Numer. Math. 14, 403–420 (1970)

Handbook Series Linear Algebra

Singular Value Decomposition and Least Squares Solutions*

Contributed by

G. H. GOLUB** and C. REINSCH

1. Theoretical Background

1.1. Introduction

Let A be a real $m \times n$ matrix with $m \geqq n$. It is well known (cf. [4]) that

$$A = U \Sigma V^T \tag{1}$$

where

$$U^T U = V^T V = V V^T = I_n \quad \text{and} \quad \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n).$$

The matrix U consists of n orthonormalized eigenvectors associated with the n largest eigenvalues of $A A^T$, and the matrix V consists of the orthonormalized eigenvectors of $A^T A$. The diagonal elements of Σ are the non-negative square roots of the eigenvalues of $A^T A$; they are called *singular values*. We shall assume that

$$\sigma_1 \geqq \sigma_2 \geqq \cdots \geqq \sigma_n \geqq 0.$$

Thus if $rank(A) = r$, $\sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_n = 0$. The decomposition (1) is called the *singular value decomposition* (SVD).

There are alternative representations to that given by (1). We may write

$$A = U_c \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T \quad \text{with} \quad U_c^T U_c = I_m$$

or

$$A = U_r \Sigma_r V_r^T \quad \text{with} \quad U_r^T U_r = V_r^T V_r = I_r \quad \text{and} \quad \Sigma_r = \operatorname{diag}(\sigma_1, \ldots, \sigma_r).$$

We use the form (1), however, since it is most useful for computational purposes.

If the matrix U is not needed, it would appear that one could apply the usual diagonalization algorithms to the symmetric matrix $A^T A$ which has to be formed explicitly. However, as in the case of linear least squares problems, the com-

* *Editor's note.* In this fascicle, prepublication of algorithms from the Linear Algebra series of the Handbook for Automatic Computation is continued. Algorithms are published in ALGOL 60 reference language as approved by the IFIP. Contributions in this series should be styled after the most recently published ones.

** The work of this author was in part supported by the National Science Foundation and Office of Naval Research.

putation of A^TA involves unnecessary numerical inaccuracy. For example, let

$$A = \begin{bmatrix} 1 & 1 \\ \beta & 0 \\ 0 & \beta \end{bmatrix},$$

then

$$A^TA = \begin{bmatrix} 1+\beta^2 & 1 \\ 1 & 1+\beta^2 \end{bmatrix}$$

so that

$$\sigma_1(A) = (2+\beta^2)^{\frac{1}{2}}, \quad \sigma_2(A) = |\beta|.$$

If $\beta^2 < \varepsilon_0$, the machine precision, the computed A^TA has the form $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, and the best one may obtain from diagonalization is $\tilde{\sigma}_1 = \sqrt{2}$, $\tilde{\sigma}_2 = 0$.

To compute the singular value decomposition of a given matrix A, Forsythe and Henrici [2], Hestenes [8], and Kogbetliantz [9] proposed methods based on plane rotations. Kublanovskaya [10] suggested a QR-type method. The program described below first uses Householder transformations to reduce A to bidiagonal form, and then the QR algorithm to find the singular values of the bidiagonal matrix. The two phases properly combined produce the singular value decomposition of A.

1.2. Reduction to Bidiagonal Form

It was shown in [6] how to construct two finite sequences of Householder transformations

$$P^{(k)} = I - 2x^{(k)}x^{(k)T} \quad (k = 1, 2, \ldots, n)$$

and

$$Q^{(k)} = I - 2y^{(k)}y^{(k)T} \quad (k = 1, 2, \ldots, n-2)$$

(where $x^{(k)T}x^{(k)} = y^{(k)T}y^{(k)} = 1$) such that

$$P^{(n)} \ldots P^{(1)} A Q^{(1)} \ldots Q^{(n-2)} = \begin{bmatrix} q_1 & e_2 & 0 & \cdots & 0 \\ & q_2 & e_3 & \ddots & \vdots \\ & & \ddots & \ddots & 0 \\ & & & \ddots & e_n \\ & & & & q_n \\ \hline & & 0 & & \end{bmatrix} \equiv J^{(0)},$$

$\left.\begin{bmatrix} \\ \\ \\ \\ \\ \hline \\ \end{bmatrix}\right\} (m-n) \times n$

an upper bidiagonal matrix. If we let $A^{(1)} = A$ and define

$$A^{(k+\frac{1}{2})} = P^{(k)}A^{(k)} \quad (k = 1, 2, \ldots, n)$$

$$A^{(k+1)} = A^{(k+\frac{1}{2})}Q^{(k)} \quad (k = 1, 2, \ldots, n-2)$$

then $P^{(k)}$ is determined such that

$$a_{ik}^{(k+\frac{1}{2})} = 0 \quad (i = k+1, \ldots, m)$$

and $Q^{(k)}$ such that

$$a_{kj}^{(k+1)} = 0 \quad (j = k+2, \ldots, n).$$

The singular values of $J^{(0)}$ are the same as those of A. Thus, if the singular value decomposition of

$$J^{(0)} = G \Sigma H^T$$

then

$$A = PG \Sigma H^T Q^T$$

so that $U = PG$, $V = QH$ with $P \equiv P^{(1)} \dots P^{(n)}$, $Q \equiv Q^{(1)} \dots Q^{(n-2)}$.

1.3. Singular Value Decomposition of the Bidiagonal Matrix

By a variant of the QR algorithm, the matrix $J^{(0)}$ is iteratively diagonalized so that

$$J^{(0)} \to J^{(1)} \to \dots \to \Sigma$$

where

$$J^{(i+1)} = S^{(i)T} J^{(i)} T^{(i)},$$

and $S^{(i)}$, $T^{(i)}$ are orthogonal. The matrices $T^{(i)}$ are chosen so that the sequence $M^{(i)} = J^{(i)T} J^{(i)}$ converges to a diagonal matrix while the matrices $S^{(i)}$ are chosen so that all $J^{(i)}$ are of the bidiagonal form. In [7], another technique for deriving $\{S^{(i)}\}$ and $\{T^{(i)}\}$ is given but this is equivalent to the method described below.

For notational convenience, we drop the suffix and use the notation

$$J \equiv J^{(i)}, \quad \bar{J} \equiv J^{(i+1)}, \quad S \equiv S^{(i)}, \quad T \equiv T^{(i)}, \quad M \equiv J^T J, \quad \bar{M} \equiv \bar{J}^T \bar{J}.$$

The transition $J \to \bar{J}$ is achieved by application of Givens rotations to J alternately from the right and the left. Thus

$$\bar{J} = \underbrace{S_n^T S_{(n-1)}^T \dots S_2^T}_{S^T} J \underbrace{T_2 T_3 \dots T_n}_{T} \tag{2}$$

where

$$S_k = \begin{bmatrix} 1 & 0 & & & & & \\ 0 & \ddots & & & & 0 & \\ & & 1 & & & & \\ & & & \cos\theta_k & -\sin\theta_k & & \\ & & & \sin\theta_k & \cos\theta_k & & \\ & & & & & 1 & \\ & 0 & & & & \ddots & 0 \\ & & & & & 0 & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ (k-1) \\ (k) \\ \\ \\ \\ \end{matrix}$$

and T_k is defined analogously to S_k with φ_k instead of θ_k.

Let the first angle, φ_2, be arbitrary while all the other angles are chosen so that $\bar{J}$ has the same form as J. Thus,

$$\begin{aligned} &T_2 \text{ annihilates nothing,} && \text{generates an entry } \{J\}_{21}, \\ &S_2^T \text{ annihilates } \{J\}_{21}, && \text{generates an entry } \{J\}_{13}, \\ &T_3 \text{ annihilates } \{J\}_{13}, && \text{generates an entry } \{J\}_{32}, \\ &\quad \vdots \end{aligned} \tag{3}$$

and finally

S_n^T annihilates $\{J\}_{n,n-1}$, and generates nothing.

(See Fig. 1.)

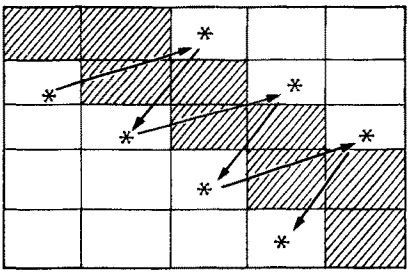

Fig. 1

This process is frequently described as "chasing". Since $\bar{J}=S^T J T$,

$$\bar{M}=\bar{J}^T\bar{J}=T^T M T$$

and $\bar{M}$ is a tri-diagonal matrix just as M is. We show that the first angle, φ_2, which is still undetermined, can be chosen so that the transition $M\to\bar{M}$ is a QR transformation with a given shift s.

The usual QR algorithm with shifts is described as follows:

$$\begin{aligned} M-sI&=T_s R_s\\ R_s T_s+sI&=\bar{M}_s \end{aligned} \tag{4}$$

where $T_s^T T_s=I$ and R_s is an upper triangular matrix. Thus $\bar{M}_s=T_s^T M T_s$. It has been shown by Francis [5] that it is not necessary to compute (4) explicitly but it is possible to perform the shift implicitly. Let T be for the moment an arbitrary matrix such that

$$\{T_s\}_{k,1}=\{T\}_{k,1}\quad (k=1,2,\ldots,n),$$

(i.e., the elements of the first column of T_s are equal to the first column of T) and

$$T^T T=I.$$

Then we have the following theorem (Francis): If

i) $\bar{M}=T^T M T$,

ii) $\bar{M}$ is a tri-diagonal matrix,

iii) the sub-diagonal elements of M are non-zero,

it follows that $\bar{M}=D\bar{M}_s D$ where D is a diagonal matrix whose diagonal elements are ± 1.

Thus choosing T_2 in (3) such that its first column is proportional to that of $M-sI$, the same is true for the first column of the product $T=T_2 T_3\ldots T_n$ which therefore is identical to that of T_s. Hence, if the sub-diagonal of M does not contain any non-zero entry the conditions of the Francis theorem are fulfilled and T is therefore identical to T_s (up to a scaling of column ± 1). Thus the transition (2) is equivalent to the QR transformation of $J^T J$ with a given shift s.

The shift parameter s is determined by an eigenvalue of the lower 2×2 minor of M. Wilkinson [13] has shown that for this choice of s, the method converges globally and almost always cubically.

1.4. Test for Convergence

If $|e_n| \leqq \delta$, a prescribed tolerance, then $|q_n|$ is accepted as a singular value, and the order of the matrix is dropped by one. If, however, $|e_k| \leqq \delta$ for $k \neq n$, the matrix breaks into two, and the singular values of each block may be computed independently.

If $q_k = 0$, then at least one singular value must be equal to zero. In the absence of roundoff error, the matrix will break if a shift of zero is performed. Now, suppose at some stage

$$|q_k| \leqq \delta.$$

At this stage an extra sequence of Givens rotations is applied from the left to J involving rows $(k, k+1)$, $(k, k+2)$, ..., (k, n) so that

$e_{k+1} \equiv \{J\}_{k,k+1}$ is annihilated, but $\{J\}_{k,k+2}$, $\{J\}_{k+1,k}$ are generated,
$\{J\}_{k,k+2}$ is annihilated, but $\{J\}_{k,k+3}$, $\{J\}_{k+2,k}$ are generated,
$\vdots$
and finally $\{J\}_{k,n}$ is annihilated, and $\{J\}_{n,k}$ is generated.

The matrix obtained thusly has the form

$$\bar{J} = \begin{bmatrix} & & & & (k) & & \\ q_1 & e_2 & & & & 0 & \\ & \ddots & \ddots & & & & \\ & & \ddots & e_k & & & \\ & & & \bar{q}_k & 0 & & \\ & & & \delta_{k+1} & \bar{q}_{k+1} & \bar{e}_{k+2} & \\ & & & \vdots & & \ddots & \ddots \\ & 0 & & \vdots & & & \bar{e}_n \\ & & & \delta_n & & & \bar{q}_n \end{bmatrix} (k).$$

Note that by orthogonality

$$\bar{q}_k^2 + \delta_{k+1}^2 + \cdots + \delta_n^2 = q_k^2 \leqq \delta^2.$$

Thus choosing $\delta = \|J^{(0)}\|_\infty \varepsilon_0$ (ε_0, the machine precision) ensures that all δ_k are less in magnitude than $\varepsilon_0 \|J^{(0)}\|_\infty$. Elements of $\bar{J}$ not greater than this are neglected. Hence $\bar{J}$ breaks up into two parts which may be treated independently.

2. Applicability

There are a large number of applications of the singular value decomposition; an extensive list is given in [7]. Some of these are as follows:

2.1. Pseudoinverse (Procedure SVD)

Let A be a real $m \times n$ matrix. An $n \times m$ matrix X is said to be the pseudoinverse of A if X satisfies the following four properties:

i) $AXA = A$,
ii) $XAX = X$,
iii) $(AX)^T = AX$,
iv) $(XA)^T = XA$.

The unique solution is denoted by A^+. It is easy to verify that if $A = U\Sigma V^T$, then $A^+ = V\Sigma^+ U^T$ where $\Sigma^+ = \text{diag}(\sigma_i^+)$ and

$$\sigma_i^+ = \begin{cases} 1/\sigma_i & \text{for} \quad \sigma_i > 0 \\ 0 & \text{for} \quad \sigma_i = 0. \end{cases}$$

Thus the pseudoinverse may easily be computed from the output provided by the procedure *SVD*.

2.2. Solution of Homogeneous Equations (Procedure SVD or Procedure Minfit)

Let A be a matrix of rank r, and suppose we wish to solve

$$A x_i = \theta \quad \text{for} \quad i = r+1, \ldots, n$$

where θ denotes the null vector.

Let

$$U = [u_1, u_2, \ldots, u_n] \quad \text{and} \quad V = [v_1, v_2, \ldots, v_n].$$

Then since $A v_i = \sigma_i u_i$ $(i = 1, 2, \ldots n)$,

$$A v_i = \theta \quad \text{for} \quad i = r+1, \ldots, n$$

and $x_i = v_i$.

Here the procedure *SVD* or the procedure *Minfit* with $p = 0$ may be used for determining the solution. If the rank of A is known, then a modification of the algorithm of Businger and Golub [1] may be used.

2.3. Solutions of Minimal Length (Procedure Minfit)

Let b_1 be a given vector. Suppose we wish to determine a vector x so that

$$\|b_1 - A x\|_2 = \min \tag{5}$$

If the rank of A is less than n then there is no unique solution. Thus we require amongst all x which satisfy (5) that

$$\|\hat{x}\|_2 = \min$$

and this solution is unique. It is easy to verify that

$$\hat{x} = A^+ b_1 = V\Sigma^+ U^T b_1 \equiv V\Sigma^+ c_1.$$

The procedure *Minfit* with $p > 0$ will yield $V, \Sigma, c_1, \ldots, c_p$. Thus the user is able to determine which singular values are to be declared as zero.

2.4. A Generalization of the Least Squares Problem (Procedure SVD)

Let A be a real $m \times n$ matrix of rank n and let b be a given vector. We wish to construct a vector x such that

$$(A + \Delta A)\, x = b + \Delta b$$

and

$$\text{trace}(\Delta A^T \Delta A) + K^2 \Delta b^T \Delta b = \min.$$

Here $K>0$ is a given weight and the standard problem is obtained for $K\to 0$. Introducing the augmented matrices $\bar{A}=(A, Kb)$ and $\Delta\bar{A}=(\Delta A, K\Delta b)$ and the vector

$$\bar{x}=\begin{pmatrix} x \\ -1/K \end{pmatrix},$$

we have to minimize $\text{trace}(\Delta\bar{A}^T\Delta\bar{A})$ under the constraint $(\bar{A}+\Delta\bar{A})\bar{x}=0$. For fixed $\bar{x}$ the minimum is attained for $\Delta\bar{A}=-\bar{A}\bar{x}\bar{x}^T/\bar{x}^T\bar{x}$ and it has the value $\bar{x}^T\bar{A}^T\bar{A}\bar{x}/\bar{x}^T\bar{x}$. Minimizing with respect to $\bar{x}$ amounts to the computation of the smallest singular value of the matrix $\bar{A}$ and $\bar{x}$ is the corresponding column of the matrix $\bar{V}$ in the decomposition (1) with proper normalization [3].

3. Formal Parameter List

3.1. *Input to Procedure SVD*

m	number of rows of A, $m\geqq n$.
n	number of columns of A.
withu	**true** if U is desired, **false** otherwise.
withv	**true** if V is desired, **false** otherwise.
eps	a constant used in the test for convergence (see Section 5, (iii)); should not be smaller than the machine precision ε_0, i.e., the smallest number for which $1+\varepsilon_0>1$ in computer arithmetic.
tol	a machine dependent constant which should be set equal to β/ε_0 where β is the smallest positive number representable in the computer, see [11].
$a[1:m, 1:n]$	represents the matrix A to be decomposed.

Output of procedure *SVD*.

$q[1:n]$	a vector holding the singular values of A; they are non-negative but not necessarily ordered in decreasing sequence.
$u[1:m, 1:n]$	represents the matrix U with orthonormalized columns (if *withu* is **true**, otherwise u is used as a working storage).
$v[1:n, 1:n]$	represents the orthogonal matrix V (if *withv* is **true**, otherwise v is not used).

3.2. *Input to Procedure Minfit*

m	number of rows of A.
n	number of columns of A.
p	number of columns of B, $p\geqq 0$.
eps	same as for procedure *SVD*.
tol	same as for procedure *SVD*.
$ab[1:max(m,n), 1:n+p]$	$ab[i, j]$ represents $a_{i,j}$, $1\leqq i\leqq m$, $1\leqq j\leqq n$, $ab[i, n+j]$ represents $b_{i,j}$, $1\leqq i\leqq m$, $1\leqq j\leqq p$.

Output of procedure *Minfit*.

$ab[1:max(m,n), 1:n+p]$	$ab[i, j]$ represents $v_{i,j}$, $1 \leqq i \leqq n$, $1 \leqq j \leqq n$, $ab[i, n+j]$ represents $c_{i,j}$, $1 \leqq i \leqq max(m,n)$, $1 \leqq j \leqq p$ viz. $C = U_c^T B$.
$q[1:n]$	same as for procedure *SVD*.

4. ALGOL Programs

```
procedure SVD (m, n, withu, withv, eps, tol) data: (a) result: (q, u, v);
    value m, n, withu, withv, eps, tol;
    integer m, n;
    Boolean withu, withv;
    real eps, tol;
    array a, q, u, v;
comment Computation of the singular values and complete orthogonal decom-
        position of a real rectangular matrix A,
                A = U diag(q) V^T,    U^T U = V^T V = I,
        where the arrays a[1:m, 1:n], u[1:m, 1:n], v[1:n, 1:n], q[1:n] re-
        present A, U, V, q respectively. The actual parameters corresponding
        to a, u, v may all be identical unless withu = withv = true. In this
        case, the actual parameters corresponding to u and v must differ.
        m ≧ n is assumed;
begin
    integer i, j, k, l, l1;
    real c, f, g, h, s, x, y, z;
    array e[1:n];
    for i := 1 step 1 until m do
    for j := 1 step 1 until n do u[i, j] := a[i, j];
comment Householder's reduction to bidiagonal form;
    g := x := 0;
    for i := 1 step 1 until n do
    begin
        e[i] := g; s := 0; l := i + 1;
        for j := i step 1 until m do s := s + u[j, i]↑2;
        if s < tol then g := 0 else
        begin
            f := u[i, i]; g := if f < 0 then sqrt(s) else -sqrt(s);
            h := f × g - s; u[i, i] := f - g;
            for j := l step 1 until n do
            begin
                s := 0;
                for k := i step 1 until m do s := s + u[k, i] × u[k, j];
                f := s/h;
                for k := i step 1 until m do u[k, j] := u[k, j] + f × u[k, i]
            end j
        end s;
```

```
        q[i] := g; s := 0;
        for j := l step 1 until n do s := s + u[i, j]↑2;
        if s < tol then g := 0 else
        begin
            f := u[i, i+1]; g := if f < 0 then sqrt(s) else -sqrt(s);
            h := f × g - s; u[i, i+1] := f - g;
            for j := l step 1 until n do e[j] := u[i, j]/h;
            for j := l step 1 until m do
            begin
                s := 0;
                for k := l step 1 until n do s := s + u[j, k] × u[i, k];
                for k := l step 1 until n do u[j, k] := u[j, k] + s × e[k]
            end j
        end s;
        y := abs(q[i]) + abs(e[i]); if y > x then x := y
    end i;

comment accumulation of right-hand transformations;
    if withv then for i := n step -1 until 1 do
    begin
        if g ≠ 0 then
        begin
            h := u[i, i+1] × g;
            for j := l step 1 until n do v[j, i] := u[i, j]/h;
            for j := l step 1 until n do
            begin
                s := 0;
                for k := l step 1 until n do s := s + u[i, k] × v[k, j];
                for k := l step 1 until n do v[k, j] := v[k, j] + s × v[k, i]
            end j
        end g;
        for j := l step 1 until n do v[i, j] := v[j, i] := 0;
        v[i, i] := 1; g := e[i]; l := i
    end i;

comment accumulation of left-hand transformations;
    if withu then for i := n step -1 until 1 do
    begin
        l := i + 1; g := q[i];
        for j := l step 1 until n do u[i, j] := 0;
        if g ≠ 0 then
        begin
            h := u[i, i] × g;
            for j := l step 1 until n do
```

```
            begin
                s := 0;
                for k := l step 1 until m do s := s + u[k, i] × u[k, j];
                f := s/h;
                for k := i step 1 until m do u[k, j] := u[k, j] + f × u[k, i]
            end j;
            for j := i step 1 until m do u[j, i] := u[j, i]/g
        end g
        else for j := i step 1 until m do u[j, i] := 0;
        u[i, i] := u[i, i] + 1
    end i;

comment diagonalization of the bidiagonal form;
    eps := eps × x;
    for k := n step −1 until 1 do
    begin
        test f splitting:
            for l := k step −1 until 1 do
            begin
                if abs(e[l]) ≦ eps then goto test f convergence;
                if abs(q[l − 1]) ≦ eps then goto cancellation
            end l;

comment cancellation of e[l] if l > 1;
    cancellation:
        c := 0; s := 1; l1 := l − 1;
        for i := l step 1 until k do
        begin
            f := s × e[i]; e[i] := c × e[i];
            if abs(f) ≦ eps then goto test f convergence;
            g := q[i]; h := q[i] := sqrt(f × f + g × g); c := g/h; s := −f/h;
            if withu then for j := 1 step 1 until m do
            begin
                y := u[j, l1]; z := u[j, i];
                u[j, l1] := y × c + z × s; u[j, i] := −y × s + z × c
            end j
        end i;
    test f convergence:
        z := q[k]; if l = k then goto convergence;

comment shift from bottom 2 × 2 minor;
        x := q[l]; y := q[k − 1]; g := e[k − 1]; h := e[k];
        f := ((y − z) × (y + z) + (g − h) × (g + h))/(2 × h × y); g := sqrt(f × f + 1);
        f := ((x − z) × (x + z) + h × (y/(if f < 0 then f − g else f + g) − h))/x;

comment next Q R transformation;
        c := s := 1;
        for i := l + 1 step 1 until k do
```

```
        begin
            g := e[i]; y := q[i]; h := s×g; g := c×g;
            e[i−1] := z := sqrt(f×f+h×h); c := f/z; s := h/z;
            f := x×c+g×s; g := −x×s+g×c; h := y×s; y := y×c;
            if withv then for j := 1 step 1 until n do
            begin
                x := v[j, i−1]; z := v[j, i];
                v[j, i−1] := x×c+z×s; v[j, i] := −x×s+z×c
            end j;
            q[i−1] := z := sqrt(f×f+h×h); c := f/z; s := h/z;
            f := c×g+s×y; x := −s×g+c×y;
            if withu then for j := 1 step 1 until m do
            begin
                y := u[j, i−1]; z := u[j, i];
                u[j, i−1] := y×c+z×s; u[j, i] := −y×s+z×c
            end j
        end i;
        e[l] := 0; e[k] := f; q[k] := x; goto test f splitting;
    convergence:
        if z < 0 then
        begin comment q[k] is made non-negative;
            q[k] := −z;
            if withv then for j := 1 step 1 until n do v[j, k] := −v[j, k]
        end z
    end k
end SVD;

procedure Minfit (m, n, p, eps, tol) trans: (ab) result: (q);
    value m, n, p, eps, tol;
    integer m, n, p;
    real eps, tol;
    array ab, q;
```

comment Computation of the matrices diag(q), V, and C such that for given real $m \times n$ matrix A and $m \times p$ matrix B

$$U_c^T A V = \text{diag}(q) \text{ and } U_c^T B = C \text{ with orthogonal matrices } U_c \text{ and } V.$$

The singular values and the matrices V and C may be used to determine $\bar{X}$ minimizing (1) $\|AX-B\|_F$ and (2) $\|X\|_F$ with the solution

$$\bar{X} = V \times \text{Pseudo-inverse of } \text{diag}(q) \times C.$$

The procedure can also be used to determine the complete solution of an underdetermined linear system, i.e., rank$(A) = m < n$.

The array $q[1:n]$ represents the matrix diag(q). A and B together are to be given as the first m rows of the array $ab[1:max(m,n), 1:n+p]$. V is returned in the first n rows and columns of ab while C is returned in the last p columns of ab (if $p > 0$);

```
begin
    integer i, j, k, l, l1, n1, np;
    real c, f, g, h, s, x, y, z;
    array e[1:n];
comment Householder's reduction to bidiagonal form;
    g := x := 0; np := n + p;
    for i := 1 step 1 until n do
    begin
        e[i] := g; s := 0; l := i + 1;
        for j := i step 1 until m do s := s + ab[j, i]↑2;
        if s < tol then g := 0 else
        begin
            f := ab[i, i]; g := if f < 0 then sqrt(s) else -sqrt(s);
            h := f × g - s; ab[i, i] := f - g;
            for j := l step 1 until np do
            begin
                s := 0;
                for k := i step 1 until m do s := s + ab[k, i] × ab[k, j];
                f := s/h;
                for k := i step 1 until m do ab[k, j] := ab[k, j] + f × ab[k, i]
            end j
        end s;
        q[i] := g; s := 0;
        if i ≦ m then for j := l step 1 until n do s := s + ab[i, j]↑2;
        if s < tol then g := 0 else
        begin
            f := ab[i, i + 1]; g := if f < 0 then sqrt(s) else -sqrt(s);
            h := f × g - s; ab[i, i + 1] := f - g;
            for j := l step 1 until n do e[j] := ab[i, j]/h;
            for j := l step 1 until m do
            begin
                s := 0;
                for k := l step 1 until n do s := s + ab[j, k] × ab[i, k];
                for k := l step 1 until n do ab[j, k] := ab[j, k] + s × e[k]
            end j
        end s;
        y := abs(q[i]) + abs(e[i]); if y > x then x := y
    end i;
comment accumulation of right-hand transformations;
    for i := n step -1 until 1 do
    begin
        if g ≠ 0 then
        begin
            h := ab[i, i + 1] × g;
            for j := l step 1 until n do ab[j, i] := ab[i, j]/h;
            for j := l step 1 until n do
```

```
            begin
                s := 0;
                for k := l step 1 until n do s := s + ab[i, k] × ab[k, j];
                for k := l step 1 until n do ab[k, j] := ab[k, j] + s × ab[k, i]
            end j
        end g;
        for j := l step 1 until n do ab[i, j] := ab[j, i] := 0;
        ab[i, i] := 1; g := e[i]; l := i
    end i;
    eps := eps × x; n1 := n + 1;
    for i := m + 1 step 1 until n do
    for j := n1 step 1 until np do ab[i, j] := 0;

comment diagonalization of the bidiagonal form;
    for k := n step −1 until 1 do
    begin
        test f splitting:
            for l := k step −1 until 1 do
            begin
                if abs(e[l]) ≦ eps then goto test f convergence;
                if abs(q[l − 1]) ≦ eps then goto cancellation
            end l;

comment cancellation of e[l] if l > 1;
        cancellation:
            c := 0; s := 1; l1 := l − 1;
            for i := l step 1 until k do
            begin
                f := s × e[i]; e[i] := c × e[i];
                if abs(f) ≦ eps then goto test f convergence;
                g := q[i]; q[i] := h := sqrt(f × f + g × g); c := g/h; s := −f/h;
                for j := n1 step 1 until np do
                begin
                    y := ab[l1, j]; z := ab[i, j];
                    ab[l1, j] := c × y + s × z; ab[i, j] := −s × y + c × z
                end j
            end i;
        test f convergence:
            z := q[k]; if l = k then goto convergence;

comment shift from bottom 2 × 2 minor;
            x := q[l]; y := q[k − 1]; g := e[k − 1]; h := e[k];
            f := ((y − z) × (y + z) + (g − h) × (g + h))/(2 × h × y); g := sqrt(f × f + 1);
            f := ((x − z) × (x + z) + h × (y/(if f < 0 then f − g else f + g) − h))/x;

comment next Q R transformation;
            c := s := 1;
            for i := l + 1 step 1 until k do
```

```
        begin
            g := e[i]; y := q[i]; h := s×g; g := c×g;
            e[i−1] := z := sqrt(f×f+h×h); c := f/z; s := h/z;
            f := x×c+g×s; g := −x×s+g×c; h := y×s; y := y×c;
            for j := 1 step 1 until n do
            begin
                x := ab[j, i−1]; z := ab[j, i];
                ab[j, i−1] := x×c+z×s; ab[j, i] := −x×s+z×c
            end j;
            q[i−1] := z := sqrt(f×f+h×h); c := f/z; s := h/z;
            f := c×g+s×y; x := −s×g+c×y;
            for j := n1 step 1 until np do
            begin
                y := ab[i−1, j]; z := ab[i, j];
                ab[i−1, j] := c×y+s×z; ab[i, j] := −s×y+c×z
            end j
        end i;
        e[l] := 0; e[k] := f; q[k] := x; goto test f splitting;
    convergence:
        if z<0 then
        begin comment q[k] is made non-negative;
            q[k] := −z;
            for j := 1 step 1 until n do ab[j, k] := −ab[j, k]
        end z
    end k
end Minfit;
```

5. Organizational and Notational Details

(i) The matrix U consists of the first n columns of an orthogonal matrix U_c. The following modification of the procedure *SVD* would produce U_c instead of U: After

comment accumulation of left-hand transformations;
insert a statement

```
if withu then for i := n+1 step 1 until m do
begin
    for j := n+1 step 1 until m do u[i, j] := 0;
    u[i, i] := 1
end i;
```

Moreover, replace n by m in the fourth and eighth line after that, i.e., write twice **for** $j := l$ **step** 1 **until** m **do**.

(ii) $m \geqq n$ is assumed for procedure *SVD*. This is no restriction; if $m < n$, store A^T, i.e., use an array $at[1:n, 1:m]$ where $at[i, j]$ represents $a_{j,i}$ and call *SVD* (*n, m, withv, withu, eps, tol, at, q, v, u*) producing the $m \times m$ matrix U and the $n \times m$ matrix V. There is no restriction on the values of m and n for the procedure *Minfit*.

(iii) In the iterative part of the procedures an element of $J^{(i)}$ is considered to be negligible and is consequently replaced by zero if it is not larger in magnitude than εx where ε is the given tolerance and

$$x = \max_{1 \leqq i \leqq n} (|q_i| + |e_i|).$$

The largest singular value σ_1 is bounded by $x/\sqrt{2} \leqq \sigma_1 \leqq x\sqrt{2}$.

(iv) A program organization was chosen which allows us to save storage locations. To this end the actual parameters corresponding to a and u may be identical. In this event the original information stored in a is overwritten by information on the reduction. This, in turn, is overwritten by u if the latter is desired. Likewise, the actual parameters corresponding to a and v may agree. Then v is stored in the upper part of a if it is desired, otherwise a is not changed. Finally, all three parameters a, u, and v may be identical unless *withu* = *withv* = **true**.

This special feature, however, increases the number of multiplications needed to form U roughly by a factor m/n.

(v) Shifts are evaluated in a way as to reduce the danger of overflow or underflow of exponents.

(vi) The singular values as delivered in the array q are not necessarily ordered. Any sorting of them should be accompanied by the corresponding sorting of the columns of U and V, and of the rows of C.

(vii) The formal parameter list may be completed by the addition of a limit for the number of iterations to be performed, and by the addition of a failure exit to be taken if no convergence is reached after the specified number of iterations (e.g., 30 per singular value).

6. Numerical Properties

The stability of the Householder transformations has been demonstrated by Wilkinson [12]. In addition, he has shown that in the absence of roundoff the QR algorithm has global convergence and asymptotically is almost always cubically convergent.

The numerical experiments indicate that the average number of complete QR iterations on the bidiagonal matrix is usually less than two per singular value. Extra consideration must be given to the implicit shift technique which fails for a split matrix. The difficulties arise when there are small q_k's or e_k's. Using the techniques of Section 1.4, there cannot be numerical instability since stable orthogonal transformations are used but under special circumstances there may be a slowdown in the rate of convergence.

7. Test Results

Tests were carried out on the UNIVAC 1108 Computer of the Andrew R. Jennings Computing Center of Case Western Reserve University. Floating point numbers are represented by a normalized 27 bit mantissa and a 7 bit exponent to the radix 2, whence $eps = 1.5_{10} - 8$, $tol = {}_{10} - 31$. In the following, computed values are marked by a tilde and $m(A)$ denotes $\max|a_{i,j}|$.

First example:

$$A=\begin{bmatrix} 22 & 10 & 2 & 3 & 7 \\ 14 & 7 & 10 & 0 & 8 \\ -1 & 13 & -1 & -11 & 3 \\ -3 & -2 & 13 & -2 & 4 \\ 9 & 8 & 1 & -2 & 4 \\ 9 & 1 & -7 & 5 & -1 \\ 2 & -6 & 6 & 5 & 1 \\ 4 & 5 & 0 & -2 & 2 \end{bmatrix}, \quad B=\begin{bmatrix} -1 & 1 & 0 \\ 2 & -1 & 1 \\ 1 & 10 & 11 \\ 4 & 0 & 4 \\ 0 & -6 & -6 \\ -3 & 6 & 3 \\ 1 & 11 & 12 \\ 0 & -5 & -5 \end{bmatrix},$$

$$\sigma_1=\sqrt{1248}, \quad \sigma_2=20, \quad \sigma_3=\sqrt{384}, \quad \sigma_4=\sigma_5=0.$$

The homogeneous system $A\,x=\theta$ has two linearly independent solutions. Six QR transformations were necessary to drop all off-diagonal elements below the internal tolerance $46.4_{10}-8$. Table 1 gives the singular values in the sequence as computed by procedures *SVD* and *Minfit*. The accuracy of the achieved decomposition is characterized by

$$m(A-\tilde{U}\tilde{\Sigma}\tilde{V}^T)=238_{10}-8, \quad m(\tilde{U}^T\tilde{U}-I)=8.1_{10}-8, \quad m(\tilde{V}^T\tilde{V}-I)=3.3_{10}-8.$$

Because two singular values are equal to zero, the procedures *SVD* and *Minfit* may lead to other orderings of the singular values for this matrix when other tolerances are used.

Table 1

$\tilde{\sigma}_k$	$\sigma_k-\tilde{\sigma}_k$
$0.96_{10}-7$	-9.6
19.595916	191
19.999999	143
$1.97_{10}-7$	-19.7
35.327038	518

$\times 10^{-8}$

The computed solutions of the homogeneous system are given by the first and fourth columns of the matrix $\tilde{V}$ (Table 2).

Table 2

$\tilde{v}_1$	$\tilde{v}_4$	$v_1-\tilde{v}_1$	$v_4-\tilde{v}_4$
−0.4190 9545	0	−1.5	0 (Def.)
0.4405 0912	0.4185 4806	1.7	0.6
−0.0520 0457	0.3487 9006	1.2	−1.3
0.6760 5915	0.2441 5305	1.0	0.3
0.4129 7730	−0.8022 1713	1.3	−0.8

$\times 10^{-8}$

Procedure *Minfit* was used to compute the solutions of the minimization problem of Section 2.3 corresponding to the three right-hand sides as given by the columns of the matrix B. Table 3 lists the exact solutions and the results obtained when the first and fourth values in Table 1 are replaced by zero.

Table 3

x_1	x_2	x_3	$\tilde{x}_1$	$\tilde{x}_2$	$\tilde{x}_3$
$-1/12$	0	$-1/12$	$-0.0833\ 3333$	$0.17_{10}-8$	$-0.0833\ 3333$
0	0	0	$-0.58_{10}-8$	$-1.09_{10}-8$	$-1.11_{10}-8$
1/4	0	1/4	$0.2500\ 0002$	$1.55_{10}-8$	$0.2500\ 0003$
$-1/12$	0	$-1/12$	$-0.0833\ 3332$	$0.74_{10}-8$	$-0.0833\ 3332$
1/12	0	1/12	$0.0833\ 3334$	$0.33_{10}-8$	$0.0833\ 3334$
	Residual				
0	$8\sqrt{5}$	$8\sqrt{5}$			

A second example is the 20×21 matrix with entries

$$a_{i,j}=\begin{cases}0 & \text{if } i>j\\ 21-i & \text{if } i=j\\ -1 & \text{if } i<j\end{cases}\qquad \begin{matrix}1\leqq i\leqq 20\\ 1\leqq j\leqq 21\end{matrix}$$

which has orthogonal rows and singular values $\sigma_{21-k}=\sqrt{k(k+1)}$, $k=0,\ldots,20$. Theoretically, the Householder reduction should produce a matrix $J^{(0)}$ with diagonal $-20, 0, \ldots, 0$ and super-diagonal $-\sqrt{20}, \sigma_2, \ldots, \sigma_{20}$. Under the influence of rounding errors a totally different matrix results. However, within working accuracy its singular values agree with those of the original matrix. Convergence is reached after 32 QR transformations and the $\tilde{\sigma}_k$, $k=1,\ldots,20$ are correct within several units in the last digit, $\tilde{\sigma}_{21}=1.61_{10}-11$.

A third example is obtained if the diagonal of the foregoing example is changed to

$$a_{i,i}=1,\qquad 1\leqq i\leqq 20.$$

This matrix has a cluster of singular values, σ_{10} to σ_{19} lying between 1.5 and 1.6, $\sigma_{20}=\sqrt{2}$, $\sigma_{21}=0$. Clusters, in general, have a tendency to reduce the number of required iterations; in this example, 26 iterations were necessary for convergence. $\tilde{\sigma}_{21}=1.49_{10}-8$ is found in eighteenth position and the corresponding column of $\tilde{V}$ differs from the unique solution of the homogeneous system by less than $3.4_{10}-8$ in any component.

A second test was made by Dr. Peter Businger on the CDC 6600.

A third test was performed on the IBM 360/67 at Stanford University. The example used was the 30×30 matrix with entries

$$a_{ij}=\begin{cases}0 & \text{if } i>j\\ 1 & \text{if } i=j\\ -1 & \text{if } i<j.\end{cases}$$

The computed singular values are given in Table 4.

Table 4. *Singular values*

18.2029 0555 7529 2200	6.2231 9652 2604 2340	3.9134 8020 3335 6160	2.9767 9450 2557 7960
2.4904 5062 9660 3570	2.2032 0757 4479 9280	2.0191 8365 4054 5860	1.8943 4154 7685 6890
1.8059 1912 6612 3070	1.7411 3576 7747 9500	1.6923 5654 4395 2610	1.6547 9302 7369 3370
1.6253 2089 2877 9290	1.6018 3335 6666 2670	1.5828 6958 8713 6990	1.5673 9214 4480 0070
1.5546 4889 0109 3720	1.5440 8471 4076 0510	1.5352 8356 5544 9020	1.5279 2951 2160 3040
1.5217 8003 9063 4950	1.5166 4741 2836 7840	1.5123 8547 3899 6950	1.5088 8015 6801 8850
1.5060 4262 0723 9700	1.5038 0424 3812 6520	1.5021 1297 6754 0060	1.5009 3071 1977 0610
1.5002 3143 4775 4370	0.0000 0000 2793 9677		

Note that $\sigma_{30}/\sigma_1 \approx 1.53 \times 10^{-10}$ so that this matrix is very close to being a matrix of rank 29 even though the determinant equals 1.

Acknowledgement. The authors wish to thank Dr. Peter Businger of Bell Telephone Laboratories for his stimulating comments.

References

1. Businger, P., Golub, G.: Linear least squares solutions by Householder transformations. Numer. Math. **7**, 269–276 (1965).
2. Forsythe, G.E., Henrici, P.: The cyclic Jacobi method for computing the principal values of a complex matrix. Proc. Amer. Math. Soc. **94**, 1–23 (1960).
3. — Golub, G.: On the stationary values of a second-degree polynomial on the unit sphere. J. Soc. Indust. Appl. Math. **13**, 1050–1068 (1965).
4. — Moler, C.B.: Computer solution of linear algebraic systems. Englewood Cliffs, New Jersey: Prentice-Hall 1967.
5. Francis, J.: The QR transformation. A unitary analogue to the LR transformation. Comput. J. **4**, 265–271 (1961, 1962).
6. Golub, G., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. J. SIAM. Numer. Anal., Ser. B **2**, 205–224 (1965).
7. — Least squares, singular values, and matrix approximations. Aplikace Matematiky **13**, 44–51 (1968).
8. Hestenes, M.R.: Inversion of matrices by biorthogonalization and related results. J. Soc. Indust. Appl. Math. **6**, 51–90 (1958).
9. Kogbetliantz, E.G.: Solution of linear equations by diagonalization of coefficients matrix. Quart. Appl. Math. **13**, 123–132 (1955).
10. Kublanovskaja, V.N.: Some algorithms for the solution of the complete problem of eigenvalues. V. Vyčisl. Mat. i. Mat. Fiz. **1**, 555–570 (1961).
11. Martin, R.S., Reinsch, C., Wilkinson, J.H.: Householder's tridiagonalization of a symmetric matrix. Numer. Math. **11**, 181–195 (1968).
12. Wilkinson, J.: Error analysis of transformations based on the use of matrices of the form $I-2w\,w^H$. Error in digital computation, vol. II, L.B. Rall, ed., p. 77–101. New York: John Wiley & Sons, Inc. 1965.
13. — Global convergence of QR algorithm. Proceedings of IFIP Congress, 1968.

Prof. G.H. Golub
Computer Science Dept.
Stanford University
Stanford, California 94305
USA

Dr. C. Reinsch
Math. Institut
der Techn. Hochschule
8000 München 2
Arcisstr. 21