LEAST-SQUARES LITE FOR THE BUDDING AFICIONADO: ART AND PRACTICE

©Carl Heiles and Deepthi Gorthi[1]
February 8, 2018

In our never-ending attempt to make your life easier, we present you with this highly instructive, time-saving, and labor-saving informative document! Here we give heuristic derivations, discussions, examples, and the prescription for doing least-squares the easy way using matrix techniques generally, and specifically in Python. This prescription is given as an example in §4, and the *power-user* can skip the details and go directly there.

This document contains selected sections from the more complete *LEAST-SQUARES AND CHI-SQUARE FOR THE BUDDING AFICIONADO: ART AND PRACTICE*, which also covers advanced topics including covariance in chi-square fitting, singular value decomposition, median and Minimum Weighted Absolute Residuals Sum (MWARS) fitting, and fitting when both x and y have uncertainties. The full document runs to 62 pages, which is a little intimidating for the novice. We occasionally refer to the books Bevington and Robinson (1992; BR), Cowan (1998), Press et al. (2001; Numerical Recipes, NR) and Taylor (1997; T97), and we update the notation to partially conform with NR.

We begin with least-squares in the classic sense, meaning we minimize the sum of squares instead of minimizing χ^2[2]. In astronomy, more often than not you don't have an independent assessment of the intrinsic uncertainty in the data, which means you cannot evaluate χ^2, and the least squares approach is the only option. However, often in astronomy you do want to weight observations differently, e.g. because of integration time, and this requires an approach similar to the χ^2 one. In later sections we generalize to the χ^2 and this other weighted-observations case.

Contents

[1]Translation from IDL to Python

[2]The χ^2 is the sum of squares down-weighted by uncertainity in the measurement. Roughly because, poor measurements should influence the outcome to a smaller extent

0. LEAST-SQUARES FITTING FOR TWO PARAMETERS, AS WITH A STRAIGHT LINE.

0.1. The closed-form expressions for a straight-line fit

First consider the least-squares fit to a straight line. Let y_m be the m^{th} measurement of the observed quantity (in this example, y_m is distance; t_m be the time of the m^{th} measurement; $M =$ the total number of observations, i.e. m runs from 0 to $M-1$. Remember that in the least-squares technique, quantities such as t_m are regarded to be known with high accuracy while the quantity y_m has uncertainties in its measurement.

We expect distance y_m to change linearly with time as follows:

$$A + Bt_m = y_m \tag{0.1}$$

Given this, one does the maximum likelihood (ML) estimate assuming Gaussian statistics. When all measurements have the same intrinsic uncertainty, this is the same as looking for the solution that minimizes the sum of the squares of the residuals (which we will define later). This leads to the pair of equations (Taylor 8.8, 8.9), called the *normal equations*

$$AN + B \sum t_m = \sum y_m \tag{0.2a}$$

$$A \sum t_m + B \sum t_m^2 = \sum t_m y_m \ . \tag{0.2b}$$

Two equations and two unknowns—easy to solve! The closed-form equations for (A, B) are Taylor's equations 8.10 to 8.12.

0.2. Better is the following generalized notation.

We want a way to generalize this approach to include any functional dependence on t and even other variables, and to have an arbitrarily large number of unknown coefficients instead of just the two (A, B). This is very easy using matrix math. We will ease into this matrix technique gently, by first carrying through an intermediate stage of notation.

First generalize the straight-line fit slightly by having two functional dependences instead of one. We have something other than the time t_m; call it s_m. For example, we could have $s_m = \cos(t_m)$ or $s_m = t_m^2$; or we could have $s_m = x_m$, where x_m is the position from which the observation was taken. To correspond to equation 0.1, $s_m = 1$. Then we rewrite equation 0.1 to include this extra dependence

$$As_m + Bt_m = y_m \ . \tag{0.3}$$

There are still only two unknown parameters, so this is an almost trivial generalization; later we'll generalize to more parameters.

We have M equations like equation 0.3, one for each measurement. They are known as the *equations of condition* because they are the equations that specify the theoretical model to which we are fitting the data. There are M equations of condition and only two unknowns (A and B). This is too many equations! We have to end up with a system in which the number of equations is equal to the number of unknowns.

To accomplish this, from equation 0.3 we form the *normal equations.* The number of normal equations is equal to the number of unknowns, so in this case we will have two. We could carry through the same ML derivation to derive equations equivalent to equation 0.2; the result is

$$A \sum s_m^2 + B \sum s_m t_m = \sum s_m y_m \tag{0.4a}$$

$$A \sum s_m t_m + B \sum t_m^2 = \sum t_m y_m \ . \tag{0.4b}$$

We can rewrite these equations using the notation $[st] = \sum s_m t_m$, etc.:

$$A[s^2] + B[st] = [sy] \tag{0.5a}$$

$$A[st] + B[t^2] = [ty] \ . \tag{0.5b}$$

This is, of course, precisely analogous to equation 0.2. And now it's clear how to generalize to more parameters!

1. LEAST-SQUARES FITTING FOR MANY PARAMETERS, AS WITH A CUBIC

With this notation it's easy to generalize to more (N) unknowns: the method is obvious because in each equation of condition (like equation 0.3) we simply add equivalent additional terms such as Cu_m, Dv_m, etc; and in the normal equations (equation 0.5) we have more products and also more normal equations.

Let's take an example with four unknowns ($N = 4$), which we will denote by A, B, C, D; this would be like fitting a cubic. With $N = 4$ we need at least five datapoints ($M = 5$), so there must be at least five equations of condition. The generalization of equation 0.4 is the M equations

$$As_m + Bt_m + Cu_m + Dv_m = y_m \ , \tag{1.1}$$

with $m = 0 \to (M - 1)$. Again, the least-squares-fitting process assumes that the s_m, t_m, u_m, v_m are known with zero uncertainty; all of the uncertainties are in the measurements of y_m. We then form the four normal equations; the generalization of equation 0.5 written in matrix format is:

$$\begin{bmatrix} [ss] & [st] & [su] & [sv] \\ [ts] & [tt] & [tu] & [tv] \\ [us] & [ut] & [uu] & [uv] \\ [vs] & [vt] & [vu] & [vv] \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} [sy] \\ [ty] \\ [uy] \\ [vy] \end{bmatrix} \tag{1.2}$$

The $N \times N$ matrix on the left is symmetric. With N equations and N unknowns, you can actually *solve* for A, B, C, D!

2. FAR, FAR BEST AND EASIEST: MATRIX ALGEBRA

The above equations are terribly cumbersome to solve in a computer code because they require lots of loops. However, it becomes trivial if we use matrices. Here we designate a **matrix** by **boldface** type.

We illustrate the matrix method by carrying through the above $N = 4$ example, and we assume that there are 5 independent measurements ($M = 5$). We first define the matrices

$$\mathbf{X} = \begin{bmatrix} s_0 & t_0 & u_0 & v_0 \\ s_1 & t_1 & u_1 & v_1 \\ s_2 & t_2 & u_2 & v_2 \\ s_3 & t_3 & u_3 & v_3 \\ s_4 & t_4 & u_4 & v_4 \end{bmatrix} \tag{2.1a}$$

$$\mathbf{a} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \tag{2.1b}$$

$$\mathbf{Y} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \tag{2.1c}$$

so, in matrix form, the equations of condition (equation 1.1) reduce to the single matrix equation

$$\mathbf{X} \cdot \mathbf{a} = \mathbf{Y} \ . \tag{2.2}$$

The notation for these equations corresponds to NR's. We write them with subscripts σ to emphasize that they are calculated without dividing by σ_{meas}, i.e. that we are doing least squares instead of chi-square fitting. For chi-square fitting, see §8 and 8.1.

Our matrix **X** corresponds to NR's "design matrix" **A** of Figure 15.4.1, except that our elements are not divided by $\sigma_{meas,m}$, and the matrix equation of condition (equation 2.2) is identical to the expression inside the square brackets of NR's equation 15.4.6. The differences arise because here we are discussing least-squares fitting instead of chi-square fitting, i.e. we have omitted the factors involving $\sigma_{meas,m}$, the intrinsic measurement uncertainties (§8).

Again, there are more equations than unknowns so we can't solve this matrix equation directly. So next we form the normal equations from these matrices. In matrix form, the normal equations (equation 1.2) reduce to the single equation

$$[\alpha] \cdot \mathbf{a} = [\beta] \ , \tag{2.3}$$

(NR equation 15.4.10), where

$$[\alpha] = \mathbf{X^T} \cdot \mathbf{X} \tag{2.4a}$$

$$[\beta] = \mathbf{X^T} \cdot \mathbf{Y} \ . \tag{2.4b}$$

The matrix $[\alpha]$ is known as the *curvature matrix* because each element is twice the curvature of σ^2 (or χ^2) plotted against the corresponding product of variables.

The number of equations is equal to the number of unknowns, so the solution of the matrix equation is easy—just rewrite it by multiplying both sides by the inverse of $[\alpha]$ (that is, by $[\alpha]^{-1}$), which gives

$$\mathbf{a} = [\alpha]^{-1} {\cdot} [\beta] \; . \tag{2.5}$$

All of these matrix operations are trivially easy in Python (§4).

3. UNCERTAINTIES IN THE DERIVED COEFFICIENTS

How about the uncertainties in the derived quantities contained in the matrix **a**?

The first thing to do is derive the *sample* variance s^2 (the square of standard deviation, or mean error, or dispersion, etc) of the individual datapoints using the generalization of the usual definition for a straight average of x, $s^2 = [\sum_0^{M-1}(x_m - \overline{x_m})^2/(M-1)]$. The generalization is, simply, to replace the $M-1$ in the denominator by $\nu = M - N$. In the straight-average case, $N = 1$ so this fits. Here ν is known as the number of *degrees of freedom* and N, the number of unknown coefficients, is known as the number of *constraints*. So we have

$$s^2 = \frac{1}{M-N} \sum_{m=0}^{M-1} (y_m - \overline{y_m})^2 \; , \tag{3.1}$$

where $\overline{y_m}$ are the values for y_m *predicted by the derived quantities* **a**. Note the difference: y_m are the *observed* values, while $\overline{y_m}$ are the values *predicted by the least-squares fit*. The predicted values are those that are computed from the derived coefficients $A, B, C \ldots$ The M quantities

$$\delta y_m = y_m - \overline{y_m} \tag{3.2}$$

are called the *residuals* or *deviations* from the fit.

It's worth reiterating some essentials about s^2, and in particular the denominator $(M - N)$. First consider the case of a single-parameter fit, e.g. $N = 1$. Then we cannot possibly derive a sample variance from only one measurement $M = 1$; but we can from two $M = 2$. So the denominator makes sense from that standpoint. The same goes for $N > 1$.

Next consider the effect of using $(M - N)$ in the denominator: it increases s^2 by the ratio $\frac{M}{M-N}$ over what you'd get if you just took a straight average and used M. This compensates for the fact that you are subtracting $\overline{y_m}$, which is derived from the data, instead of the *truly* correct value y^*. (In formal statistical language, y^* is the mean of the parent population from which your set of measurements is drawn.) If you used the truly correct value y^*, then the sum would be larger than when using $\overline{y_m}$. The use of $M - N$ in the denominator compensates for this larger value in exactly the right way: the expectation value $E_{(s^2)}$ for a large number of experiments is precisely equal to the normal variance σ^2, which you'd get by using $[y^*$ and $M]$ instead of $[\overline{y_m}$ and $(M - N)]$ in equation 3.2; see Cowan equations 5.9 and 5.10. So s^2 is, in fact, exactly the number we want: an unbiased estimate of the true variance of our sample. Why not use $[y^*$ and $M]$ in equation 3.2? The reason is obvious: we don't know y^*! (If we did, we wouldn't be doing this analysis!)

It's easy to calculate the $\overline{y_m}$ with matrices. First define the matrix $\overline{\mathbf{Y}}$ that contains these values:

$$\overline{\mathbf{Y}} = \begin{bmatrix} \overline{y_0} \\ \overline{y_1} \\ \overline{y_2} \\ \overline{y_3} \\ \overline{y_4} \end{bmatrix} \tag{3.3}$$

Calculating $\overline{\mathbf{Y}}$ is simple:

$$\overline{\mathbf{Y}} = \mathbf{X} \cdot \mathbf{a} \ . \tag{3.4}$$

Note that $\mathbf{X}$ is already defined (equation 2.1) and $\mathbf{a}$ was solved for in equation 2.5. It's convenient to define the residual matrix

$$\delta\mathbf{Y} = \mathbf{Y} - \overline{\mathbf{Y}} \tag{3.5}$$

so we can write

$$s^2 = \frac{1}{M - N} \delta\mathbf{Y}^{\mathbf{T}} \cdot \delta\mathbf{Y} \ . \tag{3.6}$$

This is the sample variance of the datapoints, not the variances in the derived coefficients. We can obtain these as before, by generalizing the results from the two-parameter case like the straight-line fit discussed in §0. We won't go through the derivation here; you can find it in Taylor §8.4 and equation 8.16, 8.17. The result is

$$\mathbf{s_a}^2 = s^2 diag\{[\alpha]^{-1}\} \ . \tag{3.7}$$

Or, to put it simply in words: to get the variance of coefficient n in the matrix $\mathbf{a}$, multiply s^2 by the n^{th} diagonal element of $[\alpha]^{-1}$.

Although the above equation for $\mathbf{s_a}^2$ is correct, there is more to the story because of covariances, which are the off-diagonal elements. We return to this topic in §5 and §8.1.

4. A NUMERICAL EXAMPLE AND ITS SOLUTION IN Python

If the following sounds like Greek to you, take a look at §2 and 3.

4.1. Generation of the numerical example

Suppose that we make four measurements of the angle y and we want to fit to a parabolic function in time t. In the notation of equation 1.1, s would be unity, t the time, and u the time squared, so the number of unknowns is three ($N = 3$). Because there are four independent measurements ($M = 4$) the subscripts run from $m = 0 \rightarrow 3$. Suppose that the four values of time are 5, 7, 9, 11.

First we create the matrix $\mathbf{X}$ in Python

$$\mathbf{X = np.linspace(0, N, M) = np.linspace(0, 3, 4)} \tag{4.1}$$

and then we populate it with numbers. In your own work, you would normally do this by reading a data file and transferring the numbers to the matrix using Python commands; to work through this example, you might manually type them in. After populating the matrix, in direct correspondence with equation 2.1a we have $s_m = 1$, $t_m = time_m$, $u_m = time_m^2$:

$$\mathbf{X} = \left[\begin{array}{ccc} 1 & 5 & 25 \\ 1 & 7 & 49 \\ 1 & 9 & 81 \\ 1 & 11 & 121 \end{array} \right] . \tag{4.2a}$$

Suppose that the four measured values of y are (equation 2.1c)

$$\mathbf{Y} = \left[\begin{array}{c} 142 \\ 168 \\ 211 \\ 251 \end{array} \right] . \tag{4.3a}$$

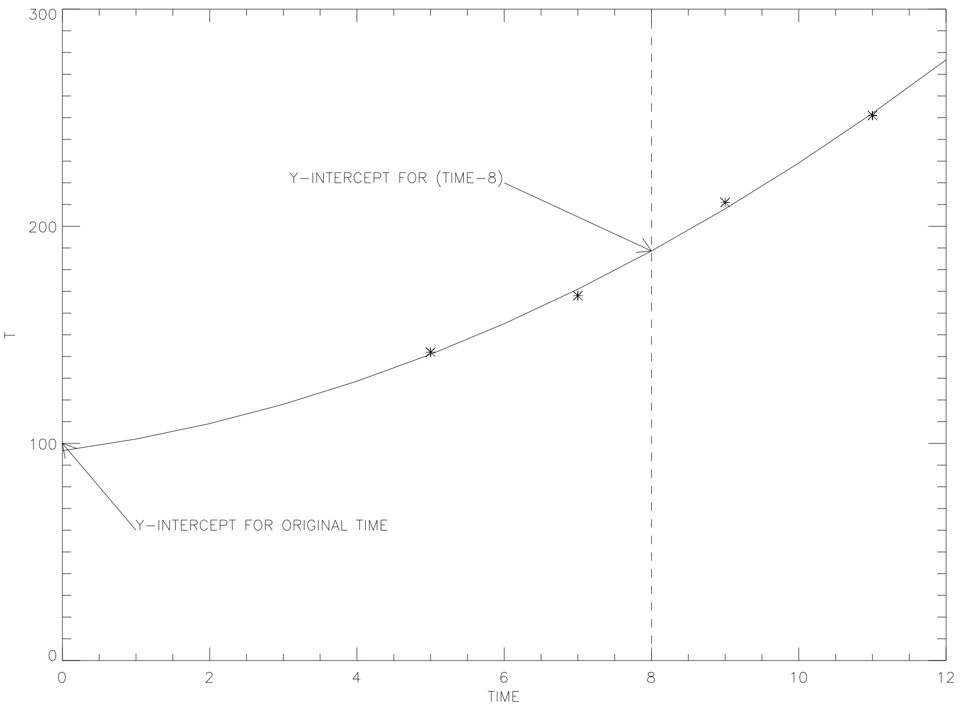

Fig. 4.1.— Our numerical example. Stars are the four datapoints; the solid line is the fit. We perform two fits: one uses the original definition of time; the other uses $(time - 8)$, in effect moving the y-axis to the dashed line. The two fits give the same line but the coefficients and their errors differ greatly.

Figure 4.1 shows the datapoints, together with the fitted curve.

One word of caution here: in Python, to get these into a column matrix, which is how we've treated $\mathbf{Y}$ above, you have to define $\mathbf{Y}$ as a two-dimensional array because the second dimension represents the column.

4.2. Solution of the Numerical Example in Python

In Python we calculate the normal equation matrices and denote the $[\alpha]$ in equation 2.4a by **xx**:

$$\mathbf{xx} = \mathbf{np.dot(X.T, X)} \ , \tag{4.4a}$$

and we denote the $[\beta]$ in equation 2.4b by **xy**:

$$\mathbf{xy} = \mathbf{np.dot}(\mathbf{X.T}, \mathbf{Y}) \ . \tag{4.4b}$$

In Python we take the inverse of $[\alpha]$ (same as **xx**) by

$$\mathbf{xxi} = \mathbf{np.linalg.inv}(\mathbf{xx}) \ . \tag{4.5}$$

The least-squares fitted quantities are in the matrix **a** (equation 2.5), which we obtain in Python with

$$\mathbf{a} = \mathbf{np.dot}(\mathbf{xxi}, \mathbf{xy}) \ . \tag{4.6}$$

In Python we denote the matrix of predicted values $\overline{y_m}$ by **ybar**, which is

$$\mathbf{ybar} = \mathbf{np.dot}(\mathbf{X}, \mathbf{a}) \ , \tag{4.7}$$

and we can also define the residuals in **Y** as

$$\mathbf{dely} = \mathbf{Y} - \mathbf{ybar} \ . \tag{4.8}$$

In Python we denote s^2 in equations 3.1 and 3.6 by s_sq and write

$$s\_sq = \mathbf{np.dot}(\mathbf{dely.T}, \mathbf{dely})/(M-N) \ , \tag{4.9a}$$

or

$$s\_sq = np.sum(\mathbf{dely} **2)/(M-N) \ . \tag{4.9b}$$

It is *always* a good idea to plot all three quantities (the measured values **Y**, the fitted values **ybar**, and the residuals **dely**) to make sure your fit looks reasonable and to check for bad datapoints.

To get the error in the derived coefficients we need a way to select the diagonal elements of a matrix. Obviously, any $N \times N$ matrix has N diagonal elements; a convenient way to get them is

$$diag\ elements\ of\ \mathbf{xxi} = \mathbf{np.diag}(\mathbf{xxi}) \ . \tag{4.10}$$

In Python, we define the variances of the N derived coefficients by **vardc** (think of "**var**iances of **d**erived **c**oefficients"). You can get this as in equation 3.7 from[3]

$$\mathbf{vardc} = s\_sq * \mathbf{np}.\mathbf{diag}(\mathbf{xxi}) \ . \tag{4.11}$$

4.3. Discussion of the numerical example

For this numerical example, the solution (the array of derived coefficients) is

$$\mathbf{a} = \begin{bmatrix} 96.6250 \\ 4.5000 \\ 0.8750 \end{bmatrix} \tag{4.12a}$$

and the errors in the derived coefficients [the square root of the σ^2's of the derived coefficients, i.e. $[\sigma_n^2]^{1/2}$ or, in Python, $sqrt(\mathbf{vardc})$ in equations 4.11] are:

$$\sigma_{\mathbf{A}} = \begin{bmatrix} 34.012 \\ 9.000 \\ 0.5590 \end{bmatrix} \ . \tag{4.12b}$$

These results look *horrible*: the uncertainties are large fractions of the derived coefficients,

The reason: we have specifically chosen an example with terrible covariance. And the great thing is this can be fixed easily (at least in this case—certainly not always), without taking more data!

5. THE COVARIANCE MATRIX AND ITS NORMALIZED COUNTERPART

First we provide a general discussion, then we apply it to the above numerical example.

5.1. Definition of the normalized covariance (or correlation) matrix

The variances in the derived coefficients are obtained from the diagonal elements of **xxi**. The off-diagonal elements represent the *covariances* between the derived coefficients. Covariance means,

[3]If you used equation 4.9a instead of 4.9b, then Python considers s_sq an array and you need to use `np.dot` instead of a $*$ in this equation.

specifically, the degree to which the *uncertainty* in *one* derived coefficient affects the uncertainty in *another* derived coefficient.

Because the covariance matrix elements relate pairwise to the various coefficients, the units of the matrix elements are all different. This makes it convenient to reduce all the matrix elements to a standard set of units—namely, no units at all. So before discussing the covariance matrix *per se*, we first discuss its normalized counterpart.

The normalized covariance matrix[4] **ncov** is derived from **xxi** by dividing each element by the square root of the product of the corresponding diagonal elements. Let **ncov** be the normalized covariance matrix; then

$$ncov_{ik} = \frac{XXI_{ik}}{\sqrt{XXI_{ii}\ XXI_{kk}}} \ . \tag{5.1}$$

This is the same normalization that one does with the Pearson linear correlation coefficient of two variables. In fact, the elements of the normalized covariance matrix *are* the correlation coefficients. So it makes sense to call this matrix the *correlation matrix*, and many people do. In Python, you do the following:

$$\mathbf{dc} = \mathbf{np.diag(xxi)} \tag{5.2a}$$

$$\mathbf{ncov} = \mathbf{xxi/np.sqrt(np.outer(dc,dc))} \ . \tag{5.2b}$$

In the above, $\mathbf{np.outer(dc,dc)}$ is an $N \times N$ matrix consisting of products of the diagonals of **xxi**, so dividing **xxi** by $\mathbf{np.sqrt(np.dot(dc,dc))}$ generates the normalized version.

Because **ncov** is a *normalized* covariance matrix, you might think that its non-normalized parent is **xxi**—and you'd be *almost* right. For the least-squares case we are discussing, the true covariance matrix **C** is[5]

$$\mathbf{C} = s^2\ \mathbf{xxi} \ . \tag{5.3}$$

In **ncov**, the diagonal elements are all unity and the off-diagonal elements reflect the interdependence of the derived coefficients on each other. The off-diagonal elements can range from $-1 \to 1$. Each matrix element is the correlation coefficient between the *uncertainties* of its two parameters. In particular, suppose that the data happen to produce a coefficient that differs from

[4]It is a pleasure to thank Doug Finkbeiner for introducing me to this concept.

[5]For chi-square, you use σ^2_{meas} instead of s^2; see §8.

its true value by some positive number. If the normalized matrix element is negative, then the other coefficient will tend to differ from its true value by a negative number.

Here's a more detailed discussion of what the covariance means. Suppose you are least-squares fitting for two derived coefficients (A_0 and A_1). When you do a least-squares fit to a set of data, you are fitting one set of data out of a possible infinity of possible sets that you'd get by repeating the experiment, and your particular set of data happens to produce specific values of $\overline{A_0}$ and $\overline{A_1}$, which differ from the *true* values (A_0^*, A_1^*) by amounts δA_0 and δA_1. If their covariance is zero, then in the infinity of data sets you'd find that δA_0 is uncorrelated with δA_1. But if it is nonzero, these two quantities would be correlated.

A high covariance is bad because the derived variables depend on each other. For one, this means that with noisy data power can be shared or passed from one parameter to/from its covariant counterpart(s). As we shall see in §8.1, it also significantly influences the uncertainties in derived coefficients. Often a high covariance results from a poor choice of functions that you are fitting or even a bad choice of the zero point of the independent variable—as in our numerical example (see the next subsection). And, as in that example, you can sometimes eliminate the bad covariance by reformulating the problem—you don't even need to take more data! The best reformulation involves using a set of orthonormal functions. However, sometimes your interest is a specific set of functions that are *not* orthogonal, and in such cases it makes no sense to convert to orthogonal functions—because you just have to convert back again and do the error propagation after-the-fact instead of letting the least-squares process do it for you.

5.2. The covariance in our numerical example

Apply equation 5.2 to obtain the covariance matrix for our numerical example:

$$\mathbf{ncov} = \begin{bmatrix} 1 & -.989848 & .969717 \\ -.989848 & 1 & -.993808 \\ .969717 & -.993808 & 1 \end{bmatrix} . \tag{5.4}$$

The off-diagonal elements are *huge*. This is the reason why our derived coefficients have such large uncertainties. Note, however, that the fitted predicted fit is a good fit even with these large uncertaintis.

In this seemingly innocuous example we have an excellent case of a poor choice of zero point for the independent variable (the time). The reason is clear upon a bit of reflection. We are fitting for $y = A_0 + A_1 t + A_2 t^2$. The coefficient A_0 is the y-intercept and A_1 is the slope. Inspection of Figure 4.1 makes it very clear that an error in the slope has a big effect on the y-intercept.

Now we retry the example, making the zero point of the time equal to the mean of all the times, that is we set ($time_m = time_m - 8$). We get the same fitted line, but the derived coefficients are completely different—and amazingly better! We get

$$\mathbf{a} = \begin{bmatrix} 188.625 \\ 18.500 \\ 0.87500 \end{bmatrix} \tag{5.5a}$$

$$\sigma_{\mathbf{A}} = \begin{bmatrix} 3.58 \\ 1.00 \\ 0.559 \end{bmatrix} . \tag{5.5b}$$

In redefining the origin of the independent variable, we have made the zero intercept completely independent of the slope: changing the slope has no affect at all on the intercept. You can see this from the normalized covariance matrix, which has become

$$\mathbf{ncov} = \begin{bmatrix} 1 & 0 & -0.78086881 \\ 0 & 1 & 0 \\ -0.78086881 & 0 & 1 \end{bmatrix} , \tag{5.6}$$

which is nice, but not perfect: Our step is *partial* because the second-order coefficient A_2 affects A_0 because, over the range of $[(time - 8) = -3 \rightarrow +3]$, the quantity $[A_2 \; \Sigma(time_m - 8)^2]$ is always positive and is thereby correlated with A_0.

We could complete the process of orthogonalization by following the prescription in BR chapter 7.3, which discusses the general technique of orthogonalizing the functions in least-squares fitting. The general case is a royal pain, analytically speaking, so much so that we won't even carry it through for our example. But for numerical work you accomplish the orthogonalization using Singular Value Decomposition (SVD), which is of course trivial in Python `np.linalg.svd`.

For some particular cases, standard pre-defined functions are orthogonal. For example, if t_m is a set of uniformly spaced points between $(-1 \rightarrow 1)$ and you are fitting a polynomial, then the appropriate orthogonal set is Legendre polynomials. This is good if your only goal is to represent a bunch of points by a polynomial function, because the coefficients of low-order polynomials are independent of the higher ones. However, it's more work and, moreover, often you are interested in the coefficients for specific functions that don't happen to be orthogonal; in such cases, you should just forge ahead.

But *always* look at the normalized covariance matrix. Suppose one pair of off-diagonal elements departs significantly from zero. Then their corresponding functions are far from being orthogonal and the variances of the derived coefficients will suffer as a result. You might be able to eliminate one of the parameters to make the fit more robust. For example, suppose one function is $t\cos(t)$ and the other is $\sin(t)\cos(t)$. If the range of t is small, these two functions are indistinguishable and have a large covariance; you should eliminate one from the fit. If the range of t is large, there is no problem.

For further discussion of covariance, see §8.1. Also, you might also want to try out another example in Taylor's §8.5.

6. REJECTING BAD DATAPOINTS I.: CHAUVENET'S CRITERION

Least-squares fitting is derived from the maximum likelihood argument assuming the datapoint residuals δy_m have a Gaussian pdf. This means that the errors are distributed as

$$p_{(\delta y;\sigma)} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\left(\frac{\delta y^2}{2\sigma^2}\right)} \ , \tag{6.1}$$

where σ^2 is the true variance of the datapoints, i.e. s^2 in equation 3.1 (to be precise, s^2 needs to be averaged over many experiments).

More importantly, the probability of finding datapoints inside the limits $\pm\Delta y$ is

$$P_{(|\delta y|<\Delta y)} = \int_{-\Delta y}^{+\Delta y} p_{(\delta y;\sigma)} d(\delta y) = \text{erf}\left(\frac{\Delta y}{\sqrt{2}\sigma}\right) \ , \tag{6.2}$$

where we use the commonly-defined error function $\text{erf}(X) = \frac{1}{\sqrt{\pi}} \int_{-X}^{+X} e^{-x^2} dx$. A particularly important value is for $\Delta y = \sigma$, for which

$$P_{(|\delta y|<\sigma)} = 0.683 \ . \tag{6.3}$$

If we have an experiment with M datapoints, then the number of datapoints we expect to lie outside the interval $\pm\Delta y$ is

$$M_{(outside\ \Delta y)} = M\left[1 - \text{erf}\left(\frac{\Delta y}{\sqrt{2}\sigma}\right)\right] \ . \tag{6.4}$$

Chauvenet's criterion simply says:

1. Find Δy such that $M_{(outside\ \Delta y)} = 0.5$. This is given by

$$\frac{\Delta y}{\sigma} = \sqrt{2}\ \mathrm{erf}^{-1}\left(1 - \frac{1}{2M}\right) \ . \tag{6.5}$$

This criterion leads to the numbers in the associated table, which is a moderately interesting set of numbers. Many astronomers adopt 3σ, which is clearly inappropriate for large N!

Chauvenet's criterion versus M

M	$\frac{\Delta y}{\sigma}$
100	2.81
1000	3.48
10^4	4.06
10^5	4.56

2. Discard all datapoints outside this range.

We offer the following important *Comments*:

- This assumes data are Gaussian-distributed. In real life this doesn't often happen because of "glitches". Examples of glitches can be interference in radio astronomy, meteors in optical astronomy, and cosmic rays on CCD chips. These glitches produce bad points that depart from Gaussian statistics. They are often called *outliers*.

  It is very important to get rid of the outliers because the least-squares process minimizes the *squares* of the residuals. Outliers, being the points with the largest residuals, have a disproportionately evil effect on the result.

  On the other hand, if your data don't follow Gaussian statistics as their *intrinsic* pdf, then you should think twice before using least squares! (Like, maybe you should try the median fitting discussed in the more complete version of this document.)

- You may wish to relax Chauvenet's criterion by *increasing* the Δx beyond which you discard points. This is being conservative and, in the presence of some non-Gaussian statistics, not a bad idea. But think about why you are doing this before you do it. Maybe the intrinsic statistics aren't Gaussian?

  You should *never* make Chauvenet's criterion more stringent by *decreasing* the Δx beyond which you discard points. This rule hardly needs elaboration: it means you are discarding datapoints that follow the assumed pdf!

- Most statistics books (e.g. Taylor, BR) harp on the purity aspect. One extreme: don't throw out any datum without examining it from all aspects to see if discarding it is justified. The other extreme: apply Chauvenet's criterion, but do it *only once* and certainly not repeatedly.

Being real-life astronomers, our approach is different. There *do* exist outliers. They increase the calculated value of σ. When you discard them, you are left with a more nearly perfect approximation to Gaussian statistics and the new σ calculated therefrom will be smaller than when including the outliers. Because the original σ was too large, there may be points that should have been discarded that weren't. So our approach is: repeatedly apply Chauvenet's criterion until it converges.

If it doesn't converge, or if it discards an inordinately large number of datapoints, you've got real problems and need to look at the situation from a global perspective.

- Many observers use the 3σ criterion: discard any points with residuals exceeding 3σ. This is definitely *not* a good idea: the limit 3σ is Chauvenet's criterion for $M = 185$ datapoints. Very often M exceeds this, often by a lot.

- To apply Chauvenet's criterion it's most convenient to calculate the inverse error function. For this, you have two choices. One (for sissies like myself), you can use **scipy.special.erfinv** from the Scipy package. But the real he-man will want to learn about using a root-finding algorithm such as Newton's method (NR §9.4 and 9.6) together with the error function. You at least ought to skim lightly some of NR's chapter 9 about root finding, because some day you'll need it.

7. NONLINEAR LEAST SQUARES

The least-squares formulation requires that the data values depend *linearly* on the unknown coefficients. For example, in equation 0.1, the unknown coefficients A and B enter linearly.

Suppose you have a nonlinear dependence, such as wanting to solve for A and B with equations of condition that look like

$$\sin(At_m) + Bt_m = y_m \; . \tag{7.1}$$

What do you do here? You linearize the process, using the following procedure.

First, assume trial values for A and B; call these A_0 and B_0. You should pick values that are close to the correct ones. In our example you wouldn't need to do this for B, but it's easier to treat all coefficients identically. These trial values produce *predicted* values $y_{0,m}$:

$$\sin(A_0 t_m) + B_0 t_m = y_{0,m} \; . \tag{7.2}$$

Subtract equation 7.2 from 7.1, and express the differences as derivatives. Letting $\delta A = A - A_0$ and $\delta B = B - B_0$, this gives

$$\delta A[t_m \cos(A_0 t_m)] + \delta B t_m = y_m - y_{0,m} \; . \qquad (7.3)$$

This is linear in $(\delta A, \delta B)$ so you can solve for them using standard least squares. Increment the original guessed values to calculate $A_{0,new} = A_0 + \delta A$ and $B_{0,new} = B_0 + \delta B$, These won't be exact because higher derivatives (including cross derivatives) come into play, so you need to use these new values to repeat the process. This is an iterative procedure and you keep going until the changes become "small". The generalization to an arbitrarily large number of unknown coefficients is obvious.

We now offer some cautionary and practical remarks.

(0) In linear least squares, the curvature and covariance matrices are set by the values of the independent variable, which here is denoted by t, and are independent of the datapoint values. Here, the matrix elements change from one iteration to the next because they depend on the guessed parameters, and sometimes they even depend on the datapoint values.

(1) Multiple minima: Nonlinear problems often have multiple minima in σ^2. A classical case is fitting multiple Gaussians to a spectral line profile. Gaussians are most definitely not orthogonal functions and in some cases several solutions may give almost comparably good values of σ^2, each one being a local minimum. For example, for the case of two blended Gaussians, one can often fit two narrow Gaussians or the combination of a wide and narrow Gaussian, the two fits giving almost equal σ^2. The lower of these is the real minimum but, given the existence of systematic errors and such, not necessarily the best solution. The best solution is often determined by physical considerations; in this case, for example, you might have physical reasons to fit a broad plus narrow Gaussian, so you'd choose this one even if its σ^2 weren't the true minimum.

(2) The Initial Guess: When there are multiple minima, the one to which the solution converges is influenced by your initial guess. To fully understand the range of possible solutions, you should try different initial guesses and see what happens. If the solutions always converge to the same answer, then you can have some confidence (but not *full* confidence) that the solution is unique.

(3) Iterative stability: If your initial guess is too far from the true solution, then the existence of higher derivatives means that the computed corrections can be too large and drive the iterative solution into instability. It is often a good idea to multiply the derived correction factors (δA and δB above) by a factor $\mathcal{F}$ less than unity, for example $\mathcal{F} = 0.5$ or 0.75. This increases the number of iterations required for convergence but often allows convergence instead of producing instability.

(4) Convergence criteria: How do you know when the solution has converged? One way: for each iteration, calculate the uncertainties in the derived coefficients. If the uncertainty exceeds the correction, then you are getting close. An alternate way, which I usually use: if the fractional correction (e.g. $\frac{\delta A}{A_0}$) decreases below some threshold, say 1%, you're close (some parameters, such as angles, need a threshold that is absolute instead of fractional). At this point, if you are using $\mathcal{F} \neq 1$, set $\mathcal{F} = 1$, do a few more iterations, and you're done.

(5) Numerical derivatives: Sometimes the equations of condition are so complicated that taking the derivatives, as in obtaining equation 7.3, is a huge job and subject to mistakes. So you can take numerical derivatives instead of analytic ones. Be careful, though; it's safest to use double precision and think a bit about numerical accuracy; take a look at NR's section 5.7 on evaluating numerical derivatives.

(6) Canned nonlinear least squares (particularly Levenberg-Marquardt, and various Gaussian fit routines): Packages like Scipy offer canned nonlinear least squares routines. They are designed to work well for a wide range of different problems. However, for the specific problem at hand you can often do better by tailoring things (such as the factor $\mathcal{F}$ and convergence criteria above).

When convergence is slow or doesn't occur because your functions are complicated, you might wish to try the Levenberg-Marquardt method (NR §15.5). This technique involves increasing the diagonal elements of the curvature matrix by a set of suitably chosen factors; when you get close to the minimum, it resets these factors to unity. LM is the gold standard for nonlinear least-squares fitting because it is supposed to converge faster than other methods. Because of its sterling reputation, many people think it's the panacea. How many times have I seen journal articles saying that the LM method was used—as if that's all one needs to know—but without saying anything about the important stuff, such as how parameter space was explored to determine uniqueness of the solution! See the discussion in NR. I've done lots of nonlinear fits and have never had to resort to any tactic other than the simple, straightforward linearization process discussed above.

(7) Be careful and LOOK at the solution before accepting it! These nonlinear problems can produce surprising results, sometimes completely meaningless results. Don't rely on them to be automatic or foolproof!

(8) Reformulate! (?) Sometimes you can avoid all this by reformulating the problem. There are two cases: the harmless case and the not-so-harmless case.

An example of the harmless case is fitting for the phase ϕ in the function $y = \cos(\theta + \phi)$. This is definitely a nonlinear fit! But its easy to reformulate it in a linear fit using the usual trig identities to write $y = A\cos\theta - B\sin\theta$, where $\frac{B}{A} = \tan\phi$. Solve for (A, B) using linear least squares, calculate ϕ, and propagate the uncertainties.

An example of the not-so-harmless case is in NR's §15.4 example: fit for (A, B) with equations of condition $y_m = Ae^{-Bx_m}$. They suggest linearizing by rewriting as $\log(y_m) = C - Bx_m$, solving for (B, C), and deriving A after-the-fact. This is not-so-harmless because you are applying a nonlinear function to the *observed* values y_m; thus the associated errors $\sigma_{meas,m}$ are *also* affected. This means you have to do weighted fitting, which is discussed in §8 below. Suppose that $A = 1$, your datapoints all have $\sigma_{meas,m} = 0.05$, and the observed y_m ranges from 0.05 to 1. The datapoint with $y_m = 0.05$ has a manageable σ_{meas_m}, but what is the corresponding value of $\sigma_{meas,m}$ for $\log y_m = \log 0.05$? It's ill-defined and asymmetric about the central value. Or even, God forbid, you have an observed y_m that's *negative*??? Even for y_m not near zero, you need to calculate new $\sigma_{meas,m}$ by error propagation; in this case, you need to reassign $\sigma(\log y) = \frac{d \log y}{dy}\sigma(y) = \frac{\sigma(y)}{y}$. This is OK when y_m is large enough so that the linear approximation is accurate, but if not the converted noise becomes non-Gaussian.

You should regard your datapoints as sacrosanct and never apply any nonlinear function to them.

8. CHI-SQUARE FITTING AND WEIGHTED FITTING: DISCUSSION IGNORING COVARIANCE

In least-squares fitting, the derived parameters minimize the sum of squares of residuals as in equation 3.1, which we repeat here:

$$s^2 = \frac{1}{M-N} \sum_{m=0}^{M-1} \delta y_m^2 \, .$$

where the m^{th} residual $\delta y_m = (y_m - \overline{y_m})$. Chi-square fitting is similar except for two differences. One, we divide each residual by its intrinsic measurement error σ_m; and two, we define χ^2 as the sum

$$\chi^2 = \sum_{m=0}^{M-1} \frac{\delta y_m^2}{\sigma_m^2} \, . \tag{8.1a}$$

Along with χ^2 goes the *reduced* chi square $\widehat{\chi^2} = \frac{\chi^2}{M-N}$

$$\widehat{\chi^2} = \frac{1}{M-N} \sum_{m=0}^{M-1} \frac{\delta y_m^2}{\sigma_m^2} \, , \tag{8.1b}$$

which is more directly analogous to the definition of s^2.

Chi-square fitting is very much like our least-squares fitting except that we divide each datapoint by its intrinsic measurement uncertainty σ_m. Thus, the reduced chi-square ($\widehat{\chi^2}$) is equal to the ratio of the *variance of the datapoint residuals* (s^2) to the *adopted intrinsic measurement variances* (σ_m^2). So it should be obvious that in chi-square fitting, you must know the measurement uncertainties σ_m of the individual datapoints beforehand. If you want to give the various datapoints weights based on something other than σ_m, then that is just like chi-square fitting except that you can adopt an arbitrary scale factor for the uncertainties (section 8.5).

Chi-square fitting treats uncertainties of the derived parameters in a surprising way. Getting the coefficient uncertainties with chi-square fitting is a tricky business because

1. With the standard treatments, the errors in the derived parameters don't depend on the residuals of the datapoints from the fit (!).
2. The errors in the derived parameters can depend on their mutual covariances. This discussion requires a separate section, which we provide below in §8.1.

In this section we treat chi-square fitting ignoring covariance. We begin by illustrating the difference between least squares and chi-square fitting by discussing the simplest chi-square fitting case of a weighted mean; then we generalize to the multivariate chi-square fitting case.

8.1. The weighted mean: the simplest chi-square fit

First, recall the formulas for an ordinary *unweighted* average in which the value of each point is y_m and the residual of each point from the weighted mean is δy_m:

$$mean = \frac{\sum y_m}{M} \tag{8.2a}$$

$$s^2 = \frac{\sum \delta y_m^2}{M-1} \tag{8.2b}$$

$$s_{mean}^2 = \frac{s^2}{M} = \frac{\sum \delta y_m^2}{M(M-1)} \ , \tag{8.2c}$$

where s_{mean}^2 is the variance of the mean and s^2 is the variance of the datapoints around the mean. Recall that in this case the mean is the least-squares fit to the data, so to use least squares jargon we can also describe s_{mean} as the error in the derived coefficient for this single-parameter least-squares fit.

Now for a *weighted* average in which the weight of each point is $w_{meas,m} = \frac{1}{\sigma_m^2}$. Applying maximum likelihood, in an unweighted average the quantity that is minimized is $\sum \delta y_m^2$; in a weighted average the quantity minimized is $\chi^2 = \sum \frac{\delta y_m^2}{\sigma_m^2} = \sum w_{meas,m} \delta y_m^2 \rightarrow w_{meas,m} \sum \delta y_m^2$, where to the right of the arrow we assume all $w_{meas,m}$ are identical. So your intuition says that the three equations corresponding to the above would become

$$mean_{w,intuit} = \frac{\sum w_{meas,m} y_m}{\sum w_{meas,m}} \rightarrow \frac{\sum y_m}{M} \tag{8.3a}$$

Again, to the right of the arrow we assume all $w_{meas,m}$ are identical and the subscript *intuit* means "intuitive". For the variances the intuitive expressions are

$$s^2_{w,intuit} = \frac{M}{M-1} \frac{\sum w_{meas,m} \delta y_m^2}{\sum w_{meas,m}} = \frac{\widehat{\chi^2}}{(\sum w_{meas,m}/M)} \rightarrow \frac{\sum \delta y_m^2}{M-1} \tag{8.3b}$$

$$s^2_{mean,intuit} = \frac{s^2_{w,intuit}}{M} = \frac{\sum w_{meas,m} \delta y_m^2}{(M-1)\sum w_{meas,m}} = \frac{\widehat{\chi^2}}{\sum w_{meas,m}} \rightarrow \frac{\sum \delta y_m^2}{M(M-1)} \ . \tag{8.3c}$$

In fact, after a formal derivation, the first two equations (8.3a and 8.3b) are correct, so we will drop the additional subscipts *intuit* and *formal* on *mean* and s^2_w. However, after a formal derivation, the last of these equations becomes, and is always written (e.g. BR equation 4.19; Taylor equation 7.12)

$$s^2_{mean,formal} = \frac{1}{\sum w_{meas,m}} \rightarrow \frac{\sigma^2_{meas}}{M} \ . \tag{8.4}$$

This is a problem, for the following reason.

Note the excruciatingly painful difference between the intuitive equation 8.3c and the formally correct equation 8.4: the former depends on the *variance of the datapoint residuals* s^2_w, as you'd think it should, while the latter depends on only the *adopted intrinsic measurement variances of the data* σ^2_m, which are chosen by the guy doing the fit. If you do an unweighted average, and derive a certain variance, and next do a weighted average in which you choose some values for σ_m that happen to be wrong, the two fits give different results for s^2_{mean}. This is crazy.

To get around this difficulty, we follow the procedure in BR equations 4.20 to 4.26. This introduces an arbitrary multiplicative factor for the weights and goes through the ML calculation to derive, instead of equation 8.4, the *far superior*

$$s^2_{mean,BR} = \frac{s^2_w}{M} = \frac{\widehat{\chi^2}}{\sum w_{meas,m}} \ , \tag{8.5}$$

which is precisely the same as our intuitive guess, equation 8.3c. The difference between the formal equation 8.5 and the intuitive equations 8.3b and 8.4 is the numerator, which contains the reduced chi-square $\widehat{\chi^2}$; for the case where all $\sigma_{meas,m}$ are identical, $\widehat{\chi^2} = \frac{s^2_w}{\sigma^2_{meas}}$. Note that χ^2 and $\widehat{\chi^2}$ are defined in equations 8.1.

8.2. The multivariate chi-square fit

Here we generalize §8.1, which dealt with the weighted average, to the multivariate case. In this case, chi-square fitting is just like least-squares fitting except for the following:

1. In the least-squares matrix $\mathbf{X}$ of equation 2.1a, each row m is a different measurement with a different intrinsic variance σ_m. For chi-square fitting you generate a new matrix $\mathbf{X}_\chi$, which is identical to $\mathbf{X}$ except that each row m (which contains a particular equation of condition) is divided by σ_m. This new matrix is the same as NR's *design matrix* (Figure 15.4.1), which they denote by $\mathbf{A}$.

2. For chi-square fitting, divide each datapoint y_m in equation 2.1b by σ_m. You are generating a new data vector $\mathbf{Y}_\chi$, which is identical to $\mathbf{Y}$ except that each datapoint is divided by σ_m. This new data vector is the same as NR's vector $\mathbf{b}$.

3. *Note* that the above two steps can be accomplished matrixwise by defining the $M \times M$ diagonal matrix $[\sigma]$ in which the diagonal elements are σ_m.

$$[\sigma] = \begin{bmatrix} \sigma_0 & 0 & \dots & 0 \\ 0 & \sigma_1 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \sigma_{M-1} \end{bmatrix} \tag{8.6}$$

   in which case we can write

$$\mathbf{X}_\chi = [\sigma]^{-1} {\cdot} \mathbf{X} \tag{8.7a}$$

$$\mathbf{Y}_\chi = [\sigma]^{-1} {\cdot} \mathbf{Y} \ . \tag{8.7b}$$

4. Carry through the matrix calculations in equations 8.8 below (using the matrices subscripted with χ).

You've divided each row, i.e. the equation of condition for each row m, by a common factor, so the solution of that particular equation of condition is unchanged. However, in the grand scheme of things—i.e. the normal equations—it receives a greater or lesser weight by a factor $\frac{1}{\sigma_m^2}$.

To perform the chi-square fit, we use the following equations:

$$\mathbf{X}_\chi = [\sigma]^{-1} {\cdot} \mathbf{X} \tag{8.8a}$$

$$\mathbf{Y}_\chi = [\sigma]^{-1} \cdot \mathbf{Y} \ . \tag{8.8b}$$

$$\mathbf{X}_\chi \cdot \mathbf{a} = \mathbf{Y}_\chi \tag{8.8c}$$

$$[\alpha] = \mathbf{X}_\chi^{\mathbf{T}} \cdot \mathbf{X}_\chi \tag{8.8d}$$

$$[\beta] = \mathbf{X}_\chi^{\mathbf{T}} \cdot \mathbf{Y}_\chi \tag{8.8e}$$

$$\mathbf{a} = [\alpha]^{-1} \cdot [\beta] \ . \tag{8.8f}$$

Having calculated the derived coefficients **a**, we can calculate the residuals. In doing so we must recall that $\mathbf{X}_\chi$ and $\mathbf{Y}_\chi$ contain factors of $\frac{1}{\sigma_m}$ and $[\alpha]^{-1}$ contains factors of σ_m^2. With all this, we can write the chi-square fit predicted data values as

$$\overline{\mathbf{Y}_\chi} = \mathbf{X}_\chi \cdot \mathbf{a} \tag{8.8g}$$

and the chi-square residuals as

$$\delta\mathbf{Y}_\chi = \mathbf{Y}_\chi - \overline{\mathbf{Y}_\chi} \tag{8.8h}$$

Because the data vector $\mathbf{Y}_\chi$ contains factors of $\frac{1}{\sigma_m}$, so do the residuals $\delta\mathbf{Y}_\chi$. You should, of course, always look at the residuals from the fit, so *remember these scale factors affect the residual values!* For example, if all σ_m are identical and equal to σ, then $\mathbf{Y}_\chi = \frac{\mathbf{Y}}{\sigma}$. If they don't, then when you plot the residuals $\delta\mathbf{Y}_\chi$ *each one will have a different scale factor!*

Moving on, we have

$$\chi^2 = \delta\mathbf{Y}_\chi^{\mathbf{T}} \cdot \delta\mathbf{Y}_\chi \tag{8.8i}$$

$$\widehat{\chi^2} = \frac{\delta\mathbf{Y}_\chi^{\mathbf{T}} \cdot \delta\mathbf{Y}_\chi}{M - N} \ . \tag{8.8j}$$

Finally, we have the analogy of equation 8.5 expressed in matrix form as in equation 3.7:

$$\mathbf{s_{a,intuit}}^2 = \widehat{\chi^2}\ diag\{[\alpha]^{-1}\}\ . \tag{8.9}$$

This *intuitively-derived* result is in contrast to the result derived from a *formal derivation*, which is the analogy to equation 8.4; again, it omits the $\widehat{\chi^2}$ factor:

$$\mathbf{s_{a,formal}}^2 = diag\{[\alpha]^{-1}\}\ . \tag{8.10}$$

This formally-derived result is what's quoted in textbooks (e.g. NR equation 15.4.15, BR equation 7.25). It provides parameter errors that are independent of the datapoint residuals, and leads to the same difficulties discussed above for the weighted mean case.

8.3. Which equation—8.9 or 8.10?

In most cases—but not all—we recommend that you use equation 8.9. Equation 8.9 is very reasonable. Suppose, for example, that the least-squares fit model is perfect and the only deviations from the fitted curve result from measurement error. Then by necessity we have $s^2 \approx \sigma^2_{meas}$ and $\widehat{\chi^2} \approx 1$. (We write "≈" instead of "=" because different experiments produce somewhat different values of s^2 because of statistical fluctuations; an average gives $\sigma^2 = \langle s^2 \rangle$.) In this situation, though, equations 8.9 and 8.10 are identical. However, if the least-squares fit model is *not correct*, meaning that it doesn't apply to the data, then the residuals will be larger than the intrinsic measurement errors, which will lead to larger values of χ^2 and $\widehat{\chi^2}$.

However, equation 8.9 is not a panacea. The numerical value of $\widehat{\chi^2}$ is subject to statistical variation. If the number of datapoints M is small (or, more properly, if the number of degrees of freedom $(M - N)$ is small), then the fractional statistical variation in $\widehat{\chi^2}$ is large and this affects the normalization inherent in equation 8.9. Alternatively, if you *really do know* the experimental errors equation 8.10 is appropriate.

Use your head!

8.4. Datapoints with known *relative* but unknown *absolute* dispersions

Here the σ_m are all different. The m^{th} row of the equation-of-condition matrix $\mathbf{X}$ and the m^{th} element of the data vector $\mathbf{Y}$ get divided by their corresponding σ_m. The equation embodied in each row of the matrix equation 2.2 remains unchanged, but the different rows are weighted differently with respect to each other.

Consider two measurements with intrinsic measurement uncertainties (σ_1, σ_2); suppose $\sigma_1 < \sigma_2$. After being divided by their respective σ_m's, all of the numbers in row 1 are larger than those in row 2. In all subsequent matrix operations, these larger numbers contribute more to all of the matrix-element products and sums. Thus, the measurement with smaller uncertainty has more influence on the final result, as it should.

Suppose that the above two measurements were taken under identical conditions except that measurement 1 received more integration time than measurement 2; we have $\frac{\sigma_1}{\sigma_2} = \left(\frac{\tau_1}{\tau_2}\right)^{-1/2}$, so the rows of $\mathbf{X}_\chi$ are weighted as $\tau^{1/2}$. This means that during the computation of $[\alpha] = \mathbf{X}_\chi^\mathbf{T} \cdot \mathbf{X}_\chi$, the self-products of row 1 are weighted as τ_1. This means that each datapoint is weighted as τ, which is exactly what you'd expect! Note that this is also exactly the same weighting scheme used in a weighted average, in which the weights are proportional to $\left(\frac{1}{\sigma_m}\right)^2$. We conclude that the weighting scheme of the first two steps in section 8.2 agrees with common sense.

Suppose you don't know the intrinsic measurement dispersion σ_m, but you *do* know the *relative* dispersion of the various measurements. For example, this would be the case if the datapoints were taken under identical conditions except for integration time; then $\sigma_m \propto \tau^{-1/2}$. In this case, multiply each row by its weight $w \propto \frac{1}{\sigma_m}$ and proceed as above. (The factors $\frac{1}{\sigma_m}$ in the equations of condition become $\frac{1}{\sigma_m^2}$ in the normal equations.)

8.5. Persnickety Diatribe on Choosing σ_m

8.5.1. Choosing and correcting σ_m

In the previous section, equation 8.10 taught us that—formally, at least—the variances in the derived fit parameters (or their uncertainties, which are the square roots) depend only on the adopted uncertainties σ_m and not on the *actual variance* of the *datapoints.*

Are you bothered by the fact that the variances of the derived parameters $\mathbf{s_a}$ are independent of the data residuals? You should be: it is obvious that the residuals should affect $\mathbf{s_a}$.

Formally, $\mathbf{s_a}$ depends only on the *adopted* uncertainties σ_m, which are chosen beforehand by you—you're supposed be such a good experimentalist that you really do know the intrinsic uncertainty in your measured values. Moreover, you are assuming that there are no other sources of uncertainty—such as "cosmic scatter" or an inappropriate model to which you are fitting the data. Suppose your adopted values of σ_m are off by a common scale factor, i.e. if $\sigma_{m,adopted} = f\sigma_{m,true}$. Then $\widehat{\chi^2} \approx f^{-2}$ instead of $\widehat{\chi^2} \approx 1$. And to obtain the parameter errors from $\delta\chi^2$, you must find the offset δx such that $\Delta\chi^2 = f^{-2} \approx \widehat{\chi^2}$.

You can correct for this erroneous common factor f by dividing your adopted values of σ_m by f. Of course, you don't know what this factor f is until you do the chi square fit. Dividing them by f is equivalent to multiplying them by $\widehat{\chi}$. And, of course, the same as multiplying σ_m^2 by $\widehat{\chi^2}$.

8.5.2. When you're using equation 8.9. . .

To be kosher, after having run through the problem once with the adopted σ_m, calculate the $\widehat{\chi^2}$; multiply all σ_m by $\widehat{\chi}$; and redo the problem so that the new $\widehat{\chi^2} = 1$. Then the derived variance $\mathbf{s_a}$ is also correct. You can obtain it either as the corresponding diagonal to the covariance matrix (equations 8.9 and 8.10, which are identical in this case) or by finding what departure from x_0 is necessary to make $\Delta\chi^2 = 1$.[6] This redoing the fit may seem like unnecessary work, but when we deal with multiparameter error estimation it's the best way to go to keep yourself from getting confused.

8.5.3. Think about your results!

In the case $\Delta\chi^2 \approx 1$ (and $\widehat{\chi^2} \approx 1$) the dispersions of the observed points s_m are equal to the intrinsic dispersions of the datapoints σ_m and the mathematical model embodied in the least-squares fit is perfect. That, at least, is the *theoretical* conclusion. In practice, however, your obtaining such a low, good value for $\widehat{\chi^2}$ might mean instead that you are using too large values for σ_m: you are ascribing more error to your datapoints than they really have, perhaps by not putting enough faith in your instrument.

But there is *another way* you can get artificially small values for $\widehat{\chi^2}$. This will occur if your *measurements are correlated.* Suppose, for example, that by mistake you include the same measurements several times in your fit. Then your measurements are no longer independent. Cowan discusses this possibility in his §7.6.

High values of $\widehat{\chi^2}$ indicate that the model is not perfect and could be improved by the use of a different model, such as the addition of more parameters—or, alternatively, that you think your equipment works better than it really does and you are ascribing *less* error to your datapoints than they really have. And in this case, using equation 8.10 instead of 8.9 is disastrous.

Think about your results.

8.5.4. When your measurements are correlated. . .

One more point, a rather subtle one. There are circumstances in which your datapoints are not independent. Then the formulation of chi-square fitting (and least-squares fitting, for that matter) is more complicated. You need to calculate the covariance matrix for the measured values y_m; call this covariance matrix $\mathbf{V}$. If this matrix is not unitary, then χ^2 is no longer given by equation 8.8i. Rather, it is given by

[6]To understand this comment about $\Delta\chi^2 = 1$, see §8.1.

$$\chi^2 = \delta\mathbf{Y^T} \cdot \mathbf{V^{-1}} \cdot \delta\mathbf{Y} \ . \tag{8.11a}$$

Of course, this leads to a different expression for **a**, which replaces equation 8.8f,

$$\mathbf{a} = (\mathbf{X^T} \cdot \mathbf{V^{-1}} \cdot \mathbf{X})^{-1} \cdot \mathbf{X^T} \cdot \mathbf{V^{-1}} \cdot \mathbf{Y} \ , \tag{8.11b}$$

and also to a different equation for the covariance matrix,

$$[\alpha]^{-1} = (\mathbf{X^T} \cdot \mathbf{V^{-1}} \cdot \mathbf{X})^{-1} \ . \tag{8.11c}$$

Correlated datapoints can occur when the measured y_m are affected by systematic errors or instrumental effects. Cowan §7.6 discusses this case. For example, suppose you take an image with a known point-spread function (psf) and want to fit an analytic function to this image. Example: a background intensity that changes linearly across the field plus a star. Here the independent variables in the function are the (x, y) pixel positions and the data are the intensities in each pixel. You'd take the intensity in each individual pixel and fit the assumed model. But here your data values are correlated because of the psf. Because you know the psf, you know the correlation between the various pixels. Such a formulation is required for CBR measurements because of the sidelobes of the radio telescope (which is just another way of saying "pdf").

Another case of correlated measurements occurs when your assumed model is incorrect. This is the very definition of correlation, because the residual δy_m is correlated with the data value y_m. But how do you calculate **V**? If you could do a large number J of experiments, each with M datapoints producing measured values $y_{m,j}$, each measured at different values of x_m, then each element of the covariance matrix would be $V_{mn} = \sum_j (y_{m,j} - \overline{y_m})(y_{n,j} - \overline{y_n})$. You don't normally have this opportunity. Much better is to look at your residuals; if the model doesn't fit, use another one!

Normally, and in particular we assume everywhere in this tutorial, the measurements are uncorrelated, so one takes $\mathbf{V} = \mathbf{I}$ (the unitary matrix).

9. BRUTE FORCE CHI-SQUARE AND THE CURVATURE MATRIX

9.1. Parameter Uncertainties in Brute Force chi-square Fitting

There are times when "brute force" least squares is appropriate. For example, if you have a nonlinear problem in which taking derivatives is complicated, and if the number of unknown coefficients is small, then it might be easier to search through the coefficient parameter space, calculate the χ^2 or s^2 for each combination of parameters, and find the minimum. This provides the best-fit parameter values.

How about the parameter uncertainties? Here we describe the case for a single parameter fit; call this parameter a. Generalizing to more parameters is straightforward.

For a chi-square fit, getting the uncertainty is easy. Calculate χ^2 as a function of the guessed values of a. As usual, define $\Delta\chi^2$ as χ^2 minus its minimum value; the minimum value gives the best estimate of a. The uncertainty in a is that offset where $\Delta\chi^2 = 1$. See §8.1.

For a least-squares fit, it's exactly the same idea. A least-squares fit implies that the measurement uncertainties σ_m are all identical, equal to σ. Thus, the sample variance $s^2 = \frac{1}{M-1}\sum \Delta y_m^2$ is equal to $\frac{\chi^2\sigma^2}{(M-1)}$. In other words, $\chi^2 = \frac{(M-1)}{\sigma^2}s^2$, which has expectation value $(M-1)$. Therefore, the uncertainty in a is that offset where $\Delta\chi^2 = 1$, i.e. where $\Delta s^2 = \frac{\sigma^2}{(M-1)}$.

To be totally explicit: For the fitted value of a_{fit}, the sample variance is

$$s_{min}^2 = \frac{1}{M-1}\sum(y_m - a_{fit})^2 \tag{9.1}$$

As a is moved from its fitted value, s^2 increases, so we can speak of the minimum sample variance s_{min}^2. As we move a from its fitted value by amounts Δa, the uncertainty in a is that value of Δa for which s^2 increases by $\frac{s_{min}^2}{M-1}$, i.e. that value of Δa for which

$$\Delta s^2 = s^2 - s_{min}^2 = \frac{s_{min}^2}{M-1} \tag{9.2}$$

10. NOTATION COMPARISON WITH NUMERICAL RECIPES

I learned least squares from Appendix A of Chauvenet (1863). He didn't use χ^2 and didn't use matrix techniques, but §0 and 1 follows his development quite closely. I wrote the first version of this document before knowing of NR's treatment, which explains my orientation towards least squares instead of chi-square. I'm fortunate in this approach because it made me realize the pitfalls one can get into with chi-square, as I discuss in §8.

On the other hand, NR describe the least-squares approach with some disdain in the discussion of equation (15.1.6) and warn that it is "dangerous" because you aren't comparing the residuals to the intrinsic inaccuracies of the data. In astronomy, though, more often than not you don't have an independent assessment of σ_m. But you might know the relative weights, and this is a plus for chi-square fitting. In any case, heed our warnings about chi-square fitting in §8.

In this writeup I have revised my old notation to agree, partially, with NR's. This effort wasn't completely successful because I didn't read NR very carefully before starting. To make it easier to cross-reference this document with NR, I provide the following table of correspondences (left of the double arrow is ours, to the right is theirs):

$$\mathbf{X} \longleftrightarrow \mathbf{A} \tag{10.1a}$$

$$\mathbf{Y} \longleftrightarrow \mathbf{b} \tag{10.1b}$$

$$\mathbf{X^T} \cdot \mathbf{X} = \mathbf{XX} = [\alpha] \longleftrightarrow \mathbf{A^T} \cdot \mathbf{A} = [\alpha] \tag{10.1c}$$

$$\mathbf{XX}^{-1} = \mathbf{XXI} = [\alpha]^{-1} \longleftrightarrow [\alpha]^{-1} = [C] = \mathbf{C} \tag{10.1d}$$

I use M for the number of measurements and N for the number of unknown coefficients; NR uses the opposite, so we have

$$N \longleftrightarrow M \tag{10.1e}$$

$$M \longleftrightarrow N \tag{10.1f}$$

Confusing, hey what?

It is a great pleasure to thank Tim Robishaw for his considerable effort in providing detailed comments on several aspects of this paper. These comments led to significant revisions and improvements. He also fixed bad formatting and manuscript glitches.

REFERENCES

Bevington, P.R. & Robinson, D. 1992, Data Reduction and Error Analysis for the Physical Sciences (WCB/McGraw-Hill).

Chauvenet, W. 1863, A Manual of Spherical and Practical Astronomy, Dover Press.

Cowan, G. 1998, Statistical Data Analysis, Clarendeon Press.

Press, W.H., Flannery, B.P., Teukolsky, S.A., & Vetterling, W.T. 2001, Numerical Recipes (second edition), Cambridge University Press.

Taylor, J.R. 1997, An Introduction to Error Analysis, University Science Books.

This preprint was prepared with the AAS LaTeX macros v5.0.