Numerical Recipes in C

The Art of Scientific Computing

William H. Press

Harvard-Smithsonian Center for Astrophysics

Brian P. Flannery

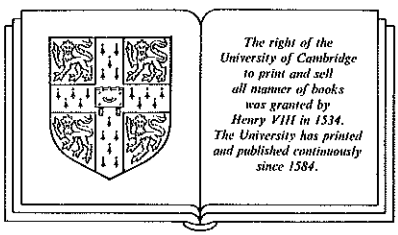
EXXON Research and Engineering Company

Saul A. Teukolsky

Department of Physics, Cornell University

William T. Vetterling

Polaroid Corporation



CAMBRIDGE UNIVERSITY PRESS
Cambridge
New York Port Chester
Melbourne Sydney

Chapter 14. Modeling of Data

14.0 Introduction

Given a set of observations, one often wants to condense and summarize the data by fitting it to a "model" that depends on adjustable parameters. Sometimes the model is simply a convenient class of functions, such as polynomials or Gaussians, and the fit supplies the appropriate coefficients. Other times, the model's parameters come from some underlying theory that the data are supposed to satisfy; examples are coefficients of rate equations in a complex network of chemical reactions, or orbital elements of a binary star. Modeling can also be used as a kind of constrained interpolation, where you want to extend a few data points into a continuous function, but with some underlying idea of what that function should look like.

The basic approach in all cases is usually the same: You choose or design a *figure-of-merit function* ("merit function," for short) that measures the agreement between the data and the model with a particular choice of parameters. The merit function is conventionally arranged so that small values represent close agreement. The parameters of the model are then adjusted to achieve a minimum in the merit function, yielding *best-fit parameters*. The adjustment process is thus a problem in minimization in many dimensions. This optimization was the subject of Chapter 10; however, there exist special, more efficient, methods that are specific to modeling, and we will discuss these in this chapter.

There are important issues that go beyond the mere finding of best-fit parameters. Data are generally not exact. They are subject to *measurement errors* (called *noise* in the context of signal-processing). Thus, typical data never exactly fit the model that is being used, even when that model is correct. We need the means to assess whether or not the model is appropriate, that is, we need to test the *goodness-of-fit* against some useful statistical standard.

We usually also need to know the accuracy with which parameters are determined by the data set. In other words, we need to know the likely errors of the best-fit parameters.

Finally, it is not uncommon in fitting data to discover that the merit function is not unimodal, with a single minimum. In some cases, we may be interested in global, rather than local questions. Not, "how good is this fit?", but rather, "how sure am I that there is not a *very much better* fit in some

corner of parameter space?" As we have seen in Chapter 10, especially §10.7, this kind of problem is generally quite difficult to solve.

The important message we want to deliver is that fitting of parameters is not the end-all of parameter estimation. To be genuinely useful, a fitting procedure should provide (i) parameters, (ii) error estimates on the parameters, and (iii) a statistical measure of goodness-of-fit. When the third item suggests that the model is an unlikely match to the data, then items (i) and (ii) are probably worthless. Unfortunately, many practitioners of parameter estimation never proceed beyond item (i)! They deem a fit acceptable if a graph of data and model "looks good." This approach is known as *chi-by-eye*. Luckily, its practitioners get what they deserve.

REFERENCES AND FURTHER READING:

Bevington, Philip R. 1969, *Data Reduction and Error Analysis for the Physical Sciences* (New York: McGraw-Hill).

Brownlee, K.A. 1965, *Statistical Theory and Methodology*, 2nd ed. (New York: Wiley).

Martin, B.R. 1971, *Statistics for Physicists* (New York: Academic Press).

von Mises, Richard. 1964, *Mathematical Theory of Probability and Statistics* (New York: Academic Press), Chapter X.

Korn, G.A., and Korn, T.M. 1968, *Mathematical Handbook for Scientists and Engineers*, 2nd ed. (New York: McGraw-Hill), Chapters 18–19.

14.1 Least Squares as a Maximum Likelihood Estimator

Suppose that we are fitting N data points (x_i, y_i) $i = 1, \ldots, N$, to a model which has M adjustable parameters a_j, $j = 1, \ldots, M$. The model predicts a functional relationship between the measured independent and dependent variables,

$$y(x) = y(x; a_1 \ldots a_M) \tag{14.1.1}$$

where the dependence on the parameters is indicated explicitly on the right-hand side.

What, exactly, do we want to minimize to get fitted values for the a_j's? The first thing that comes to mind is the familiar least-squares fit,

$$\text{minimize over } a_1 \ldots a_M : \quad \sum_{i=1}^{N} [y_i - y(x_i; a_1 \ldots a_M)]^2 \tag{14.1.2}$$

But where does this come from? What general principles is it based on? The answer to these questions takes us into the subject of *maximum likelihood estimators*.

Given a particular data set of x_i's and y_i's, we have the intuitive feeling that some parameter sets $a_1 \ldots a_M$ are very unlikely — those for which the model function $y(x)$ looks *nothing like* the data — while others may be very likely — those which closely resemble the data. How can we quantify this intuitive feeling? How can we select fitted parameters that are "most likely" to be correct? It is not meaningful to ask the question, "What is the probability that a particular set of fitted parameters $a_1 \ldots a_M$ is correct?" The reason is that there is no statistical universe of models from which the parameters are drawn. There is just one model, the correct one, and a statistical universe of data sets that are drawn from it!

That being the case, we can, however, turn the question around, and ask, "*Given a particular set of parameters*, what is the probability that this data set could have occurred?" If the y_i's take on continuous values, the probability will always be zero unless we add the phrase, "...plus or minus some fixed Δy on each data point." So let's always take this phrase as understood. If the probability of obtaining the data set is infinitesimally small, then we can conclude that the parameters under consideration are "unlikely" to be right. Conversely, our intuition tells us that the data set should not be too improbable for the correct choice of parameters.

In other words, we identify the probability of the data given the parameters (which is a mathematically computable number), as the *likelihood* of the parameters given the data. This identification is entirely based on intuition. It has no formal mathematical basis in and of itself; as we already remarked, statistics is *not* a branch of mathematics!

Once we make this intuitive identification, however, it is only a small further step to decide to fit for the parameters $a_1 \ldots a_M$ precisely by finding those values that *maximize* the likelihood defined in the above way. This form of parameter estimation is *maximum likelihood estimation*.

We are now ready to make the connection to (14.1.2). Suppose that each data point y_i has a measurement error that is independently random and distributed as a normal (Gaussian) distribution around the "true" model $y(x)$. And suppose that the standard deviations σ of these normal distributions are the same for all points. Then the probability of the data set is the product of the probabilities of each point,

$$P = \prod_{i=1}^{N} \left\{ \exp\left[-\frac{1}{2} \left(\frac{y_i - y(x_i)}{\sigma} \right)^2 \right] \Delta y \right\} \tag{14.1.3}$$

Notice that there is a factor Δy in each term in the product. Maximizing (14.1.3) is equivalent to maximizing its logarithm, or minimizing the negative of its logarithm, namely,

$$\left[\sum_{i=1}^{N} \frac{[y_i - y(x_i)]^2}{2\sigma^2} \right] - N \log \Delta y \tag{14.1.4}$$

Since N, σ and Δy are all constants, minimizing this equation is equivalent to minimizing (14.1.2).

What we see is that least-squares fitting *is* a maximum likelihood estimation of the fitted parameters *if* the measurement errors are independent and normally distributed with constant standard deviation. Notice that we made no assumption about the linearity or nonlinearity of the model $y(x; a_1 \ldots)$ in its parameters $a_1 \ldots a_M$. Just below, we will relax our assumption of constant standard deviations and obtain the very similar formulas for what is called "chi-square fitting" or "weighted least-squares fitting." First, however, let us discuss further our very stringent assumption of a normal distribution.

For a hundred years or so, mathematical statisticians have been in love with the fact that the probability distribution of the sum of a very large number of very small random deviations always converges to a normal distribution. (For precise statements of this *central limit theorem*, consult von Mises or other standard works on mathematical statistics.) This infatuation tended to focus interest away from the fact that, for real data, the normal distribution is often rather poorly realized, if it is realized at all. We are often taught, rather casually, that, on average, measurements will fall within $\pm\sigma$ of the true value 68 percent of the time, within $\pm 2\sigma$ 95 percent of the time, and within $\pm 3\sigma$ 99.7 percent of the time. Extending this, one would expect a measurement to be off by $\pm 20\sigma$ only one time out of 2×10^{88}. We all know that "glitches" are much more likely than *that*!

In some instances, the deviations from a normal distribution are easy to understand and quantify. For example, in measurements obtained by counting events, the measurement errors are usually distributed as a Poisson distribution, whose cumulative probability function was already discussed in §6.2. When the number of counts going into one data point is large, the Poisson distribution converges toward a Gaussian. However, the convergence is not uniform when measured in fractional accuracy. The more standard deviations out on the tail of the distribution, the larger the number of counts must be before a value close to the Gaussian is realized. The sign of the effect is always the same: the Gaussian predicts that "tail" events are much less likely than they actually (by Poisson) are. This causes such events, when they occur, to skew a least-squares fit much more than they ought.

Other times, the deviations from a normal distribution are not so easy to understand in detail. Experimental points are occasionally just *way off*. Perhaps the power flickered during a point's measurement, or someone kicked the apparatus, or someone wrote down a wrong number. Points like this are called *outliers*. They can easily turn a least-squares fit on otherwise adequate data into nonsense. Their probability of occurrence in the assumed Gaussian model is so small that the maximum likelihood estimator is willing to distort the whole curve to try to bring them, mistakenly, into line.

The subject of *robust statistics* deals with cases where the normal or Gaussian model is a bad approximation, or cases where outliers are important. We will discuss robust methods briefly in §14.6. All the sections between this one and that one assume, one way or the other, a Gaussian model for the measurement errors in the data. It it quite important that you keep the

limitations of that model in mind, even as you use the very useful methods which follow from assuming it.

Finally, note that our discussion of measurement errors has been limited to *statistical* errors, the kind that will average away if we only take enough data. Measurements are also susceptible to *systematic* errors that will not go away with any amount of averaging. For example, the calibration of a metal meter stick might depend on its temperature. If we take all our measurements at the same wrong temperature, then no amount of averaging or numerical processing will correct for this unrecognized systematic error.

Chi-Square Fitting

We considered the chi-square statistic once before, in §13.5. Here it arises in a slightly different context.

If each data point (x_i, y_i) has its own standard deviation σ_i, then equation (14.1.3) is modified only by putting a subscript i on the symbol σ. That subscript also propagates docilely into (14.1.4), so that the maximum likelihood estimate of the model parameters is obtained by minimizing the quantity

$$\chi^2 \equiv \sum_{i=1}^{N} \left(\frac{y_i - y(x_i; a_1 \ldots a_M)}{\sigma_i} \right)^2 \tag{14.1.5}$$

called the "chi-square."

To whatever extent the measurement errors actually *are* normally distributed, the quantity χ^2 is correspondingly a sum of N squares of normally distributed quantities, each normalized to unit variance. Once we have adjusted the $a_1 \ldots a_M$ to minimize the value of χ^2, the terms in the sum are not all statistically independent. However it turns out that the probability distribution for different values of χ^2 at its minimum can nevertheless be derived analytically, and is the *chi-square distribution for $N - M$ degrees of freedom.* We learned how to compute this probability function using the incomplete gamma function `gammq` in §6.2. In particular, equation (6.2.18) gives the probability Q that the chi-square should exceed a particular value χ^2 by chance, where $\nu = N - M$ is the *number of degrees of freedom.* The quantity Q, or its complement $P \equiv 1 - Q$ is frequently tabulated in appendices to statistics books, but we generally find it easier to use `gammq` and compute our own values: q=`gammq(0.5*`ν`,0.5*`χ^2`)`.

This computed probability gives a quantitative measure for the goodness-of-fit of the model. If Q is a very small probability for some particular data set, then the apparent discrepancies are unlikely to be chance fluctuations. Much more probably either (i) the model is wrong — can be statistically rejected, or (ii) someone has lied to you about the size of the measurement errors σ_i — they are really larger than stated.

It is an important point that the chi-square probability Q does not directly measure the credibility of the assumption that the measurement errors

are normally distributed. It assumes they are. In most, but not all, cases, however, the effect of nonnormal errors is to create an abundance of outlier points. These decrease the probability Q, so that we can add another possible, though less definitive, conclusion to the above list: (iii) the measurement errors may not be normally distributed.

Possibility (iii) is fairly common, and also fairly benign. It is for this reason that reasonable experimenters are often rather tolerant of low probabilities Q. It is not uncommon to deem acceptable on equal terms any models with, say, $Q > 0.001$. This is not as sloppy as it sounds: truly *wrong* models will often be rejected with vastly smaller values of Q, 10^{-18}, say. However, if day-in and day-out you find yourself accepting models with $Q \sim 10^{-3}$, you really should track down the cause.

If you happen to know the actual distribution law of your measurement errors, then you might wish to *Monte Carlo simulate* some data sets drawn from a particular model, cf. §7.2–§7.3. You can then subject these synthetic data sets to your actual fitting procedure, so as to determine both the probability distribution of the χ^2 statistic, and also the accuracy with which your model parameters are reproduced by the fit. We discuss this further in §14.5. The technique is very general, but it can also be very expensive.

At the opposite extreme, it sometimes happens that the probability Q is too large, too near to 1, literally too good to be true! Nonnormal measurement errors cannot in general produce this disease, since the normal distribution is about as "compact" as a distribution can be. Almost always, the cause of too good a chi-square fit is that the experimenter, in a "fit" of conservativism, has *overestimated* his or her measurement errors. Very rarely, too good a chi-square signals actual fraud, data that has been "fudged" to fit the model.

A rule of thumb is that a "typical" value of χ^2 for a "moderately" good fit is $\chi^2 \approx \nu$. More precise is the statement that, asymptotically for large ν, the statistic χ^2 becomes normally distributed with a mean ν and a standard deviation $\sqrt{2\nu}$.

In some cases the uncertainties associated with a set of measurements are not known in advance, and considerations related to χ^2 fitting are used to derive a value for σ. If we assume that all measurements have the same standard deviation, $\sigma_i = \sigma$, and that the model does fit well, then we can proceed by first assigning an arbitrary constant σ to all points, next fitting for the model parameters by minimizing χ^2, and finally recomputing

$$\sigma^2 = \sum_{i=1}^{N} [y_i - y(x_i)]^2 / N \tag{14.1.6}$$

Obviously, this approach prohibits an independent assessment of goodness-of-fit, a fact occasionally missed by its adherents. When, however, the measurement error is not known, this approach at least allows *some* kind of error bar to be assigned to the points.

If we take the derivative of equation (14.1.5) with respect to the parameters a_k, we obtain equations which must hold at the chi-square minimum,

$$0 = \sum_{i=1}^{N} \left(\frac{y_i - y(x_i)}{\sigma_i^2} \right) \left(\frac{\partial y(x_i; \ldots a_k \ldots)}{\partial a_k} \right) \qquad k = 1, \ldots, M \qquad (14.1.7)$$

Equation (14.1.7) is, in general, a set of M nonlinear equations for the M unknown a_k. Various of the procedures described subsequently in this chapter derive from (14.1.7) and its specializations.

REFERENCES AND FURTHER READING:

Bevington, Philip R. 1969, *Data Reduction and Error Analysis for the Physical Sciences* (New York: McGraw-Hill), Chapters 1-4.

von Mises, Richard. 1964, *Mathematical Theory of Probability and Statistics* (New York: Academic Press), §VI.C.

14.2 Fitting Data to a Straight Line

A concrete example will make the considerations of the previous section more meaningful. We consider the problem of fitting a set of N data points (x_i, y_i) to a straight-line model

$$y(x) = y(x; a, b) = a + bx \qquad (14.2.1)$$

This problem is often called *linear regression*, a terminology that originated, long ago, in the social sciences. We assume that the uncertainty σ_i associated with each measurement y_i is known, and that the x_i's (values of the dependent variable) are known exactly.

To measure how well the model agrees with the data, we use the chi-square merit function (14.1.5), which in this case is

$$\chi^2(a, b) = \sum_{i=1}^{N} \left(\frac{y_i - a - bx_i}{\sigma_i} \right)^2 \qquad (14.2.2)$$

If the measurement errors are normally distributed, then this merit function will give maximum likelihood parameter estimations of a and b; if the errors are not normally distributed, then the estimations are not maximum likelihood, but may still be useful in a practical sense. In §14.6, we will treat the case where outlier points are so numerous as to render the χ^2 merit function useless.

Equation (14.2.2) is minimized to determine a and b. At its minimum, derivatives of $\chi^2(a,b)$ with respect to a, b vanish.

$$0 = \frac{\partial \chi^2}{\partial a} = -2\sum_{i=1}^{N} \frac{y_i - a - bx_i}{\sigma_i^2}$$
$$0 = \frac{\partial \chi^2}{\partial b} = -2\sum_{i=1}^{N} \frac{x_i(y_i - a - bx_i)}{\sigma_i^2} \tag{14.2.3}$$

These conditions can be rewritten in a convenient form if we define the following sums,

$$S \equiv \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \quad S_x \equiv \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} \quad S_y \equiv \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2}$$
$$S_{xx} \equiv \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} \quad S_{xy} \equiv \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} \tag{14.2.4}$$

With these definitions (14.2.3) becomes

$$aS + bS_x = S_y$$
$$aS_x + bS_{xx} = S_{xy} \tag{14.2.5}$$

The solution of these two equations in two unknowns is calculated as

$$\Delta \equiv SS_{xx} - (S_x)^2$$
$$a = \frac{S_{xx}S_y - S_x S_{xy}}{\Delta} \tag{14.2.6}$$
$$b = \frac{SS_{xy} - S_x S_y}{\Delta}$$

Equation (14.2.6) gives the solution for the best-fit model parameters a and b.

We are not done, however. We must estimate the probable uncertainties in the estimates of a and b, since obviously the measurement errors in the data must introduce some uncertainty in the determination of those parameters. If the data are independent, then each contributes its own bit of uncertainty to the parameters. Consideration of propagation of errors shows that the variance σ_f^2 in the value of any function will be

$$\sigma_f^2 = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial f}{\partial y_i}\right)^2 \tag{14.2.7}$$

For the straight line, the derivatives of a and b with respect to y_i can be directly evaluated from the solution:

$$\frac{\partial a}{\partial y_i} = \frac{S_{xx} - S_x x_i}{\sigma_i^2 \Delta}$$
$$\frac{\partial b}{\partial y_i} = \frac{S x_i - S_x}{\sigma_i^2 \Delta} \tag{14.2.8}$$

Summing over the points as in (14.2.7), we get

$$\sigma_a^2 = S_{xx}/\Delta$$
$$\sigma_b^2 = S/\Delta \tag{14.2.9}$$

which are the variances in the estimates of a and b, respectively. We will see in §14.5 that an additional number is also needed to characterize properly the probable uncertainty of the parameter estimation. That number is the *covariance* of a and b, and (as we will see below) is given by

$$\mathrm{Cov}(a, b) = -S_x/\Delta \tag{14.2.10}$$

The coefficient of correlation between the uncertainty in a and the uncertainty in b, which is a number between -1 and 1, follows from (14.2.10) (compare equation 13.7.1),

$$r_{ab} = \frac{-S_x}{\sqrt{S S_{xx}}} \tag{14.2.11}$$

A positive value of r_{ab} indicates that the errors in a and b are likely to have the same sign, while a negative value indicates the errors are anticorrelated, likely to have opposite signs.

We are *still* not done. We must estimate the goodness-of-fit of the data to the model. Absent this estimate, we have not the slightest indication that the parameters a and b in the model have any meaning at all! The probability Q that a value of chi-square as *poor* as the value (14.2.2) should occur by chance is

$$Q = \texttt{gammq}\left(\frac{N-2}{2}, \frac{\chi^2}{2}\right) \tag{14.2.12}$$

Here `gammq` is our routine for the incomplete gamma function $Q(a, x)$, §6.2. If Q is larger than, say, 0.1, then the goodness-of-fit is believable. If it is larger than, say, 0.001, then the fit *may* be acceptable if the errors are nonnormal or

have been moderately underestimated. If Q is less than 0.001 then the model and/or estimation procedure can rightly be called into question. In this latter case, turn to §14.6 to proceed further.

If you do not know the individual measurement errors of the points σ_i, and are proceeding (dangerously) to use equation (14.1.6) for estimating these errors, then here is the procedure for estimating the probable uncertainties of the parameters a and b: Set $\sigma_i \equiv 1$ in all equations through (14.2.6), and multiply σ_a and σ_b, as obtained from equation (14.2.9), by the additional factor $\sqrt{\chi^2/(N-2)}$, where χ^2 is computed by (14.2.2) using the fitted parameters a and b. As discussed above, this procedure is equivalent to *assuming* a good fit, so you get no independent goodness-of-fit probability Q.

In §13.7 we promised a relation between the linear correlation coefficient r (equation 13.7.1) and a goodness-of-fit measure, χ^2 (equation 14.2.2). For unweighted data (all $\sigma_i = 1$), that relation is

$$\chi^2 = (1 - r^2)\mathrm{NVar}\,(y_1 \ldots y_N) \tag{14.2.13}$$

where

$$\mathrm{NVar}\,(y_1 \ldots y_N) \equiv \sum_{i=1}^{N} (y_i - \overline{y})^2 \tag{14.2.14}$$

For data with varying weights σ_i, the above equations remain valid if the sums in equation (13.7.1) are weighted by $1/\sigma_i^2$.

The following function, `fit`, carries out exactly the operations that we have discussed. When the weights σ are known in advance, the calculations exactly correspond to the formulas above. However, when weights σ are unavailable, the routine *assumes* equal values of σ for each point and *assumes* a good fit, as discussed in §14.1.

The formulas (14.2.6) are susceptible to roundoff error. Accordingly, we rewrite them as follows: Define

$$t_i = \frac{1}{\sigma_i}\left(x_i - \frac{S_x}{S}\right), \qquad i = 1, 2, \ldots, N \tag{14.2.15}$$

and

$$S_{tt} = \sum_{i=1}^{N} t_i^2 \tag{14.2.16}$$

Then, as you can verify by direct substitution,

$$b = \frac{1}{S_{tt}} \sum_{i=1}^{N} \frac{t_i y_i}{\sigma_i} \tag{14.2.17}$$

$$a = \frac{S_y - S_x b}{S} \tag{14.2.18}$$

$$\sigma_a^2 = \frac{1}{S}\left(1 + \frac{S_x^2}{SS_{tt}}\right) \tag{14.2.19}$$

$$\sigma_b^2 = \frac{1}{S_{tt}} \tag{14.2.20}$$

$$\mathrm{Cov}(a,b) = -\frac{S_x}{SS_{tt}} \tag{14.2.21}$$

$$r_{ab} = \frac{\mathrm{Cov}(a,b)}{\sigma_a \sigma_b} \tag{14.2.22}$$

```
#include <math.h>

static float sqrarg;
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)

void fit(x,y,ndata,sig,mwt,a,b,siga,sigb,chi2,q)
float x[],y[],sig[],*a,*b,*siga,*sigb,*chi2,*q;
int ndata,mwt;
```

Given a set of points `x[1..ndata]`, `y[1..ndata]` with standard deviations `sig[1..ndata]`, fit them to a straight line `y=a+bx` by minimizing χ^2. Returned are `a,b` and their respective probable uncertainties `siga` and `sigb`, the chi-square `chi2`, and the goodness-of-fit probability `q` (that the fit would have χ^2 this large or larger). If `mwt=0` on input, then the standard deviations are assumed to be unavailable: `q` is returned as `1.0` and the normalization of `chi2` is to unit standard deviation on all points.

```
{
    int i;
    float wt,t,sxoss,sx=0.0,sy=0.0,st2=0.0,ss,sigdat;
    float gammq();

    *b=0.0;
    if (mwt) {                              Accumulate sums ...
        ss=0.0;
        for (i=1;i<=ndata;i++) {            ...with weights
            wt=1.0/SQR(sig[i]);
            ss += wt;
            sx += x[i]*wt;
            sy += y[i]*wt;
        }
    } else {
        for (i=1;i<=ndata;i++) {            ...or without weights.
            sx += x[i];
            sy += y[i];
        }
        ss=ndata;
    }
    sxoss=sx/ss;
    if (mwt) {
        for (i=1;i<=ndata;i++) {
            t=(x[i]-sxoss)/sig[i];
            st2 += t*t;
            *b += t*y[i]/sig[i];
        }
    } else {
        for (i=1;i<=ndata;i++) {
            t=x[i]-sxoss;
            st2 += t*t;
            *b += t*y[i];
        }
    }
    *b /= st2;                              Solve for A, B, σ_a and σ_b.
```

```
        *a=(sy-sx*(*b))/ss;
        *siga=sqrt((1.0+sx*sx/(ss*st2))/ss);
        *sigb=sqrt(1.0/st2);
        *chi2=0.0;                                  Calculate χ².
        if (mwt == 0) {
            for (i=1;i<=ndata;i++)
                *chi2 += SQR(y[i]-(*a)-(*b)*x[i]);
            *q=1.0;
            sigdat=sqrt((*chi2)/(ndata-2));         For unweighted data evaluate typical sig us-
            *siga *= sigdat;                        ing chi2, and adjust the standard devia-
            *sigb *= sigdat;                        tions.
        } else {
            for (i=1;i<=ndata;i++)
                *chi2 += SQR((y[i]-(*a)-(*b)*x[i])/sig[i]);
            *q=gammq(0.5*(ndata-2),0.5*(*chi2));    §6.2
        }
    }
```

REFERENCES AND FURTHER READING:

Bevington, Philip R. 1969, *Data Reduction and Error Analysis for the Physical Sciences* (New York: McGraw-Hill), Chapter 6.

14.3 General Linear Least Squares

An immediate generalization of the previous section is to fit a set of data points (x_i, y_i) to a model which is not just a linear combination of 1 and x (namely $a + bx$), but rather a linear combination of *any* M specified functions of x. For example, the functions could be $1, x, x^2, \ldots, x^{M-1}$, in which case their general linear combination,

$$y(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_M x^{M-1} \tag{14.3.1}$$

is a polynomial of degree $M-1$. Or, the functions could be sines and cosines, in which case their general linear combination can be a harmonic series.

The general form of this kind of model is

$$y(x) = \sum_{k=1}^{M} a_k X_k(x) \tag{14.3.2}$$

where $X_1(x), \ldots, X_M(x)$ are arbitrary fixed functions of x, called the *basis functions.*

Note that the functions $X_k(x)$ can be wildly nonlinear functions of x. In this discussion "linear" only refers to the model's dependence on its *parameters* a_k.

For these linear models we generalize the discussion of the previous section by defining a merit function

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{M} a_k X_k(x_i)}{\sigma_i} \right]^2 \tag{14.3.3}$$

As before σ_i is the measurement error (standard deviation) of the i^{th} data point, presumed to be known. If the measurement errors are not known, they may all, as before, be set to the constant value $\sigma = 1$.

Once again, we will pick as best parameters those that minimize χ^2. There are several different techniques available for finding this minimum. Two are particularly useful, and we will discuss both in this section. To introduce them and elucidate their relationship, we need some notation.

Let $\mathbf{A}$ be a matrix whose $N \times M$ components are constructed from the M basis functions evaluated at the N abscissas x_i, and from the N measurement errors σ_i, by the prescription

$$A_{ij} = \frac{X_j(x_i)}{\sigma_i} \tag{14.3.4}$$

The matrix $\mathbf{A}$ is called the *design matrix* of the fitting problem. Notice that in general $\mathbf{A}$ has more rows than columns, $N \geq M$, since there must be more data points than model parameters to be solved for. (You can fit a straight line to two points, but not a very meaningful quintic!) The design matrix is shown schematically in Figure 14.3.1.

Also define a vector $\mathbf{b}$ of length N by

$$b_i = \frac{y_i}{\sigma_i} \tag{14.3.5}$$

and denote the M vector whose components are the parameters to be fitted, $a_1, \ldots, a_M$, by $\mathbf{a}$.

Solution by Use of the Normal Equations

The minimum of (14.3.3) occurs where the derivative of χ^2 with respect to all M parameters a_k vanishes. Specializing equation (14.1.7) to the case of the model (14.3.2), this condition yields the M equations

$$0 = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{M} a_j X_j(x_i) \right] X_k(x_i) \qquad k = 1, \ldots, M \tag{14.3.6}$$

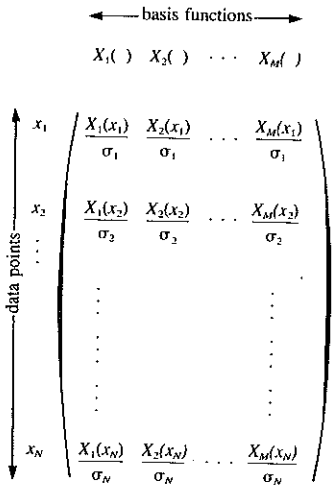


Figure 14.3.1. Design matrix for the least-squares fit of a linear combination of M basis functions to N data points. The matrix elements involve the basis functions evaluated at the values of the independent variable at which measurements are made, and the standard deviations of the measured dependent variable. The measured values of the dependent variable do not enter the design matrix.

Interchanging the order of summations, we can write (14.3.6) as the matrix equation

$$\sum_{j=1}^{M} \alpha_{kj} a_j = \beta_k \tag{14.3.7}$$

where

$$\alpha_{kj} = \sum_{i=1}^{N} \frac{X_j(x_i) X_k(x_i)}{\sigma_i^2} \qquad \text{or equivalently} \qquad [\alpha] = \mathbf{A}^T \cdot \mathbf{A} \tag{14.3.8}$$

an $M \times M$ matrix, and

$$\beta_k = \sum_{i=1}^{N} \frac{y_i X_k(x_i)}{\sigma_i^2} \qquad \text{or equivalently} \qquad [\beta] = \mathbf{A}^T \cdot \mathbf{b} \tag{14.3.9}$$

a vector of length M.

The equations (14.3.6) or (14.3.7) are called the *normal equations* of the least-squares problem. They can be solved for the vector of parameters $\mathbf{a}$ by the standard methods of Chapter 2, notably LU decomposition and backsubstitution or Gauss-Jordan elimination. In matrix form, the normal equations

can be written as either

$$[\alpha] \cdot \mathbf{a} = [\beta] \qquad \text{or as} \qquad \left(\mathbf{A}^T \cdot \mathbf{A}\right) \cdot \mathbf{a} = \mathbf{A}^T \cdot \mathbf{b} \tag{14.3.10}$$

The inverse matrix $C_{jk} \equiv [\alpha]^{-1}_{jk}$ is closely related to the probable (or, more precisely, *standard*) uncertainties of the estimated parameters $\mathbf{a}$. To estimate these uncertainties, consider that

$$a_j = \sum_{k=1}^{M} [\alpha]^{-1}_{jk} \beta_k = \sum_{k=1}^{M} C_{jk} \left[\sum_{i=1}^{N} \frac{y_i X_k(x_i)}{\sigma_i^2} \right] \tag{14.3.11}$$

and that the variance associated with the estimate a_j can be found as in (14.2.7) from

$$\sigma^2(a_j) = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial a_j}{\partial y_i} \right)^2 \tag{14.3.12}$$

Note that α_{jk} is independent of y_i, so that

$$\frac{\partial a_j}{\partial y_i} = \sum_{k=1}^{M} C_{jk} X_k(x_i)/\sigma_i^2 \tag{14.3.13}$$

Consequently, we find that

$$\sigma^2(a_j) = \sum_{k=1}^{M} \sum_{l=1}^{M} C_{jk} C_{jl} \left[\sum_{i=1}^{N} \frac{X_k(x_i) X_l(x_i)}{\sigma_i^2} \right] \tag{14.3.14}$$

The final term in brackets is just the matrix $[\alpha]$. Since this is the matrix inverse of $[C]$, (14.3.14) reduces immediately to

$$\sigma^2(a_j) = C_{jj} \tag{14.3.15}$$

In other words, the diagonal elements of $[C]$ are the variances (squared uncertainties) of the fitted parameters $\mathbf{a}$. It should not surprise you to learn that the off diagonal elements C_{jk} are the covariances between a_j and a_k (cf. 14.2.10); but we shall defer discussion of these to §14.5.

We will now give a routine that implements the above formulas for the general linear least-squares problem, by the method of normal equations. Since we wish to compute not only the solution vector $\mathbf{a}$ but also the covariance matrix $[C]$, it is most convenient to use Gauss-Jordan elimination

(routine gaussj of §2.1) to perform the linear algebra. The operation count, in this application, is no larger than that for LU decomposition. If you have no need for the covariance matrix, however, you can save a factor of 3 on the linear algebra by switching to LU decomposition, without computation of the matrix inverse.

We need to warn you that the solution of a least-squares problem directly from the normal equations is rather susceptible to roundoff error. An alternative, and preferred, technique involves QR decomposition (§11.3 and §11.6) of the design matrix $\mathbf{A}$. This is essentially what we did at the end of §14.2 for fitting data to a straight line, but without invoking all the machinery of QR to derive the necessary formulas. Later in this section, we will discuss other difficulties in the least-squares problem, for which the cure is *singular value decomposition* (SVD), of which we give an implementation. It turns out that SVD also fixes the roundoff problem, so it is our recommended technique for all but "easy" least-squares problems. It is for these easy problems that the following routine, which solves the normal equations, is intended.

The routine below introduces one bookkeeping complication that is quite useful in practical work. Frequently it is a matter of "art" to decide which parameters a_k in a model should be fit from the data set, and which should be held constant at fixed values, for example values predicted by a theory or measured in a previous experiment. One wants, therefore, to have a convenient means for "freezing" and "unfreezing" the parameters a_k. In the following routine the total number of parameters a_k is denoted ma (called M above), while mfit is the number of parameters which are to be adjusted in minimizing the best fit. As input to the routine, you supply a list lista. The first mfit elements of lista contain the numbers of the parameters that are to be adjusted. The remaining ma−mfit parameters will be held fixed at their input values. For example, if ma=8, mfit=4, and lista[1..4] contains the numbers 3,1,7,5, then the parameters a_3, a_1, a_7, a_5 will be adjusted. The other parameters (a_2, a_4, a_6, a_8) will be held fixed at their input values; notice that you *must* therefore initialize these input values before calling the program. On output, any frozen variable will have its variance and all its covariances set to zero in the covariance matrix.

```
static float sqrarg;
#define SQR(a) (sqrarg=(a),sqrarg*sqrarg)

void lfit(x,y,sig,ndata,a,ma,lista,mfit,covar,chisq,funcs)
int ndata,ma,lista[],mfit;
float x[],y[],sig[],a[],**covar,*chisq;
void (*funcs)();    /* ANSI: void (*funcs)(float,float *,int); */
```

Given a set of points x[1..ndata], y[1..ndata] with individual standard deviations given by sig[1..ndata], use χ^2 minimization to determine mfit of the coefficients a[1..ma] of a function that depends linearly on a, $y = \sum_i a_i \times$afunc$_i(x)$. The array lista[1..ma] renumbers the parameters so that the first mfit elements correspond to the parameters actually being determined; the remaining ma-mfit elements are held fixed at their input values. The program returns values for the ma fit parameters a, $\chi^2 =$ chisq, and the elements [1..mfit][1..mfit] of the covariance matrix covar[1..ma][1..ma]. The user supplies a routine funcs(x,afunc,ma) that returns the ma basis functions evaluated at $x =$x in the array afunc[1..ma].

```
{
    int k,kk,j,ihit,i;
    float ym,wt,sum,sig2i,**beta,*afunc;
    void gaussj(),covsrt(),nrerror(),free_matrix(),free_vector();
```

```
    float **matrix(),*vector();

    beta=matrix(1,ma,1,1);
    afunc=vector(1,ma);
    kk=mfit+1;                          Check to see that lista contains a proper permutation of
    for (j=1;j<=ma;j++) {                   the coefficients and fill in any missing members.
        ihit=0;
        for (k=1;k<=mfit;k++)
            if (lista[k] == j) ihit++;
        if (ihit == 0)
            lista[kk++]=j;
        else if (ihit > 1) nrerror("Bad LISTA permutation in LFIT-1");
    }
    if (kk != (ma+1)) nrerror("Bad LISTA permutation in LFIT-2");
    for (j=1;j<=mfit;j++) {             Initialize the (symmetric) matrix.
        for (k=1;k<=mfit;k++) covar[j][k]=0.0;
        beta[j][1]=0.0;
    }
    for (i=1;i<=ndata;i++) {            Loop over data to accumulate coefficients of the normal
        (*funcs)(x[i],afunc,ma);            equations.
        ym=y[i];
        if (mfit < ma)                  Subtract off dependences on known pieces of the fitting
            for (j=(mfit+1);j<=ma;j++)      function.
                ym -= a[lista[j]]*afunc[lista[j]];
        sig2i=1.0/SQR(sig[i]);
        for (j=1;j<=mfit;j++) {
            wt=afunc[lista[j]]*sig2i;
            for (k=1;k<=j;k++)
                covar[j][k] += wt*afunc[lista[k]];
            beta[j][1] += ym*wt;
        }
    }
    if (mfit > 1)
        for (j=2;j<=mfit;j++)           Fill in above the diagonal from symmetry.
            for (k=1;k<=j-1;k++)
                covar[k][j]=covar[j][k];
    gaussj(covar,mfit,beta,1);              Matrix solution.
    for (j=1;j<=mfit;j++) a[lista[j]]=beta[j][1];   Partition solution to appropriate
    *chisq=0.0;                                         coefficients a.
    for (i=1;i<=ndata;i++) {            Evaluate \chi^2 of the fit.
        (*funcs)(x[i],afunc,ma);
        for (sum=0.0,j=1;j<=ma;j++) sum += a[j]*afunc[j];
        *chisq += SQR((y[i]-sum)/sig[i]);
    }
    covsrt(covar,ma,lista,mfit);            Sort covariance matrix to true order of fitting
    free_vector(afunc,1,ma);                    coefficients.
    free_matrix(beta,1,ma,1,1);
}
```

That last call to a function `covsrt` is only for the purpose of spreading the `mfit`×`mfit` covariances back into the full `ma`×`ma` covariance matrix, sorted into the proper rows and columns and with zero variances and covariances set for variables which were held frozen. Thus, e.g., the variance of variable a_i will be in its natural place `covar[`i`][`i`]`. If, instead, you are willing to look up variances via the index `lista`, then you can omit the call. In that case, e.g., the variance of variable number `lista[`j`]` will be in `covar[`j`][`j`]`.

The function `covsrt` is as follows.

```
void covsrt(covar,ma,lista,mfit)
float **covar;
int ma,lista[],mfit;
Given the covariance matrix covar[1..ma][1..ma] of a fit for mfit of ma total parameters, and
their ordering lista[1..ma], repack the covariance matrix to the true order of the parameters.
Elements associated with fixed parameters will be zero.
{
    int i,j;
    float swap;

    for (j=1;j<ma;j++)                    Zero all elements below diagonal.
        for (i=j+1;i<=ma;i++) covar[i][j]=0.0;
    for (i=1;i<mfit;i++)                  Repack off-diagonal elements of fit into correct loca-
        for (j=i+1;j<=mfit;j++) {             tions below diagonal.
            if (lista[j] > lista[i])
                covar[lista[j]][lista[i]]=covar[i][j];
            else
                covar[lista[i]][lista[j]]=covar[i][j];
        }
    swap=covar[1][1];                     Temporarily store original diagonal elements in top row,
    for (j=1;j<=ma;j++) {                     and zero the diagonal.
        covar[1][j]=covar[j][j];
        covar[j][j]=0.0;
    }
    covar[lista[1]][lista[1]]=swap;       Now sort elements into proper order on diagonal.
    for (j=2;j<=mfit;j++) covar[lista[j]][lista[j]]=covar[1][j];
    for (j=2;j<=ma;j++)                   Finally, fill in above diagonal by symmetry.
        for (i=1;i<=j-1;i++) covar[i][j]=covar[j][i];
}
```

Solution by Use of Singular Value Decomposition

In some applications, the normal equations are perfectly adequate for linear least-squares problems. However, in many cases the normal equations are very close to singular. A zero pivot element may be encountered during the solution of the linear equations (e.g. in `gaussj`), in which case you get no solution at all. Or a very small pivot may occur, in which case you typically get fitted parameters a_k with very large magnitudes that are delicately (and unstably) balanced to cancel out almost precisely when the fitted function is evaluated.

Why does this commonly occur? The reason is that, more often than experimenters would like to admit, data do not clearly distinguish between two or more of the basis functions provided. If two such functions, or two different combinations of functions, happen to fit the data about equally well — or equally badly — then the matrix $[\alpha]$, unable to distinguish between them, neatly folds up its tent and becomes singular. There is a certain mathematical irony in the fact that least-squares problems are *both* overdetermined (number of data points greater than number of parameters) *and* underdetermined (ambiguous combinations of parameters exist); but that is how it frequently is. The ambiguities can be extremely hard to notice *a priori* in complicated problems.

Enter singular value decomposition (SVD). This would be a good time for you to review the material in §2.9, which we will not repeat here. In

the case of an overdetermined system, SVD produces a solution that is the best approximation in the least-squares sense, cf. equation (2.9.10). That is exactly what we want. In the case of an underdetermined system, SVD produces a solution whose values (for us, the a_k's) are smallest in the least-squares sense, cf. equation (2.9.8). That is also what we want: when some combination of basis functions is irrelevant to the fit, that combination will be driven down to a small, innocuous, value, rather than pushed up to delicately canceling infinities.

In terms of the design matrix $\mathbf{A}$ (equation 14.3.4) and the vector $\mathbf{b}$ (equation 14.3.5), minimization of χ^2 in (14.3.3) can be written as

$$\text{find} \quad \mathbf{a} \quad \text{which minimizes} \quad \chi^2 = |\mathbf{A} \cdot \mathbf{a} - \mathbf{b}|^2 \tag{14.3.16}$$

Comparing to equation (2.9.9), we see that this is precisely the problem which routines `svdcmp` and `svbksb` are designed to solve. The solution, which is given by equation (2.9.12), can be rewritten as follows: If $\mathbf{U}$ and $\mathbf{V}$ enter the SVD decomposition of $\mathbf{A}$ according to equation (2.9.1), as computed by `svdcmp`, then let the vectors $\mathbf{U}_{(i)}\ i = 1, \ldots, M$ denote the *columns* of $\mathbf{U}$ (each one a vector of length N); and let the vectors $\mathbf{V}_{(i)}; i = 1, \ldots, M$ denote the *columns* of $\mathbf{V}$ (each one a vector of length M). Then the solution (2.9.12) of the least-squares problem (14.3.16) can be written as

$$\mathbf{a} = \sum_{i=1}^{M} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{w_i} \right) \mathbf{V}_{(i)} \tag{14.3.17}$$

where the w_i are, as in §2.9, the singular values calculated by `svdcmp`.

Equation (14.3.17) says that the fitted parameters $\mathbf{a}$ are linear combinations of the columns of $\mathbf{V}$, with coefficients obtained by forming dot products of the columns of $\mathbf{U}$ with the weighted data vector (14.3.5). Though it is beyond our scope to prove here, it turns out that the standard (loosely, "probable") errors in the fitted parameters are also linear combinations of the columns of $\mathbf{V}$. In fact, equation (14.3.17) can be written in a form displaying these errors as

$$\mathbf{a} = \left[\sum_{i=1}^{M} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{w_i} \right) \mathbf{V}_{(i)} \right] \pm \frac{1}{w_1} \mathbf{V}_{(1)} \pm \cdots \pm \frac{1}{w_M} \mathbf{V}_{(M)} \tag{14.3.18}$$

Here each $\pm$ is followed by a standard deviation. The amazing fact is that, decomposed in this fashion, the standard deviations are all mutually independent (uncorrelated). Therefore they can be added together in root-mean-square fashion. What is going on is that the vectors $\mathbf{V}_{(i)}$ are the principal axes of the error ellipsoid of the fitted parameters $\mathbf{a}$ (see §14.5).

It follows that the variance in the estimate of a parameter a_j is given by

$$\sigma^2(a_j) = \sum_{i=1}^{M} \frac{1}{w_i^2} [\mathbf{V}_{(i)}]_j^2 = \sum_{i=1}^{M} \left(\frac{V_{ji}}{w_i} \right)^2 \tag{14.3.19}$$

whose result should be identical with (14.3.14). As before, you should not be surprised at the formula for the covariances, here given without proof,

$$\mathrm{Cov}(a_j, a_k) = \sum_{i=1}^{M} \left(\frac{V_{ji} V_{ki}}{w_i^2} \right) \tag{14.3.20}$$

We introduced this subsection by noting that the normal equations can fail by encountering a zero pivot. We have not yet, however, mentioned how SVD overcomes this problem. The answer is: If any singular value w_i is zero, its reciprocal in equation (14.3.18) should be set to zero, not infinity. (Compare the discussion preceding equation 2.9.7). This corresponds to adding to the fitted parameters **a** a *zero* multiple, rather than some random large multiple, of any linear combination of basis functions which are degenerate in the fit. It is a good thing to do!

Moreover, if a singular value w_i is nonzero but very small, you should also define *its* reciprocal to be zero, since its apparent value is probably an artifact of roundoff error, not a meaningful number. A plausible answer to the question "how small is small?", is to edit in this fashion all singular values whose ratio to the largest singular value is less than N times the machine precision ϵ. (You might argue for $\sqrt{N}$, or a constant, instead of N as the multiple; that starts getting into hardware-dependent questions.)

There is another reason for editing even *additional* singular values, ones large enough that roundoff error is not a question. Singular value decomposition allows you to identify linear combinations of variables which just happen not to contribute much to reducing the χ^2 of your data set. Editing these can sometimes reduce the probable error on your coefficients quite significantly, while increasing the minimum χ^2 only negligibly. We will learn more about identifying and treating such cases in §14.5. In the following routine, the point at which this kind of editing would occur is indicated.

Generally speaking, we recommend that you always use SVD techniques instead of using the normal equations. SVD's only significant disadvantage is that it requires an extra array of size $N \times M$ to store the whole design matrix. This storage is overwritten by the matrix **U**. Storage is also required for the $M \times M$ matrix **V**, but this is instead of the same-sized matrix for the coefficients of the normal equations. SVD can be significantly slower than solving the normal equations; however its great advantage, that it (theoretically) *cannot fail* more than makes up for the speed disadvantage.

In the routine that follows, the matrices `u`, `v` and the vector `w` are input as working space. The logical dimensions of the problem are `ndata` data points by `ma` basis functions (and fitted parameters). If you care only about the

values a of the fitted parameters, then u,v,w contain no useful information on output. If you want probable errors for the fitted parameters, read on.

```
#define TOL 1.0e-5

void svdfit(x,y,sig,ndata,a,ma,u,v,w,chisq,funcs)
float x[],y[],sig[],a[],**u,**v,w[],*chisq;
int ndata,ma;
void (*funcs)();    /* ANSI: void (*funcs)(float,float *,int); */
Given a set of points x[1..ndata], y[1..ndata] with individual standard deviations given
by sig[1..ndata], use χ² minimization to determine the coefficients a[1..ma] of the fitting
function y = Σ_i a_i × afunc_i(x). Here we solve the fitting equations using singular value decom-
position of the ndata by ma matrix, as in §2.9. Arrays u[1..ndata][1..ma], v[1..ma][1..ma],
and w[1..ma] provide workspace on input; on output they define the singular value decom-
position, and can be used to obtain the covariance matrix. The program returns values for
the ma fit parameters a, and χ², chisq. The user supplies a routine funcs(x,afunc,ma) that
returns the ma basis functions evaluated at x =x in the array afunc[1..ma].
{
    int j,i;
    float wmax,tmp,thresh,sum,*b,*afunc,*vector();
    void svdcmp(),svbksb(),free_vector();

    b=vector(1,ndata);
    afunc=vector(1,ma);
    for (i=1;i<=ndata;i++) {                    Accumulate coefficients of the fitting matrix.
        (*funcs)(x[i],afunc,ma);
        tmp=1.0/sig[i];
        for (j=1;j<=ma;j++) u[i][j]=afunc[j]*tmp;
        b[i]=y[i]*tmp;
    }
    svdcmp(u,ndata,ma,w,v);                     Singular value decomposition.
    wmax=0.0;                                   Edit the singular values, given TOL from the #define
    for (j=1;j<=ma;j++)                             statement, between here ...
        if (w[j] > wmax) wmax=w[j];
    thresh=TOL*wmax;
    for (j=1;j<=ma;j++)
        if (w[j] < thresh) w[j]=0.0;            ...and here.
    svbksb(u,w,v,ndata,ma,b,a);
    *chisq=0.0;                                 Evaluate chi-square.
    for (i=1;i<=ndata;i++) {
        (*funcs)(x[i],afunc,ma);
        for (sum=0.0,j=1;j<=ma;j++) sum += a[j]*afunc[j];
        *chisq += (tmp=(y[i]-sum)/sig[i],tmp*tmp);
    }
    free_vector(afunc,1,ma);
    free_vector(b,1,ndata);
}
```

Feeding the matrix v and vector w output by the above program into the following short routine, you easily obtain variances and covariances of the fitted parameters a. The square roots of the variances are the standard deviations of the fitted parameters. The routine straightforwardly implements equation (14.3.20) above, with the convention that singular values equal to zero are recognized as having been edited out of the fit.

```
void svdvar(v,ma,w,cvm)
float **v,w[],**cvm;
int ma;
To evaluate the covariance matrix cvm[1..ma][1..ma] of the fit for ma parameters obtained by
svdfit, call this routine with matrices v[1..ma][1..ma], w[1..ma] as returned from svdfit.
{
    int k,j,i;
    float sum,*wti,*vector();
    void free_vector();

    wti=vector(1,ma);
    for (i=1;i<=ma;i++) {
        wti[i]=0.0;
        if (w[i]) wti[i]=1.0/(w[i]*w[i]);
    }
    for (i=1;i<=ma;i++) {          Sum contributions to covariance matrix (14.3.20).
        for (j=1;j<=i;j++) {
            for (sum=0.0,k=1;k<=ma;k++) sum += v[i][k]*v[j][k]*wti[k];
            cvm[j][i]=cvm[i][j]=sum;
        }
    }
    free_vector(wti,1,ma);
}
```

Examples

Be aware that some apparently nonlinear problems can be expressed so that they are linear. For example, an exponential model with two parameters a and b,

$$y(x) = a\exp(-bx) \tag{14.3.21}$$

can be rewritten as

$$\log[y(x)] = c - bx \tag{14.3.22}$$

which is linear in its parameters c and b.

Also watch out for "non-parameters," as in

$$y(x) = a\exp(-bx + d) \tag{14.3.23}$$

Here the parameters a and d are, in fact, indistinguishable. This is a good example of where the normal equations will be exactly singular, and where SVD will find a zero singular value. SVD will then make a "least-squares" choice for setting a balance between a and d (or, rather, their equivalents in the linear model derived by taking the logarithms). However — and this is true whenever SVD gives back a zero singular value — you are better advised to figure out analytically where the degeneracy is among your basis functions, and then make appropriate deletions in the basis set.

Here are two examples for user-supplied routines `funcs`. The first one is trivial and fits a general polynomial to a set of data:

```
void fpoly(x,p,np)
float x,p[];
int np;
Fitting routine for a polynomial of degree NP-1, with coefficients in the array p[1..np].
{
    int j;

    p[1]=1.0;
    for (j=2;j<=np;j++) p[j]=p[j-1]*x;
}
```

The second example is slightly less trivial. It is used to fit Legendre polynomials up to some order `nl-1` through a data set.

```
void fleg(x,pl,nl)
float x,pl[];
int nl;
Fitting routine for an expansion with nl Legendre polynomials pl, evaluated using the recur-
rence relation as in §4.5.
{
    int j;
    float twox,f2,f1,d;

    pl[1]=1.0;
    pl[2]=x;
    if (nl > 2) {
        twox=2.0*x;
        f2=x;
        d=1.0;
        for (j=3;j<=nl;j++) {
            f1=d;
            d += 1.0;
            f2 += twox;
            pl[j]=(f2*pl[j-1]-f1*pl[j-2])/d;
        }
    }
}
```

REFERENCES AND FURTHER READING:

Bevington, Philip R. 1969, *Data Reduction and Error Analysis for the Physical Sciences* (New York: McGraw-Hill), Chapters 8,9.

Lawson, Charles L., and Hanson, Richard J. 1974, *Solving Least Squares Problems* (Englewood Cliffs, N.J.: Prentice-Hall).

Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, N.J.: Prentice-Hall), Chapter 9.

14.4 Nonlinear Models

We now consider fitting when the model depends *nonlinearly* on the set of M unknown parameters $a_k, k = 1, 2, \ldots, M$. We use the same approach as in previous sections, namely to define a χ^2 merit function and determine best-fit parameters by its minimization. With nonlinear dependences, however, the minimization must proceed iteratively. Given trial values for the parameters, we develop a procedure that improves the trial solution. The procedure is then repeated until χ^2 stops (or effectively stops) decreasing.

How is this problem different from the general nonlinear function minimization problem already dealt with in Chapter 10? Superficially, not at all: Sufficiently close to the minimum, we expect the χ^2 function to be well approximated by a quadratic form, which we can write as

$$\chi^2(\mathbf{a}) \approx \gamma - \mathbf{d} \cdot \mathbf{a} + \frac{1}{2}\mathbf{a} \cdot \mathbf{D} \cdot \mathbf{a} \tag{14.4.1}$$

where $\mathbf{d}$ is an M-vector and $\mathbf{D}$ is an $M \times M$ matrix. (Compare equation 10.6.1.) If the approximation is a good one, we know how to jump from the current trial parameters $\mathbf{a}_{cur}$ to the minimizing ones $\mathbf{a}_{min}$ in a single leap, namely

$$\mathbf{a}_{min} = \mathbf{a}_{cur} + \mathbf{D}^{-1} \cdot \left[-\nabla\chi^2(\mathbf{a}_{cur})\right] \tag{14.4.2}$$

(Compare equation 10.7.4, and reread the discussion leading up to it.)

On the other hand, (14.4.1) might be a poor local approximation to the shape of the function that we are trying to minimize at $\mathbf{a}_{cur}$. In that case, about all we can do is take a step down the gradient, as in the steepest descent method (§10.6). In other words,

$$\mathbf{a}_{next} = \mathbf{a}_{cur} - \text{constant} \times \nabla\chi^2(\mathbf{a}_{cur}) \tag{14.4.3}$$

where the constant is small enough not to exhaust the downhill direction.

To use (14.4.2) or (14.4.3), we must be able to compute the gradient of the χ^2 function at any set of parameters $\mathbf{a}$. To use (14.4.2) we also need the matrix $\mathbf{D}$, which is the second derivative matrix (Hessian matrix) of the χ^2 merit function, at any $\mathbf{a}$.

Now, this is the crucial difference from Chapter 10: There, we had no way of directly evaluating the Hessian matrix. We were only given the ability to evaluate the function to be minimized and (in some cases) its gradient. Therefore, we had to resort to iterative methods *not just* because our function was nonlinear, *but also* in order to build up information about the Hessian matrix. Sections 10.7 and 10.6 concerned themselves with two different techniques for building up this information.

Here, life is much simpler. We *know* exactly the form of χ^2, since it is based on a model function that we ourselves have specified. Therefore the Hessian matrix is known to us. Thus we are free to use (14.4.2) whenever we care to do so. The only reason to use (14.4.3) will be failure of (14.4.2) to improve the fit, signaling failure of (14.4.1) as a good local approximation.

Calculation of the Gradient and Hessian

The model to be fitted is

$$y = y(x; \mathbf{a}) \tag{14.4.4}$$

and the χ^2 merit function is

$$\chi^2(\mathbf{a}) = \sum_{i=1}^{N} \left[\frac{y_i - y(x_i; \mathbf{a})}{\sigma_i} \right]^2 \tag{14.4.5}$$

The gradient of χ^2 with respect to the parameters $\mathbf{a}$, which will be zero at the χ^2 minimum, has components

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_{i=1}^{N} \frac{[y_i - y(x_i; \mathbf{a})]}{\sigma_i^2} \frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \qquad k = 1, 2, \ldots, M \tag{14.4.6}$$

Taking an additional partial derivative gives

$$\frac{\partial^2 \chi^2}{\partial a_k \partial a_l} = 2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \frac{\partial y(x_i; \mathbf{a})}{\partial a_l} - [y_i - y(x_i; \mathbf{a})] \frac{\partial^2 y(x_i; \mathbf{a})}{\partial a_l \partial a_k} \right] \tag{14.4.7}$$

It is conventional to remove the factors of 2 by defining

$$\beta_k \equiv -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k} \qquad \alpha_{kl} \equiv \frac{1}{2} \frac{\partial^2 \chi^2}{\partial a_k \partial a_l} \tag{14.4.8}$$

making $[\alpha] = \frac{1}{2}\mathbf{D}$ in equation (14.4.2), in terms of which that equation can be rewritten as the set of linear equations

$$\sum_{l=1}^{M} \alpha_{kl} \, \delta a_l = \beta_k \tag{14.4.9}$$

This set is solved for the increments δa_l that, added to the current approximation, give the next approximation. In the context of least-squares, the matrix $[\alpha]$, equal to one-half times the Hessian matrix, is usually called the *curvature matrix*.

Equation (14.4.3), the steepest descent formula, translates to

$$\delta a_l = \text{constant} \times \beta_l \tag{14.4.10}$$

Note that the components α_{kl} of the Hessian matrix (14.4.7) depend both on the first derivatives and on the second derivatives of the basis functions with respect to their parameters. Some treatments proceed to ignore the second derivative without comment. We will ignore it also, but only *after* a few comments.

Second derivatives occur because the gradient (14.4.6) already has a dependence on $\partial y/\partial a_k$, so the next derivative simply must contain terms involving $\partial^2 y/\partial a_l \partial a_k$. The second derivative term can be dismissed when it is zero (as in the linear case of equation 14.3.8), or small enough to be negligible when compared to the term involving the first derivative. It also has an additional possibility of being ignorably small in practice: The term multiplying the second derivative in equation (14.4.7) is $[y_i - y(x_i; \mathbf{a})]$. For a successful model, this term should just be the random measurement error of each point. This error can have either sign, and should in general be uncorrelated with the model. Therefore, the second derivative terms tend to cancel out when summed over i.

Inclusion of the second-derivative term can in fact be destabilizing if the model fits badly or is contaminated by outlier points that are unlikely to be offset by compensating points of opposite sign. From this point on, we will always use as the definition of α_{kl} the formula

$$\alpha_{kl} = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \frac{\partial y(x_i; \mathbf{a})}{\partial a_l} \right] \tag{14.4.11}$$

This expression more closely resembles its linear cousin (14.3.8). You should understand that minor (or even major) fiddling with $[\alpha]$ has no effect at all on what final set of parameters $\mathbf{a}$ is reached, but only affects the iterative route that is taken in getting there. The condition at the χ^2 minimum, that $\beta_k = 0$ for all k, is independent of how $[\alpha]$ is defined.

Levenberg-Marquardt Method

Marquardt has put forth an elegant method, related to an earlier suggestion of Levenberg, for varying smoothly between the extremes of the inverse-Hessian method (14.4.9) and the steepest descent method (14.4.10). The latter method is used far from the minimum, switching continuously to the former as the minimum is approached. This *Levenberg-Marquardt method* (also called

Marquardt method) works very well in practice and has become the standard of nonlinear least-squares routines.

The method is based on two elementary, but important, insights. Consider the "constant" in equation (14.4.10). What should it be, even in order of magnitude? What sets its scale? There is no information about the answer in the gradient. That tells only the slope, not how far that slope extends. Marquardt's first insight is that the components of the Hessian matrix, even if they are not usable in any precise fashion, give *some* information about the order-of-magnitude scale of the problem.

The quantity χ^2 is nondimensional, i.e. is a pure number; this is evident from its definition (14.4.5). On the other hand, β_k has the dimensions of $1/a_k$, which may well be dimensional, i.e. have units like cm^{-1}, or kilowatt-hours, or whatever. (In fact, each component of β_k can have different dimensions!) The constant of proportionality between β_k and δa_k must therefore have the dimensions of a_k^2. Scan the components of $[\alpha]$ and you see that there is only one obvious quantity with these dimensions, and that is $1/\alpha_{kk}$, the reciprocal of the diagonal element. So that must set the scale of the constant. But that scale might itself be too big. So let's divide the constant by some (nondimensional) fudge factor λ, with the possibility of setting $\lambda \gg 1$ to cut down the step. In other words, replace equation (14.4.10) by

$$\delta a_l = \frac{1}{\lambda \alpha_{ll}} \beta_l \qquad \text{or} \qquad \lambda \alpha_{ll}\, \delta a_l = \beta_l \tag{14.4.12}$$

It is necessary that a_{ll} be positive, but this is guaranteed by definition (14.4.11) — another reason for adopting that equation.

Marquardt's second insight is that equations (14.4.12) and (14.4.9) can be combined if we define a new matrix α' by the following prescription

$$\begin{aligned} \alpha'_{jj} &\equiv \alpha_{jj}(1+\lambda) \\ \alpha'_{jk} &\equiv \alpha_{jk} \qquad (j \neq k) \end{aligned} \tag{14.4.13}$$

and then replace both (14.4.12) and (14.4.9) by

$$\sum_{l=1}^{M} \alpha'_{kl}\, \delta a_l = \beta_k \tag{14.4.14}$$

When λ is very large, the matrix α' is forced into being *diagonally dominant*, so equation (14.4.14) goes over to be identical to (14.4.12). On the other hand, as λ approaches zero, equation (14.4.14) goes over to (14.4.9).

Given an initial guess for the set of fitted parameters $\mathbf{a}$, the recommended Marquardt recipe is as follows:

- Compute $\chi^2(\mathbf{a})$.
- Pick a modest value for λ, say $\lambda = 0.001$.

- (†) Solve the linear equations (14.4.14) for $\delta\mathbf{a}$ and evaluate $\chi^2(\mathbf{a}+\delta\mathbf{a})$.
- If $\chi^2(\mathbf{a}+\delta\mathbf{a}) \geq \chi^2(\mathbf{a})$, *increase* λ by a factor of 10 (or any other substantial factor) and go back to (†).
- If $\chi^2(\mathbf{a}+\delta\mathbf{a}) < \chi^2(\mathbf{a})$, *decrease* λ by a factor of 10, update the trial solution $\mathbf{a} \leftarrow \mathbf{a}+\delta\mathbf{a}$, and go back to (†).

Also necessary is a condition for stopping. Iterating to convergence (to machine accuracy or to the roundoff limit) is generally wasteful and unnecessary since the minimum is at best only a statistical estimate of the parameters $\mathbf{a}$. As we will see in §14.5, a change in the parameters that changes χ^2 by an amount $\ll 1$ is *never* statistically meaningful.

Furthermore, it is not uncommon to find the parameters wandering around near the minimum in a flat valley of complicated topology. The reason is that Marquardt's method generalizes the method of normal equations (§14.3), hence has the same problem as that method with regard to near-degeneracy of the minimum. Outright failure by a zero pivot is possible, but unlikely. More often, a small pivot will generate a large correction which is then rejected, the value of λ being then increased. For sufficiently large λ the matrix $[\alpha']$ is positive definite and can have no small pivots. Thus the method does tend to stay away from zero pivots, but at the cost of a tendency to wander around doing steepest descent in very un-steep degenerate valleys.

These considerations suggest that, in practice, one might as well stop iterating on the first or second occasion that χ^2 decreases by a negligible amount, say either less than 0.1 absolutely or (in case roundoff prevents that being reached) some fractional amount like 10^{-3}. Don't stop after a step where χ^2 *increases*: that only shows that λ has not yet adjusted itself optimally.

Once the acceptable minimum has been found, one wants to set $\lambda = 0$ and compute the matrix

$$[C] \equiv [\alpha]^{-1} \tag{14.4.15}$$

which, as before, is the estimated covariance matrix of the standard errors in the fitted parameters $\mathbf{a}$ (see next section).

The following pair of functions encodes Marquardt's method for nonlinear parameter estimation. Much of the organization matches that used in `lfit` of §14.3. In particular the array `lista` should have as its first elements a list of the `mfit` parameters, out of `ma` total, that are desired to be fitted, the remaining parameters being held at their input values.

The routine `mrqmin` performs one iteration of Marquardt's method. It is first called (once) with `alamda` < 0, which signals the routine to initialize. `alamda` is set on the first and all subsequent calls to the suggested value of λ for the next iteration; `a` and `chisq` are always given back as the best parameters found so far and their χ^2. When convergence is deemed satisfactory, set `alamda` to zero before a final call. The matrices `alpha` and `covar` (which were used as workspace in all previous calls) will then be set to the curvature and covariance matrices for the converged parameter values. The arguments `alpha`, `a`, and `chisq` must not be modified between calls, nor should `alamda` be, except to set it to zero for the final call. When an uphill step is taken,

chisq and a are given back with their input (best) values, but alamda is set to an increased value.

The routine mrqmin calls the routine mrqcof for the computation of the matrix $[\alpha]$ (equation 14.4.11) and vector β (equations 14.4.6 and 14.4.8). In turn mrqcof calls the user-supplied routine funcs(x,a,y,dyda) which for input values $\mathbf{x} \equiv x_i$ and $\mathbf{a} \equiv \mathbf{a}$ calculates the model function $\mathbf{y} \equiv y(x_i; \mathbf{a})$ and the vector of derivatives dyda $\equiv \partial y / \partial a_k$.

```
void mrqmin(x,y,sig,ndata,a,ma,lista,mfit,covar,alpha,chisq,funcs,alamda)
float x[],y[],sig[],a[],**covar,**alpha,*chisq,*alamda;
int ndata,ma,lista[],mfit;
void (*funcs)();
```

Levenberg-Marquardt method, attempting to reduce the value χ^2 of a fit between a set of points x[1..ndata], y[1..ndata] with individual standard deviations sig[1..ndata], and a nonlinear function dependent on coefficients a[1..ma]. The array lista[1..ma] numbers the parameters a such that the first mfit elements correspond to values actually being adjusted; the remaining ma-mfit parameters are held fixed at their input value. The program returns current best-fit values for the ma fit parameters a, and $\chi^2 =$ chisq. The [1..mfit][1..mfit] elements of the arrays covar[1..ma][1..ma], alpha[1..ma][1..ma] are used as working space during most iterations. Supply a routine funcs(x,a,yfit,dyda,ma) that evaluates the fitting function yfit, and its derivatives dyda[1..ma] with respect to the fitting parameters a at x. On the first call provide an initial guess for the parameters a, and set alamda<0 for initialization (which then sets alamda=.001). If a step succeeds chisq becomes smaller and alamda decreases by a factor of 10. If a step fails alamda grows by a factor of 10. You must call this routine repeatedly until convergence is achieved. Then, make one final call with alamda=0, so that covar returns the covariance matrix, and alpha the curvature matrix.

```
{
    int k,kk,j,ihit;
    static float *da,*atry,**oneda,*beta,ochisq;
    float *vector(),**matrix();
    void mrqcof(),gaussj(),covsrt(),nrerror(),free_matrix(),free_vector();

    if (*alamda < 0.0) {                    Initialization.
        oneda=matrix(1,mfit,1,1);           These variables are not freed until the last call, when
        atry=vector(1,ma);                      alamda = 0.
        da=vector(1,ma);
        beta=vector(1,ma);
        kk=mfit+1;
        for (j=1;j<=ma;j++) {               Does lista contain a proper permutation of the coef-
            ihit=0;                             ficients?
            for (k=1;k<=mfit;k++)
                if (lista[k] == j) ihit++;
            if (ihit == 0)
                lista[kk++]=j;
            else if (ihit > 1) nrerror("Bad LISTA permutation in MRQMIN-1");
        }
        if (kk != ma+1) nrerror("Bad LISTA permutation in MRQMIN-2");
        *alamda=0.001;
        mrqcof(x,y,sig,ndata,a,ma,lista,mfit,alpha,beta,chisq,funcs);
        ochisq=(*chisq);
    }
    for (j=1;j<=mfit;j++) {                 Alter linearized fitting matrix, by augmenting diagonal
        for (k=1;k<=mfit;k++) covar[j][k]=alpha[j][k];      elements.
        covar[j][j]=alpha[j][j]*(1.0+(*alamda));
        oneda[j][1]=beta[j];
    }
    gaussj(covar,mfit,oneda,1);             Matrix solution.
    for (j=1;j<=mfit;j++)
        da[j]=oneda[j][1];
    if (*alamda == 0.0) {                   Once converged evaluate covariance matrix with alamda=0.
        covsrt(covar,ma,lista,mfit);
```

```
        free_vector(beta,1,ma);
        free_vector(da,1,ma);
        free_vector(atry,1,ma);
        free_matrix(oneda,1,mfit,1,1);
        return;
    }
    for (j=1;j<=ma;j++) atry[j]=a[j];
    for (j=1;j<=mfit;j++)                    Did the trial succeed?
        atry[lista[j]] = a[lista[j]]+da[j];
    mrqcof(x,y,sig,ndata,atry,ma,lista,mfit,covar,da,chisq,funcs);
    if (*chisq < ochisq) {                   Success, accept the new solution.
        *alamda *= 0.1;
        ochisq=(*chisq);
        for (j=1;j<=mfit;j++) {
            for (k=1;k<=mfit;k++) alpha[j][k]=covar[j][k];
            beta[j]=da[j];
            a[lista[j]]=atry[lista[j]];
        }
    } else {                                 Failure, increase alamda and return.
        *alamda *= 10.0;
        *chisq=ochisq;
    }
    return;
}
```

Notice the use of the routine `covsrt` from §14.3. This is only for rearranging the covariance matrix `covar` into the order of all `ma` parameters. If you are willing to look up nonzero components corresponding to the `mfit` fitted variables through the index `lista`, then you can omit all reference to `covsrt`. The above routine also makes use of

```
void mrqcof(x,y,sig,ndata,a,ma,lista,mfit,alpha,beta,chisq,funcs)
float x[],y[],sig[],a[],**alpha,beta[],*chisq;
int ndata,ma,lista[],mfit;
void (*funcs)();    /* ANSI: void (*funcs)(float,float *,float *,float *,int); */
Used by mrqmin to evaluate the linearized fitting matrix alpha[1..mfit][1..mfit], and vector
beta[1..mfit] as in (14.4.8).
{
    int k,j,i;
    float ymod,wt,sig2i,dy,*dyda,*vector();
    void free_vector();

    dyda=vector(1,ma);
    for (j=1;j<=mfit;j++) {                  Initialize (symmetric) alpha, beta.
        for (k=1;k<=j;k++) alpha[j][k]=0.0;
        beta[j]=0.0;
    }
    *chisq=0.0;
    for (i=1;i<=ndata;i++) {                 Summation loop over all data.
        (*funcs)(x[i],a,&ymod,dyda,ma);
        sig2i=1.0/(sig[i]*sig[i]);
        dy=y[i]-ymod;
        for (j=1;j<=mfit;j++) {
            wt=dyda[lista[j]]*sig2i;
            for (k=1;k<=j;k++)
                alpha[j][k] += wt*dyda[lista[k]];
            beta[j] += dy*wt;
        }
```

```
            (*chisq) += dy*dy*sig2i;          And find χ².
        }
        for (j=2;j<=mfit;j++)                 Fill in the symmetric side.
            for (k=1;k<=j-1;k++) alpha[k][j]=alpha[j][k];
        free_vector(dyda,1,ma);
}
```

Example

The following function `fgauss` is an example of a user-supplied function `funcs`. Used with the above routine `mrqmin` (in turn using `mrqcof`, `covsrt`, and `gaussj`) it fits for the model

$$y(x) = \sum_{k=1}^{K} B_k \exp\left[-\left(\frac{x - E_k}{G_k}\right)^2\right] \tag{14.4.16}$$

which is a sum of K Gaussians, each having a variable position, amplitude, and width. We store the parameters in the order $B_1, E_1, G_1, B_2, E_2, G_2, \ldots, B_K, E_K, G_K$.

```
#include <math.h>

void fgauss(x,a,y,dyda,na)
float x,a[],*y,dyda[];
int na;
y(x;a) is the sum of na/3 Gaussians (14.4.16). The amplitude, center, and width of the
Gaussians are stored in consecutive locations of a: a[i]= B_k, a[i+1]= E_k, a[i+2]= G_k,
k = 1,...,na/3. The dimensions of the arrays are a[1..na], dyda[1..na]
{
    int i;
    float fac,ex,arg;

    *y=0.0;
    for (i=1;i<=na-1;i+=3)
    {
        arg=(x-a[i+1])/a[i+2];
        ex=exp(-arg*arg);
        fac=a[i]*ex*2.0*arg;
        *y += a[i]*ex;
        dyda[i]=ex;
        dyda[i+1]=fac/a[i+2];
        dyda[i+2]=fac*arg/a[i+2];
    }
}
```

REFERENCES AND FURTHER READING:

Bevington, Philip R. 1969, *Data Reduction and Error Analysis for the Physical Sciences* (New York: McGraw-Hill), Chapter 11.

Marquardt, D. W. 1963, *J. Soc. Ind. Appl. Math.*, vol. 11, pp. 431-441.

Jacobs, David A.H., ed. 1977, *The State of the Art in Numerical Analysis* (London: Academic Press), chapter III.2 (by J.E. Dennis).

14.5 Confidence Limits on Estimated Model Parameters

Several times already in this chapter we have made statements about the standard errors, or uncertainties, in a set of M estimated parameters $\mathbf{a}$. We have given some formulas for computing standard deviations or variances of individual parameters (equations 14.2.9, 14.3.15, 14.3.19), as well as some formulas for covariances between pairs of parameters (equation 14.2.10; remark following equation 14.3.15; equation 14.4.15).

In this section, we want to be more explicit regarding the precise meaning of these quantitative uncertainties, and to give further information about how quantitative confidence limits on fitted parameters can be estimated. The subject can get somewhat technical, and even somewhat confusing, so we will try to make precise statements, even when they must be offered without proof.

Figure 14.5.1 shows the conceptual scheme of an experiment which "measures" a set of parameters. There is some underlying true set of parameters $\mathbf{a}_{true}$ which are known to Mother Nature but hidden from the experimenter. These true parameters are statistically realized, along with random measurement errors, as a measured data set, which we will symbolize as $\mathcal{D}_{(0)}$. The data set $\mathcal{D}_{(0)}$ *is* known to the experimenter. He or she fits the data to a model by χ^2 minimization or some other technique, and obtains measured, i.e. fitted, values for the parameters, which we here denote $\mathbf{a}_{(0)}$.

Because measurement errors have a random component, $\mathcal{D}_{(0)}$ is not a unique realization of the true parameters $\mathbf{a}_{true}$. Rather, there are infinitely many other realizations of the true parameters as "hypothetical data sets" each of which *could* have been the one measured, but happened not to be. Let us symbolize these by $\mathcal{D}_{(1)}, \mathcal{D}_{(2)}, \ldots$. Each one, had it been realized, would have given a slightly different set of fitted parameters, $\mathbf{a}_{(1)}, \mathbf{a}_{(2)}, \ldots$, respectively. These parameter sets $\mathbf{a}_{(i)}$ therefore occur with some probability distribution in the M-dimensional space of all possible parameter sets $\mathbf{a}$. The actual measured set $\mathbf{a}_{(0)}$ is one member drawn from this distribution.

Even more interesting than the probability distribution of $\mathbf{a}_{(i)}$ would be the distribution of the difference $\mathbf{a}_{(i)} - \mathbf{a}_{true}$. This distribution differs from the former one by a translation that puts Mother Nature's true value at the origin. If we knew *this* distribution, we would know everything that there is to know about the quantitative uncertainties in our experimental measurement $\mathbf{a}_{(0)}$.

So the name of the game is to find some way of estimating or approximating the probability distribution of $\mathbf{a}_{(i)} - \mathbf{a}_{true}$ without knowing $\mathbf{a}_{true}$ and without having available to us an infinite universe of hypothetical data sets.

General Case: Confidence Limits by Monte Carlo Simulation

There is really only one way of making the desired estimation. That one way sometimes comes dressed up with fancy analytical formulas, or sometimes naked as a purely numerical procedure; but conceptually it is the same in both cases. When various extra mathematical assumptions are known to hold, the

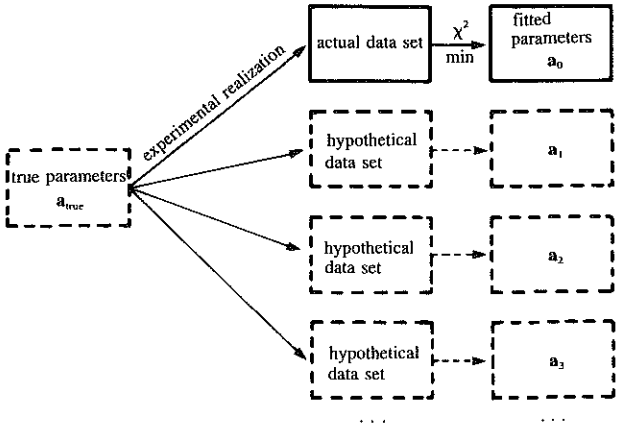


Figure 14.5.1. A statistical universe of data sets from an underlying model. True parameters $\mathbf{a}_{true}$ are realized in a data set, from which fitted (observed) parameters $\mathbf{a}_0$ are obtained. If the experiment were repeated many times, new data sets and new values of the fitted parameters would be obtained.

one way can be proved to give an "accurate" estimate; when they fail, its estimate may be crude. But in either case it is just about the only game in town. Here it is:

Although the measured parameter set $\mathbf{a}_{(0)}$ is not the true one, let us consider a fictitious world in which it *was* the true one. Since we hope that our measured parameters are not *too* wrong, we hope that that fictitious world is not too different from the actual world with parameters $\mathbf{a}_{true}$. In particular, let us hope — no, let us *assume* — that the shape of the probability distribution $\mathbf{a}_{(i)} - \mathbf{a}_{(0)}$ in the fictitious world is the same, or very nearly the same, as the shape of the probability distribution $\mathbf{a}_{(i)} - \mathbf{a}_{true}$ in the real world. Notice that we are not assuming that $\mathbf{a}_{(0)}$ and $\mathbf{a}_{true}$ are equal; they are certainly not. We are only assuming that the way in which random errors enter the experiment and data analysis does not vary rapidly as a function of $\mathbf{a}_{true}$, so that $\mathbf{a}_{(0)}$ can serve as a reasonable surrogate.

Now the distribution of $\mathbf{a}_{(i)} - \mathbf{a}_{(0)}$ in the fictitious world *is* within our power to calculate (see Figure 14.5.2). Starting with our parameters $\mathbf{a}_{(0)}$, we can *simulate* our own sets of "synthetic" realizations of these parameters as "synthetic data sets." The procedure is to draw random numbers from appropriate distributions (cf. §7.2–§7.3) so as to mimic our best understanding of the measurement errors in our apparatus. With such random draws, we construct data sets with exactly the same numbers of measured points, and precisely the same values of all control (independent) variables, as our actual data set $\mathcal{D}_{(0)}$. Let us call these simulated data sets $\mathcal{D}^S_{(1)}, \mathcal{D}^S_{(2)}, \ldots$. By construction these are supposed to have exactly the same statistical relationship to $\mathbf{a}_{(0)}$ as the $\mathcal{D}_{(i)}$'s have to $\mathbf{a}_{true}$.

Next, for each $\mathcal{D}^S_{(j)}$, perform exactly the same procedure for estimation of parameters, e.g. χ^2 minimization, as was performed on the actual data to get the parameters $\mathbf{a}_{(0)}$, giving simulated measured parameters $\mathbf{a}^S_{(1)}, \mathbf{a}^S_{(2)}, \ldots$.

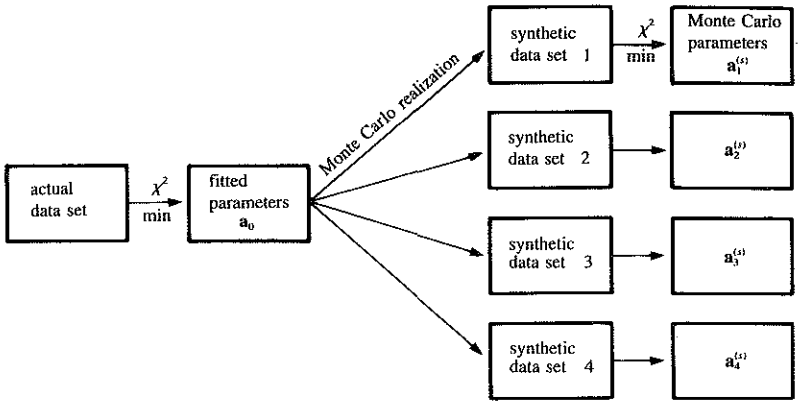


Figure 14.5.2. Monte Carlo simulation of an experiment. The fitted parameters from an actual experiment are used as surrogates for the true parameters. Computer-generated random numbers are used to simulate many synthetic data sets. Each of these is analyzed to obtain its fitted parameters. The distribution of these fitted parameters around the (known) surrogate true parameters is thus studied.

Each simulated measured parameter set yields a point $\mathbf{a}^S_{(i)} - \mathbf{a}_{(0)}$. Simulate enough data sets and enough derived simulated measured parameters, and you map out the desired probability distribution in M dimensions.

In fact, the ability to do *Monte Carlo simulations* in this fashion has revolutionized many fields of modern experimental science. Not only is one able to characterize the errors of parameter estimation in a very precise way. One can also try out on the computer different methods of parameter estimation, or different data reduction techniques, and seek to minimize the uncertainty of the result according to any desired criteria. Offered the choice between mastery of a five-foot shelf of analytical statistics books and middling ability at performing statistical Monte Carlo simulations, we would surely choose to have the latter skill.

Nevertheless, there are a few important analytic results which we will mention just below.

Rather than present all details of the probability distribution of errors in parameter estimation, it is common practice to summarize the distribution in the form of *confidence limits*. The full probability distribution is a function defined on the M-dimensional space of parameters $\mathbf{a}$. A *confidence region* (or *confidence interval*) is just a region of that M-dimensional space (hopefully a small region) that contains a certain (hopefully large) percentage of the total probability distribution. You point to a confidence region and say, e.g., "there is a 99 percent chance that the true parameter values fall within this region around the measured value."

It is worth emphasizing that you, the experimenter, get to pick both the *confidence level* (99 percent in the above example), and the shape of the confidence region. The only requirement is that your region does include the stated percentage of probability. Certain percentages are, however, customary in scientific usage: 68.3 percent (the lowest confidence worthy of quoting), 90 percent, 95.4 percent, 99 percent, and 99.73 percent. Higher confidence

levels are conventionally "ninety-nine point nine ... nine." As for shape, obviously you want a region that is compact and reasonably centered on your measurement $\mathbf{a}_{(0)}$, since the whole purpose of a confidence limit is to inspire confidence in that measured value. In one dimension, the convention is to use a line segment centered on the measured value; in higher dimensions, ellipses or ellipsoids are most frequently used.

You might suspect, correctly, that the numbers 68.3 percent, 95.4 percent, and 99.73 percent, and the use of ellipsoids, have some connection with a normal distribution. That is true historically, but not always relevant nowadays. In general, the probability distribution of the parameters will not be normal, and the above numbers, used as levels of confidence, are purely matters of convention.

Figure 14.5.3 sketches a possible probability distribution for the case $M = 2$. Shown are three different confidence regions which might usefully be given, all at the same confidence level. The two vertical lines enclose a band (horizontal inverval) which represents the 68 percent confidence interval for the variable a_1 without regard to the value of a_2. Similarly the horizontal lines enclose a 68 percent confidence interval for a_2. The ellipse shows a 68 percent confidence interval for a_1 and a_2 jointly. Notice that to enclose the same probability as the two bands, the ellipse must necessarily extend outside of both of them (a point we will return to below).

Use of Constant Chi-Square Boundaries as Confidence Limits

When the method used to estimate the parameters $\mathbf{a}_{(0)}$ is chi-square minimization, as in the previous sections of this chapter, then there is a natural choice for the shape of confidence intervals, whose use is almost universal. For the observed data set $\mathcal{D}_{(0)}$, the value of χ^2 is a minimum at $\mathbf{a}_{(0)}$. Call this minimum value χ^2_{min}. If the vector $\mathbf{a}$ of parameter values is perturbed away from $\mathbf{a}_{(0)}$, then χ^2 increases. The region within which χ^2 increases by no more than a set amount $\Delta\chi^2$ defines some M dimensional confidence region around $\mathbf{a}_{(0)}$. If $\Delta\chi^2$ is set to be a large number, this will be a big region; if it is small, it will be small. Somewhere in between there will be choices of $\Delta\chi^2$ which cause the region to contain, variously, 68 percent, 90 percent, etc. of probability distribution for $\mathbf{a}$'s, as defined above. These regions are taken as the confidence regions for the parameters $\mathbf{a}_{(0)}$.

Very frequently one is interested not in the full M-dimensional confidence region, but in individual confidence regions for some smaller number ν of parameters. For example, one might be interested in the confidence interval of each parameter taken separately (the bands in Figure 14.5.3), in which case $\nu = 1$. In that case, the natural confidence regions in the ν-dimensional subspace of the M-dimensional parameter space are the *projections* of the M-dimensional regions defined by fixed $\Delta\chi^2$ into the ν-dimensional spaces of interest. In Figure 14.5.4, for the case $M = 2$, we show regions corresponding to several values of $\Delta\chi^2$. The one-dimensional confidence interval in a_2 corresponding to the region bounded by $\Delta\chi^2 = 1$ lies between the lines A and A'.

Notice that the projection of the higher-dimensional region on the lower-dimension space is used, not the intersection. The intersection would be the

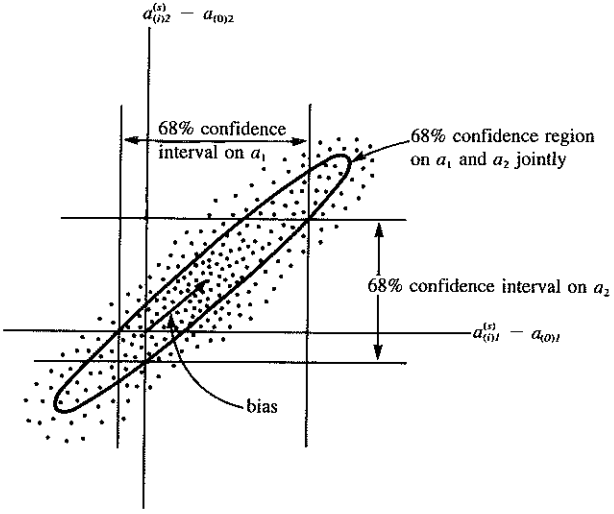


Figure 14.5.3. Confidence intervals in 1 and 2 dimensions. The same fraction of measured points (here 68%) lies (i) between the two vertical lines, (ii) between the two horizontal lines, (iii) within the ellipse.

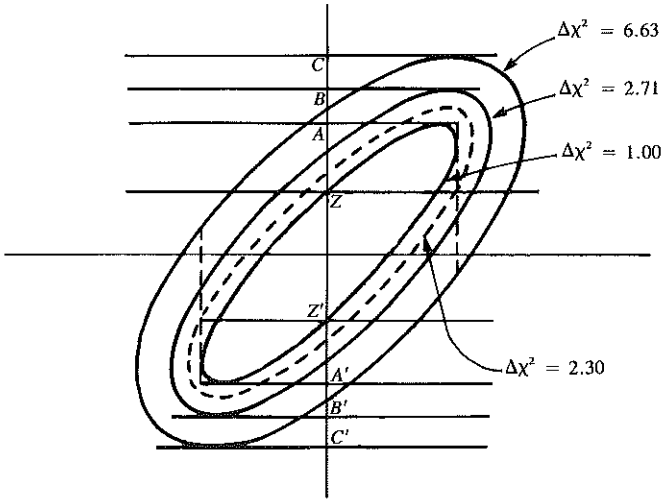


Figure 14.5.4. Confidence region ellipses corresponding to values of chi-square larger than the fitted minimum. The solid curves, with $\Delta\chi^2 = 1.00, 2.71, 6.63$ project onto one-dimensional intervals AA', BB', CC'. These intervals – not the ellipses themselves – contain 68.3%, 90%, and 99% of normally distributed data. The ellipse that contains 68.3% of normally distributed data is shown dashed, and has $\Delta\chi^2 = 2.30$. For additional numerical values, see accompanying table.

band between Z and Z'. It is *never* used. It is shown in the figure only for the purpose of making this cautionary point, that it should not be confused with the projection.

Probability Distribution of Parameters in the Normal Case

You may be wondering why we have, in this section up to now, made no connection at all with the error estimates that come out of the χ^2 fitting procedure, most notably the covariance matrix C_{ij}. The reason is this: χ^2 minimization is a useful means for estimating parameters even if the measurement errors are not normally distributed. While normally distributed errors are required if the χ^2 parameter estimate is to be a maximum likelihood estimator (§14.1), one is often willing to give up that property in return for the relative convenience of the χ^2 procedure. Only in extreme cases, measurement error distributions with very large "tails," is χ^2 minimization abandoned in favor of more robust techniques, as will be discussed in §14.6.

However, the formal covariance matrix that comes out of a χ^2 minimization has meaning *only* if (or to the extent that) the measurement errors actually are normally distributed. In the case of *non*normal errors, you are "allowed"

- to fit for parameters by minimizing χ^2
- to use a contour of constant $\Delta\chi^2$ as the boundary of your confidence region
- to use Monte Carlo simulation or detailed analytic calculation in determining *which* contour $\Delta\chi^2$ is the correct one for your desired confidence level
- to give the covariance matrix C_{ij} as the "formal covariance matrix of the fit on the assumption of normally distributed errors."

You are *not* allowed

- to interpret C_{ij} as the actual squared standard errors of the parameter estimation
- to use formulas that we now give for the case of normal errors, which establish quantitative relationships among $\Delta\chi^2$, C_{ij}, and the confidence level.

Here are the key theorems that hold when (i) the measurement errors are normally distributed, and either (ii) the model is linear in its parameters or (iii) the sample size is large enough that the uncertainties in the fitted parameters **a** do not extend outside a region in which the model could be replaced by a suitable linearized model. [Note that condition (iii) does not preclude your use of a nonlinear routine like `mqrfit` to *find* the fitted parameters.]

Theorem A. χ^2_{min} is distributed as a chi-square distribution with $N-M$ degrees of freedom, where N is the number of data points and M is the number of fitted parameters. This is the basic theorem which lets you evaluate the goodness-of-fit of the model, as discussed above in §14.1. We list it first to remind you that unless the goodness-of-fit is credible, the whole estimation of parameters is suspect.

Theorem B. If $\mathbf{a}^S_{(j)}$ is drawn from the universe of simulated data sets with actual parameters $\mathbf{a}_{(0)}$, then the probability distribution of $\delta\mathbf{a} \equiv \mathbf{a}^S_{(j)} - \mathbf{a}_{(0)}$ is the multivariate normal distribution

$$P(\delta\mathbf{a})\, da_1 \dots da_M = \text{const.} \times \exp\left(-\frac{1}{2}\delta\mathbf{a} \cdot [\alpha] \cdot \delta\mathbf{a}\right)\, da_1 \dots da_M$$

where $[\alpha]$ is the curvature matrix defined in equation (14.4.8).

Theorem C. If $\mathbf{a}^S_{(j)}$ is drawn from the universe of simulated data sets with actual parameters $\mathbf{a}_{(0)}$, then the quantity $\Delta\chi^2 \equiv \chi^2(\mathbf{a}_{(j)}) - \chi^2(\mathbf{a}_{(0)})$ is distributed as a chi-square distribution with M degrees of freedom. Here the χ^2's are all evaluated using the fixed (actual) data set $\mathcal{D}_{(0)}$. This theorem makes the connection between particular values of $\Delta\chi^2$ and the fraction of the probability distribution that they enclose as an M-dimensional region, i.e., the confidence level of the M-dimensional confidence region.

Theorem D. Suppose that $\mathbf{a}^S_{(j)}$ is drawn from the universe of simulated data sets (as above), that its first ν components $a_1, \dots, a_\nu$ are held fixed, and that its remaining $M - \nu$ components are varied so as to minimize χ^2. Call this minimum value χ^2_ν. Then $\Delta\chi^2_\nu \equiv \chi^2_\nu - \chi^2_{min}$ is distributed as a chi-square distribution with ν degrees of freedom. If you consult Figure 14.5.4, you will see that this theorem connects the *projected* $\Delta\chi^2$ region with a confidence level. In the figure, a point that is held fixed in a_2 and allowed to vary in a_1 minimizing χ^2 will seek out the ellipse whose top or bottom edge is tangent to the line of constant a_2, and is therefore the line that projects it onto the smaller dimensional space.

As a first example, let us consider the case $\nu = 1$, where we want to find the confidence interval of a single parameter, say a_1. Notice that the chi-square distribution with $\nu = 1$ degree of freedom is the same distribution as that of the square of a single normally distributed quantity. Thus $\Delta\chi^2_\nu < 1$ occurs 68.3 percent of the time (1-σ for the normal distribution), $\Delta\chi^2_\nu < 4$ occurs 95.4 percent of the time (2-σ for the normal distribution), $\Delta\chi^2_\nu < 9$ occurs 99.73 percent of the time (3-σ for the normal distribution), etc. In this manner you find the $\Delta\chi^2_\nu$ which corresponds to your desired confidence level. (Additional values are given in the accompanying table.)

Let $\delta\mathbf{a}$ be a change in the parameters whose first component is arbitrary, δa_1, but the rest of whose components are chosen to minimize the $\Delta\chi^2$. Then Theorem D applies. The value of $\Delta\chi^2$ is given in general by

$$\Delta\chi^2 = \delta\mathbf{a} \cdot [\alpha] \cdot \delta\mathbf{a} \tag{14.5.1}$$

which follows from equation (14.4.8) applied at χ^2_{min} where $\beta_k = 0$. Since $\delta\mathbf{a}$ by hypothesis minimizes χ^2 in all but its first component, the second through M^{th} components of the normal equations (14.4.9) continue to hold,

$\Delta\chi^2$ *as a Function of Confidence Level and Degrees of Freedom*

	ν					
p	1	2	3	4	5	6
68.3%	1.00	2.30	3.53	4.72	5.89	7.04
90%	2.71	4.61	6.25	7.78	9.24	10.6
95.4%	4.00	6.17	8.02	9.70	11.3	12.8
99%	6.63	9.21	11.3	13.3	15.1	16.8
99.73%	9.00	11.8	14.2	16.3	18.2	20.1
99.99%	15.1	18.4	21.1	23.5	25.7	27.8

Therefore, the solution of (14.4.9) is

$$\delta\mathbf{a} = [\alpha]^{-1} \cdot \begin{pmatrix} c \\ 0 \\ \vdots \\ 0 \end{pmatrix} = [C] \cdot \begin{pmatrix} c \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{14.5.2}$$

where c is one arbitrary constant that we get to adjust to make (14.5.1) give the desired left-hand value. Plugging (14.5.2) into (14.5.1) and using the fact that $[C]$ and $[\alpha]$ are inverse matrices of one another, we get

$$c = \delta a_1 / C_{11} \qquad \text{and} \qquad \Delta\chi^2_\nu = (\delta a_1)^2 / C_{11} \tag{14.5.3}$$

or

$$\delta a_1 = \pm\sqrt{\Delta\chi^2_\nu}\,\sqrt{C_{11}} \tag{14.5.4}$$

At last! A relation between the confidence interval $\pm\delta a_1$ and the formal standard error $\sigma_1 \equiv \sqrt{C_{11}}$. Not unreasonably, we find that the 68 percent confidence interval is $\pm\sigma_1$, the 95 percent confidence interval is $\pm 2\sigma_1$, etc.

These considerations hold not just for the individual parameters a_i, but also for any linear combination of them: If

$$b \equiv \sum_{k=1}^{M} c_i a_i = \mathbf{c} \cdot \mathbf{a} \tag{14.5.5}$$

then the 68 percent confidence interval on b is

$$\delta b = \pm\sqrt{\mathbf{c} \cdot [C] \cdot \mathbf{c}} \tag{14.5.6}$$

However, these simple, normal-sounding numerical relationships do *not* hold in the case $\nu > 1$. In particular, $\Delta\chi^2 = 1$ is not the boundary, nor does it project onto the boundary, of a 68.3 percent confidence region when $\nu > 1$. If you want to calculate not confidence intervals in one parameter, but

confidence ellipses in two parameters jointly, or ellipsoids in three, or higher, then you must follow the following prescription for implementing Theorems C and D above:

- Let ν be the number of fitted parameters whose joint confidence region you wish to display, $\nu \leq M$. Call these parameters the "parameters of interest."
- Let p be the confidence limit desired, e.g. $p = 0.68$ or $p = 0.95$.
- Find Δ (i.e. $\Delta\chi^2$) such that the probability of a chi-square variable with ν degrees of freedom being less than Δ is p. For some useful values of p and ν, Δ is given in the table. For other values, you can use the routine **gammq** and a simple root-finding routine (e.g. bisection) to find Δ such that $\texttt{gammq}(\nu/2, \Delta/2) = 1 - p$.
- Take the $M \times M$ covariance matrix $[C] = [\alpha]^{-1}$ of the chi-square fit. Copy the intersection of the ν rows and columns corresponding to the parameters of interest into a $\nu \times \nu$ matrix denoted $[C_{proj}]$.
- Invert the matrix $[C_{proj}]$. (In the one-dimensional case this was just taking the reciprocal of the element C_{11}.)
- The equation for the elliptical boundary of your desired confidence region in the ν-dimensional subspace of interest is

$$\Delta = \delta\mathbf{a}' \cdot [C_{proj}]^{-1} \cdot \delta\mathbf{a}' \tag{14.5.7}$$

where $\delta\mathbf{a}'$ is the ν-dimensional vector of parameters of interest.

If you are confused at this point, you may find it helpful to compare Figure 14.5.4 and the accompanying table, considering the case $M = 2$ with $\nu = 1$ and $\nu = 2$. You should be able to verify the following statements: (i) The horizontal band between C and C' contains 99 percent of the probability distribution, so is a confidence limit on a_2 alone at this level of confidence. (ii) Ditto the band between B and B' at the 90 percent confidence level. (iii) The dashed ellipse, labeled by $\Delta\chi^2 = 2.30$, contains 68.3 percent of the probability distribution, so is a confidence region for a_1 and a_2 jointly, at this level of confidence.

Confidence Limits from Singular Value Decomposition

When you have obtained your χ^2 fit by singular value decomposition (§14.3), the information about the fit's formal errors comes packaged in a somewhat different, but generally more convenient, form. The columns of the matrix $\mathbf{V}$ are an orthonormal set of M vectors which are the principal axes of the $\Delta\chi^2 =$ constant ellipsoids. We denote the columns as $\mathbf{V}_{(1)} \ldots \mathbf{V}_{(M)}$. The lengths of those axes are inversely proportional to the corresponding singular values $w_1 \ldots w_M$; see Figure 14.5.5. The boundaries of the ellipsoids are thus given by

$$\Delta\chi^2 = w_1^2(\mathbf{V}_{(1)} \cdot \delta\mathbf{a})^2 + \cdots + w_M^2(\mathbf{V}_{(M)} \cdot \delta\mathbf{a})^2 \tag{14.5.8}$$

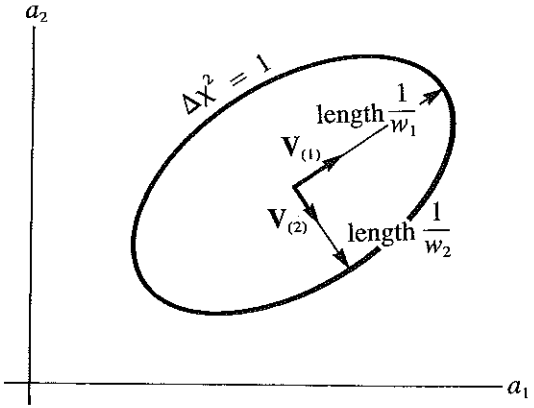


Figure 14.5.5. Relation of the confidence region ellipse $\Delta\chi^2 = 1$ to quantities computed by singular value decomposition. The vectors $\mathbf{V}_{(i)}$ are unit vectors along the principal axes of the confidence region. The semi-axes have lengths equal to the reciprocal of the singular values w_i. If the axes are all scaled by some constant factor α, $\Delta\chi^2$ is scaled by the factor α^2.

which is the justification for writing equation (14.3.18) above. Keep in mind that it is *much* easier to plot an ellipsoid given a list of its vector principal axes, than given its matrix quadratic form!

The formula for the covariance matrix $[C]$ in terms of the columns $\mathbf{V}_{(i)}$ is

$$[C] = \sum_{i=1}^{M} \frac{1}{w_i^2} \mathbf{V}_{(i)} \otimes \mathbf{V}_{(i)} \tag{14.5.9}$$

or, in components,

$$C_{jk} = \sum_{i=1}^{M} \frac{1}{w_i^2} V_{ji} V_{ki} \tag{14.5.10}$$

REFERENCES AND FURTHER READING:

Avni, Y. 1976, *Astrophysical Journal*, vol. 210, pp. 642–646.

Lampton, M., Margon, M., and Bowyer, S. 1976, *Astrophysical Journal*, vol. 208, pp. 177–190.

Brownlee, K.A. 1965, *Statistical Theory and Methodology*, 2nd ed. (New York: Wiley).

Martin, B.R. 1971, *Statistics for Physicists* (New York: Academic Press).

14.6 Robust Estimation

The concept of *robustness* has been mentioned in passing several times already. In §13.2 we noted that the median was a more robust estimator of central value than the mean; in §13.8 it was mentioned that rank correlation is more robust than linear correlation. The concept of outlier points as exceptions to a Gaussian model for experimental error was discussed in §14.1.

The term "robust" was coined in statistics by G.E.P. Box in 1953. Various definitions of greater or lesser mathematical rigor are possible for the term, but in general, referring to a statistical estimator, it means "insensitive to small departures from the idealized assumptions for which the estimator is optimized." The word "small" can have two different interpretations, both important: either fractionally small departures for all data points, or else fractionally large departures for a small number of data points. It is the latter interpretation, leading to the notion of outlier points, that is generally the most stressful for statistical procedures.

Statisticians have developed various sorts of robust statistical estimators. Many, if not most, can be grouped in one of three categories.

M-estimates follow from maximum-likelihood arguments very much as equations (14.1.5) and (14.1.7) followed from equation (14.1.3). M-estimates are usually the most relevant class for model-fitting, that is, estimation of parameters. We therefore consider these estimates in some detail below.

L-estimates are "linear combinations of order statistics." These are most applicable to estimations of central value and central tendency, though they can occasionally be applied to some problems in estimation of parameters. Two "typical" L-estimates will give you the general idea. They are (i) the median, and (ii) *Tukey's trimean*, defined as the weighted average of the first, second, and third quartile points in a distribution, with weights 1/4, 1/2, and 1/4 respectively.

R-estimates are estimates based on rank tests. For example, the equality or inequality of two distributions can be estimated by the *Wilcoxon test* of computing the mean rank of one distribution in a combined sample of both distributions. The Kolmogorov-Smirnov statistic (equation 13.5.4) and the Spearman rank-order correlation coefficient (13.8.1) are R-estimates in essence, if not always by formal definition.

Some other kinds of robust techniques, coming from the fields of optimal control and filtering rather than from the field of mathematical statistics, are mentioned at the end of this section. Some examples where robust statistical methods are desirable are shown in Figure 14.6.1.

Estimation of Parameters by Local M-estimates

Suppose we know that our measurement errors are not normally distributed. Then, in deriving a maximum-likelihood formula for the estimated

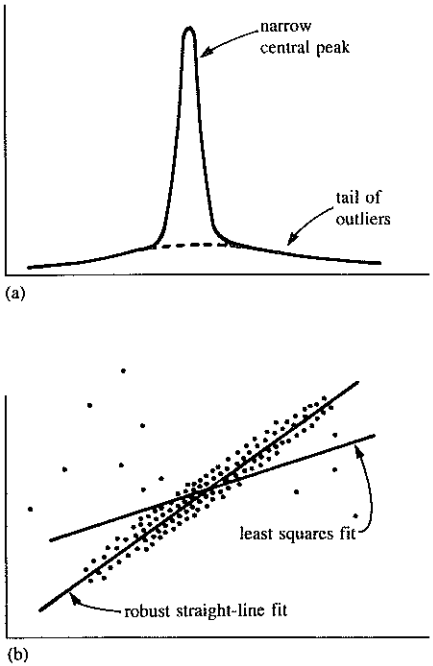


Figure 14.6.1. Examples where robust statistical methods are desirable: (a) A one-dimensional distribution with a tail of outliers; statistical fluctuations in these outliers can prevent accurate determination of the position of the central peak. (b) A distribution in two dimensions fitted to a straight line; non-robust techniques such as least-squares fitting can have undesired sensitivity to outlying points.

parameters $\mathbf{a}$ in a model $y(x;\mathbf{a})$, we would write instead of equation (14.1.3)

$$P = \prod_{i=1}^{N} \{\exp[-\rho(y_i, y\{x_i;\mathbf{a}\})]\,\Delta y\} \tag{14.6.1}$$

where the function ρ is the negative logarithm of the probability density. Taking the logarithm of (14.6.1) analogously with (14.1.4), we find that we want to minimize the expression

$$\sum_{i=1}^{N} \rho(y_i, y\{x_i;\mathbf{a}\}) \tag{14.6.2}$$

Very often, it is the case that the function ρ depends not independently on its two arguments, measured y_i and predicted $y(x_i)$, but only on their difference, at least if scaled by some weight factors σ_i which we are able to

assign to each point. In this case the M-estimate is said to be *local*, and we can replace (14.6.2) by the prescription

$$\text{minimize over } \mathbf{a} \qquad \sum_{i=1}^{N} \rho\left(\frac{y_i - y(x_i;\mathbf{a})}{\sigma_i}\right) \tag{14.6.3}$$

where the function $\rho(z)$ is a function of a single variable $z \equiv [y_i - y(x_i)]/\sigma_i$.

If we now define the derivative of $\rho(z)$ to be a function $\psi(z)$,

$$\psi(z) \equiv \frac{d\rho(z)}{dz} \tag{14.6.4}$$

then the generalization of (14.1.7) to the case of a general M-estimate is

$$0 = \sum_{i=1}^{N} \frac{1}{\sigma_i}\psi\left(\frac{y_i - y(x_i)}{\sigma_i}\right)\left(\frac{\partial y(x_i;\mathbf{a})}{\partial a_k}\right) \qquad k = 1,\ldots,M \tag{14.6.5}$$

If you compare (14.6.3) to (14.1.3), and (14.6.5) to (14.1.7), you see at once that the specialization for normally distributed errors is

$$\rho(z) = \frac{1}{2}z^2 \qquad \psi(z) = z \qquad \text{(normal)} \tag{14.6.6}$$

If the errors are distributed as a *double* or *two-sided exponential*, namely

$$\text{Prob}\,\{y_i - y(x_i)\} \sim \exp\left(-\left|\frac{y_i - y(x_i)}{\sigma_i}\right|\right) \tag{14.6.7}$$

then, by contrast,

$$\rho(x) = |z| \qquad \psi(z) = \text{sgn}(z) \qquad \text{(double exponential)} \tag{14.6.8}$$

Comparing to equation (14.6.3), we see that in this case the maximum likelihood estimator is obtained by minimizing the *mean absolute deviation*, rather than the mean square deviation. Here the tails of the distribution, although exponentially decreasing, are asymptotically much larger than any corresponding Gaussian.

A distribution with even more extensive — therefore sometimes even more realistic — tails is the *Cauchy* or *Lorentzian* distribution,

$$\text{Prob}\,\{y_i - y(x_i)\} \sim \frac{1}{1 + \frac{1}{2}\left(\frac{y_i - y(x_i)}{\sigma_i}\right)^2} \tag{14.6.9}$$

This implies

$$\rho(z) = \log\left(1 + \frac{1}{2}z^2\right) \qquad \psi(z) = \frac{z}{1 + \frac{1}{2}z^2} \qquad \text{(Lorentzian)} \qquad (14.6.10)$$

Notice that the ψ function occurs as a weighting function in the generalized normal equations (14.6.5). For normally distributed errors, equation (14.6.6) says that the more deviant the points, the greater the weight. By contrast, when tails are somewhat more prominent, as in (14.6.7), then (14.6.8) says that all deviant points get the same relative weight, with only the sign information used. Finally, when the tails are even larger, (14.6.10) says the ψ increases with deviation, then starts *decreasing*, so that very deviant points — the true outliers — are not counted at all in the estimation of the parameters.

This general idea, that the weight given individual points should first increase with deviation, then decrease, motivates some additional prescriptions for ψ which do not especially correspond to standard, textbook probability distributions. Two examples are

Andrew's sine

$$\psi(z) = \begin{cases} \sin(z/c) & |z| < c\pi \\ 0 & |z| > c\pi \end{cases} \qquad (14.6.11)$$

If the measurement errors happen to be normal after all, with standard deviations σ_i, then it can be shown that the optimal value for the constant c is $c = 2.1$.

Tukey's biweight

$$\psi(z) = \begin{cases} z(1 - z^2/c^2)^2 & |z| < c \\ 0 & |z| > c \end{cases} \qquad (14.6.12)$$

where the optimal value of c for normal errors is $c = 6.0$.

Numerical Calculation of M-estimates

To fit a model by means of an M-estimate, you first decide which M-estimate you want, that is, which matching pair ρ, ψ you want to use. We rather like (14.6.8) or (14.6.10).

You then have to make Hobson's choice between two fairly difficult problems. Either find the solution of the nonlinear set of M equations (14.6.5), or else minimize the single function in M variables (14.6.3).

Notice that the function (14.6.8) has a discontinuous ψ, and a discontinuous derivative for ρ. Such discontinuities frequently wreak havoc on both general nonlinear equation solvers and general function minimizing routines. You might now think of rejecting (14.6.8) in favor of (14.6.10), which is smoother. However, you will find that the latter choice is also bad news for many general

equation solving or minimization routines: small changes in the fitted parameters can drive $\psi(z)$ off its peak into one or the other of its asymptotically small regimes. Therefore, different terms in the equation spring into or out of action (almost as bad as analytic discontinuities).

Don't despair. If your computer budget (or, for personal computers, patience) is up to it, this is an excellent application for the downhill simplex minimization algorithm exemplified in `amoeba` §10.4. That algorithm makes no assumptions about continuity, it just oozes downhill. It will work for virtually any sane choice of the function ρ.

It is very much to your (financial) advantage to find good starting values, however. Often this is done by first fitting the model by the standard χ^2 (nonrobust) techniques, e.g. as described in §14.3 or §14.4. The fitted parameters thus obtained are then used as starting values in `amoeba`, now using the robust choice of ρ and minimizing the expression (14.6.3).

Fitting a Line by Minimizing Absolute Deviation

Occasionally there is a special case that happens to be much easier than is suggested by the general strategy outlined above. The case of equations (14.6.7)–(14.6.8), when the model is a simple straight line

$$y(x; a, b) = a + bx \tag{14.6.13}$$

and where the weights σ_i are all equal, happens to be such a case. The problem is precisely the robust version of the problem posed in equation (14.2.1) above, namely fit a straight line through a set of data points. The merit function to be minimized is

$$\sum_{i=1}^{N} |y_i - a - bx_i| \tag{14.6.14}$$

rather than the χ^2 given by equation (14.2.2).

The key simplification is based on the following fact: The median c_M of a set of numbers c_i is also that value which minimizes the sum of the absolute deviations

$$\sum_i |c_i - c_M|$$

(Proof: Differentiate the above expression with respect to c_M and set it to zero.)

It follows that, for fixed b, the value of a which minimizes (14.6.14) is

$$a = \text{median}\,\{y_i - bx_i\} \tag{14.6.15}$$

Equation (14.6.5) for the parameter b is

$$0 = \sum_{i=1}^{N} x_i \, \mathrm{sgn}(y_i - a - bx_i) \tag{14.6.16}$$

If we replace a in this equation by the implied function $a(b)$ of (14.6.15), then we are left with an equation in a single variable which can be solved by bracketing and bisection, as described in §9.1. (In fact, it is dangerous to use any fancier method of root-finding, because of the discontinuities in equation 14.6.16.)

Here is a routine which does all this. It calls `sort` (§8.2) to find the median by the sorting method, cf. §13.2. The bracketing and bisection are built in to the following routine, as is the χ^2 solution which generates the initial guesses for a and b. Notice that the evaluation of the right-hand side of (14.6.16) occurs in the function `rofunc`, with communication via global (top-level) variables.

```
#include <math.h>

int ndatat=0;    /* defining declaration */
float *xt=0,*yt=0,aa=0.0,abdevt=0.0;    /* defining declaration */

void medfit(x,y,ndata,a,b,abdev)
float *x,*y,*a,*b,*abdev;
int ndata;
Fits y = a + bx by the criterion of least absolute deviations. The arrays x[1..ndata] and
y[1..ndata] are the input experimental points. The fitted parameters a and b are output,
along with abdev which is the mean absolute deviation (in y) of the experimental points from
the fitted line. This routine uses the routine rofunc, with communication via global variables.
{
    int j;
    float bb,b1,b2,del,f,f1,f2,sigb,temp;
    float sx=0.0,sy=0.0,sxy=0.0,sxx=0.0,chisq=0.0;
    float rofunc();

    ndatat=ndata;
    xt=x;
    yt=y;
    for (j=1;j<=ndata;j++) {          As a first guess for a and b, we will find the least-squares
        sx += x[j];                       fitting line.
        sy += y[j];
        sxy += x[j]*y[j];
        sxx += x[j]*x[j];
    }
    del=ndata*sxx-sx*sx;
    aa=(sxx*sy-sx*sxy)/del;               Least-squares solutions.
    bb=(ndata*sxy-sx*sy)/del;
    for (j=1;j<=ndata;j++)
        chisq += (temp=y[j]-(aa+bb*x[j]),temp*temp);
    sigb=sqrt(chisq/del);                 The standard deviation will give some idea of how big
    b1=bb;                                    an iteration step to take.
    f1=rofunc(b1);
    b2=bb+((f1 > 0.0) ? fabs(3.0*sigb) : -fabs(3.0*sigb));      Guess bracket as 3-σ
    f2=rofunc(b2);                            away, in the downhill direction known from f1.
    while (f1*f2 > 0.0) {                 Bracketing.
```

```
        bb=2.0*b2-b1;
        b1=b2;
        f1=f2;
        b2=bb;
        f2=rofunc(b2);
    }
    sigb=0.01*sigb;                    Refine until error a negligible number of standard devi-
    while (fabs(b2-b1) > sigb) {           ations.
        bb=0.5*(b1+b2);                Bisection.
        if (bb == b1 || bb == b2) break;
        f=rofunc(bb);
        if (f*f1 >= 0.0) {
            f1=f;
            b1=bb;
        } else {
            f2=f;
            b2=bb;
        }
    }
    *a=aa;
    *b=bb;
    *abdev=abdevt/ndata;
}
```

```
#include <math.h>

extern int ndatat;    /* defined in MEDFIT */
extern float *xt,*yt,aa,abdevt;

float rofunc(b)
float b;
Evaluates the right-hand side of equation (14.6.16) for a given value of b. Communication
with the program medfit is through global variables.
{
    int j,n1,nmh,nml;
    float *arr,d,sum=0.0,*vector();
    void sort(),free_vector();

    arr=vector(1,ndatat);
    n1=ndatat+1;
    nml=n1/2;
    nmh=n1-nml;
    for (j=1;j<=ndatat;j++) arr[j]=yt[j]-b*xt[j];
    sort(ndatat,arr);
    aa=0.5*(arr[nml]+arr[nmh]);
    abdevt=0.0;
    for (j=1;j<=ndatat;j++) {
        d=yt[j]-(b*xt[j]+aa);
        abdevt += fabs(d);
        sum += d > 0.0 ? xt[j] : -xt[j];
    }
    free_vector(arr,1,ndatat);
    return sum;
}
```

Other Robust Techniques

Sometimes you may have *a priori* knowledge about the probable values and probable uncertainties of some parameters that you are trying to estimate from a data set. In such cases you may want to perform a fit that takes this advance information properly into account, neither completely freezing a parameter at a predetermined value (as in `lfit` §14.3) nor completely leaving it to be determined by the data set. The formalism for doing this is called "use of a priori covariances."

A related problem occurs in signal processing and control theory, where it is sometimes desired to "track" (i.e. maintain an estimate of) a time-varying signal in the presence of noise. If the signal is known to be characterized by some number of parameters that vary only slowly, then the formalism of *Kalman filtering* tells how the incoming, raw measurements of the signal should be processed to produce best parameter estimates as a function of time. For example, if the signal is a frequency-modulated sine wave, then the slowly varying parameter might be the instantaneous frequency. The Kalman filter for this case is called a *phase-locked loop* and is implemented in the circuitry of good radio receivers.

Consult Bryson and Ho, or Jazwinski for details on these and other techniques.

REFERENCES AND FURTHER READING:

Huber, P.J. 1981, *Robust Statistics* (New York: Wiley).

Launer, R.L., and Wilkinson, G.N., eds. 1979, *Robustness in Statistics* (New York: Academic Press).

Bryson, A. E., and Ho, Y.C. 1969, *Applied Optimal Control* (Waltham, Mass.: Ginn).

Jazwinski, A. H. 1970, *Stochastic Processes and Filtering Theory* (New York: Academic Press).