15.4 General Linear Least Squares

An immediate generalization of §15.2 is to fit a set of data points $(x_i,\ y_i)$ to a model that 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,

$$y(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_M x^{M-1} \tag{15.4.1}$$

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

The general form of this kind of model is

$$y(x) = \sum_{k=1}^{M} a_k X_k(x) \tag{15.4.2}$$

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" refers only 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

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{M} a_k X_k(x_i)}{\sigma_i} \right]^2 \tag{15.4.3}$$

As before, σ_i is the measurement error (standard deviation) of the ith data point, presumed to be known. If the measurement errors are not known, they may all (as discussed at the end of §15.1) 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

$$A_{ij} = \frac{X_j(x_i)}{\sigma_i} \tag{15.4.4}$$

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 15.4.1.

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

$$b_i = \frac{y_i}{\sigma_i} \tag{15.4.5}$$

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

$$\begin{array}{ccc}
 & & \longleftarrow \text{basis functions} \longrightarrow \\
 & & \begin{array}{cccc} X_1(\ ) & X_2(\ ) & \cdots & X_M(\ ) \end{array} \\
 & & \\
\text{data points} & \begin{array}{c} \uparrow\ x_1 \\ \\ x_2 \\ \\ \vdots \\ \\ \downarrow\ x_N \end{array} & \begin{pmatrix}
\dfrac{X_1(x_1)}{\sigma_1} & \dfrac{X_2(x_1)}{\sigma_1} & \cdots & \dfrac{X_M(x_1)}{\sigma_1} \\[2ex]
\dfrac{X_1(x_2)}{\sigma_2} & \dfrac{X_2(x_2)}{\sigma_2} & \cdots & \dfrac{X_M(x_2)}{\sigma_2} \\[2ex]
\vdots & & & \vdots \\
\vdots & & & \vdots \\
\vdots & & & \vdots \\[2ex]
\dfrac{X_1(x_N)}{\sigma_N} & \dfrac{X_2(x_N)}{\sigma_N} & \cdots & \dfrac{X_M(x_N)}{\sigma_N}
\end{pmatrix}
\end{array}$$

Figure 15.4.1. 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.

Solution by Use of the Normal Equations

The minimum of (15.4.3) occurs where the derivative of χ^2 with respect to all M parameters a_k vanishes. Specializing equation (15.1.7) to the case of the model (15.4.2), this condition yields the M equations

$$0 = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[y_i - \sum_{j=1}^{M} a_j X_j(x_i) \right] X_k(x_i) \qquad k = 1, \ldots, M \tag{15.4.6}$$

Interchanging the order of summations, we can write (15.4.6) as the matrix equation

$$\sum_{j=1}^{M} \alpha_{kj} a_j = \beta_k \tag{15.4.7}$$

where

$$\alpha_{kj} = \sum_{i=1}^{N} \frac{X_j(x_i) X_k(x_i)}{\sigma_i^2} \qquad \text{or equivalently} \qquad [\alpha] = \mathbf{A}^T \cdot \mathbf{A} \tag{15.4.8}$$

an $M \times M$ matrix, and

$$\beta_k = \sum_{i=1}^{N} \frac{y_i X_k(x_i)}{\sigma_i^2} \qquad \text{or equivalently} \qquad [\beta] = \mathbf{A}^T \cdot \mathbf{b} \tag{15.4.9}$$

a vector of length M.

The equations (15.4.6) or (15.4.7) are called the *normal equations* of the least-squares problem. They can be solved for the vector of parameters **a** by the standard methods of Chapter 2, notably LU decomposition and backsubstitution, Choleksy decomposition, or Gauss-Jordan elimination. In matrix form, the normal equations can be written as either

$$[\alpha] \cdot \mathbf{a} = [\beta] \qquad \text{or as} \qquad \left(\mathbf{A}^T \cdot \mathbf{A}\right) \cdot \mathbf{a} = \mathbf{A}^T \cdot \mathbf{b} \tag{15.4.10}$$

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

$$a_j = \sum_{k=1}^{M} [\alpha]^{-1}_{jk} \beta_k = \sum_{k=1}^{M} C_{jk} \left[\sum_{i=1}^{N} \frac{y_i X_k(x_i)}{\sigma_i^2} \right] \tag{15.4.11}$$

and that the variance associated with the estimate a_j can be found as in (15.2.7) from

$$\sigma^2(a_j) = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial a_j}{\partial y_i} \right)^2 \tag{15.4.12}$$

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

$$\frac{\partial a_j}{\partial y_i} = \sum_{k=1}^{M} C_{jk} X_k(x_i)/\sigma_i^2 \tag{15.4.13}$$

Consequently, we find that

$$\sigma^2(a_j) = \sum_{k=1}^{M} \sum_{l=1}^{M} C_{jk} C_{jl} \left[\sum_{i=1}^{N} \frac{X_k(x_i) X_l(x_i)}{\sigma_i^2} \right] \tag{15.4.14}$$

The final term in brackets is just the matrix $[\alpha]$. Since this is the matrix inverse of $[C]$, (15.4.14) reduces immediately to

$$\sigma^2(a_j) = C_{jj} \tag{15.4.15}$$

In other words, the diagonal elements of $[C]$ are the variances (squared uncertainties) of the fitted parameters **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. 15.2.10); but we shall defer discussion of these to §15.6.

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 **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. In theory, since $\mathbf{A}^T \cdot \mathbf{A}$ is positive definite, Cholesky decomposition is the most efficient way to solve the normal equations. However, in practice most of the computing time is spent in looping over the data to form the equations, and Gauss-Jordan is quite adequate.

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 (§2.10, §11.3, and §11.6) of the design matrix $\mathbf{A}$. This is essentially what we did at the end of §15.2 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 trick 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). As input to the routine, you supply an array `ia(1:ma)`, whose components are either zero or nonzero (e.g., 1). Zeros indicate that you want the corresponding elements of the parameter vector `a(1:ma)` to be held fixed at their input values. Nonzeros indicate parameters that should be fitted for. On output, any frozen parameters will have their variances, and all their covariances, set to zero in the covariance matrix.

```
      SUBROUTINE lfit(x,y,sig,ndat,a,ia,ma,covar,npc,chisq,funcs)
      INTEGER ma,ia(ma),npc,ndat,MMAX
      REAL chisq,a(ma),covar(npc,npc),sig(ndat),x(ndat),y(ndat)
      EXTERNAL funcs
      PARAMETER (MMAX=50)            Set to the maximum number of coefficients ma.
C     USES covsrt,gaussj
         Given a set of data points x(1:ndat), y(1:ndat) with individual standard deviations
         sig(1:ndat), use χ² minimization to fit for some or all of the coefficients a(1:ma) of a
         function that depends linearly on a, y = Σ_i a_i × afunc_i(x). The input array ia(1:ma) in-
         dicates by nonzero entries those components of a that should be fitted for, and by zero entries
         those components that should be held fixed at their input values. The program returns values
         for a(1:ma), χ² = chisq, and the covariance matrix covar(1:ma,1:ma). (Parameters
         held fixed will return zero covariances.) npc is the physical dimension of covar(npc,npc)
         in the calling routine. The user supplies a subroutine funcs(x,afunc,ma) that returns
         the ma basis functions evaluated at x = x in the array afunc.
      INTEGER i,j,k,l,m,mfit
      REAL sig2i,sum,wt,ym,afunc(MMAX),beta(MMAX)
      mfit=0
      do 11 j=1,ma
          if(ia(j).ne.0) mfit=mfit+1
      enddo 11
      if(mfit.eq.0) pause 'lfit: no parameters to be fitted'
      do 13 j=1,mfit                 Initialize the (symmetric) matrix.
          do 12 k=1,mfit
              covar(j,k)=0.
          enddo 12
          beta(j)=0.
      enddo 13
```

```
do 17 i=1,ndat                        Loop over data to accumulate coefficients of the normal
    call funcs(x(i),afunc,ma)             equations.
    ym=y(i)
    if(mfit.lt.ma) then               Subtract off dependences on known pieces of the fitting
        do 14 j=1,ma                      function.
            if(ia(j).eq.0) ym=ym-a(j)*afunc(j)
        enddo 14
    endif
    sig2i=1./sig(i)**2
    j=0
    do 16 l=1,ma
        if (ia(l).ne.0) then
            j=j+1
            wt=afunc(l)*sig2i
            k=0
            do 15 m=1,l
                if (ia(m).ne.0) then
                    k=k+1
                    covar(j,k)=covar(j,k)+wt*afunc(m)
                endif
            enddo 15
            beta(j)=beta(j)+ym*wt
        endif
    enddo 16
enddo 17
do 19 j=2,mfit                        Fill in above the diagonal from symmetry.
    do 18 k=1,j-1
        covar(k,j)=covar(j,k)
    enddo 18
enddo 19
call gaussj(covar,mfit,npc,beta,1,1)    Matrix solution.
j=0
do 21 l=1,ma
    if(ia(l).ne.0) then
        j=j+1
        a(l)=beta(j)                  Partition solution to appropriate coefficients a.
    endif
enddo 21
chisq=0.                              Evaluate χ^2 of the fit.
do 23 i=1,ndat
    call funcs(x(i),afunc,ma)
    sum=0.
    do 22 j=1,ma
        sum=sum+a(j)*afunc(j)
    enddo 22
    chisq=chisq+((y(i)-sum)/sig(i))**2
enddo 23
call covsrt(covar,npc,ma,ia,mfit)       Sort covariance matrix to true order of fitting
return                                      coefficients.
END
```

That last call to a subroutine `covsrt` is only for the purpose of spreading the covariances back into the full $\mathtt{ma} \times \mathtt{ma}$ covariance matrix, in the proper rows and columns and with zero variances and covariances set for variables which were held frozen.

The subroutine `covsrt` is as follows.

```
SUBROUTINE covsrt(covar,npc,ma,ia,mfit)
INTEGER ma,mfit,npc,ia(ma)
REAL covar(npc,npc)
    Expand in storage the covariance matrix covar, so as to take into account parameters that
    are being held fixed. (For the latter, return zero covariances.)
INTEGER i,j,k
REAL swap
```

```
do 12 i=mfit+1,ma
    do 11 j=1,i
        covar(i,j)=0.
        covar(j,i)=0.
    enddo 11
enddo 12
k=mfit
do 15 j=ma,1,-1
    if(ia(j).ne.0)then
        do 13 i=1,ma
            swap=covar(i,k)
            covar(i,k)=covar(i,j)
            covar(i,j)=swap
        enddo 13
        do 14 i=1,ma
            swap=covar(k,i)
            covar(k,i)=covar(j,i)
            covar(j,i)=swap
        enddo 14
        k=k-1
    endif
enddo 15
return
END
```

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 §2.6, 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 (2.6.10). 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 (2.6.8). 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 15.4.4) and the vector $\mathbf{b}$ (equation 15.4.5), minimization of χ^2 in (15.4.3) can be written as

$$\text{find} \quad \mathbf{a} \quad \text{that minimizes} \quad \chi^2 = |\mathbf{A}\cdot\mathbf{a} - \mathbf{b}|^2 \tag{15.4.16}$$

Comparing to equation (2.6.9), we see that this is precisely the problem that routines `svdcmp` and `svbksb` are designed to solve. The solution, which is given by equation (2.6.12), can be rewritten as follows: If $\mathbf{U}$ and $\mathbf{V}$ enter the SVD decomposition of $\mathbf{A}$ according to equation (2.6.1), as computed by `svdcmp`, then let the vectors $\mathbf{U}_{(i)}$ $i = 1, \dots, M$ denote the *columns* of $\mathbf{U}$ (each one a vector of length N); and let the vectors $\mathbf{V}_{(i)}; i = 1, \dots, M$ denote the *columns* of $\mathbf{V}$ (each one a vector of length M). Then the solution (2.6.12) of the least-squares problem (15.4.16) can be written as

$$\mathbf{a} = \sum_{i=1}^{M} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{w_i} \right) \mathbf{V}_{(i)} \tag{15.4.17}$$

where the w_i are, as in §2.6, the singular values returned by `svdcmp`.

Equation (15.4.17) 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 (15.4.5). 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 (15.4.17) can be written in a form displaying these errors as

$$\mathbf{a} = \left[\sum_{i=1}^{M} \left(\frac{\mathbf{U}_{(i)} \cdot \mathbf{b}}{w_i} \right) \mathbf{V}_{(i)} \right] \pm \frac{1}{w_1}\mathbf{V}_{(1)} \pm \cdots \pm \frac{1}{w_M}\mathbf{V}_{(M)} \tag{15.4.18}$$

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 §15.6).

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

$$\sigma^2(a_j) = \sum_{i=1}^{M} \frac{1}{w_i^2} [\mathbf{V}_{(i)}]_j^2 = \sum_{i=1}^{M} \left(\frac{V_{ji}}{w_i} \right)^2 \tag{15.4.19}$$

whose result should be identical with (15.4.14). As before, you should not be surprised at the formula for the covariances, here given without proof,

$$\text{Cov}(a_j, a_k) = \sum_{i=1}^{M} \left(\frac{V_{ji} V_{ki}}{w_i^2} \right) \tag{15.4.20}$$

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 (15.4.18) should be set to zero, not infinity. (Compare the discussion preceding equation 2.6.7.) 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 that 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 that 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 §15.6. 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. `np` and `mp` are their various physical dimensions. 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.

```
      SUBROUTINE svdfit(x,y,sig,ndata,a,ma,u,v,w,mp,np,
     *     chisq,funcs)
      INTEGER ma,mp,ndata,np,NMAX,MMAX
      REAL chisq,a(ma),sig(ndata),u(mp,np),v(np,np),w(np),
     *     x(ndata),y(ndata),TOL
      EXTERNAL funcs
      PARAMETER (NMAX=1000,MMAX=50,TOL=1.e-5)    Max expected ndata and ma.
C     USES svbksb,svdcmp
```

Given a set of data points `x(1:ndata)`,`y(1:ndata)` with individual standard deviations `sig(1:ndata)`, use χ^2 minimization to determine the `ma` coefficients `a` of the fitting function $y = \sum_i \mathtt{a}_i \times \mathtt{afunc}_i(x)$. Here we solve the fitting equations using singular value decomposition of the `ndata` by `ma` matrix, as in §2.6. Arrays `u(1:mp,1:np)`,`v(1:np,1:np)`, `w(1:np)` provide workspace on input; on output they define the singular value decomposition, and can be used to obtain the covariance matrix. `mp`,`np` are the physical dimensions of the matrices `u`,`v`,`w`, as indicated above. It is necessary that `mp`$\geq$`ndata`, `np` $\geq$ `ma`. The program returns values for the `ma` fit parameters `a`, and χ^2, `chisq`. The user supplies a subroutine `funcs(x,afunc,ma)` that returns the `ma` basis functions evaluated at $x =$ `x` in the array `afunc`.

```
      INTEGER i,j
      REAL sum,thresh,tmp,wmax,afunc(MMAX),b(NMAX)
```

```
do 12 i=1,ndata                          Accumulate coefficients of the fitting ma-
    call funcs(x(i),afunc,ma)                trix.
    tmp=1./sig(i)
    do 11 j=1,ma
        u(i,j)=afunc(j)*tmp
    enddo 11
    b(i)=y(i)*tmp
enddo 12
call svdcmp(u,ndata,ma,mp,np,w,v)        Singular value decomposition.
wmax=0.                                  Edit the singular values, given TOL from the
do 13 j=1,ma                                 parameter statement, between here ...
    if(w(j).gt.wmax)wmax=w(j)
enddo 13
thresh=TOL*wmax
do 14 j=1,ma
    if(w(j).lt.thresh)w(j)=0.
enddo 14                                 ...and here.
call svbksb(u,w,v,ndata,ma,mp,np,b,a)
chisq=0.                                 Evaluate chi-square.
do 16 i=1,ndata
    call funcs(x(i),afunc,ma)
    sum=0.
    do 15 j=1,ma
        sum=sum+a(j)*afunc(j)
    enddo 15
    chisq=chisq+((y(i)-sum)/sig(i))**2
enddo 16
return
END
```

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 (15.4.20) above, with the convention that singular values equal to zero are recognized as having been edited out of the fit.

```
SUBROUTINE svdvar(v,ma,np,w,cvm,ncvm)
INTEGER ma,ncvm,np,MMAX
REAL cvm(ncvm,ncvm),v(np,np),w(np)
PARAMETER (MMAX=20)            Set to the maximum number of fit parameters.
  To evaluate the covariance matrix cvm of the fit for ma parameters obtained by svdfit,
  call this routine with matrices v,w as returned from svdfit. np,ncvm give the physical
  dimensions of v,w,cvm as indicated.
INTEGER i,j,k
REAL sum,wti(MMAX)
do 11 i=1,ma
    wti(i)=0.
    if(w(i).ne.0.) wti(i)=1./(w(i)*w(i))
enddo 11
do 14 i=1,ma                   Sum contributions to covariance matrix (15.4.20).
    do 13 j=1,i
        sum=0.
        do 12 k=1,ma
            sum=sum+v(i,k)*v(j,k)*wti(k)
        enddo 12
        cvm(i,j)=sum
        cvm(j,i)=sum
    enddo 13
enddo 14
return
END
```

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,

$$y(x) = a\exp(-bx) \tag{15.4.21}$$

can be rewritten as

$$\log[y(x)] = c - bx \tag{15.4.22}$$

which is linear in its parameters c and b. (Of course you must be aware that such transformations do not exactly take Gaussian errors into Gaussian errors.)

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

$$y(x) = a\exp(-bx + d) \tag{15.4.23}$$

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 returns 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:

```
SUBROUTINE fpoly(x,p,np)
INTEGER np
REAL x,p(np)
   Fitting routine for a polynomial of degree np-1, with np coefficients.
INTEGER j
p(1)=1.
do 11 j=2,np
   p(j)=p(j-1)*x
enddo 11
return
END
```

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

```
SUBROUTINE fleg(x,pl,nl)
INTEGER nl
REAL x,pl(nl)
   Fitting routine for an expansion with nl Legendre polynomials pl, evaluated using the
   recurrence relation as in §5.5.
INTEGER j
REAL d,f1,f2,twox
pl(1)=1.
pl(2)=x
if(nl.gt.2) then
   twox=2.*x
   f2=x
   d=1.
```

```
    do 11 j=3,nl
        f1=d
        f2=f2+twox
        d=d+1.
        pl(j)=(f2*pl(j-1)-f1*pl(j-2))/d
    enddo 11
endif
return
END
```

Multidimensional Fits

If you are measuring a single variable y as a function of more than one variable — say, a *vector* of variables $\mathbf{x}$, then your basis functions will be functions of a vector, $X_1(\mathbf{x}), \dots, X_M(\mathbf{x})$. The χ^2 merit function is now

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{M} a_k X_k(\mathbf{x}_i)}{\sigma_i} \right]^2 \tag{15.4.24}$$

All of the preceding discussion goes through unchanged, with x replaced by $\mathbf{x}$. In fact, if you are willing to tolerate a bit of programming hack, you can use the above programs without any modification: In both `lfit` and `svdfit`, the only use made of the array elements `x(i)` is that each element is in turn passed to the user-supplied routine `funcs`, which duly returns the values of the basis functions at that point. If you set `x(i)=i` before calling `lfit` or `svdfit`, and independently provide `funcs` with the true vector values of your data points (e.g., in a `COMMON` block), then `funcs` can translate from the fictitious `x(i)`'s to the actual data points before doing its work.

CITED REFERENCES AND FURTHER READING:

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

Lawson, C.L., and Hanson, R. 1974, *Solving Least Squares Problems* (Englewood Cliffs, NJ: Prentice-Hall).

Forsythe, G.E., Malcolm, M.A., and Moler, C.B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 9.

15.5 Nonlinear Models

We now consider fitting when the model depends *nonlinearly* on the set of M unknown parameters $a_k, k = 1, 2, \dots, M$. 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.