3.2. A SINGULAR-VALUE DECOMPOSITION ALGORITHM

It may seem odd that the first algorithm to be described in this work is designed to compute the singular-value decomposition (svd) of a matrix. Such computations are topics well to the back of most books on numerical linear algebra. However, it was the algorithm below which first interested the author in the capabilities of small computers. Moreover, while the svd is somewhat of a sledgehammer method for many nutshell problems, its versatility in finding the eigensolutions of a real symmetric matrix, in solving sets of simultaneous linear equations or in computing minimum-length solutions to least-squares problems makes it a valuable building block in programs used to tackle a variety of real problems.

This versatility has been exploited in a single small program suite of approximately 300 lines of BASIC code to carry out the above problems as well as to find inverses and generalised inverses of matrices and to solve nonlinear least-squares problems (Nash 1984b, 1985).

The mathematical problem of the svd has already been stated in §2.5. However, for computational purposes, an alternative viewpoint is more useful. This considers the possibility of finding an orthogonal matrix **V,** *n* by *n,* which transforms the real m by *n* matrix **A** into another real m by *n* matrix **B** whose columns are orthogonal. That is, it is desired to find **V** such that

$$\mathbf{B} = \mathbf{AV} = (\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_n) \tag{3.1}$$

where

$$\boldsymbol{b}_i^{\mathrm{T}} \boldsymbol{b}_j = S_i^2 \delta_{ij} \tag{3.2}$$

and

$$\mathbf{V}\mathbf{V}^{\mathrm{T}} = \mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{1}_n . \tag{3.3}$$

The Kronecker delta takes values

$$d_{\mathrm{ij}} = \begin{cases} 0 & \text{for } i \neq j \\ 1 & \text{for } i = j. \end{cases} \tag{3.4}$$

The quantities S_i may, as yet, be either positive or negative, since only their square is defined by equation (3.2). They will henceforth be taken arbitrarily as positive and will be called *singular values* of the matrix **A.** The vectors

$$\boldsymbol{u}_j = \boldsymbol{b}_j / S_j \tag{3.5}$$

which can be computed when none of the S_j is zero, are unit orthogonal vectors. Collecting these vectors into a real m by *n* matrix, and the singular values into a diagonal *n* by *n* matrix, it is possible to write

$$\mathbf{B} = \mathbf{US} \tag{3.6}$$

where

$$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{1}_n \tag{3.7}$$

is a unit matrix of order *n.*

In the case that some of the S_j are zero, equations (3.1) and (3.2) are still valid, but the columns of **U** corresponding to zero singular values must now be

constructed such that they are orthogonal to the columns of **U** computed via equation (3.5) and to each other. Thus equations (3.6) and (3.7) are also satisfied. An alternative approach is to set the columns of **U** corresponding to zero singular values to null vectors. By choosing the first k of the singular values to be the non-zero ones, which is always possible by simple permutations within the matrix **V,** this causes the matrix $\mathbf{U}^{\mathrm{T}}\mathbf{U}$ to be a unit matrix of order k augmented to order n with zeros. This will be written

$$\mathbf{U}^{\mathrm{T}}\mathbf{U} = \begin{pmatrix} \mathbf{1}_k & \\ & \mathbf{0}_{n-k} \end{pmatrix}. \tag{3.8}$$

While not part of the commonly used definition of the svd, it is useful to require the singular values to be sorted, so that

$$S_{11} \geq S_{22} \geq S_{33} \geq \ldots \geq S_{kk} \geq \ldots \geq S_{nn}.$$

This allows (2.53) to be recast as a summation

$$\mathbf{A} = \sum_{j=1}^{n} u_j S_{jj} v_j^{\mathrm{T}}. \tag{2.53a}$$

Partial sums of this series give a sequence of approximations

$$\tilde{\mathbf{A}}_1, \tilde{\mathbf{A}}_2, \ldots, \tilde{\mathbf{A}}_n.$$

where, obviously, the last member of the sequence

$$\tilde{\mathbf{A}}_n = \mathbf{A}$$

since it corresponds to a complete reconstruction of the svd. The rank-one matrices

$$u_j S_{jj} v^{\mathrm{T}}{}_j$$

can be referred to as singular planes, and the partial sums (in order of decreasing singular values) are partial svds (Nash and Shlien 1987).

A combination of (3.1) and (3.6) gives

$$\mathbf{AV} = \mathbf{US} \tag{3.9}$$

or, using (3.3), the orthogonality of **V,**

$$\mathbf{A} = \mathbf{USV}^{\mathrm{T}} \tag{2.53}$$

which expresses the svd of **A.**

The preceding discussion is conditional on the existence and computability of a suitable matrix **V.** The next section shows how this task may be accomplished.

3.3. ORTHOGONALISATION BY PLANE ROTATIONS

The matrix **V** sought to accomplish the orthogonalisation (3.1) will be built up as

a product of simpler matrices

$$\mathbf{V} = \prod_{k=1}^{z} \mathbf{V}^{(k)} \tag{3.10}$$

where z is some index not necessarily related to the dimensions m and n of $\mathbf{A}$, the matrix being decomposed. The matrices used in this product will be plane rotations. If $\mathbf{V}^{(k)}$ is a rotation of angle f in the ij plane, then all elements of $\mathbf{V}^{(k)}$ will be the same as those in a unit matrix of order n except for

$$\begin{aligned} V_{ii}^{(k)} &= \cos\phi = V_{jj}^{(k)} \\ -V_{ij}^{(k)} &= \sin\phi = V_{ji}^{(k)}. \end{aligned} \tag{3.11}$$

Thus $\mathbf{V}^{(k)}$ affects only two columns of any matrix it multiplies from the right. These columns will be labelled $\boldsymbol{x}$ and $\boldsymbol{y}$. Consider the effect of a single rotation involving these two columns

$$(\boldsymbol{x}, \boldsymbol{y}) \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} = (\boldsymbol{X}, \boldsymbol{Y}). \tag{3.12}$$

Thus we have

$$\begin{aligned} \boldsymbol{X} &= \boldsymbol{x}\cos f + \boldsymbol{y}\sin f \\ \boldsymbol{Y} &= -\boldsymbol{x}\sin f + \boldsymbol{y}\cos f. \end{aligned} \tag{3.13}$$

If the resulting vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ are to be orthogonal, then

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} = 0 = -(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y})\sin f \cos f + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y}(\cos^2 f - \sin^2 f). \tag{3.14}$$

There is a variety of choices for the angle f, or more correctly for the sine and cosine of this angle, which satisfy (3.14). Some of these are mentioned by Hestenes (1958), Chartres (1962) and Nash (1975). However, it is convenient if the rotation can order the columns of the orthogonalised matrix $\mathbf{B}$ by length, so that the singular values are in decreasing order of size and those which are zero (or infinitesimal) are found in the lower right-hand corner of the matrix $\mathbf{S}$ as in equation (3.8). Therefore, a further condition on the rotation is that

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} \geq 0. \tag{3.15}$$

For convenience, the columns of the product matrix

$$\mathbf{A}\prod_{j=1}^{k-1} \mathbf{V}^{(j)} \tag{3.16}$$

will be donated $\boldsymbol{a}_i$, $i = 1, 2, \ldots, n$. The progress of the orthogonalisation is then observable if a measure Z of the non-orthogonality is defined

$$Z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{a}_j)^2. \tag{3.17}$$

Since two columns orthogonalised in one rotation may be made non-orthogonal in subsequent rotations, it is essential that this measure be reduced at each rotation.

Because only two columns are involved in the kth rotation, we have

$$Z^{(k)} = Z^{(k-1)} + (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y})^2 - (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})^2. \tag{3.18}$$

But condition (3.14) implies

$$Z^{(k)} = Z^{(k-1)} - (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{y})^2 \tag{3.19}$$

so that the non-orthogonality is reduced at each rotation.

The specific formulae for the sine and cosine of the angle of rotation are (see e.g. Nash 1975) given in terms of the quantities

$$p = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{y} \tag{3.20}$$

$$q = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} \tag{3.21}$$

and

$$v = (4p^2 + q^2)^{1/2}. \tag{3.22}$$

They are

$$\cos f = [(v + q)/(2v)]^{1/2} \qquad \text{for } q \geq 0 \tag{3.23}$$

$$\sin f = p/(v \cos f) \tag{3.24}$$

$$\sin f = \mathrm{sgn}(p)[(v - q)/(2v)]^{1/2} \qquad \text{for } q < 0 \tag{3.25}$$

$$\cos f = p/(u \sin f) \tag{3.26}$$

where

$$\mathrm{sgn}(p) = \begin{cases} 1 & \text{for } p \geq 0 \\ -1 & \text{for } p < 0. \end{cases} \tag{3.27}$$

Note that having two forms for the calculation of the functions of the angle of rotation permits the subtraction of nearly equal numbers to be avoided. As the matrix nears orthogonality p will become small, so that q and v are bound to have nearly equal magnitudes.

In the first edition of this book, I chose to perform the computed rotation only when $q \geq r$, and to use

$$\sin(f) = 1 \qquad \cos(f) = 0 \tag{3.28}$$

when $q < 0$. This effects an interchange of the columns of the current matrix **A.** However, I now believe that it is more efficient to perform the rotations as defined in the code presented. The rotations (3.28) were used to force nearly null columns of the final working matrix to the right-hand side of the storage array. This will occur when the original matrix **A** suffers from linear dependencies between the columns (that is, is rank deficient). In such cases, the rightmost columns of the working matrix eventually reflect the lack of information in the data in directions corresponding to the null space of the matrix **A.** The current methods cannot do much about this lack of information, and it is not sensible to continue computations on these columns. In the current implementation of the method (Nash and Shlien 1987), we prefer to ignore columns at the right of the working matrix which become smaller than a

specified tolerance. This has a side effect of speeding the calculations significantly when rank deficient matrices are encountered.

3.4. A FINE POINT

Equations (3.15) and (3.19) cause the algorithm just described obviously to proceed *towards* both an orthogonalisation and an ordering of the columns of the resulting matrix $\mathbf{A}^{(z)}$. However the rotations must be arranged in some sequence to carry this task to completion. Furthermore, it remains to be shown that some sequences of rotations will not place the columns in disorder again. For suppose $\boldsymbol{a}_1$ is orthogonal to all other columns and larger than any of them individually. A sequential arrangement of the rotations to operate first on columns (1, 2), then (1, 3), (1, 4), . . . , (1, n), followed by (2, 3), . . . , (2, n), (3, 4), . . . , (($n-1$), n) will be called a cycle or sweep. Such a sweep applied to the matrix described can easily yield a new $\boldsymbol{a}_2$ for which

$$\boldsymbol{a}_2^{\mathrm{T}}\boldsymbol{a}_2 > \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{a}_1 \tag{3.29}$$

if, for instance, the original matrix has $\boldsymbol{a}_2 = \boldsymbol{a}_3$ and the norm of these vectors is greater than $2^{-\frac{1}{2}}$ times the norm of $\boldsymbol{a}_1$. Another sweep of rotations will put things right in this case by exchanging $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$. However, once two columns have achieved a separation related in a certain way to the non-orthogonality measure (3.17), it can be shown that no subsequent rotation can exchange them.

Suppose that the algorithm has proceeded so far that the non-orthogonality measure Z satisfies the inequality

$$Z < t^2 \tag{3.30}$$

where t is some positive tolerance. Then, for any subsequent rotation the parameter p, equation (3.21), must obey

$$p^2 < t^2. \tag{3.31}$$

Suppose that all adjacent columns are separated in size so that

$$\boldsymbol{a}_{k-1}^{\mathrm{T}}\boldsymbol{a}_{k-1} - \boldsymbol{a}_k^{\mathrm{T}}\boldsymbol{a}_k > t. \tag{3.32}$$

Then a rotation which changes $\boldsymbol{a}_k$ (but not $\boldsymbol{a}_{k-1}$) cannot change the ordering of the two columns. If $\boldsymbol{x} = \boldsymbol{a}_k$, then straightforward use of equations (3.23) and (3.24) or (3.25) and (3.26) gives

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} = (v - q)/2 \geq 0. \tag{3.33}$$

Using (3.31) and (3.22) in (3.33) gives

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} \leqslant [(4t^2 + q^2)^{1/2} - q]/2 \leqslant [(2t + q) - q]/2 \leqslant t. \tag{3.34}$$

Thus, once columns become sufficiently separated by size and the non-orthogonality sufficiently diminished, the column ordering is stable. When some columns are equal in norm but orthogonal, the above theorem can be applied to columns separated by size.

The general question of convergence in the case of equal singular values has been

investigated by T Hoy Booker (Booker 1985). The proof in exact arithmetic is incomplete. However, for a method such as the algorithm presented here, which uses tolerances for zero, Booker has shown that the cyclic sweeps must eventually terminate.

Algorithm 1. Singular-value decomposition

```
procedure NashSVD(nRow, nCo1: integer; {size of problem}
                  var W: wmatrix; {working matrix}
                  var Z: rvector); {squares of singular values}
(alg01.pas ==
      form a singular value decomposition of matrix A which is stored in the
      first nRow rows of working array W and the nCo1 columns of this array.
      The first nRow rows of W will become the product U * S of a
      conventional svd, where S is the diagonal matrix of singular values.
      The last nCo1 rows of W will be the matrix V of a conventional svd.
      On return, Z will contain the squares of the singular values. An
      extended form of this commentary can be displayed on the screen by
      removing the comment braces on the writeln statements below.

                  Copyright 1988 J. C. Nash
}
Var
   i, j, k, EstColRank, RotCount, SweepCount, slimit : integer;
   eps, e2, tol, vt, p, h2, x0, y0, q, r, c0, s0, c2, d1, d2 : real;
procedure rotate; (STEP 10 as a procedure}
(This rotation acts on both U and V, by storing V at the bottom of U}
begin (<< rotation )
   for i := 1 to nRow+nCol do
   begin
      D1 := W[i,j]; D2 := W[i,k];
     W[i,j] := D1*c0+D2*s0; W[i,k] := -D1*s0+D2*c0
   end; { rotation >>}
end; { rotate }
begin { procedure SVD }
{ -- remove the comment braces to allow message to be displayed --
   writeln('Nash Singular Value Decomposition (NashSVD).');
   writeln;
   writeln('The program takes as input a real matrix A.');
   writeln;
   writeln('Let U and V be orthogonal matrices, & S');
   writeln('a diagonal matrix, such that U" A V = S.');
   writeln('Then A = U S V" is the decomposition.');
   writeln('A is assumed to have more rows than columns. If it');
   writeln('does not, the svd of the transpose A" gives the svd');
   writeln('of A, since A" = V S U".');
   writeln;
   writeln('If A has nRow rows and nCo1 columns, then the matrix');
   writeln('is supplied to the program in a working array W large');
   writeln('enough to hold nRow+nCol rows and nCo1 columns.');
   writeln('Output comprises the elements of Z, which are the');
   writeln('squares of the elements of the vector S, together');
   writeln('with columns of W that correspond to non-zero elements');
```

Algorithm 1. Singular-value decomposition (cont.)

```
    writeln('of Z. The final array W contains the decomposition in a');
    writeln('special form, namely,');
    writeln;
    writeln(' ( U S ) ');
    writeln(' W = ( ) ');
    writeln(' ( V ) ');
    writeln;
    writeln('The matrices U and V are extracted from W, and S is');
    writeln('found from Z. However, the (U S) matrix and V matrix may');
    writeln('also be used directly in calculations, which we prefer');
    writeln('since fewer arithmetic operations are then needed.');
    writeln;
}
{STEP 0 Enter nRow, nCo1, the dimensions of the matrix to be decomposed.}
    writeln('alg01.pas--NashSVD');
    eps := Calceps; {Set eps, the machine precision.}
    slimit := nCo1 div 4; if slimit< then slimit := 6;
    {Set slimit, a limit on the number of sweeps allowed. A suggested
    limit is max([nCol/4], 6).}
    SweepCount := 0; {to count the number of sweeps carried out}
    e2 := 10.0*nRow*eps*eps;
    tol := eps*0.1;
    {Set the tolerances used to decide if the algorithm has converged.
        For further discussion of this, see the commentary under STEP 7.}
    EstColRank := nCo1; {current estimate of rank};
    {Set V matrix to the unit matrix of order nCo1.
    V is stored in rows (nRow+1) to (nRow+nCol) of array W.}
    for i := 1 to nCo1 do
    begin
        for j := 1 to nCo1 do
        W[nRow+i,j] := 0.0; W[nRow+i,i] := 1.0;
    end; {loop on i, and initialization of V matrix}
    {Main SVD calculations}
    repeat {until convergence is achieved or too many sweeps are carried out}
        RotCount := EstColRank*(EstColRank-1) div 2; {STEP 1 -- rotation counter}
        SweepCount := SweepCount+1;
        for j := 1 to EstColRank-1 do {STEP 2 -- main cyclic Jacobi sweep}
            begin {STEP 3}
            for k := j+l to EstColRank do
            begin {STEP 4}
                p := 0.0; q := 0.0; r := 0.0;
                for i := 1 to nRow do {STEP 5}
                begin
                    x0 := W[i,j]; y0 := W[i,k];
                    p := p+x0*y0; q := q+x0*x0; r := r+y0*y0;
                end;
                Z[j] := q; Z[k] := r;
                {Now come important convergence test considerations. First we
                will decide if rotation will exchange order of columns.}
                if q >= r then {STEP 6 -- check if the columns are ordered.}
                begin {STEP 7 Columns are ordered, so try convergence test.}
                    if (q<=e2*2[1]) or (abs(p)<= tol*q) then RotCount := RotCount-1
                    {There is no more work on this particular pair of columns in the
```

Algorithm 1. Singular-value decomposition (cont.)

```
                    current sweep. That is, we now go to STEP 11. The first
                    condition checks for very small column norms in BOTH columns, for
                    which no rotation makes sense. The second condition determines
                    if the inner product is small with respect to the larger of the
                    columns, which implies a very small rotation angle.)
                    else {columns are in order, but their inner product is not small}
                    begin {STEP 8}
                        p := p/q; r := 1-r/q; vt := sqrt(4*p*p + r*r);
                        c0 := sqrt(0.5*(1+r/vt)); s0 := p/(vt*c0);
                        rotate;
                    end
                end {columns in order with q>=r}
                else {columns out of order -- must rotate}
                begin {STEP 9}
                    {note: r > q, and cannot be zero since both are sums of squares for
                    the svd. In the case of a real symmetric matrix, this assumption
                    must be questioned.}
                    p := p/r; q := q/r-1; vt := sqrt(4*p*p + q*q);
                    s0 := sqrt(0.5*(1-q/vt));
                    if p<0 then s0 := -s0;
                    co := p/(vt*s0);
                    rotate; {The rotation is STEP 10.}
                end;
                {Both angle calculations have been set up so that large numbers do
                not occur in intermediate quantities. This is easy in the svd case,
                since quantities x2,y2 cannot be negative. An obvious scaling for
                the eigenvalue problem does not immediately suggest itself.)
            end; {loop on K -- end-loop is STEP 11}
        end; {loop on j -- end-loop is STEP 12}
        writeln('End of Sweep #', SweepCount,
                '- no. of rotations performed =', RotCount);
        {STEP 13 -- Set EstColRank to largest column index for which
            Z[column index] > (Z[1]*tol + tol*tol)
            Note how Pascal expresses this more precisely.}
        while (EstColRank >= 3) and (Z[EstColRank] <= Z[1]*tol + tol*tol)
                do EstColRank := EstColRank-1;
            {STEP 14 -- Goto STEP 1 to repeat sweep if rotations have been
                performed and the sweep limit has not been reached.}
    until (RotCount=0) or (SweepCount>slimit);
    {STEP 15 -- end SVD calculations}
    if (SweepCount > slimit) then writeln('**** SWEEP LIMIT EXCEEDED');
    if (SweepCount > slimit) then
    {Note: the decomposition may still be useful, even if the sweep
    limit has been reached.}
end; {alg01.pas == NashSVD}
```

3.5. AN ALTERNATIVE IMPLEMENTATION OF THE SINGULAR-VALUE DECOMPOSITION

One of the most time-consuming steps in algorithm 1 is the loop which comprises