

2.6 Singular Value Decomposition

There exists a very powerful set of techniques for dealing with sets of equations or matrices that are either singular or else numerically very close to singular. In many cases where Gaussian elimination and *LU* decomposition fail to give satisfactory results, this set of techniques, known as *singular value decomposition*, or *SVD*, will diagnose for you precisely what the problem is. In some cases, *SVD* will not only diagnose the problem, it will also solve it, in the sense of giving you a useful numerical answer, although, as we shall see, not necessarily "the" answer that you thought you should get.

SVD is also the method of choice for solving most *linear least-squares* problems. We will outline the relevant theory in this section, but defer detailed discussion of the use of *SVD* in this application to Chapter 15, whose subject is the parametric modeling of data.

SVD methods are based on the following theorem of linear algebra, whose proof is beyond our scope: Any $M \times N$ matrix **A** whose number of rows M is greater than or equal to its number of columns N , can be written as the product of an $M \times N$ column-orthogonal matrix **U**, an $N \times N$ diagonal matrix **W** with positive or zero elements (the *singular values*), and the transpose of an $N \times N$ orthogonal matrix **V**. The various shapes of these matrices will be made clearer by the following tableau:

$$\begin{pmatrix} \mathbf{A} \end{pmatrix} = \begin{pmatrix} \mathbf{U} \end{pmatrix} \cdot \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \dots & \\ & & & w_N \end{pmatrix} \cdot \begin{pmatrix} \mathbf{V}^T \end{pmatrix} \tag{2.6.1}$$

The matrices **U** and **V** are each orthogonal in the sense that their columns are orthonormal,

$$\sum_{i=1}^M U_{ik}U_{in} = \delta_{kn} \quad \begin{matrix} 1 \leq k \leq N \\ 1 \leq n \leq N \end{matrix} \tag{2.6.2}$$

st amplification of length that it is

(2.5.12)

side **b**, as one wants a matrix inverse
ngence is

(2.5.13)

ϵ_0 is any sufficiently small constant

(2.5.14)

11; we give here only the briefest
so it has real, positive eigenvalues.

(2.5.15)

11; we give here only the briefest
so it has real, positive eigenvalues.

(2.5.16)

n equation (2.5.16). Pan and Reif
e directly from **A**. The following

(2.5.17)

Pan and Reif derive by noting that
 L_2 norm, but can instead be either
e their work for details.
success, is to estimate the largest
veral unit vector v_i 's with randomly
1 then be bounded by the maximum
te the sample variance and mean,

2nd ed. (Reading, MA: Addison-
2nd ed. (Baltimore: Johns Hopkins
nglewood Cliffs, NJ: Prentice-Hall),
f *Linear Algebraic Systems* (Engle-
erical Analysis, 2nd ed. (New York:

SVD of a Square Matrix

If the matrix A is square, $N \times N$ say, then U , V , and W are all square matrices of the same size. Their inverses are also trivial to compute: U and V are orthogonal, so their inverses are equal to their transposes; W is diagonal, so its inverse is the diagonal matrix whose elements are the reciprocals of the elements w_j . From (2.6.1) it now follows immediately that the inverse of A is

$$A^{-1} = V \cdot [\text{diag} (1/w_j)] \cdot U^T \tag{2.6.5}$$

The only thing that can go wrong with this construction is for one of the w_j 's to be zero, or (numerically) for it to be so small that its value is dominated by roundoff error and therefore unknowable. If more than one of the w_j 's have this problem, then the matrix is even more singular. So, first of all, SVD gives you a clear diagnosis of the situation.

Formally, the *condition number* of a matrix is defined as the ratio of the largest (in magnitude) of the w_j 's to the smallest of the w_j 's. A matrix is singular if its condition number is infinite, and it is *ill-conditioned* if its condition number is too large, that is, if its reciprocal approaches the machine's floating-point precision (for example, less than 10^{-6} for single precision or 10^{-12} for double).

For singular matrices, the concepts of *nullspace* and *range* are important. Consider the familiar set of simultaneous equations

$$A \cdot x = b \tag{2.6.6}$$

where A is a square matrix, b and x are vectors. Equation (2.6.6) defines A as a linear mapping from the vector space x to the vector space b . If A is singular, then there is some subspace of x , called the nullspace, that is mapped to zero, $A \cdot x = 0$. The dimension of the nullspace (the number of linearly independent vectors x that can be found in it) is called the *nullity* of A .

Now, there is also some subspace of b that can be "reached" by A , in the sense that there exists some x which is mapped there. This subspace of b is called the range of A . The dimension of the range is called the *rank* of A . If A is nonsingular, then its range will be all of the vector space b , so its rank is N . If A is singular, then the rank will be less than N . In fact, the relevant theorem is "rank plus nullity equals N ."

What has this to do with SVD? SVD explicitly constructs orthonormal bases for the nullspace and range of a matrix. Specifically, the columns of U whose same-numbered elements w_j are *nonzero* are an orthonormal set of basis vectors that span the range; the columns of V whose same-numbered elements w_j are *zero* are an orthonormal basis for the nullspace.

Now let's have another look at solving the set of simultaneous linear equations (2.6.6) in the case that A is singular. First, the set of *homogeneous* equations, where $b = 0$, is solved immediately by SVD: Any column of V whose corresponding w_j is *zero* yields a solution.

When the vector b on the right-hand side is not zero, the important question is whether it lies in the range of A or not. If it does, then the singular set of equations *does* have a solution x ; in fact it has more than one solution, since any vector in

$$V^T \cdot \begin{pmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{pmatrix} \cdot \begin{pmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{pmatrix}$$

$$1 \quad \left. \begin{matrix} \vdots \\ \vdots \\ \vdots \end{matrix} \right\} \tag{2.6.4}$$

$V^T = 1$.
 out when $M < N$. In this case all zero, and the corresponding olds only for $k, n \leq M$.
 ie, no matter how singular the , it is unique up to (i) making ts of W , and columns of V (or any columns of U and V whose al. An important consequence $< N$, a numerical algorithm for $= M + 1, \dots, N$; the $N - M$ tions $j = 1, 2, \dots, N$.
 vdcmp, that performs SVD on e same shape) and giving back d on a routine by Forsythe et of Golub and Reinsch, found, in es include extensive discussion e of black-box routines, we are take us too far afield to cover o say that the algorithm is very ave. Most of the concepts that gonal form, diagonalization by in Chapter 11.
 you will want to verify yourself sy to do: Generate an arbitrary multiplication that (2.6.1) and the only defining requirements delete end-to-end check.

If we want to single out one particular member of this solution-set of vectors as a representative, we might want to pick the one with the smallest length $|\mathbf{x}|^2$. Here is how to find that vector using SVD: Simply replace $1/w_j$ by zero if $w_j = 0$. (It is not very often that one gets to set $\infty = 0$!) Then compute (working from right to left)

$$\mathbf{x} = \mathbf{V} \cdot [\text{diag}(1/w_j)] \cdot (\mathbf{U}^T \cdot \mathbf{b}) \quad (2.6.7)$$

This will be the solution vector of smallest length; the columns of \mathbf{V} that are in the nullspace complete the specification of the solution set.

Proof: Consider $|\mathbf{x} + \mathbf{x}'|$, where \mathbf{x}' lies in the nullspace. Then, if \mathbf{W}^{-1} denotes the modified inverse of \mathbf{W} with some elements zeroed,

$$\begin{aligned} |\mathbf{x} + \mathbf{x}'| &= |\mathbf{V} \cdot \mathbf{W}^{-1} \cdot \mathbf{U}^T \cdot \mathbf{b} + \mathbf{x}'| \\ &= |\mathbf{V} \cdot (\mathbf{W}^{-1} \cdot \mathbf{U}^T \cdot \mathbf{b} + \mathbf{V}^T \cdot \mathbf{x}')| \\ &= |\mathbf{W}^{-1} \cdot \mathbf{U}^T \cdot \mathbf{b} + \mathbf{V}^T \cdot \mathbf{x}'| \end{aligned} \quad (2.6.8)$$

Here the first equality follows from (2.6.7), the second and third from the orthonormality of \mathbf{V} . If you now examine the two terms that make up the sum on the right-hand side, you will see that the first one has nonzero j components only where $w_j \neq 0$, while the second one, since \mathbf{x}' is in the nullspace, has nonzero j components only where $w_j = 0$. Therefore the minimum length obtains for $\mathbf{x}' = 0$, q.e.d.

If \mathbf{b} is not in the range of the singular matrix \mathbf{A} , then the set of equations (2.6.6) has no solution. But here is some good news: If \mathbf{b} is not in the range of \mathbf{A} , then equation (2.6.7) can still be used to construct a "solution" vector \mathbf{x} . This vector \mathbf{x} will not exactly solve $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. But, among all possible vectors \mathbf{x} , it will do the closest possible job in the least squares sense. In other words (2.6.7) finds

$$\mathbf{x} \text{ which minimizes } r \equiv |\mathbf{A} \cdot \mathbf{x} - \mathbf{b}| \quad (2.6.9)$$

The number r is called the *residual* of the solution.

The proof is similar to (2.6.8): Suppose we modify \mathbf{x} by adding some arbitrary \mathbf{x}' . Then $\mathbf{A} \cdot \mathbf{x} - \mathbf{b}$ is modified by adding some $\mathbf{b}' \equiv \mathbf{A} \cdot \mathbf{x}'$. Obviously \mathbf{b}' is in the range of \mathbf{A} . We then have

$$\begin{aligned} |\mathbf{A} \cdot \mathbf{x} - \mathbf{b} + \mathbf{b}'| &= |(\mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^T) \cdot (\mathbf{V} \cdot \mathbf{W}^{-1} \cdot \mathbf{U}^T \cdot \mathbf{b}) - \mathbf{b} + \mathbf{b}'| \\ &= |(\mathbf{U} \cdot \mathbf{W} \cdot \mathbf{W}^{-1} \cdot \mathbf{U}^T - 1) \cdot \mathbf{b} + \mathbf{b}'| \\ &= |\mathbf{U} \cdot [(\mathbf{W} \cdot \mathbf{W}^{-1} - 1) \cdot \mathbf{U}^T \cdot \mathbf{b} + \mathbf{U}^T \cdot \mathbf{b}']| \\ &= |(\mathbf{W} \cdot \mathbf{W}^{-1} - 1) \cdot \mathbf{U}^T \cdot \mathbf{b} + \mathbf{U}^T \cdot \mathbf{b}'| \end{aligned} \quad (2.6.10)$$

Now, $(\mathbf{W} \cdot \mathbf{W}^{-1} - 1)$ is a diagonal matrix which has nonzero j components only for $w_j = 0$, while $\mathbf{U}^T \mathbf{b}'$ has nonzero j components only for $w_j \neq 0$, since \mathbf{b}' lies in the range of \mathbf{A} . Therefore the minimum obtains for $\mathbf{b}' = 0$, q.e.d.

Figure 2.6.1 summarizes our discussion of SVD thus far.

Equations

norm-set of vectors as
 st length $\|x\|^2$. Here is
 zero if $w_j = 0$. (It is not
 from right to left)

(2.6.7)

ns of V that are in the
 Then, if W^{-1} denotes

(2.6.8)

third from the orthonor-
 take up the sum on the
 components only where
 s nonzero j components
 for $x'_j = 0$, q.e.d.
 set of equations (2.6.6)
 in the range of A , then
 vector x . This vector x
 vectors x , it will do the
 rds (2.6.7) finds

(2.6.9)

by adding some arbitrary
 x' . Obviously b' is in

(2.6.10)

zero j components only for
 $w_j \neq 0$ since b' lies in the

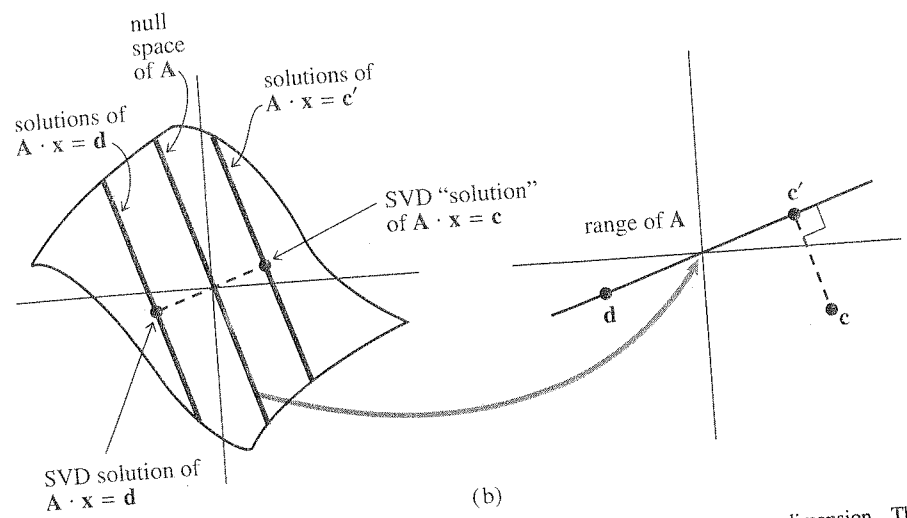
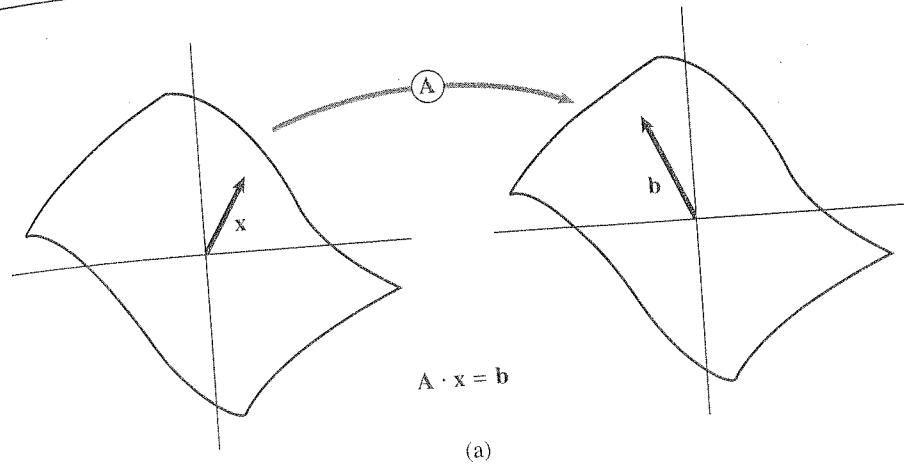


Figure 2.6.1. (a) A nonsingular matrix A maps a vector space into one of the same dimension. The vector x is mapped into b , so that x satisfies the equation $A \cdot x = b$. (b) A singular matrix A maps a vector space into one of lower dimensionality, here a plane into a line, called the "range" of A . The "nullspace" of A is mapped to zero. The solutions of $A \cdot x = d$ consist of any one particular solution plus any vector in the nullspace, here forming a line parallel to the nullspace. Singular value decomposition (SVD) selects the particular solution closest to zero, as shown. The point c lies outside of the range of A , so $A \cdot x = c$ has no solution. SVD finds the least-squares best compromise solution, namely a solution of $A \cdot x = c'$, as shown.

In the discussion since equation (2.6.6), we have been pretending that a matrix either is singular or else isn't. That is of course true analytically. Numerically, however, the far more common situation is that some of the w_j 's are very small but nonzero, so that the matrix is ill-conditioned. In that case, the direct solution methods of LU decomposition or Gaussian elimination may actually give a formal solution to the set of equations (that is, a zero pivot may not be encountered); but the solution vector may have wildly large components whose algebraic cancellation, when multiplying by the matrix A , may give a very poor approximation to the solution vector x obtained by zeroing the

small w_j 's and then using equation (2.6.7) is very often better (in the sense of the residual $\|A \cdot x - b\|$ being smaller) than *both* the direct-method solution *and* the SVD solution where the small w_j 's are left nonzero.

It may seem paradoxical that this can be so, since zeroing a singular value corresponds to throwing away one linear combination of the set of equations that we are trying to solve. The resolution of the paradox is that we are throwing away precisely a combination of equations that is so corrupted by roundoff error as to be at best useless; usually it is worse than useless since it "pulls" the solution vector way off towards infinity along some direction that is almost a nullspace vector. In doing this, it compounds the roundoff problem and makes the residual $\|A \cdot x - b\|$ larger.

SVD cannot be applied blindly, then. You have to exercise some discretion in deciding at what threshold to zero the small w_j 's, and/or you have to have some idea what size of computed residual $\|A \cdot x - b\|$ is acceptable.

As an example, here is a "backsubstitution" routine `svbksb` for evaluating equation (2.6.7) and obtaining a solution vector x from a right-hand side b , given that the SVD of a matrix A has already been calculated by a call to `svdcmp`. Note that this routine presumes that *you* have already zeroed the small w_j 's. It does not do this for you. If you *haven't* zeroed the small w_j 's, then this routine is just as ill-conditioned as any direct method, and you are misusing SVD.

```
#include "nrutil.h"

void svbksb(float **u, float w[], float **v, int m, int n, float b[], float x[])
Solves  $A \cdot X = B$  for a vector  $X$ , where  $A$  is specified by the arrays  $u[1..m][1..n]$ ,  $w[1..n]$ ,
 $v[1..n][1..n]$  as returned by svdcmp.  $m$  and  $n$  are the dimensions of  $a$ , and will be equal for
square matrices.  $b[1..m]$  is the input right-hand side.  $x[1..n]$  is the output solution vector.
No input quantities are destroyed, so the routine may be called sequentially with different  $b$ 's.
{
    int jj,j,i;
    float s,*tmp;

    tmp=vector(1,n);
    for (j=1;j<=n;j++) {          Calculate  $U^T B$ .
        s=0.0;
        if (w[j]) {              Nonzero result only if  $w_j$  is nonzero.
            for (i=1;i<=m;i++) s += u[i][j]*b[i];
            s /= w[j];           This is the divide by  $w_j$ .
        }
        tmp[j]=s;
    }
    for (j=1;j<=n;j++) {        Matrix multiply by  $V$  to get answer.
        s=0.0;
        for (jj=1;jj<=n;jj++) s += v[j][jj]*tmp[jj];
        x[j]=s;
    }
    free_vector(tmp,1,n);
}
```

Note that a typical use of `svdcmp` and `svbksb` superficially resembles the typical use of `ludcmp` and `lubksb`: In both cases, you decompose the left-hand matrix A just once, and then can use the decomposition either once or many times with different right-hand sides. The crucial difference is the "editing" of the singular values before `svbksb` is called:

given by (2.6.7), which, with nonsquare matrices, looks like this,

$$\begin{pmatrix} \mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{V} \end{pmatrix} \cdot \begin{pmatrix} \text{diag}(1/w_j) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{U}^T \end{pmatrix} \cdot \begin{pmatrix} \mathbf{b} \end{pmatrix} \quad (2.6.12)$$

In general, the matrix \mathbf{W} will not be singular, and no w_j 's will need to be set to zero. Occasionally, however, there might be column degeneracies in \mathbf{A} . In this case you will need to zero some small w_j values after all. The corresponding column in \mathbf{V} gives the linear combination of \mathbf{x} 's that is then ill-determined even by the supposedly overdetermined set.

Sometimes, although you do not need to zero any w_j 's for *computational* reasons, you may nevertheless want to take note of any that are unusually small: Their corresponding columns in \mathbf{V} are linear combinations of \mathbf{x} 's which are insensitive to your data. In fact, you may then wish to zero these w_j 's, to reduce the number of free parameters in the fit. These matters are discussed more fully in Chapter 15.

Constructing an Orthonormal Basis

Suppose that you have N vectors in an M -dimensional vector space, with $N \leq M$. Then the N vectors span some subspace of the full vector space. Often you want to construct an orthonormal set of N vectors that span the same subspace. The textbook way to do this is by Gram-Schmidt orthogonalization, starting with one vector and then expanding the subspace one dimension at a time. Numerically, however, because of the build-up of roundoff errors, naive Gram-Schmidt orthogonalization is *terrible*.

The right way to construct an orthonormal basis for a subspace is by SVD: Form an $M \times N$ matrix \mathbf{A} whose N columns are your vectors. Run the matrix through `svdcmp`. The columns of the matrix \mathbf{U} (which in fact replaces \mathbf{A} on output from `svdcmp`) are your desired orthonormal basis vectors.

You might also want to check the output w_j 's for zero values. If any occur, then the spanned subspace was not, in fact, N dimensional; the columns of \mathbf{U} corresponding to zero w_j 's should be discarded from the orthonormal basis set.

(QR factorization, discussed in §2.10, also constructs an orthonormal basis, see [5].)

Approximation of Matrices

Note that equation (2.6.1) can be rewritten to express any matrix A_{ij} as a sum of outer products of columns of \mathbf{U} and rows of \mathbf{V}^T , with the "weighting factors" being the singular values w_j ,

$$A_{ij} = \sum_{k=1}^N w_k U_{ik} V_{jk} \quad (2.6.13)$$

Th
Ed
Ar
col
res
tex
sel
an
col
we
the
mc
col

```

    for (k=1;k<=n;k++) {
        a[i][k] /= scale;
        s += a[i][k]*a[i][k];
    }
    f=a[i][1];
    g = -SIGN(sqrt(s),f);
    h=f*g-s;
    a[i][1]=f-g;
    for (k=1;k<=n;k++) rv1[k]=a[i][k]/h;
    for (j=1;j<=m;j++) {
        for (s=0.0,k=1;k<=n;k++) s += a[j][k]*a[i][k];
        for (k=1;k<=n;k++) a[j][k] += s*rv1[k];
    }
    for (k=1;k<=n;k++) a[i][k] *= scale;
}
}
anorm=FMAX(anorm,(fabs(w[i])+fabs(rv1[i])));
}
for (i=n;i>=1;i--) {
    if (i < n) {
        if (g) {
            for (j=1;j<=n;j++)
                v[j][i]=(a[i][j]/a[i][1])/g;
            for (j=1;j<=n;j++) {
                for (s=0.0,k=1;k<=n;k++) s += a[i][k]*v[k][j];
                for (k=1;k<=n;k++) v[k][j] += s*v[k][i];
            }
        }
        for (j=1;j<=n;j++) v[i][j]=v[j][i]=0.0;
    }
    v[i][i]=1.0;
    g=rv1[i];
    l=i;
}
for (i=IMIN(m,n);i>=1;i--) {
    l=i+1;
    g=w[i];
    for (j=1;j<=n;j++) a[i][j]=0.0;
    if (g) {
        g=1.0/g;
        for (j=1;j<=n;j++) {
            for (s=0.0,k=1;k<=m;k++) s += a[k][i]*a[k][j];
            f=(s/a[i][i])*g;
            for (k=i;k<=m;k++) a[k][j] += f*a[k][i];
        }
        for (j=i;j<=m;j++) a[j][i] *= g;
    } else for (j=i;j<=m;j++) a[j][i]=0.0;
    ++a[i][i];
}
}
for (k=n;k>=1;k--) {
    for (its=1;its<=30;its++) {
        flag=1;
        for (l=k;l>=1;l--) {
            nm=l-1;
            if ((float)(fabs(rv1[l])+anorm) == anorm) {
                flag=0;
                break;
            }
        }
        if ((float)(fabs(w[nm])+anorm) == anorm) break;
    }
    if (flag) {
        c=0.0;
        s=1.0;
        for (i=1;i<=k;i++) {
            Accumulation of right-hand transformations.
            Double division to avoid possible underflow.
            Accumulation of left-hand transformations.
            Diagonalization of the bidiagonal form: Loop over
            singular values, and over allowed iterations.
            Test for splitting.
            Note that rv1[1] is always zero.
            Cancellation of rv1[1], if 1 > 1.

```

```

;
[j][k]*a[i][k];
*rv1[k];
;
;
of right-hand transformations.
n to avoid possible underflow.
[i][k]*v[k][j];
v[k][i];
of left-hand transformations.
]*a[k][j];
[i];
of the bidiagonal form: Loop over
lues, and over allowed iterations.
ig.
[1] is always zero.
norm) {
orm) break;

```

```

f=s*rv1[i];
rv1[i]=c*rv1[i];
if ((float)(fabs(f)+anorm) == anorm) break;
g=w[i];
h=pythag(f,g);
w[i]=h;
h=1.0/h;
c=g*h;
s = -f*h;
for (j=1;j<=n;j++) {
    y=a[j][nm];
    z=a[j][i];
    a[j][nm]=y*c+z*s;
    a[j][i]=z*c-y*s;
}
}
z=w[k];
if (l == k) {
    if (z < 0.0) {
        w[k] = -z;
        for (j=1;j<=n;j++) v[j][k] = -v[j][k];
    }
    break;
}
if (its == 30) nrerror("no convergence in 30 svdcmp iterations");
x=w[l];
nm=k-1;
y=w[nm];
g=rv1[nm];
h=rv1[k];
f=((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
g=pythag(f,1.0);
f=((x-z)*(x+z)+h*((y/(f+SIGN(g,f)))-h))/x;
c=s=1.0;
for (j=1;j<=nm;j++) {
    i=j+1;
    g=rv1[i];
    y=w[i];
    h=s*g;
    g=c*g;
    z=pythag(f,h);
    rv1[j]=z;
    c=f/z;
    s=h/z;
    f=x*c+g*s;
    g = g*c-x*s;
    h=y*s;
    y *= c;
    for (jj=1;jj<=n;jj++) {
        x=v[jj][j];
        z=v[jj][i];
        v[jj][j]=x*c+z*s;
        v[jj][i]=z*c-x*s;
    }
    z=pythag(f,h);
    w[j]=z;
    if (z) {
        z=1.0/z;
        c=f*z;
        s=h*z;

```

Convergence.
Singular value is made nonnegative.

Shift from bottom 2-by-2 minor.

Next QR transformation:

Rotation can be arbitrary if z = 0.

Th
Ed
An
col
res
tex
sel
an
co:
we
the
mc
co

```
        for (jj=1;jj<=m;jj++) {  
            y=a[jj][j];  
            z=a[jj][i];  
            a[jj][j]=y*c+z*s;  
            a[jj][i]=z*c-y*s;  
        }  
    }  
    rv1[1]=0.0;  
    rv1[k]=f;  
    w[k]=x;  
}  
}  
free_vector(rv1,1,n);  
}
```

```
#include <math.h>  
#include "nrutil.h"
```

```
float pythag(float a, float b)  
Computes  $(a^2 + b^2)^{1/2}$  without destructive underflow or overflow.  
{  
    float absa,absb;  
    absa=fabs(a);  
    absb=fabs(b);  
    if (absa > absb) return absa*sqrt(1.0+SQR(absb/absa));  
    else return (absb == 0.0 ? 0.0 : absb*sqrt(1.0+SQR(absa/absb)));  
}
```

(Double precision versions of svdcmp, svbksb, and pythag, named dsvdcmp, dsvbksb, and dpythag, are used by the routine ratlsq in §5.13. You can easily make the conversions, or else get the converted routines from the *Numerical Recipes* diskette.)

CITED REFERENCES AND FURTHER READING:

- Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press), §8.3 and Chapter 12.
- Lawson, C.L., and Hanson, R. 1974, *Solving Least Squares Problems* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 18.
- Forsythe, G.E., Malcolm, M.A., and Moler, C.B. 1977, *Computer Methods for Mathematical Computations* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 9. [1]
- Wilkinson, J.H., and Reinsch, C. 1971, *Linear Algebra*, vol. II of *Handbook for Automatic Computation* (New York: Springer-Verlag), Chapter I.10 by G.H. Golub and C. Reinsch. [2]
- Dongarra, J.J., et al. 1979, *LINPACK User's Guide* (Philadelphia: S.I.A.M.), Chapter 11. [3]
- Smith, B.T., et al. 1976, *Matrix Eigensystem Routines — EISPACK Guide*, 2nd ed., vol. 6 of *Lecture Notes in Computer Science* (New York: Springer-Verlag).
- Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), §6.7. [4]
- Golub, G.H., and Van Loan, C.F. 1989, *Matrix Computations*, 2nd ed. (Baltimore: Johns Hopkins University Press), §5.2.6. [5]