Computing the SVD

Walter Gander & Johann Joss

November 24, 2024

Abstract

We describe and compare the first algorithms to compute the Singular Value Decomposition proposed in 1965 by G. H. Golub and W. Kahan, 1967 by G. H. Golub and P. Businger, and also in 1967 by Ch. Reinsch. We show that the algorithms of Golub-Businger and of Golub-Reinsch perform the same basic operations. This is not obvious as the implementations and the data structures are different. The current algorithm used in many software packages is the one of Golub-Reinsch. We explain the algorithms, analyze the details and run some examples. Finally we discuss the project "Handbook of Automatic Computation", the first attempt to built a reliable software library.

1 Introduction

Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$. Then there exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ and a diagonal matrix $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{m\times n}$ with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$, such that

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

holds. This is the *singular value decomposition of* A (SVD). The column vectors of $U = [\mathbf{u}_1, \ldots, \mathbf{u}_m]$ are called the *left singular vectors* and similarly $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ are the *right singular vectors*. The values σ_i are called the *singular values* of A. If $\sigma_r > 0$ is the smallest nonzero *singular value*, then the matrix A has rank r.

2 The Pioneers

The numerical analysts who were developing the first algorithms to compute the SVD (1) were Gene H. Golub, William M. Kahan and Christian Reinsch (see Figure 1).

Christian Reinsch (1934–2022) TU München

Gene H. Golub (1932–2007) Stanford

William M. Kahan *1933 Berkeley

Figure 1: The three pioneers

3 Connection to Eigenvalues

The squares of the singular values are the eigenvalues of the matrix $A^\top A$, and V contains the eigenvectors, because

$$A^\top A = (U\Sigma V^\top)^\top U\Sigma V^\top = VDV^\top, \quad D = \Sigma^\top\Sigma = \operatorname{diag}(\sigma_1^2, \dots, \sigma_n^2). \tag{2}$$

Similarly

$$AA^\top = U\Sigma V^\top (U\Sigma V^\top)^\top = U\Sigma\Sigma^\top U^\top, \tag{3}$$

thus the eigenvalues of $AA^\top$ are $\sigma_1^2, \dots, \sigma_n^2$ plus $m-n$ zeros. U contains the eigenvectors.

Since the eigenvalues of $A^\top A$ are the squares of the singular values, one could consider computing the singular values as $\sigma_i(A) = \sqrt{\lambda_i(A^\top A)}$. However, this is not a good approach as already mentioned in [4]:

> *But the calculation of $A^\top A$ using ordinary floating point arithmetic does serious violence to the smaller singular values as well as to the corresponding eigenvectors which appear in U and V.*

The following example shows this:

```
m=10; format long
A=hilb(m); A=A(:,1:7);
E=eig(A'*A); E=sqrt(E(7:-1:1));
[E svd(A)]
format short e
RelError= (E-svd(A))./svd(A)

ans =                                        RelError =
   1.703422789369242   1.703422789369242       1.3035e-16
   0.303861884355195   0.303861884355195       9.1343e-16
   0.027332449735276   0.027332449735275       2.7291e-14
   0.001576339549570   0.001576339549570       2.5517e-13
   0.000060439432051   0.000060439432540      -8.0919e-09
   0.000001483441024   0.000001483519405      -5.2835e-05
   0.000000020984384   0.000000020211193       3.8256e-02
```

We notice that the small singular values have a large relative error.

4 The Algorithm of Golub-Kahan 1965

Golub and Kahan published 1965 an article [4] in which they developed two algorithms for computing the SVD that avoid forming the matrix $A^\top A$. If $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, they consider the augmented matrix

$$\tilde{A} = \begin{pmatrix} 0 & A \\ A^\top & 0 \end{pmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}. \tag{4}$$

If μ is an eigenvalue of $\tilde{A}$ then

$$\begin{pmatrix} 0 & A \\ A^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mu \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}.$$

It follows that $A\mathbf{y} = \mu\mathbf{x}$ and $A^\top\mathbf{x} = \mu\mathbf{y}$ and therefore

$$A^\top A\mathbf{y} = A^\top \mu\mathbf{x} = \mu^2\mathbf{y}.$$

Thus if λ is an eigenvalue of $A^\top A$ then $\mu = \pm\sqrt{\lambda(A^\top A)}$, which means that the eigenvalues of $\tilde{A}$ are the singular values $\pm\sigma_k$ of A plus $m-n$ zeros.

The singular values remain the same when we transform the matrix by multiplying with orthogonal matrices: $\{\sigma_i(A)\} = \{\sigma_i(P^\top AQ)\}$ if $P^\top P = I_m$ and $Q^\top Q = I_n$.

We can choose P and Q to bidiagonalize A:

$$P^\top AQ = \begin{pmatrix} J \\ 0 \end{pmatrix}, \quad J = \begin{pmatrix} a_1 & b_1 & & & \\ & \ddots & \ddots & & \\ & & a_{n-1} & b_{n-1} \\ & & & a_n \end{pmatrix}.$$

Furthermore we may replace all elements of the bidiagonal matrix J by their absolute value without changing the singular values, as the change of signs can be performed by an orthogonal transformation.

Thus we can replace $\tilde{A}$ (4) by the $2n \times 2n$ matrix $\bar{A}$

$$\bar{A} = \begin{pmatrix} 0 & J \\ J^\top & 0 \end{pmatrix}$$

and the eigenvalues of $\bar{A}$ are $\lambda_k(\bar{A}) = \pm\sigma_k$.

Let P be the permutation matrix that transforms

$$P \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ \vdots \\ 2n-1 \\ 2n \end{pmatrix} = \begin{pmatrix} n+1 \\ 1 \\ n+2 \\ 2 \\ \vdots \\ n+n \\ n \end{pmatrix}.$$

Then applying the transformation to $\bar{A}$ we obtain

$$S = P \begin{pmatrix} 0 & J \\ J^\top & 0 \end{pmatrix} P^\top = \begin{pmatrix} 0 & a_1 & & & & & \\ a_1 & 0 & b_1 & & & & \\ & b_1 & 0 & a_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & a_{n-1} & 0 & b_{n-1} & \\ & & & & b_{n-1} & 0 & a_n \\ & & & & & a_n & 0 \end{pmatrix} \tag{5}$$

and the problem is reduced to compute the non-negative eigenvalues of a special symmetric tridiagonal matrix S.

Golub-Kahan note in [4]:

> *There are a number of methods for obtaining the eigenvalues of a tridiagonal symmetric matrix. One of the most accurate and effective methods is to use Sturm sequences; an ALGOL program is given by Wilkinson.*

They refer here to the publication of J. Wilkinson of 1962 [7], which later also was published in the Handbook [8]. In order to compare the algorithms we translated the procedure `tridibisection` to Matlab.

```
function [w,NormInf,m1]=tridibisection(c,b,gu,go)
% TRIDIBISECTION computes the m1 Eigenvalues of symmetric tridiagonal
%  matrix with diagonal c and lower/upper diagonal b laying between gu
%  and go. The eigenvalues are computed in decreasing order using
%  Sturm sequences and  bisection and stored in w.
% TRIDIBISECTION is a translation of Wilkinson's tridibisection1 from
% ALGOL to MATLAB by W. Gander, May 2019.
% Changes in ALGOL tridibisection:
              % eliminate t=number of bisection steps,
              % instead compute EV to machine precision
gamma=eps^2;  % square of machine precision
n=length(c);  % no need to be in parameter list.
              % function [q1,a1]=sturmssequence(c,p,lambda)
              % with parameters, no global variables
b(n)=0;       % add zero for same length as c

NormInf=abs(c(1))+abs(b(1));
for i=2:n
  l=abs(b(i-1))+abs(c(i))+abs(b(i));
  if l>NormInf, NormInf=l; end
end
if nargin==2                    % added by W. Gander:
   go=1.5*NormInf; gu=-go;      % if no interval specified
end                             % compute all eingenvalues
if nargin==3                    % compute all lambda>=gu
   go=1.5*NormInf;
end
if gu>go, g=gu; gu=go; go=g; end  % Rearrangement if gu>go

for i=1:n-1
  if b(i)==0, p(i+1)=gamma*NormInf*NormInf;
  else p(i+1)=b(i)*b(i);
  end
end
p(1)=0;
if gu>NormInf | go<-NormInf
  m1=0; error('noeigenvalue')
end
lambda=gu;               % Determination of required number of EV
[q1,a1]=sturmssequence(c,p,lambda);
a2=a1;
if q1==0, a2=a1+1; end
lambda=go;
[q1,a1]=sturmssequence(c,p,lambda);
m1=a2-a1;
d=a1;
if go>NormInf, go=NormInf; end
if gu<-NormInf, gu=-NormInf; end

w=[];
for k=1:m1
  d=d+1;
  g=go; h=gu; lambda=(g+h)/2;
  while h<lambda & lambda<g      % machine independent termination
    [q1,a1]=sturmssequence(c,p,lambda);
    if a1>=d, h=lambda; else g=lambda; end
    lambda=(g+h)/2;
  end
  w(k)=(g+h)/2;
end
w=w(:);

function [q1,a1]=sturmssequence(c,p,lambda)
 n=length(c);
 p1=0; q1=1; a1=0;
 for i=1:n
   y=(c(i)-lambda)*q1-p(i)*p1;
   p1=q1; q1=y;
   if p1>=0 & q1>=0 | p1<0 & q1<0
      a1=a1+1;
   end
 end
 if q1==0 & p1>0, a1=a1-1;  end
```

For the bidiagonalization of A we use the MATLAB-function `Bidiagonalize.m` given in [3]. This function uses Householder-transformations to compute the bidiagonal matrix:

```
function [q,e,A]=Bidiagonalize(A)
% BIDIAGONALIZE bidiagonalizes a matrix with Householder reflections
%   [q,e,A]=Bidiagonalize(A) computes B=diag(q)+diag(e(2:n),1) such
%   that A=P B Q' using Householder reflexions. A is overwritten with
%   the Householder-vectors.

[m,n]=size(A);
for i=1:n
  s=norm(A(i:m,i));                    % transform A(i:m,i) to
                                       % (q_i,0,...,0)
  if s==0, q(i)=0;
  else
    if A(i,i)>0, q(i)=-s; else q(i)=s; end
    fak=sqrt(s*(s+abs(A(i,i))));
    A(i,i)=A(i,i)-q(i);
    A(i:m,i)=A(i:m,i)/fak;
    A(i:m,i+1:n)=A(i:m,i+1:n)-A(i:m,i)*(A(i:m,i)'*A(i:m,i+1:n));
  end
  if i<n,
    s=norm(A(i,i+1:n));                % tranformation A(i,i+1:n) to
                                       % (e_i,0...0)
    if s==0, e(i)=0;
    else
      if A(i,i+1)>0, e(i)=-s; else e(i)=s; end
      fak=sqrt(s*(s+abs(A(i,i+1))));
      A(i,i+1)=A(i,i+1)-e(i);
      A(i,i+1:n)=A(i,i+1:n)/fak;
      A(i+1:m,i+1:n)=A(i+1:m,i+1:n) - ...
                    (A(i+1:m,i+1:n)*A(i,i+1:n)')*A(i,i+1:n);
    end
  end
end                                    % insert 0 element in e (see
e=[0 e];                               % notation of bidiagonal matrix)
```

The function `tridibisection` assumes that the off-diagonal elements are nonzero. The rounding errors help that the elements do not vanish as the following example shows. In case an off-diagonal element happens anyway to be zero, Wilkinson introduces a rounding error by replacing the zero element by $\|J\|_\infty \times \text{eps}^2$ (where eps is the machine precision).

Example

Consider the 18×12 matrix A with rank 6.

```
>> B =[5 -1 -1  6  4  0
   -3  1  4 -7 -2 -3
    1  3 -4  5  4  7
    0  4 -1  1  4  5
    4  2  3  1  6 -1
    3 -3 -5  8  0  2
    0 -1 -4  4 -1  3
   -5  4 -3 -2 -1  7
    3  4 -3  6  7  7];
>> A=[B 2*B;  3*B -B];
```

Using the function `Bidiagonalize`

```
>> [a,b]=Bidiagonalize(A)
```

we obtain the following bidiagonal matrix:

$$\left[\begin{array}{rr}
\multicolumn{1}{c}{\mathbf{a}} & \multicolumn{1}{c}{\mathbf{b}} \\
\hline
-30.659419433511783 & -45.852631656070017 \\
35.249794473507983 & 52.134575892741985 \\
9.439799374843513 & -35.868247122679563 \\
-8.601911783054952 & -36.110469925331472 \\
19.528418692750023 & -38.856756306376781 \\
-8.575731326097568 & 28.348270719033149 \\
-1.66142625743e{-12} & 2.4e-15 \\
3.6e{-15} & -1.0e{-15} \\
4.0e{-15} & 1.7e{-15} \\
-3.2e{-15} & 5.9e-16 \\
-2.9e{-15} & -1.0e{-15} \\
-1.9e{-15} &
\end{array}\right] \qquad (6)$$

The first column of the matrix (6) is the diagonal **a** and the second column the off-diagonal **b** of the bidiagonal matrix J. If we now apply `tridibisection` to the matrix S (5) and compute only the non-negative eigenvalues, we get the first version of the Golub-Kahan Algorithm to compute the singular values:

```
function q=SVDGolubKahan1(A)
% SVDGOLUBKAHAN1 singular values by the first Golub Kahan algorithm
%  svdgolubkahan1 computes the simgular values of A  by applying
%  Wilkinson's TRIDIBISECTION to the matrix S (2n x 2n)
[m,n]=size(A); if n>m, A=A'; [m,n]=size(A);end
[a,b]=Bidiagonalize(A);            % Householder bidiagonalization
sk=zeros(2*n-1,1);                 % form tridiagonal matrix (3.3)
k=1:n; sk(2*k-1)=abs(a(k));        % S on page 213
k=1:n-1; sk(2*k)=abs(b(k+1));
q=tridibisection(zeros(2*n,1),sk,0); % compute all nonnegative EV
```

Using the above matrix A we compute with `SVDGolubKahan1` the singular values:

```
>> q1=SVDGolubKahan1(A);
>> [q1, svd(A)]
ans =
  72.265903120085341  72.265903120085312
  49.630339183086051  49.630339183086058
  44.288698552845858  44.288698552845830
  36.427417335191990  36.427417335191990
  30.416324106579545  30.416324106579534
  25.017401012828763  25.017401012828767
   0.000000000000006   0.000000000000011
   0.000000000000005   0.000000000000008
   0.000000000000003   0.000000000000006
   0.000000000000003   0.000000000000003
   0.000000000000003   0.000000000000001
   0.000000000000002   0.000000000000000
```

We note that the singular values compare very well with those computed by MATLAB's function `svd`.

Golub-Kahan develop in [4] a second algorithm to compute the singular values of a matrix. After bidiagonalization they consider

$$K = J^\top J = \begin{pmatrix} a_1^2 & a_1 b_1 & & & \\ a_1 b_1 & a_2^2 + b_1^2 & a_2 b_2 & & \\ & a_2 b_2 & \ddots & \ddots & \\ & & \ddots & \ddots & a_{n-1} b_{n-1} \\ & & & a_{n-1} b_{n-1} & a_n^2 + b_{n-1}^2 \end{pmatrix}$$

and write:

Although the smaller eigenvalues of $A^\top A$ are usually poorly determined, a simple error analysis shows that all the eigenvalues of K are as well-determined as those of T [the tridiagonal matrix S]. The reason for this is that the computation of the Sturm sequences is algebraically the same for both T and K. Thus to use K is preferable since the total number of operations in calculating its eigenvalues is certainly less than in computing the eigenvalues of T.

A MATLAB-program for this second algorithm is

```
function q=SVDGolubKahan2(A)
% SVDGOLUBKAHAN2 singular values by the second Golub Kahan algorithm.
%   Applying Wilkinson's TRIDIBISECTION to matrix K (n x n)
[m,n]=size(A); if n>m, A=A'; [m,n]=size(A);end
[a,b]=Bidiagonalize(A);        % Householder bidiagonalization
a=abs(a); b=abs(b);
d=a.^2+b.^2;                   % form diagonal and
nd=a(1:n-1).*b(2:n);           % subdiagonal of K= J'J
q=tridibisection(d,nd);        % compute EV of K
q=sqrt(q);                     % sqrt(lambda_k)=sigma_k
```

Using `SVDGolubKahan2` for the example above we get

```
>> q2=SVDGolubKahan2(A)
q2 =
  72.265903120085326
  49.630339183086051
  44.288698552845858
  36.427417335191997
  30.416324106579545
  25.017401012828763
   0.000000412953092
   0.000000000000005
   0.000000000000004
   0.000000000000003
   0.000000000000003
   0.000000000000002
```

Note that the seventh singular value is $0.000000412953092 \neq 0$. It seems that our MATLAB implementation of the second algorithm has a problem with zero eigenvalues though, the nonzero eigenvalues give correct singular values.

Unfortunately the original algorithms are lost. W. Kahan wrote me in 2018:

My earliest programs to compute singular values were punched into cards for the IBM 7094 that had just replaced the 7090 at the University of Toronto. Those card decks are long gone.

I used Householder rotations/reflections to reduce a matrix to bidiagonal form, and then either QR-iteration for all singular values and vectors, or Sturm sequences for only the extreme singular values, all in Fortran. That was in the spring of 1964.

Then I attended a SHARE meeting in San Francisco; it convenes users of IBM's mainframes. I was at that time an active contributor to the SHARE library of numerical subprograms, but had not yet submitted the SVD programs. I broke away from the meeting to visit an old friend, Gene Golub, at Stanford; we had met first at the University of Illinois Urbana-Champaign in the summer of 1957. During my visit to Stanford we discovered that we had both been working on the same SVD problem, and with similar ideas.

It seemed wasteful for each of us to write things up separately, so I got permission from the U. of T. to spend two extra days away, and Gene and I wrote the SVD paper over the weekend plus Monday. Gene's versions of the program would have been run on the Stanford 7090, probably in Fortran, rather than in Algol on Stanford's Burroughs B5000 next door in Pine Hall. All that stuff has been gone long ago.

I have been throwing stuff away to lighten the load that I must carry to a shared office soon when I leave my own office to make way for a younger appointee. During that process I did not see a copy of my old SVD programs that ran on the CDC 6400 here. So, sadly, I cannot provide you with a copy of the earliest SVD.

5 The Algorithm of Golub-Businger 1967

On July 31, 1967, G. H. Golub and P. Businger published the Stanford Technical Report No. CS73. The report contains two parts. The first part by Golub has the title *Least Squares, Singular Values and Matrix Computations*. The second part is an ALGOL program written by Businger: *An ALGOL Procedure for Computing the Singular Value Decomposition*. The report is missing in the collection of CS-reports at Stanford University. I am indebted to Åke Björck who drew my attention to it and sent me a copy of the report [6].

We managed to scan/ocr and retype the ALGOL procedure by Businger and we obtained:

```
procedure singular_values_decomposition
   (a, m , n , u_desired, vt_desired, eta) results: (sigma, u, vt);
value m, n, u_desired, vt_desired, eta;
real array a, sigma, u, vt ;
integer m, n ;
boolean u_desired, vt_desired ;
real eta ;

comment Householder's and the QR method are ùsed to find all singular
values sigma[i] , (i=1, 2,... , n) of the given matrix a[1:m,1:n],
(m>=n). The orthogonal matrices u[1:m, 1:m] and vt[1:n, 1:n] which
effect the singular values decomposition a=u sigma vt are computed
individually depending on whether u desired or vt desired. The input
parameter eta is the relative machlne precision ;

begin
   procedure Householder_bidiagonalization
       (a, m, n , u_desired, vt_desired) results: (alpha, beta, u, vt);
   value m, n, u_desired, vt_desired ;
   real array a, alpha, beta, u, vt;
   integer m, n ;
   boolean u_desired, vt_desired ;

   comment Householder transformations applied ln turn on the left and
   the right reduce the given matrix a[ 1 :m, 1 :n], (m>=n) to upper
   bi- diagonal form J. The diagonal elements of J are returned as
   alpha[i], (i=l, 2, ... , n), the superdlagonal elements as beta [i]
   , (1=1, 2, ..., n-1, beta(n]=O. The orthogonal matrices u[1:m,1:m]
   and vt[1:n,1:n] which effect the decomposition a=u J vt are
   computed indivdually depending on whether u desired or vt desired ;

begin
   real procedure inner_product (i, m, n, a, b, c) ;
   value m, n, c; real a, b, c ; integer i, m, n ;
   begin
```

```
      for i := m step 1 until n do c:=c+a*b ; inner_product:=c
   end inner product ;

  real s, b ;
  integer i, j, k;
  if u_desired then
     for i:=1 step 1 until m do
     begin
        u[i,i] :=1.0 ;
        for j:=i+1 step 1 until m do u[i,j]:=u[j,i]:=0.0
     end i;
  if vt_desired then
     for i:=1 step 1 until n do
     begin
        vt[i,i] :=1.0 ;
        for j:=i+1 step 1 until n do vt[i,j]:=vt[j,i]:=0.0
     end i  ;

  for k:=1 step 1 until n do
     begin
        s:=inner_product(i,k, m, a[i,k], a[i,k], 0.0) ;
        alpha[k] :=if a[k,k]<0.0 then sqrt(s) else -sqrt(s) ;
        if s != 0.0 then
        begin comment transformation on the left;
           b:=s-a[k,k]*alpha[k] ;
           a[k,k]:=a[k,k]-alpha[k] ;
           for j:=k+1 step 1 until n do
           begin
             s:=inner_product(i,k,m,a[i,k],a[i,j], 0.0)/b ;
             for i:=k step 1 until m do
                 a[i,j]:=a[i,j]-a[i,k]*s
           end j ;
           if u_desired then
              for i:=1 step 1 until m do
              begin
                 s:=inner_product(j,k,m,u[i,j],a[j,k], 0.0)/b ;
                 for j:=k step 1 until m do
                    u[i,j]:=u[i,j]-s*a[j,k]
              end i
        end transformation on the left ;
        if k <= n-2 then
        begin
         s:=inner_product(j,k+1,n, a[k,j], a[k,i], 0.0) ;
         beta[k]:=if a[k,k+1]<0.0 then sqrt(s) else -sqrt(s) ;
         if s != 0 then
          begin comment. transforatlon on the rlght ;
            b:=s-a[k,k+1]*beta[k] ;
            a[k,k+1]:=a[k,k+1]-beta[k] ;
            for i:=k+1 step 1 until m do
            begin
              s:=inner_product(j,k+1,n, a[k,j], a[i,j], 0.0)/b ;
              for j:=k+1 step 1 until n do
                  a[i,j]:=a[i,j]-a[k,j]*s
            end i ;
            if vt_desired then
               for j:=1 step 1 until n do
               begin
                  s:=inner_product(i,k+1,n, a[k,i], vt[i,j], 0.0)/b ;
                  for i:=k+1 step 1 until n do
                     vt[i,j]:=vt[i,j]-a[k,i]*s
               end j
          end transformation on the rlght
        end k from 1 to n-2
      else beta[k]:=if k=n then 0.0 else a[k,n]
   end k
end Householder bidiagonalizat1on ;

procedure QR_diagonalization
   (gamma,m, n, u_desired, vt_desired, eta) result:(sigma)
   transients: (u, vt) ;
value m, n, u_desired, vt_desired, eta ;
real array gamma, sigma, u, vt ;
integer m, n ;
boolean u_desired, vt_desired ;

comment  The QR algorithm diagonalizes the given symmetric tridiagonal
 Matrix T of order 2n by 2n qhose diagonal elements are zero and
 whose super- and subdiagonal elements are gamma[i], (i=1,2,...,
 2n-1), gamma[0])_gamma[2n]=0. If u desired then the odd numbered
 rotations of the QR algorithm are also applied to u[1:m,1:m] from
 the right. If vt desired then the even numered rotations are also
 applied to vt[1:n,1:n] from the left. The input parameter eta is
 the relative machine precision. The nonnegative eigenvalues of T
 are returned as sigma[i], (i=1,2,...,n);

begin
   real kappa, d, r, sinphi, cosphi, g0, g1, g2, g3, epsilon, rho;
   integer i, j, k, s, s0, t, t0, t2 ;
   s:=s0:=t0:=0; t:=2*n ;
   kappa :=g1:=abs(gamma [1]) ;
   for i:=2 step 1 until t do
   begin comment find the infinity norm of the tridiagonal matrix T ;
      g2:=abs(gamma[i]) ;  d:=g1+g2 ; if d>kappa then kappa:=d;
      g1:=g2
   end i ;
   epsilon :=eta*kappa ;
inspect:
   comment scan for lower block limit t ;
   gamma[s]:=gamma[t]:=0.0 ;
   for i:=t-2 while abs(gamma [i])<= epsilon do
   begin comment pick up computed value ;
      t2:=t div 2 ;  sigma[t2]:=abs(gamma[t-1]) ;
      if gamma[t-1]<0.0 and vt_desired then
         for j:=1 step 1 until n do vt[t2,j]:=-vt[t2,j] ;
      t:=i ; gamma[t] := 0.0;
      if t=0 then goto return
   end ;
   s:=t-4 ; comment scan for upper block limit s ;
   for i:=s-2 while abs(gamma[s])>epsilon do s:=i ;
   comment did block limits s, t change ;
   if s != s0 or t != t0  then
   begin
zero_shift:
      gamma[s] := gamma[s+1] ; d:=gamma[s+2] ; goto QR_sweep
   end zero shift

   comment does matrix break ;
   if abs(gamma[s+1]*gamma[s+2]) <= epsilon then goto zero_shift ;

   for i:=s+1 step 2 until t-1 do
      if abs(gamma[i])<=epsilon then goto zero_shift ;

   comment did bottom value settle down ;
   if abs(abs(gamma[t-1])-rho) >  0.1*abs(gamma[t-1]) then
      goto zero_shift ;
   comment determine the origin shift kappa;
   g0:=gamma[t-1]^2+gamma[t-2]^2+gamma[t-3]^2 ;
   g1:=gamma[t-1]^2*gamma[t-3]^2 ;
   g2:=0.5*(g0+sqrt(g0^2-4.0*g1)) ;
   g3:=g1/g2 ;
   kappa:=if abs(gamma[t-1]^2-g2)<abs(gamma[t-1]^2-g3) then g2 else g3 ;
   gamma[s] :=gamma[s+1]^2-kappa ; d:=gamma[s+1]*gamma[s+2] ;

QR_sweep:
   comment save previous block limits and botom element ;
   s0:=s ; t0:=t ;   rho:=abs(gamma[t-1]) ;
   for  i:= s step 1 until t-3 do
   begin
      comment does matrix break ;
     if d=0.0 then goto inspect ;
     g0:=gamma[i] ;  g1 :=gamma[i+1] ;
     g2:=gamma[i+2] ; g3:=gamma[i+3] ;
     r:=sqrt(g0^2+d^2) ;
     sinphi:=d/r; cosphi:=g0/r ;
     gamma[i]:=r ;
     gamma[i+1]:=g1*cosphi+g2*sinphi ;
     gamma[i+2]:=g1*sinphi-g2*cosphi ;
     gamma[i+3]:=-g3*cosphi ;
     d:=g3*sinphi ;
     if u_desired or vt_desired then
     begin
        k:=i div 2 ;
        if i=2*k and vt_desired then
        for j:=1 step 1 until n do
        begin
           g1:=vt[k+1,j] ; g2:=vt[k+2,j] ;
           vt[k+1,j]:=g1*cosphi+g2*sinphi ;
           vt[k+2,j]:=g1*sinphi-g2*cosphi
        end j ;
        if i != 2*k and u_desired then
        for j:=1 step 1 until m do
        begin
           g1:=u[j,k+1] ; g2:=u[j,k+2] ;
           u[j,k+1]:=g1*cosphi+g2*sinphi ;
           u[j,k+2]:=g1*sinphi-g2*cosphi
        end j
```

```
    end if u_desired or vt_desired
end i ;
goto inspect ;
return:
end QR diagonalization ;

real array alpha, beta[1:n], gamma[0:2*n] ;
integer i, j ;

Householder_bidiagonalization
   (a, m, n, u_desired, vt_desired, alpha, beta, u, vt) ;

for i:=1 step 1 until n do
begin
   gamma[2*i-1] :=alpha[i] ; gamma[2*i]:=beta[i]
end
gamma[0]:=gamma(2*n]:=0.0 ;

QR_diagonalization
   (gamma, m, n, u_desired, vt_desired, eta, sigma, u, vt)

end singular values decomposition;
```

In order to test the program, we first considered to translate the procedure to Matlab. However, the ALGOL procedure contains goto-statements, even a goto into an if-clause! There are no goto statements in Matlab, so for a translation we would have to significantly change the the structure of the procedure and this would no longer represent the original ALGOL procedure.

We therefore looked around to find an ALGOL-compiler. Indeed we were successful: Jan van Katwijk, a *"Retired and hobby programmer"* as he calls himself, is *"Working on - and interested in - sdr software. Always in for a new challenge in the field of programming and software design."* His ALGOL compiler is in public domain and can be obtained by `https://github.com/JvanKatwijk/algol-60-compiler`. Using the jff-Compiler, we can test the procedure and compute singular values.

The idea of Golub is to apply the QR-Algorithm of Francis with implicit shifts to the special tridiagonal matrix (5) which we denote by K_0. We obtain a sequence of matrices K_i with $K_i = M_iR_i$ and $K_{i+1} = R_iM_i = M_i^\top K_iM_i$.

However, since the eigenvalues of K_i occur in pairs, Golub proposes to consider the QR-decomposition of

$$(K_i - s_iI)(K_i + s_i) = K_i^2 - s_i^2 I$$

so that $M_iR_i = K_i^2 - s_i^2 I$.

Note that if

$$K = \begin{bmatrix} 0 & \gamma_1 & 0 & 0 & 0 \\ \gamma_1 & 0 & \gamma_2 & 0 & 0 \\ 0 & \gamma_2 & 0 & \gamma_3 & 0 \\ 0 & 0 & \gamma_3 & 0 & \gamma_4 \\ 0 & 0 & 0 & \gamma_4 & 0 \end{bmatrix}$$

then

$$K^2 - s^2 I = \begin{bmatrix} {\gamma_1}^2 - s^2 & 0 & \gamma_1\gamma_2 & 0 & 0 \\ 0 & {\gamma_1}^2 + {\gamma_2}^2 - s^2 & 0 & \gamma_2\gamma_3 & 0 \\ \gamma_1\gamma_2 & 0 & {\gamma_2}^2 + {\gamma_3}^2 - s^2 & 0 & \gamma_3\gamma_4 \\ 0 & \gamma_2\gamma_3 & 0 & {\gamma_3}^2 + {\gamma_4}^2 - s^2 & 0 \\ 0 & 0 & \gamma_3\gamma_4 & 0 & {\gamma_4}^2 - s^2 \end{bmatrix}$$

is pentadiagonal. However, since the first diagonal after the main diagonal is zero, the pentadiagonal matrix has only three nonzero diagonals. To apply a first Givens-reflection Z_1 to annihilate the (3,1)-element $\gamma_1\gamma_2$

$$Z_1(K^2 - s^2 I)$$

we need to consider the reflection Z_1 of the form

$$Z_p = \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & \cos\Theta_p & 0 & \sin\Theta_p & & & \\ & & & 0 & 1 & 0 & & & \\ & & & \sin\Theta_p & 0 & -\cos\Theta_p & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{bmatrix} \begin{array}{l} \\ \\ \\ \leftarrow p \\ \leftarrow p+1 \\ \leftarrow p+2 \\ \\ \\ \\ \end{array} \qquad (7)$$

6 Shift Strategy and First Transformation

The shift s_i^2 used for

$$(K_i - s_i I)(K_i + s_i) = K_i^2 - s_i^2 I$$

is chosen as the square of the eigenvalue of the bottom 4×4 matrix that is closer to γ_{t-1}^2.

$$M = \begin{bmatrix} 0 & \gamma_{t-3} & 0 & 0 \\ \gamma_{t-3} & 0 & \gamma_{t-2} & 0 \\ 0 & \gamma_{t-2} & 0 & \gamma_{t-1} \\ 0 & 0 & \gamma_{t-1} & 0 \end{bmatrix}$$

Forming the characteristic polynomial and equating to zero we get a quadratic equation for λ^2:

$$\det(M - \lambda I) = \lambda^4 - \left({\gamma_{t-3}}^2 + {\gamma_{t-2}}^2 + {\gamma_{t-1}}^2\right)\lambda^2 + {\gamma_{t-3}}^2{\gamma_{t-1}}^2 = 0$$

and the solutions

$$\lambda^2 = \frac{\left({\gamma_{t-3}}^2 + {\gamma_{t-2}}^2 + {\gamma_{t-1}}^2\right) \pm \sqrt{({\gamma_{t-3}}^2 + {\gamma_{t-2}}^2 + {\gamma_{t-1}}^2)^2 - 4{\gamma_{t-3}}^2{\gamma_{t-1}}^2}}{2}.$$

Comparing this with the ALGOL statements

```
g0:=gamma[t-1]^2+gamma[t-2]^2+gamma[t-3]^2 ;
g1:=gamma[t-1]^2*gamma[t-3]^2 ;
g2:=0.5*(g0+sqrt(g0^2-4.0*g1)) ;
g3:=g1/g2 ;
kappa:=if abs(gamma[t-1]^2-g2)<abs(gamma[t-1]^2-g3) then g2 else g3 ;
```

we see that the quadratic equation is carefully solved by first solving for the larger solution `g2` and then computing the smaller `g3` by the relation of Vieta. The shift `kappa` is chosen as the solution that is closer to γ_{t-1}^2. But there is no check weather the discriminant is negative. Also it should be checked if `g2` is zero.

The first Givens-reflection Z_1 is determined to annihilate the (3,1)-element of the matrix $K_i^2 - s_i^2 I$. Thus the first column of

$$Z_1(K_i^2 - s_i^2 I)$$

is a multiple of $\mathbf{e}_1$.

This first reflection is then applied to the matrix K_i and defines the implicit shift for the QR-iteration step. It generates a bulge in position (3,1) and (1,3) as we can see in the following example.

7 Example

We consider the bidiagonal matrix

$$J = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 4 & 0 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{8}$$

The augmented matrix becomes

$$K_0 = \begin{pmatrix} 0 & 1 & & & & & & \\ 1 & 0 & 2 & & & & & \\ & 2 & 0 & 1 & & & & \\ & & 1 & 0 & 4 & & & \\ & & & 4 & 0 & 1 & & \\ & & & & 1 & 0 & 6 & \\ & & & & & 6 & 0 & 1 \\ & & & & & & 1 & 0 \end{pmatrix}$$

For $\sigma_0 = 0$ the matrix $K_0^2 - \sigma_0^2 I$ becomes

$$K_0^2 - \sigma_0^2 I = \begin{pmatrix} 1 & 0 & 2 & & & & & \\ 0 & 5 & 0 & 2 & & & & \\ 2 & 0 & 5 & 0 & 4 & & & \\ & 2 & 0 & 17 & 0 & 4 & & \\ & & 4 & 0 & 17 & 0 & 6 & \\ & & & 4 & 0 & 37 & 0 & 6 \\ & & & & 6 & 0 & 37 & 0 \\ & & & & & 6 & 0 & 1 \end{pmatrix}$$

The Givens-reflection Z_1 to transform the first column of $K_0^2 - \sigma_0^2 I$ to a multiple of $\mathbf{e}_1$ is

$$Z_1 = \begin{pmatrix} 0.4472 & 0 & 0.8944 & & & & & \\ 0 & 1.0000 & 0 & & & & & \\ 0.8944 & 0 & -0.4472 & & & & & \\ & & & 1.0000 & & & & \\ & & & & 1.0000 & & & \\ & & & & & 1.0000 & & \\ & & & & & & 1.0000 & \\ & & & & & & & 1.0000 \end{pmatrix}$$

This first transformation Z_1 is then applied to K_0 to give

$$K_1 = Z_1 K_0 Z_1 = \begin{pmatrix} 0 & 2.2361 & 0 & 0.8944 & & & & \\ 2.2361 & 0 & 0 & 0 & & & & \\ 0 & 0 & 0 & -0.4472 & & & & \\ 0.8944 & 0 & -0.4472 & 0 & 4.0000 & & & \\ & & & 4.0000 & 0 & 1.0000 & & \\ & & & & 1.0000 & 0 & 6.0000 & \\ & & & & & 6.0000 & 0 & 1.0000 \\ & & & & & & 1.0000 & 0 \end{pmatrix}$$

The bulge 0.8944 is now chased down by subsequent Givens-reflections of type (7) which preserve the zero diagonal:

$$K_i = Z_i K_{i-1} Z_i, \quad i = 2, 3, \ldots, 6$$

and thus the resulting matrix after this first QR-step is

$$K_6 = \begin{pmatrix} 0 & 2.4083 &&&&&& \\ 2.4083 & 0 & -1.4948 &&&&& \\ & -1.4948 & 0 & 3.8669 &&&& \\ && 3.8669 & 0 & 1.5423 &&& \\ &&& 1.5423 & 0 & 5.8850 && \\ &&&& 5.8850 & 0 & 0.0031 & \\ &&&&& 0.0031 & 0 & 0.0182 \\ &&&&&& 0.0182 & 0 \end{pmatrix}.$$

A second QR-step (again with zero shift) gives

$$\begin{pmatrix} 0 & 3.4919 &&&&&& \\ 3.4919 & 0 & 2.1196 &&&&& \\ & 2.1196 & 0 & 3.8644 &&&& \\ && 3.8644 & 0 & 3.4473 &&& \\ &&& 3.4473 & 0 & 4.0615 && \\ &&&& 4.0615 & 0 & 4.8402e{-08} & \\ &&&&& 4.8402e{-08} & 0 & 0.0182 \\ &&&&&& 0.0182 & 0 \end{pmatrix}$$

A third QR-step (this time with shift $\kappa = 3.3292e{-04}$) gives

$$\begin{pmatrix} 0 & 4.5505 &&&&&& \\ 4.5505 & 0 & -2.1040 &&&&& \\ & -2.1040 & 0 & 5.1932 &&&& \\ && 5.1932 & 0 & 1.5867 &&& \\ &&& 1.5867 & 0 & 2.3192 && \\ &&&& 2.3192 & 0 & 8.8827e{-18} & \\ &&&&& 6.2711e{-18} & 0 & 0.0182 \\ &&&&&& 0.0182 & 0 \end{pmatrix}$$

Notice that the element γ_6 in position (6,7) converges fast to zero

original	6
after QR-step 1	0.0031
after QR-step 2	$4.8402e{-08}$
after QR-step 3	$8.8827e{-18}$

A first singular value is computed : 0.0182 and the matrix can be deflated by setting $n := n - 2$.

The above computations were performed in Matlab with full matrices for didactic purposes to explain the algorithm.

When computing the same example with the Golub-Businger Procedure we obtain the following results. We print the array `gamma` after the label `inspect`. We also print the variables `s` and `t` that define the current block of the tridiagonal matrix being processed. In the ALGOL procedure the vector `gamma` is augmented by `gamma[0]=0` and `gamma[2n]=0`.

Note that some signs of elements of `gamma` differ from the Matlab computation. This does not matter for the QR-iteration. The elements could be replaced by their absolute values without changing the singular values of the bidiagonal matrix.

```
inspect  1
gamma
-1.0000e0  2.0000e0  1.0000e0  4.0000e0  1.0000e0  -6.0000e0  -1.0000e0
inspect  2
gamma
2.4083e0  1.4948e0  3.8669e0  1.5422e0  5.8850e0  -3.1456e-3  1.8246e-2
zero shift
inspect  3
gamma
```

```
3.4918e0  2.1196e0  3.8644e0  3.4473e0  4.0614e0  4.8402e-8  1.8246e-2
shift = 3.329227607e-4
inspect  4
gamma
4.5505e0  2.1040e0  5.1932e0  1.5866e0  2.3191e0  -1.0213e-23  1.8246e-2
```

The results are the same as computed before with full matrices in MATLAB. After two QR-iterations with zero shift and one iteration with shift `kappa 3.329227607e-4` the element $\gamma_6 = -1.0213e{-23} \approx 0$. We obtain the first singular value $\sigma_4 = 1.8246e{-2}$ and the matrix is deflated by $t := t - 2$. The deflation is done because $|\gamma_6| \leq \texttt{eta}\|K_0\|_\infty$ with `eta` the machine precision.

The algorithm of Golub-Businger preserves the zeros on the diagonal of the symmetric tridiagonal matrix. Thus in fact, the algorithm works only on the secondary diagonal with the elements $\gamma_1, \gamma_2, \cdots, \gamma_{2n-1}$.

Let's compare this Golub-Businger -algorithm with QR-steps on the corresponding bidiagonal matrix. We consider the matrix (8) and apply QR-steps with the same shifts as before.

```
A =
     1     2     0     0
     0     1     4     0
     0     0     1     6
     0     0     0     1
shift =
     0
gamma =
  -2.4083e+00  -1.4948e+00  -3.8669e+00  -1.5423e+00  -5.8850e+00  -3.1456e-03  -1.8246e-02
shift =
     0
gamma =
  -3.4919e+00  -2.1196e+00  -3.8644e+00  -3.4473e+00  -4.0615e+00  -4.8402e-08  -1.8246e-02
shift =
   3.3292e-04
gamma =
  -4.5505e+00   2.1040e+00   5.1932e+00   1.5867e+00   2.3192e+00  -3.3631e-22  -1.8246e-02
```

We see that we obtain the same results. This means that the basic Golub-Businger - and the Golub-Reinsch -Algorithm (as presented in the next section) are the same! They operate on different data structure (vector with $2n-1$ elements versus bidiagonal $n \times n$ matrix). The shift-strategy, the exception handling and the termination criteria are different. But mathematically they operate basically in the same way.

The Golub-Businger -Procedure was not accepted for the Handbook [8], because sometimes the algorithm failed. We show here a few examples.

8 Successful Examples

- For the 18×12 matrix A of Example 4, Golub-Businger produces in 18 iterations perfectly correct results:

```
7.226590312008532e1
4.963033918308604e1
4.428869855284583e1
3.642741733519196e1
3.041632410657953e1
2.501740101282876e1
5.861665712052792e-15
4.488399943021106e-15
3.445807702749013e-15
2.802195408089914e-15
2.351807372845660e-15
2.226465746728873e-15
```

- Consider the transposed of the $n \times (n+1)$- matrix, the third example on page 150 of the Handbook [8]:

```
for i=1:n
  for j=1:n
    if i==j, a(i,j)=1;
    elseif j>i, a(i,j)=0;
      else a(i,j)=-1;
    end
  end
end
for i=1:n
  a(n+1,i)=-1;
end
```

For $n = 6$ we get

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

For $n = 30$ Golub-Businger needs 66 QR-steps to compute the singular values. This means a singular value is computed in about two QR-steps. The singular values are

18.8357	1.6996	1.5223
6.4243	1.6603	1.5171
4.0239	1.6296	1.5127
3.0466	1.6052	1.5091
2.5374	1.5855	1.5062
2.2360	1.5694	1.5039
2.0426	1.5562	1.5022
1.9114	1.5453	1.5010
1.8186	1.5362	1.5002
1.7507	1.5287	1.4142

The results compare well with those of MATLAB. When comparing the norm of the vectors of the singular values we obtain

$$\frac{\|svd_{\text{Matlab}} - svd_{\text{GB}}\|}{\|svd_{\text{Matlab}}\|} = 6.6867e{-}16.$$

The ALGOL Procedure of Golub-Reinsch which we scanned from the Handbook [8] gives the same results with only 40 iterations.

- By replacing the unit diagonal by

  $$a_{ii} = n + 1 - i, \quad j = 1, \ldots, n$$

  we get an explicit expression for the singular values

  $$\sigma_{n+1-k} = \sqrt{k(k+1)}, \quad k = n, n-1, \ldots, 1.$$

  For this case Golub-Businger and Golub-Reinsch are very effective. E.g. for $n = 150$ both procedures compute with *only one* QR-step all 150 singular values to full machine precision!

9 Close and Multiple Singular Values

The Golub-Businger algorithm has problems with matrices with multiple or close singular values.

1. The following bidiagonal matrix

   $$B_1 = \begin{pmatrix} 1.614874172816116 & 9.264623902779769e{-}01 \\ 1.238486644745703 & 2.131595816650056e{-}07 \\ 1.926281858121494 & 4.598199463754764e{-}01 \\ 1.038269760777829 & \end{pmatrix}$$

   has the singular values

   $$\begin{matrix} 2.000000100000000 \\ 2.000000000000000 \\ 1.000000100000000 \\ 1.000000000000000 \end{matrix}.$$

   There are two clusters with two singular values with a gap of 10^{-7}. We print intermediate results using Golub-Businger and obtain

```
inspect 1
gamma
-1.6148e0  9.2646e-1  1.2384e0  2.1315e-7  1.9262e0  -4.5981e-1  -1.0382e0  0
zero shift
s and t:  0 8
inspect 2
gamma
1.9611e0  3.3759e-1  1.0198e0  7.4224e-7  1.9950e0  -1.2203e-1  1.0024e0  0
kappa=1.000000179
s and t:  0 8
inspect 3
gamma
2.0000e0  8.7751e-8  1.9611e0  3.3742e-1  1.0198e0  -7.4612e-9  1.0000e0  0
kappa=1.000000199
s and t:  0 8
inspect 4
gamma
2.0000e0  8.5471e-8  2.0000e0  2.3202e-8  9.9999e-1  -1.0219e-11  1.0000e0  0
kappa=1.000000198
s and t:  0 8
inspect 5
gamma
2.0000e0  8.5471e-8  2.0000e0  1.5378e-15  9.9999e-1  -5.9455e-14  1.0000e0  0
kappa=1.000000199
s and t:  0 8
inspect 6
gamma
2.0000e0  8.5471e-8  2.0000e0  1.0250e-22  9.9999e-1  -1.1886e-17  1.0000e0  0
```

After the first QR-iteration with zero shift we obtain $\gamma_6 = -1.2203e{-}1$. In the next step with shift $\kappa = 1.000000179$ it decreases nicely to $\gamma_6 = -7.4612e{-}9$.

However, the next steps with same shift show linear convergence:

$$-1.0219e{-}11, -5.9455e{-}14, -1.1886e{-}17.$$

We also observe simultaneously convergence of γ_4:

$$\text{7.4224e-7 , 3.3742e-1 , 2.3202e-8 , 1.5378e-15 , 1.0250e-22 .}$$

Thus two singular values are computed, the matrix can be deflated and computation continues with the smaller matrix.

```
zero shift
s and t:  0 4
inspect 7
gamma
2.0000e0  -8.5471e-8  2.0000e0  0
kappa=4.000000011
s and t:  0 4
inspect 8
gamma
2.0000e0  -3.3341e-9  2.0000e0  0
kappa=4.000000011
s and t:  0 4
inspect 9
gamma
2.0000e0  -9.8840e-11  2.0000e0  0
kappa=3.999999993
s and t:  0 4
inspect 10
gamma
2.0000e0  1.5748e-12  2.0000e0  0
kappa=3.999999993
s and t:  0 4
inspect 11
gamma
2.0000e0  -2.5092e-14  2.0000e0  0
kappa=3.999999993
s and t:  0 4
inspect 12
gamma
2.0000e0  3.9981e-16  2.0000e0  0
```

After a zero shift iteration, 5 QR-iterations with shift $\kappa = 4$ are performed to reduce with linear convergence the element $\gamma_2 = 3.9981e{-16}$ and all singular values are computed.

```
sigma
2.000000099999999e0
2.000000000000000e0
1.000000099999999e0
9.999999999999994e-1
```

The results are correct, but it is obvious that the algorithm has problems with close singular values.

2. Consider the bidiagonal matrix

$$B_2 = \begin{pmatrix} 1.614874124853175 & 9.264623389167206e{-01} \\ 1.238486628039565 & 2.131595964078222e{-08} \\ 1.926281841828408 & 4.598199397802367e{-01} \\ 1.038269674236179 & \end{pmatrix}$$

with the singular values (the gap is now $1e{-8}$)

$$\begin{matrix} 2.000000010000000 \\ 2.000000000000000 \\ 1.000000010000000 \\ 1.000000000000000 \end{matrix}$$

For this matrix the Golub-Businger Procedure produces an infinite loop. The reason is that after QR-step 9, the shift is assigned the value `NaN`, as we can see in the intermediate output:

```
inspect 1
gamma
-1.6148e0  9.2646e-1  1.2384e0  2.1315e-8  1.9262e0  -4.5981e-1  -1.0382e0  0
zero shift
s and t:  0 8
inspect 2
gamma
1.9611e0  3.3759e-1  1.0198e0  7.4224e-8  1.9950e0  -1.2203e-1  1.0024e0  0
determine shift
kappa=1.000000017
s and t:  0 8
inspect 3
gamma
2.0000e0  8.7751e-9  1.9611e0  3.3742e-1  1.0198e0  -7.4612e-10  1.0000e0  0
determine shift
kappa=1.000000019
s and t:  0 8
inspect 4
gamma
2.0000e0  8.5471e-9  2.0000e0  2.3202e-9  9.9999e-1  -1.0216e-12  1.0000e0  0
determine shift
kappa=1.000000009
s and t:  0 8
inspect 5
gamma
2.0000e0  8.5471e-9  2.0000e0  7.7340e-18  9.9999e-1  -1.0216e-12  1.0000e0  0
zero shift
s and t:  4 8
inspect 6
gamma
2.0000e0  8.5471e-9  2.0000e0  9.9999e-1  9.9999e-1  1.0216e-12  1.0000e0  0
determine shift
kappa=1.000000009
s and t:  4 8
inspect 7
gamma
2.0000e0  8.5471e-9  2.0000e0  9.9999e-9  9.9999e-1  1.0216e-12  1.0000e0  0
determine shift
kappa=1.000000009
s and t:  4 8
inspect 8
gamma
2.0000e0  8.5471e-9  2.0000e0  9.9999e-9  9.9999e-1  1.0216e-12  1.0000e0  0
determine shift
kappa=1.000000024
s and t:  4 8
inspect 9
gamma
```

```
2.0000e0  8.5471e-9  2.0000e0  2.4901e-8  9.9999e-1  -2.0108e-13  1.0000e0  0
determine shift
kappa= nan
s and t:  4 8
inspect 10
gamma
2.0000e0  8.5471e-9  2.0000e0  0.0000e0  0.0000e0  0.0000e0  0.0000e0  0.0000e0
determine shift
kappa= nan
s and t:  4 8
inspect 11
gamma
2.0000e0  8.5471e-9  2.0000e0  0.0000e0  0.0000e0  0.0000e0  0.0000e0  0.0000e0
determine shift
kappa= nan
s and t:  4 8
inspect 12
gamma
2.0000e0  8.5471e-9  2.0000e0  0.0000e0  0.0000e0  0.0000e0  0.0000e0  0.0000e0
```

Why is `NaN` assigned? The shift is computed by solving a quadratic equation

```
g2:=0.5*(g0+sqrt(g0^2-4.0*g1))
```

and the expression in the square root, the discriminant, becomes negative: $-8.881784197e{-16}$

The Procedure of Golub-Reinsch has no problems with this example.

3. The bidiagonal matrix

$$B_3 = \begin{pmatrix} 1.546667895215945 & 9.673182260019585e-01 \\ 1.293102421137901 & 1.845276169487005e-15 \\ 1.984647769311140 & 2.136408344093210e-01 \\ 1.007735493887760 & \end{pmatrix}$$

has the singular values $[2, 2, 1, 1]$. The Golub-Businger Procedure needs 11 QR-steps to produce the correct result

```
1.999999999999999e0
1.999999999999999e0
1.000000000000001e0
1.000000000000000e0
```

4. We now augment the multiplicity. The bidiagonal matrix

$$B_4 = \begin{pmatrix} 1.666426845302032e & 8.846508172580001e-01 \\ 1.200172696232285e & 2.323234527365937e-15 \\ 1.927953055087120e & 4.548199770714277e-01 \\ 1.037369657276030e & 1.265849009056839e-15 \\ 1.994732361709430e & 1.255160648047072e-01 \\ 1.002640774467638e & \end{pmatrix}$$

has singular values $[1, 1, 1, 2, 2, 2]$. The Golub-Businger Procedure produces an infinite loop:

```
.
.
.
inspect 10
gamma
2.0000e0  -7.7715e-16  1.9999e0  0
determine shift
kappa=3.999999999
s and t:  0 4
inspect 11
gamma
2.0000e0  -7.7715e-16  1.9999e0  0
determine shift
kappa=3.999999999
s and t:  0 4
.
.
.
```

This happens because of the termination criterion when scanning

```
for i:=t-2 while abs(gamma [i])<= epsilon do
```

The value of `epsilon=5.664412841e-16` is smaller than
`abs(gamma[2])=7.7715e-16`. Since `gamma[2]` remains constant the iteration never ends.

The Golub-Reinsch Procedure has no problems with this example and produces the results

```
2.000000000000000e0
2.000000000000000e0
1.999999999999999e0
1.000000000000001e0
1.000000000000000e0
9.999999999999998e-1
```

5. Finally we consider a Wilkinson matrix. The matrix is tridiagonal with diagonal

$$[100, 90, \dots, 20, 10, 0, 10, 20, \dots, 90, 100]$$

and the secondary diagonals have ones:

$$\begin{pmatrix} 100 & 1 & & & & \\ 1 & 90 & 1 & & & \\ & 1 & \ddots & \ddots & & \\ & & \ddots & 80 & 1 & \\ & & & 1 & 90 & 1 \\ & & & & 1 & 100 \end{pmatrix}$$

For this matrix we have $n = 21$ and the augmented matrix is $2n = 42$. After 27 QR-steps Golub-Businger has computed 7 eigenvalues

$$\begin{array}{c} 3.000000082849189e1 \\ 2.999999917290396e1 \\ 2.000049662325264e1 \\ 1.999950657441164e1 \\ 1.009659543859793e1 \\ 9.900494253375482e0 \\ 1.970928910340472e{-1} \end{array} .$$

They are all correct; however, before the next shift the discriminant is $-1.862645149e{-9}$, again negative, and an infinite loop results.

Golub-Reinsch and MATLAB deliver the same results

`Golub − Reinsch`	`Matlab`
1.000995057466245e2	1.000995057466245e+02
1.000995057466244e2	1.000995057466245e+02
9.000049342558835e1	9.000049342558840e+01
9.000049342558833e1	9.000049342558833e+01
8.000000082709604e1	8.000000082709602e+01
8.000000082709600e1	8.000000082709597e+01
7.000000000069067e1	7.000000000069073e+01
7.000000000069064e1	7.000000000069072e+01
6.000000000000036e1	6.000000000000036e+01
6.000000000000031e1	6.000000000000033e+01
5.000000000000036e1	5.000000000000034e+01
4.999999999999966e1	4.999999999999965e+01
4.000000000069121e1	4.000000000069121e+01
3.999999999930925e1	3.999999999930925e+01
3.000000082849191e1	3.000000082849191e+01
2.999999917290397e1	2.999999917290398e+01
2.000049662325266e1	2.000049662325265e+01
1.999950657441164e1	1.999950657441164e+01
1.009659543859792e1	1.009659543859793e+01
9.900494253375477e0	9.900494253375475e+00
1.970928910340454e−1	1.970928910340456e−01

10 The Algorithm of Reinsch, 1967

Independently of the Golub-Businger Algorithm, Reinsch developed his algorithm at the same time in parallel. The first step is again to bidiagonalize the matrix $A = PBQ^\top$. P and Q are orthogonal and

$$B = \begin{pmatrix} q_1 & e_2 & & & \\ & q_2 & e_3 & & \\ & & \ddots & \ddots & \\ & & & \ddots & e_n \\ & & & & q_n \end{pmatrix}$$

Since $\lambda_k(B^\top B) = \sigma_k(B)^2$ Reinsch considers applying the QR-Algorithm with implicit shift σ to

$$T = B^\top B = \begin{pmatrix} q_1^2 & q_1 e_2 & & & \\ q_1 e_2 & e_2^2 + q_2^2 & q_2 e_3 & & \\ & q_2 e_3 & \ddots & \ddots & \\ & & \ddots & e_{n-1}^2 + q_{n-1}^2 & q_{n-1} e_n \\ & & & q_{n-1} e_n & e_n^2 + q_n^2 \end{pmatrix}$$

The first QR-transformation uses the Givens rotation G_1 such that

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^\top \begin{pmatrix} q_1^2 - \sigma \\ q_1 e_2 \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}.$$

Applying G_1 and $G_1^\top$ to $T = B^\top B$ yields

$$G_1^\top B^\top BG_1 = \begin{pmatrix} x & x & x & & & \\ x & x & x & & & \\ x & x & x & x & & \\ & & x & x & x & \\ & & & \ddots & \ddots & \ddots \end{pmatrix},$$

and subsequent Givens transformations would *chase the bulge* x till the tridiagonal form is restored.

Reinsch now had the clever idea to work with B alone! He considered

$$BG_1 = \begin{pmatrix} x & x & & & & \\ x & x & x & & & \\ & & x & x & & \\ & & & x & x & \\ & & & & \ddots & \ddots \end{pmatrix}$$

and removed the bulge x by Givens-rotations to restore *bidiagonal form*:

$$P_1^\top BG_1 = \begin{pmatrix} x & x & x & & & \\ & x & x & & & \\ & & x & x & & \\ & & & x & x & \\ & & & & \ddots & \ddots \end{pmatrix},$$

$$P_1^\top BG_1G_2 = \begin{pmatrix} x & x & & & & \\ & x & x & & & \\ & x & x & x & & \\ & & & x & x & \\ & & & & \ddots & \ddots \end{pmatrix}$$

This process to restore the bidiagonal form $\tilde{B} = P_{n-1}^\top \cdots P_1^\top BG_1 \cdots G_{n-1}$ is known as "chasing the bulge". Cleve Moler made already in 1976 a movie showing this iteration. It can be found on YouTube:

`https://www.youtube.com/watch?v=R9UoFyqJca8`.

Moler also wrote a blog about this topic:

`https://blogs.mathworks.com/cleve/2012/12/10/1976-matrix-singular-value-decomposition-film/`

It can be proved by the *implicit Q-Theorem* [3] that the transformation $B \to \tilde{B}$ is mathematically the same process as a QR-step for the tridiagonal matrix $T = B^\top B \to \tilde{B}^\top \tilde{B}$.

During the iterations an off-diagonal element e_i or a diagonal element q_i may vanish. We distinguish therefore

Splitting: If $e_i = 0$ then B splits in two bidiagonal matrices:

$$B = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}, \quad \operatorname{svd}(B) = \operatorname{svd}(B_1) \cup \operatorname{svd}(B_2).$$

The singular values of B_1 and B_2 can be computed *independently* (even in parallel).

If the split is for $i = n$ then $B_2 = q_n$ and q_n *is a singular value*. The computation is continued with with B_1.

Cancellation: If $q_i = 0$ then one of the singular values is zero. One can split B using a sequence of Givens rotations of the form $\begin{pmatrix} c & -s \\ s & c \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ r \end{pmatrix}$ to annihilate e_{i+1}

$$G_{i,i+1}^\top \begin{pmatrix} q_1 & e_2 & & & & & \\ & \ddots & \ddots & & & & \\ & & q_{i-1} & e_i & & & \\ & & & 0 & e_{i+1} & & \\ & & & & q_{i+1} & \ddots & \\ & & & & & \ddots & e_n \\ & & & & & & q_n \end{pmatrix} = \begin{pmatrix} q_1 & e_2 & & & & & \\ & \ddots & \ddots & & & & \\ & & q_{i-1} & e_i & & & \\ & & & 0 & 0 & b & \\ & & & & \tilde{q}_{i+1} & \ddots & \\ & & & & & \ddots & e_n \\ & & & & & & q_n \end{pmatrix}.$$

and to remove the bulge b by further Givens rotations $G_{i,k}^\top$, $k = i+2, \ldots, n$. Thus because e_{i+1} is rotated to zero, the matrix splits again in two submatrices.

Convergence: We cannot expect e_i or q_i to become exactly zero. Therefore we need a threshold to decide when an element is zero. Golub-Reinsch recommend (with ε=machine precision) $|e_{i+1}|, |q_i| \le \varepsilon \max_i(|q_i| + |e_i|) = \varepsilon \|B\|_1$.
Deflation: If e_n is negligible then q_n is a singular value. Continue with the iteration with the submatrix of order $n-1$.

The ALGOL procedure [5] is based on these considerations:

We compute again the singular values of our example matrix A. After 15 iterations Golub-Reinsch obtains

```
 ALGOL GolubReinsch         Matlab SVD
7.226590312008532e1       7.226590312008531e+01
4.963033918308604e1       4.963033918308606e+01
4.428869855284587e1       4.428869855284583e+01
3.642741733519198e1       3.642741733519199e+01
3.041632410657953e1       3.041632410657953e+01
2.501740101282877e1       2.501740101282877e+01
1.454503113945855e-14     1.109916271779696e-14
5.111683456877152e-15     7.702896935587551e-15
4.035934999064047e-15     5.518876944731536e-15
3.703390246973525e-15     2.522335374801936e-15
3.236570651837828e-15     1.461831916547074e-15
1.763400919991642e-15     5.592859153401258e-17
```

The results compare very well with those of MATLAB's `SVD`.

11 Computing the singular vectors

It is easy to compute the singular vectors with the Algorithms of Golub-Businger or Golub-Reinsch. The Householder- and Givens-Transformations have to be accumulated. As mentioned in [6] on page 11, Jim Wilkinson noticed that by accumulating the Givens reflections in Golub-Businger with even numbers to form the matrix X and similarly forming Y by accumulating the Givens reflections with odd numbers, the bidiagonal matrix J is diagonalized: $\Sigma = XJY^\top$.

Changing the MATLAB function `SVDGolubReinsch.m` in [3] to also compute the singular vectors is presented as an exercise in [3]. Applying `SVDGolubReinschVec.m` to our example we obtain

```
>> [U,S,V]=SVDGolubReinschVec(A);
>> [u,s,v]=svd(A);
>> [norm(U*S*V'-A) norm(u*s*v'-A)]
ans =
   1.0e-14 *

    0.1491    0.1098
```

which again compares very well with MATLAB's `SVD`.

Computing the singular vectors with the two Algorithms of Golub-Kahan is more difficult. Golub-Kahan propose in [4] several methods based on inverse iteration. The main concern is to obtain orthogonal matrices, which is not easy.

12 Historical remarks

The Algorithm of Reinsch, though also developed in 1967, was only published 1970 [5]. This delay is explained in the commentary of Ch. Reinsch in [2], *Milestones in Matrix Computation: Selected Works of Gene H. Golub, with Commentaries*, a book commemorating the 75th anniversary of Gene Golub.

The *Handbook Project* in the sixties was a first attempt to build a reliable software library for numerical computations. The editors of this multi-volume book project were F. L. Bauer, A.

S. Householder, H. Rutishauser and K. Samelson. The first volume was intended to describe the reference language and its implementation: Volumes I Part a *Description of ALGOL 60* by H. Rutishauser appeared together with Volume I Part b *Translation of ALGOL 60* by A. A. Grau, U. Hill and H. Langmaack in 1967. Volume II with the title *Linear Algebra* was supposed to contain procedures by several authors for solving systems of linear equations and eigenvalue problems. Before a contribution would be accepted by the editors, it had to be refereed and prepublished in the journal *Numerische Mathematik*. Volume II of the Handbook was edited by J. H. Wilkinson and Ch. Reinsch and appeared in 1971.

Reinsch writes about his algorithm in [2]:

> *The evaluation process of the Handbook series with approval from several international editors was a very slow process, the Internet did not exist at that time, snail mail had to be used, which forth and back across the Atlantic would take more than two weeks in the most favorable cases. Thus, the Handbook article, although finished in 1967, did appear in print not before 1970. The complex version (the reduction $A \in \mathbb{C}^{m,n} \mapsto B \in \mathbb{R}^{m,n}$) was programmed in FORTRAN by Businger and Golub and needed communication paths just within the Stanford University Campus to reach the ACM Communications editor. Therefore it overtook the Handbook article and appeared earlier in print.*

Reinsch refers here to the publication by Businger and Golub [1].

Gene Golub submitted also an algorithm for the Handbook, namely the ALGOL-Procedure of Businger in [6]. This algorithm is based on the augmented matrix by the Golub-Kahan algorithm. Reinsch writes about it in [2]:

> *To this matrix [= S] Golub and Kahan applied the symmetric version of the QR-transformation with a special shift-strategy: a conventionally chosen shift s for one QR-step and the shift with $-s$ for the next QR-step, which restores the zero-diagonal. At that time the insight into the convergence properties of the QR-iteration with shifts was not yet fully developed, so that it was overlooked how disastrous this shift technique is for a special class of tridiagonal matrices frequently occurring in physical applications. It was easy to find sample matrices with unsatisfactory convergence rate and in some cases the algorithm would not converge at all.*

Reinsch writes that after the development of his SVD-algorithm it *became very popular*. Reinsch continues in [2]:

> *The editors of the mentioned Handbook series Linear Algebra had now the choice between two proposals for a SVD routine: Gene Golub had submitted in 1967 a program based on the Golub-Kahan algorithm (1965) and there was my routine with its proven convergence properties. Their wise decision let them choose my algorithm under a joint authorship.*

This has to be specified. It is true that Golub "*submitted in 1967 a program based on the Golub-Kahan algorithm*"; it was the ALGOL program of Businger [6], which uses the Golub-Kahan transformation of the tridiagonal matrix (5).

The submission was not accepted because the program failed sometimes as we have shown earlier. Since the Golub-Businger-algorithm and the Reinsch-algorithm compute basically the same iteration, the editors of the Handbook [8] made the wise proposal to Golub and Reinsch to publish the Golub-Reinsch-algorithm as a common paper [5].

The Golub-Reinsch-algorithm for computing the singular value decomposition is one of the best, most elegant and most reliable algorithms in numerical linear algebra. The article [5] has been cited over 3000 times (as of June 2019).

Acknowledgments

I wish to thank

- *Christian Reinsch* TUM, for many discussions and many e-mails. He is a real genius.
- *Jan van Katwijk*, J.vanKatwijk@gmail.com Lazy Chair Computing. Jan did a wonderful work to produce this jff-algol program which compiles the Algol sources to C code. By doing so he revives old ALGOL procedures and made it possible for us to experiment.
- *Johann Joss*, my old fellow student from ETH. He is a gifted mathematician and computer scientist. With his help we got rid of all problems and finally we had running ALGOL procedures.
- *Å. Björck*. I am indebted to Åke for sending me a paper copy of the Stanford CS Report #73. This paper by Golub and Businger is not well known and it is hard to get hold of this report.

References

[1] Peter A. Businger and Gene H. Golub. Algorithm 358: singular value decomposition of a complex matrix [f1, 4, 5]. *Communications of the ACM*, 12:564–565, 1969.

[2] Raymond H. Chan, Chen Greif, and Dianne P. O'Leary eds. *Milestones in Matrix Computation: Selected Works of Gene H. Golub, with Commentaries.* Oxford University Press, 2007.

[3] Walter Gander, Martin J. Gander, and Felix Kwok. *Scientific Computing, an Introduction Using* MAPLE *and* MATLAB. Springer, 2014.

[4] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM. J. Numer. Anal.*, 2:202–224, 1965.

[5] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numer. Math.*, 14:403–420, 1970.

[6] Gene H. Golub and P. Businger. Least squares, singular values and matrix approximations. *Stanford Technical Report No. CS73*, 1967.

[7] J. H. Wilkinson. Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection. *Numerische Mathematik*, 4:362–367, 1962.

[8] J. H. Wilkinson and Chr. Reinsch. *Handbook for Automatic Computation, Volume II: Linear Algebra.* Springer, 1971.