regression prospect:

$$y = a_1 \sin(a_2x) + a_3x \quad (3.27)$$

The culprit is the a_2 term, which appears as part of the *argument* of a non-linear function. We will deal with this kind of problem in section 3.4.

Now about this supposed easy way. Well, think of constructing a series of simultaneous equations with one equation for each measurement (we'll use the nasty little equation we gave as an example):

$$\begin{aligned} y_1 &= a_1 + a_2x_1^2 + a_3 \exp\left(\frac{x_1}{\pi}\right) + a_4 \sin(0.17x_1) \\ y_2 &= a_1 + a_2x_2^2 + a_3 \exp\left(\frac{x_2}{\pi}\right) + a_4 \sin(0.17x_2) \\ &\vdots \\ y_N &= a_1 + a_2x_N^2 + a_3 \exp\left(\frac{x_N}{\pi}\right) + a_4 \sin(0.17x_N) \end{aligned} \quad (3.28)$$

The above can be represented by a matrix called the **design matrix**, which would look like:

$$A = \begin{pmatrix} 1 & x_1^2 & \exp\left(\frac{x_1}{\pi}\right) & \sin(0.17x_1) \\ 1 & x_2^2 & \exp\left(\frac{x_2}{\pi}\right) & \sin(0.17x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N^2 & \exp\left(\frac{x_N}{\pi}\right) & \sin(0.17x_N) \end{pmatrix} \quad (3.29)$$

It would be a $N \times 4$ matrix (N measurements and 4 columns). One way to think about the problem is that the design matrix is just a set of basis functions that we are trying to fit to the data. Remember that all of the elements in this design matrix are simply numbers that you calculate from your x -data. Then you would have a column vector consisting of your 4 unknown coefficients (which you want to solve for):

$$\vec{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} \quad (3.30)$$

And the column vector of your N observations:

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{pmatrix} \quad (3.31)$$

The matrix version of these equations now reduces to $Aa = y$. Looks simple, doesn't it? But what about the weighting factors (the σ 's)? Well, just like the straight line case, you just divide your x entries and your y entries by σ^2 . We won't repeat the matrices listed above, but for example the i^{th} row of the design matrix would look like:

$$\left(\frac{1}{\sigma_i^2} \quad \frac{x_i^2}{\sigma_i^2} \quad \frac{\exp\left(\frac{x_i}{\pi}\right)}{\sigma_i^2} \quad \frac{\sin(0.17x_i)}{\sigma_i^2} \right) \quad (3.32)$$

and the i^{th} element of the y -vector would also be divided by σ_i^2 . Everything else proceeds as before.

3.3.3 Solving the Design Matrix Equation with SVD

Now that we've shown you how to build a design matrix, *etc.*, we'll show you what to do. You cannot uniquely solve this as a set of simultaneous equations because it is overdetermined. That is, there are more equations than unknowns. Now since data is always imperfect, the data points will never agree on the true coefficients. The equations will always be inconsistent to some extent. This is equivalent to saying that not all the points will all lie on the regression line. But what you want to do is to minimize the square of the distance between the regression function and the observations. That is, you want to minimize the function $(Aa - y)^2$. This is exactly what singular value decomposition does for you. So in MATLAB you enter the commands (after constructing the matrices, of course)

```
[U,S,V] = svd(A,0);           % SVD of design matrix
W = diag(1./diag(S));         % not checked for zero singular values
a = V*W*(U'*y);              % your coefficients!
Covmat = V*W.^2*V';          % compute covariance matrix
[n m] = size(A);              % size of design matrix (row X column)
redchisqr = sum((A*a-y).^2)/(n-m); % compute reduced chi squared
sa = sqrt(redchisqr*diag(Covmat)); % uncertainties in coefficients
```

And that's your answer. Note that the primes mean matrix transposes in MATLAB and in the second line we've skipped an important step of checking for zeroes in the singular value list before inverting. The second line is a bit tricky: it is equivalent to taking the diagonal of S , inverting the individual elements of the resulting vector, and then reconstructing a square matrix with those new elements on the diagonal. The calculations look obscure, but they are efficient and powerful. You

can use this 6 lines of code as an engine to a general linear least squares regression program. All you need to do is build the design matrix and data vector for the specific linear model you want to fit.

3.3.4 Multidimensional Regressions

What if you want to fit your data to higher dimensions? For example, suppose you wanted to model the distribution of dissolved oxygen at some depth level in the North Atlantic. You might do this, for example, if you had observations in an area of the North Atlantic and you were interested in calculating the large scale gradient (the rate and direction of change with distance). Supposing further that you believed that the distribution was best fit with a biquadratic function, that is, a function which was quadratic in both x (longitude) and y (latitude). Then your data, z would be modeled after:

$$z = a_1 + a_2x + a_3x^2 + a_4xy + a_5y + a_6y^2 \quad (3.33)$$

Be careful not to be confused here, because we are now using “ y ” as an independent variable, rather than an observation, and we have introduced “ z ” as your dependent variable (observation). Actually, this whole thing sounds more complicated than it is, but it is mathematically the same as the general linear least squares for which you already have the equations. You just calculate your design matrix just like before, as

$$A = \begin{pmatrix} \frac{1}{\sigma_1^2} & \frac{x_1}{\sigma_1^2} & \frac{x_1^2}{\sigma_1^2} & \frac{x_1 y_1}{\sigma_1^2} & \frac{y_1}{\sigma_1^2} & \frac{y_1^2}{\sigma_1^2} \\ \frac{1}{\sigma_2^2} & \frac{x_2}{\sigma_2^2} & \frac{x_2^2}{\sigma_2^2} & \frac{x_2 y_2}{\sigma_2^2} & \frac{y_2}{\sigma_2^2} & \frac{y_2^2}{\sigma_2^2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{\sigma_N^2} & \frac{x_N}{\sigma_N^2} & \frac{x_N^2}{\sigma_N^2} & \frac{x_N y_N}{\sigma_N^2} & \frac{y_N}{\sigma_N^2} & \frac{y_N^2}{\sigma_N^2} \end{pmatrix} \quad (3.34)$$

the right hand data column vector would be:

$$z = \begin{pmatrix} \frac{z_1}{\sigma_1^2} \\ \frac{z_2}{\sigma_2^2} \\ \vdots \\ \frac{z_N}{\sigma_N^2} \end{pmatrix} \quad (3.35)$$

and your coefficient matrix is given by:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{pmatrix} \quad (3.36)$$

which you then solve with the MATLAB code we showed you above. Not too hard. You might want to look at an m-file called [surfit.m](#) on our Web Site, which does an unweighted two dimensional fit to an arbitrary order polynomial, to see how you might extend it to weighted fits (*i.e.*, with different σ_i 's for each data point).

3.3.5 Transformably Linear Models

There is one obvious case where the model equation is non-linear in its coefficients, but you can transform your data to make the model linear. Consider the model equation:

$$y = a_1 e^{a_2 x} \quad (3.37)$$

If you take the logarithm of this equation, it reduces to:

$$\log(y) = \log(a_1) + a_2 x \quad (3.38)$$

So all you have to do is to take the log of your results (y_i 's), do a linear fit, and transform the first coefficient by taking the `exp` of the intercept. You can thus avoid doing non-linear fits sometimes in this fashion. We refer you back to Chapter 2 for a more complete discussion on the *log-normal* distribution and some of its properties.

3.3.6 Non-Coefficients

Also, beware the “*non-coefficient*” problem. That is, you can sometimes introduce two model functions that are identical in behavior. The net result is that the normal equations become *exactly singular*. The SVD approach above, however, will give you an answer (after telling you there is a problem by giving you a zero singular value). Consider the following equation:

$$y = a_1 e^{(a_2 + a_3 x)} \quad (3.39)$$

which looks reasonable, until you realize that a_1 and e^{a_2} are essentially the same basis functions (both are constants multiplying the $e^{a_3 x}$ term, *i.e.* $y = a_1 e^{a_2} e^{a_3 x}$). You need to think carefully about the basis function you've built, the problem of non-coefficients happens even to the best of us.

3.4 Non-Linear Least Squares Techniques

3.4.1 Iterative Techniques (Like a Rolling Stone ...)

What happens when the model equations are non-linear in the coefficients? You do the same thing: **MINIMIZE CHI SQUARED** (χ^2). The difference is that you have to do it iteratively. There is no simple way of writing down and solving a set of linear analytic equations.

Most non-linear fitting routines systematically search the coefficient space for a χ^2 minimum by repeatedly calculating the χ^2 for your model equation and data and then nudging the coefficients in differing directions until they reach a minimum in χ^2 (or as close to a minimum as you stipulate). The only difference between the techniques is the precise search mechanism, which can range from very crude to quite elaborate. Bevington and Robinson (1992, chapter 8) describes three basic approaches: grid search, gradient search, and expansion methods. The grid search and expansion methods work well when you are near the χ^2 minimum, but are not very efficient or effective at moving large distances in coefficient space. The gradient search moves large distances well, since it travels down the path of steepest descent, adjusting all the coefficients at once. But it tends to sometimes get trapped in long valleys and doesn't converge well near the global χ^2 minimum.

The more sophisticated routines tend to use gradient search routines initially, which adjust the size of the nudge that they give to the coefficients to the size of the change in χ^2 . Near the minimum they may perform a grid search, and the last effort usually involves some kind of polynomial expansion (interpolation) of the χ^2 surface. The price is that you need to supply them with the means not only to evaluate your model function for your data (x_i 's) but also the gradient of χ^2 . A popular method is the "*Levenberg-Marquardt Method*". We won't go into the details right now.

Think of the algorithm like a marble rolling around on a surface of hills and valleys (a little like the arcade game "marble madness"). The ball will continue to roll down hill until it reaches the lowest point (sometimes oscillating around the minimum as it goes, depending on the character of the algorithm used).

All of these routines require you to provide:

- an initial guess of values for your coefficients (often done by eyeball);
- the size of the incremental changes in the coefficients that you expect to make;
- how small a change in χ^2 that you would consider convergence;
- the name of the function (m-file) that computes your model results;
- sometimes: the maximum number of iterations;
- sometimes: the name of the function (m-file) that computes the model χ^2 gradient.

Keep one thing in mind. Nothing works better than a good initial first guess. You might get a little sloppy and tend to let the computer do the work, but the χ^2 surface in coefficient hyperspace may have more than one minimum, and your algorithm may get trapped in a local minimum, which is not nearly as good as a global minimum perhaps just over the next "hill" in hyperspace. As a corollary you should always plot your results: the eyeball is not often fooled. You may not be able to distinguish precise fits, but you can tell if the fit is far from optimal. Sometimes it is worthwhile

to do a grid search of initial starting values, or to approach the optimum from extremely different directions (*e.g.*, from positive, then negative values).

If your problem is particularly plagued by “false minima”, one alternative method to consider is *simulated annealing* (see Press *et al.*, 1992, *Numerical Recipes* for more discussion and example programs). In this and related techniques, the linear (“straight-down-the-hill”) approach is somewhat abandoned for a more probabilistic view. In essence, the search routine starts by hopping randomly around parameter-space computing χ^2 and then iteratively hones in on the hoped for (but not rigorously provable) “global minimum”. These searching techniques are practical for small to moderately sized problems, but be forewarned that the same caveats apply as for the grid and gradient searches.

For large data sets and/or complicated non-linear functions with many coefficients, the iterative techniques are computationally expensive. You may find that you have no choice. Keep in mind that a sensible choice of model equations, based on a realistic physical or chemical model of the system you are fitting will take you much further than any sophisticated mathematics or powerful computers. **Common sense** is a powerful ally.

3.4.2 Uncertainties in Coefficients

Computing the uncertainties in coefficients for non-linear regressions is a challenging problem. It involves examining the shape of the χ^2 surface in coefficient hyperspace and estimating the boundaries of confidence intervals associated with “elevation” changes in χ^2 . Many sophisticated routines return the covariance matrix (for the coefficients) on convergence from which the associated parameter uncertainties are computed. In addition to the excellent discussion by Bevington and Robinson (1992) in chapter 8, the reader is referred to section 15.6 of *Numerical Recipes* for a discussion by Press *et al.* (1992).

3.4.3 Example Calculation: Gaussian on a Constant Background

To give you an idea of how this might work, consider a data set which you suspect to be a Gaussian shaped peak embedded in a constant background, with some measurement noise. The measurements are represented by the dots in Fig. 3.8.

The first thing you do is create a MATLAB m-file whose sole purpose is to compute your model function. You can call it anything you want, but here we will call it “modfunc.m”. It is a function that takes as its arguments the x data and vector containing the coefficients and returns the model fitted data. The m-file in this example contains a whole two lines:

```
function out=modfunc(x,p)
out = p(1) + p(2)*exp(-((x-p(3))/p(4)).^2);
```

Note the following about this m-file. First the `function` statement makes it a function call rather than just a simple script. You need to do this if you want it to return any values to MATLAB. Second, note the “dot” before the “caret”. This way you don’t end up getting a matrix from

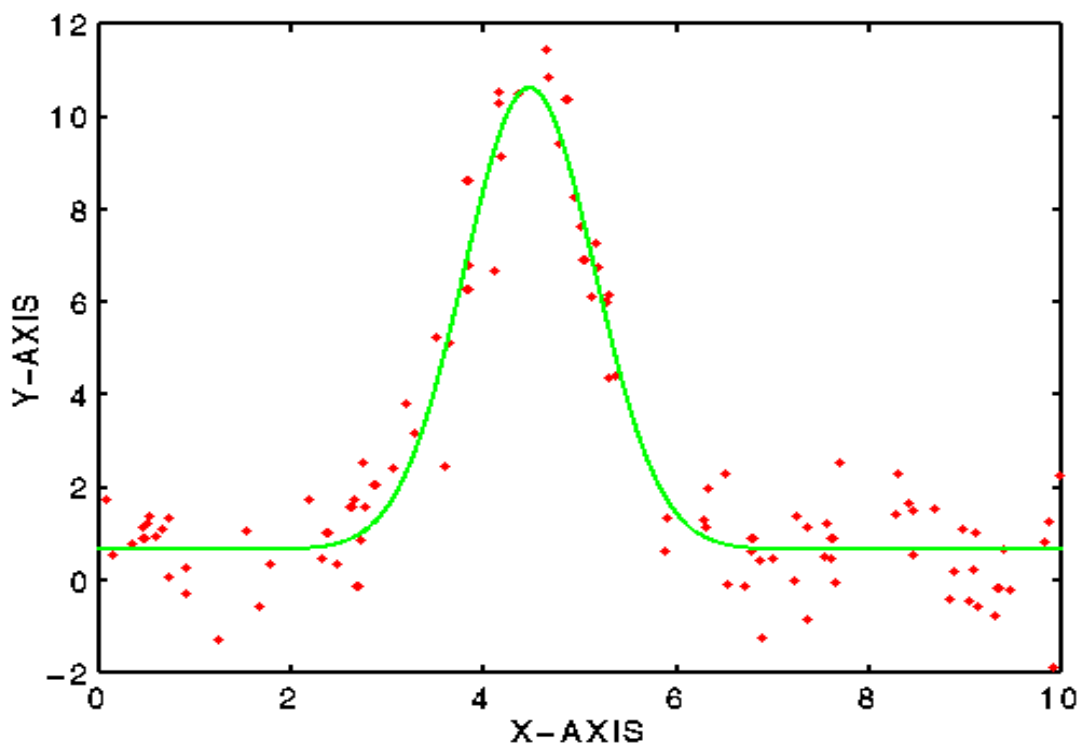


Figure 3.8: For some problems, such as the Gaussian peak in this data set, non-linear least-squares fitting methods are required.

squaring “ x ”. This statement calculates all of the y ’s from the x ’s at once. The above is equivalent to the following mathematical equation:

$$y_f = a_1 + a_2 e^{\left(-\frac{(x-a_3)^2}{2a_4^2}\right)} \quad (3.40)$$

Next, you download from our Web Site a non-linear least squares fitting routine called [nlleasqr.m](#) and a helper routine called [dfdp.m](#), which you also need. Both files ultimately came from the MathWorks Web Site, but have been fixed up a little to remove some bugs. The first file is the main engine. The second is a program that calculates the χ^2 gradient in coefficient space. It does this numerically rather than analytically. You can make your own analytic function (and call it some distinct name), which may do a better job, but it is not really necessary. Finally, there is the data [x.dat](#) and [y.dat](#) (how original!).

OK, now you have the m-files, you start up MATLAB and load in the x and y data. You then plot it up, since you need to have some idea of your initial starting guesses for parameters. You

eyeball the plot and guess what the baseline value should be (this is a_1) which we'll guess to be 1. You then guess the height of the Gaussian (a_2) to be 8. The center of the Gaussian (a_3) we'll guess to be at 5, and its width (a_4) to be 2. (Note that from equation 3.40 $a_4 = \sigma$, but watch out for that $\sqrt{2}$). So we set our initial parameter guess to be equal to `pin`.

```
load x.dat
load y.dat
pin=[1 8 5 2];
```

Note that the order is important and should be the same as you use in the m-file `modfunc.m`. Next, you type `help nllleastqr` to find out how to feed it information. Note that most of the input parameters are optional, so don't be intimidated, and don't wear out your fingers. We'll just try:

```
[f,p,kvg,iter,corp,covp,covr,stdresid,Z,r2] = nllleastqr(x,y,pin,'modfunc');
```

The stuff on the left is all the goodies you get back, but we'll only look at a few of them. Your "answer" or coefficient vector is stored in `p`. First, let's look at what kind of job this routine did for us. Let's plot it:

```
xf=[0:.1:10];           % dummy array to plot
yf=modfunc(xf,p);       % compute fit for the dummy array
plot(x,y,'*',xf,yf)     % plot both data and fit on same graph
```

Hmmm, not bad (see the green line in Fig. 3.8). About the numbers: `p` is the output coefficients, `kvg` is a flag to say if convergence was achieved before the routine gave up, `iter` is the number of iterations, `covp` is the covariance matrix for the coefficients (the square roots of the diagonal elements are the uncertainties in the parameters, and `r2` is the overall correlation coefficient. For this example, we get:

```
coefficients = [0.6595    9.9624    4.4780    0.9561]
uncertainties = [0.1220    0.3760    0.0249    0.0466]
r2 = 0.9160
```

Oh yes, the "real" values were `[0.5 10.0 4.50 1.0]`. The answer is roughly within errors of the real values, but will differ because of the noise that was added to the data. Your mileage may differ slightly when you do this experiment due to roundoff error.

3.5 Problems

All of your problems sets are served from the web page:

```
http://eos.who.i.edu/12.747/problem\_sets.html
```

which can be reach *via* a number of links from the main course web page. In addition, the date the problem set comes out, the date it is due, and the date the answers will be posted are also available in a number of locations (including the one above) on the course web page.

References

- Bevington, P.R. and D.K. Robinson, 1992, *Data Reduction and Error Analysis for the Physical Sciences*, 2nd Edition, McGraw-Hill Inc., New York, NY, 328 pp.
- Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, 1992, *Numerical Recipes*, 2nd Edition, Cambridge University Press, New York, , 818 pp.
- York, D., 1966, Least-squares fitting of straight lines, *Canad. J. Phys.*, **44**, 1079–1086.