Numerical Algorithms (2023) 93:1137–1155
https://doi.org/10.1007/s11075-022-01459-9

ORIGINAL PAPER

Singular value decomposition in extended double precision arithmetic

Christian Reinsch[1] · **Mathias Richter**[2]

Received: 28 October 2022 / Accepted: 15 November 2022 / Published online: 3 December 2022

Abstract
A well-known and successful algorithm to compute the singular value decomposition (SVD) of a matrix was published by Golub and Reinsch (*Numer. Math.* 14:403–420, 1970), together with an implementation in Algol. We give an updated implementation in extended double precision arithmetic in the C programming language. Extended double precision is native for Intel x86 processors and provides improved accuracy at full hardware speed. The complete program for computing the SVD is listed. Additionally, a comprehensive explanation of the original algorithm of Golub and Reinsch (*Numer. Math.* 14:403–420, 1970) is given at an elementary level without referring to the more general results of Francis (*Comput. J.* 4:265–271, 1961, 1962).

Keywords Singular value decomposition · Extended precision arithmetic · Implicit QR iteration

Mathematics Subject Classification (2010) 65F25

1 Introduction

The IEEE floating point standard, chiefly designed by William Kahan, recommends that hardware implementations should provide an extended precision data format. This is put into effect for x86-64 processors, which provide a native 80-bit extended double precision data format. Higher precisions will be provided in the future [4, p. 43]. In 80-bit extended double precision data format, the significand comprises 64 bits and thus 11 bits more than does the standard IEEE double precision format. In the

✉ Mathias Richter
mathias.richter@unibw.de

[1] Department of Mathematics, Technical University of Munich, Munich, Germany

[2] Department of Electrical Engineering and Information Technology, Universität der Bundeswehr München, München, Germany

C programming language, the type specifier `long double` is reserved for declaration of extended double precision variables. Some compilers, like the GNU compiler, Clang, and Intel's C/C++ compiler, implement `long double` using 80-bit extend double precision numbers on x86-architectures. Extended double precision thus becomes easily available for computations and should be used, as it offers improved accuracy at full computational speed. We give a C implementation of the singular value decomposition (SVD) in extended double precision arithmetic. The program ist based on the algorithm published by Golub and Reinsch in [3].

Let $A \in \mathbb{R}^{m,n}$ a real $m \times n$ matrix, let $\ell := \min\{m, n\}$. It is well known that

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

where $U \in \mathbb{R}^{m,m}$, $\Sigma \in \mathbb{R}^{m,n}$, and $V \in \mathbb{R}^{n,n}$ such that

$$U^T U = I_m, \quad V^T V = I_n \quad \text{and} \quad \Sigma_{i,i} = \sigma_i, \ i = 1, \ldots, \ell,$$

all other elements of Σ being zero. The numbers $\sigma_1, \ldots, \sigma_\ell$ are the non-negative square roots of the ℓ largest eigenvalues of $A^T A$. We shall assume that

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_\ell \geq 0.$$

The computation of U, V, and Σ splits into two parts. The first part is a reduction of A to a bidiagonal matrix. The second part is an implicit application of the QR algorithm (with shifts) to this bidiagonal matrix. We describe bidiagonalization in Section 2. Section 3 is a reminder about the QR algorithm for tridiagonal symmetric matrices and gives an elementary explanation of Francis' method to use implicit shifts. Section 4 explains how QR steps can implicitly be performed on a bidiagonal matrix. Section 5 gives details on how to compute the shift parameter reliably and Section 6 gives a stopping criterion for the QR iteration. Section 7 outlines how to arrange for computations in extended precision and in Section 8, a complete C function for computing the SVD is given. Test cases are investigated in Section 9 and we conclude in Section 10.

2 Bidiagonalization

$A = (a_{i,j})$ is a real matrix with m rows $i = 0, \ldots, m-1$, counting down from top to bottom, and with n columns $j = 0, \ldots, n-1$, counting across from left to right.

With the pair of rows 0 and $i = 1, \ldots, m-1$ we do the **plane rotations from left**

$$a_{0,j} = \cos \cdot a_{0,j} \ + \ \sin \cdot a_{i,j}, a_{i,j} = -\sin \cdot a_{0,j} \ + \ \cos \cdot a_{i,j}, \ j = 0, \ldots, n-1$$

overwriting the old with the new values.[1] We choose cos and sin as follows: let $p := a_{0,0}$, $q := a_{i,0}$, $r := \pm\sqrt{p^2 + q^2}$ with same sign as p, $\cos = p/r$ and $\sin = q/r$. ($\cos = 1, \sin = 0$ if $q = 0$). Then $a_{i,0}$ gets *eliminated*, i.e., its new value is 0. Note that plane rotations from left leave the sum of squares in a *column* invariant. Thus while $a_{i,0}$ gets annihilated, $a_{0,0}^2$ gets bigger. Indeed, the erasing is more some sort of

[1] $P_i^T A$ with the orthogonal $m \times m$ matrix $P_i^T = \begin{pmatrix} \cos & \sin \\ -\sin & \cos \end{pmatrix}$ in rows/columns 0 and i

collecting. After $i = m-1$, $a_{0,0}^2$ is positive unless A has a zero column. In the same way we do with the pair of columns 1 and $j = 2, \ldots, n-1$ **plane rotations from right**

$$a_{i,1} = a_{i,1} \cdot \cos + a_{i,j} \cdot \sin, a_{i,j} = -a_{i,1} \cdot \sin + a_{i,j} \cdot \cos, i = 0, \ldots, m-1$$

again overwriting the old with the new values.[2] Thus we do not only generate zeros in the leftmost column but also zeros in the top row after the first two entries, which here are called d_0 and e_1. Plane rotations from right leave the sum of squares in a *row* invariant, thus $d_0^2 + e_1^2 > 0$ unless there is a zero row.

Such plane rotations either from left or from right in order to erase a selected matrix entry are called **Givens rotations**. Together, we have the *main step number one*. More such main steps follow, all a repetition of the first one, but each time with another column and another row less.

Let us assume for the moment that $m \geq n$. Then, with $n-1$ such main steps we can reduce the matrix A to **bidiagonal** form B. Ignoring the trivial (zero) rows $i = n, \ldots, m-1$, one has

$$B = \begin{bmatrix} d_0 & e_1 & & & & \\ & d_1 & e_2 & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & \\ & & & & d_{n-2} & e_{n-1} \\ & & & & & d_{n-1} \end{bmatrix} \tag{2}$$

With $e_0 := 0$ each column of B has the same form. The remaining B is quadratic with $\det(B) = d_0 \cdots d_{n-1}$ and $\det(B^T B) = d_0^2 \cdots d_{n-1}^2$.

If required, we accumulate all plane rotations from left in U^T and all plane rotations from right in V. These are orthogonal matrices of order m and n. It is helpful for understanding the coming steps to extend the $m \times n$ matrix A to the right by the $m \times m$ unit matrix I_m and down by the $n \times n$ unit matrix I_n. With these two extensions we have a working area with $m+n$ rows/columns of length $m+n$. At the beginning it is

$$\begin{pmatrix} A & I_m \\ I_n & \end{pmatrix}$$

where $A = U B V^T$. After the plane rotations from left and right it is

$$\begin{pmatrix} B & U^T \\ V & \end{pmatrix}$$

All steps of the second part of the algorithm (to be described below) follow this scheme of applying plane rotations from left and right to further reduce B to diagonal form Σ while updating U^T and V.

The case $m < n$ had been omitted in the Algol program published in [3], but is included now. It is best to consider A^T and to apply the algorithm to the $n \times m$

[2] $A\,Q_j$ with the orthogonal $n \times n$ matrix $Q_j = \begin{pmatrix} \cos & -\sin \\ \sin & \cos \end{pmatrix}$ in rows/columns 1 and j

matrix A^T. Anyway, as described in Section 7, a copy of A in extended precision format is needed. At the moment this copy is created, one may as well copy A^T. The C function given in Section 8 does so and then computes V^T instead of U^T and U instead of V. It is easy to take this swap into account when returning the matrices U and V. Therefore, we may continue to assume $m \geq n$ in the following.

The present version of the SVD uses exclusively plane rotations. All Householder reflections from the version of 1967 [3] have been replaced, although they need only half the number of multiplications. But plane rotations will be needed in the second part of the algorithm and can all be done by calling two functions, viz.

`PREPARE()` to compute the desired values cos and sin for the next plane rotation,
`ROTATE()` to apply this plane rotation to a pair of *BLA*-vectors $x[\,]$, $y[\,]$ (defined by equidistant storage in memory, for example matrix rows, columns, diagonals, or single elements).

As said above, the second part of the algorithm will consist in applying further plane rotations from left and right to B with the goal to reduce B to a diagonal matrix. When $e_j = 0$ occurs for some index $j > 0$ (iteration indices are dropped), then the matrix B can be split into two bidiagonal submatrices of order j (rows $0, \ldots, j-1$ of B) and $n - j$ (rows $j, \ldots, n-1$ of B), which may be diagonalized independently of each other. At any time, the second part of the algorithm will iteratively diagonalize the southernmost remaining bidiagonal submatrix of B with non vanishing off-diagonal elements. The position of this submatrix is described by two indices ℓ and k, both in the range $0, \ldots, n-1$, defining three diagonal blocks of B, as illustrated in (3):

$$B = \left[\begin{array}{c|c|c} \begin{vmatrix} d_0 & e_1 & & \\ & d_1 & \cdot & \\ & & \cdot & e_{\ell-1} \\ & & & d_{\ell-1} \end{vmatrix} & & \\ \hline & \begin{vmatrix} d_\ell & e_{\ell+1} & & \\ & d_{\ell+1} & \cdot & \\ & & \cdot & e_k \\ & & & d_k \end{vmatrix} & \\ \hline & & \begin{vmatrix} d_{k+1} & \cdot & \\ & \cdot & 0 \\ & & d_{n-1} \end{vmatrix} \end{array}\right] \tag{3}$$

Here, $\ell = 0$ means an empty first block, and $k = n-1$ means an empty third block. The third "bidiagonal" block in fact already is a diagonal matrix and can be ignored for all further computations, $|d_{k+1}|, \ldots, |d_{n-1}|$ are singular values. The middle block with *lower* row index ℓ and *upper* row index k is characterized by non vanishing off-diagonal elements $e_{\ell+1}, \ldots, e_k$ and can be diagonalized independently of the upper

bidiagonal block (rows $0, \ldots, \ell-1$ of B).[3] The elements $e_1, \ldots, e_{\ell-1}$ may be zero or not. After each iteration step taken in the second part, k and ℓ are updated by scanning for zero elements e_j. When $k = \ell = 0$, then B is completely diagonalized. When $\ell = k > 0$, then $e_k = 0$, $|d_k|$ is a singular value and the trivial 1×1 matrix (e_k) may be split off the middle block in (3), thus k will be decremented by 1. When $\ell < k$, then one continues to operate on the bidiagonal submatrix of B in rows $\ell, \ldots, k$, which will be denoted by

$$B_{\ell,k} = \begin{bmatrix} d_\ell & e_{\ell+1} & & & & \\ & d_{\ell+1} & e_{\ell+2} & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & \\ & & & & d_{k-1} & e_k \\ & & & & & d_k \end{bmatrix}. \tag{4}$$

In practice, zero tests must be replaced by tests $|e_j| \le tol$. A proper choice of tol will be discussed in Section 6.

3 Symmetric QR steps with shifts

This section is a reminder about one of the most successful algorithms of Linear Algebra, the **QR iteration** for the diagonalization of matrices. Heinz Rutishauser invented it in 1961 (then in the form of an equivalent LR iteration). The name comes from the basic iteration step for the matrix to be diagonalized, which is here $X =: X^{(0)}$:

$$X^{(i)} =: Q^{(i)} R^{(i)} \text{ and } R^{(i)} Q^{(i)} =: X^{(i+1)},\ Q^{(i)} \text{ orthogonal},\ R^{(i)} \text{ upper triangular.}$$

The step from $X^{(i)}$ to $X^{(i+1)}$ is a similarity transformation: $X^{(i+1)} = Q^{(i)T} X^{(i)} Q^{(i)}$. All $X^{(i)}$ have the same eigenvalues. The absolute values of these invariant eigenvalues, arranged in decreasing order, play a decisive role in analyzing the asympotic behavior of the sequence $X^{(i)}$. The QR iteration will be applied to the symmetric tridiagonal matrix

$$X := B^T B,$$

where B is initially given by (2) and more generally by (4), but in this section it is considered in its own right. To describe a single QR step applied to a tridiagonal symmetric matrix, the iteration index $^{(i)}$ is dropped. Let

$$X = X^{(i)} = \begin{bmatrix} \delta_0 & \varepsilon_1 & & & & \\ \varepsilon_1 & \delta_1 & \varepsilon_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \varepsilon_{n-2} & \delta_{n-2} & \varepsilon_{n-1} \\ & & & & \varepsilon_{n-1} & \delta_{n-1} \end{bmatrix} \quad \text{and} \quad \overline{X} = X^{(i+1)}.$$

[3] Formally, k is defined as the lowest row index such that $e_{k+1} = \ldots = e_{n-1} = 0$ (initially $k = n-1$) and ℓ is defined as the highest row index with $\ell \le k$ and $e_\ell = 0$. Since $e_0 = 0$, ℓ must exist for any k.

As a consequence of the tridiagonal form, each QR step is rather special:

(1) Q is a product of $n-1$ rotations Q_j in plane $(j-1, j)$, $j = 1, \ldots, n-1$, to eliminate ε_j,
(2) X stays symmetric and tridiagonal (the minor steps are $X \to Q_j^T X Q_j$).
(3) The number of operations per step is only $O(n)$ rather than $O(n^3)$.
(4) The eigenvalues of X are real and non-negative.
(5) Also the upper triangular matrix R has only three non-trivial diagonals, see Fig. 1.

The basic iteration step for the QR algorithm *with shifts* is described as

$$X - sI =: QR \text{ and } RQ + sI =: \overline{X},$$

where Q and R are orthogonal and upper triangular, respectively, but not the same as above, depending on the value of s. As before, one has $\overline{X} = Q^T X Q$. The value s is called the **shift**. It should be chosen close to one of the eigenvalues of X. The closer the shift is to such an eigenvalue, the smaller is some diagonal element of R (in most cases the last one) and the smaller is the (last) row and column of $\overline{X}$. Good shifts make Rutishauser's LR or QR iteration so successful. In [5] Wilkinson proved global convergence of the sequence $X^{(i)}$ to a diagonal matrix for two different strategies to choose a shift. The convergence rate usually is cubic, i.e., for sufficiently small η follows from $|\varepsilon_{n-1}| \le \eta$ that the next iteration will reduce $|\varepsilon_{n-1}|$ to $O(|\varepsilon_{n-1}|^3)$. We devote Section 5 to Wilkinson's idea and proposals for the shift.

The factorization $X - sI = QR$ and the recombination $QR + sI = \overline{X}$ may be combined to what is known as a QR step with **implicit shift**. This was found by Francis [1] in a much more general setting, but shall be explained here on an elementary level. Assume that no off-diagonal element of X vanishes, i.e., $\varepsilon_j \neq 0$ for $j = 1, \ldots, n-1$. In this case, the orthogonal matrix Q is unique up to a scaling of its columns by ± 1. The matrix Q can always be written as a product of $n-1$ rotations Q_j in plane $(j-1, j)$, $j = 1, \ldots, n-1$. The first rotation Q_1^T is to be chosen such that (schematically, just noting the upper 2×2-minors)

$$Q_1^T (X - sI) = \begin{pmatrix} \cos & \sin \\ -\sin & \cos \end{pmatrix} \cdot \begin{pmatrix} \delta_0 - s & \varepsilon_1 \\ \varepsilon_1 & \delta_1 - s \end{pmatrix} = \begin{pmatrix} * & * \\ 0 & * \end{pmatrix},$$

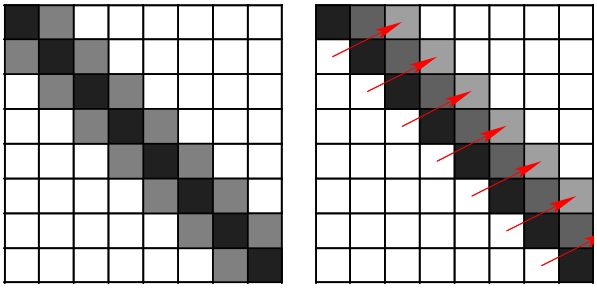

Fig. 1 $X = B^T B$ before elimination and R after elimination

which is achieved by

$$\cos = \frac{\delta_0 - s}{\sqrt{(\delta_0 - s)^2 + \varepsilon_1^2}}, \quad \sin = \frac{\varepsilon_1}{\sqrt{(\delta_0 - s)^2 + \varepsilon_1^2}} \tag{5}$$

(or the negatives of these values). Since $\varepsilon_1 \neq 0$, there cannot be a zero division in (5) and moreover $\sin \neq 0$. It is known that

$$\overline{X} = Q_{n-1}^T \cdots \underbrace{\left(Q_i^T \cdots Q_1^T \cdot A \cdot Q_1 \cdots Q_i\right)}_{=: X_i} \cdots Q_{n-1}. \tag{6}$$

is tridiagonal, but the matrices X_i, $i = 1, \ldots, n-2$, deviate from the tridiagonal form. Schematically, one has

$$X_1 = Q_1^T X Q_1 = \begin{pmatrix} * & \varepsilon_1' & \beta_2 & & & \\ \varepsilon_1' & * & * & & & \\ \beta_2 & * & * & \varepsilon_3 & & \\ & & \varepsilon_3 & * & * & \\ & & & * & * & * \\ & & & & * & * \end{pmatrix}$$

The extra element β_2 is called **bulge element**. It is defined by $\beta_2 = \sin \cdot \varepsilon_2$ with sin from (5), so that $\beta_2 \neq 0$. Now choose a rotation $\tilde{Q}_2^T$ in plane $(1, 2)$, such that multiplication of X_1 from the left with $\tilde{Q}_2^T$ eliminates β_2 in position $(2, 0)$. This is achieved by the choice

$$\cos = \frac{\varepsilon_1'}{\sqrt{(\varepsilon_1')^2 + \beta_2^2}}, \quad \sin = \frac{\beta_2}{\sqrt{(\varepsilon_1')^2 + \beta_2^2}}. \tag{7}$$

Schematically, one gets

$$\tilde{X}_2 = \tilde{Q}_2^T X_1 \tilde{Q}_2 = \begin{pmatrix} * & * & & & & \\ * & * & \varepsilon_2' & \beta_3 & & \\ & \varepsilon_2' & * & * & & \\ & \beta_3 & * & * & * & \\ & & & * & * & * \\ & & & & * & * \end{pmatrix}.$$

The bulge β_2 moved down one position along the diagonal to become $\beta_3 = \sin \cdot \varepsilon_3$ with the value sin defined in (7). Since $\beta_2 \neq 0$, one has $\sin \neq 0$ again and therefore $\beta_3 \neq 0$. Continuing this way chasing down the bulge along the diagonal, one arrives at

$$\tilde{X}_{n-2} = \tilde{Q}_{n-2}^T \tilde{X}_{n-3} \tilde{Q}_{n-2} = \begin{pmatrix} * & * & & & & \\ * & * & * & & & \\ & * & * & * & & \\ & & * & * & * & \beta_{n-1} \\ & & & * & * & * \\ & & & \beta_{n-1} & * & * \end{pmatrix}$$

again with $\beta_{n-1} \neq 0$. A last rotation $\tilde{Q}_{n-1}^T$ in plane $(n-2, n-1)$ is chosen such that multiplication of $\tilde{X}_{n-2}$ from the left with $\tilde{Q}_{n-1}^T$ eliminates β_{n-1} in position $(n-1, n-3)$. Then $\tilde{X}_{n-1} = \tilde{Q}_{n-1}^T \tilde{X}_{n-2} \tilde{Q}_{n-1}$ is tridiagonal again. Because all bulges are non-zero, all rotations $\tilde{Q}_i$, $i = 2, \ldots, n-1$, are uniquely determined (up to phase factors, i.e., up to multiplications by ± 1). This means that once Q_1 is set, the need to chase down the bulge until it disappears and to re-establish the tridiagonal form of

$$\tilde{X}_{n-1} = \tilde{Q}_{n-1}^T \cdots \tilde{Q}_2^T \cdot Q_1^T \cdot X \cdot Q_1 \cdot \tilde{Q}_2 \cdots \tilde{Q}_{n-1} \tag{8}$$

uniquely determines $\tilde{Q}_2, \ldots, \tilde{Q}_{n-1}$ (up to phase factors). But tridiagonal form will also be established by using the rotations $Q_2, \ldots, Q_{n-1}$ as in (6). The conclusion is that $\tilde{Q}_i = Q_i$, $i = 2, \ldots, n-1$ (up to phase factors). The choice of Q_1 according to (5) — which is the only place where the shift explicitly enters the computation — and the choice of $\tilde{Q}_2, \ldots, \tilde{Q}_{n-1}$ such as to chase down the bulge, will make (8) an implementation of the QR step.

Thus, the symmetric QR step with shift for the matrix $X = B^T B$ can be done as follows:

(1) Choose a good shift s.
(2) Choose Q_1^T according to (5) and compute $X_1 = Q_1^T X Q_1$.
(3) Choose rotations $\tilde{Q}_j$ in plane $(j-1, j)$ in order to chase down the bulge and successively build $X_j := \tilde{Q}_j^T X_{j-1} \tilde{Q}_j$, $j = 2, \ldots, n-1$. $\overline{X} := X_{n-1}$ is the next matrix X.

4 Implicit QR steps on bidiagonal matrix

The QR iteration could explicitly be applied to the matrix $X = B^T B$ with $B = B_{\ell,k}$ from (4) — the double index of $B_{\ell,k}$ will be dropped now. The non-trivial diagonal and off-diagonal entries of X then are

$$\begin{aligned} X_{j,j} &= d_j^2 + e_j^2, && (j = \ell, \ldots, k) \quad \text{and} \\ X_{j-1,j} &= X_{j,j-1} = d_{j-1} e_j, && (j = \ell+1, \ldots, k). \end{aligned} \tag{9}$$

However, for reasons of economy and accuracy, one should avoid dealing with $B^T B$. Rather, it is preferable to work with B alone. In fact it is possible to do the similarity transformations on X with *two-sided* plane rotations, viz. $B \to P_j^T B\, Q_j$, $j = \ell, \ldots, k-1$, where all matrices Q_j and P_j^T are Givens rotations. To be precise, assume that

$$d_{j-1} \neq 0 \quad \text{and} \quad e_j \neq 0 \quad \text{for} \quad j = \ell+1, \ldots, k, \tag{10}$$

— according to (9) this means that X has no vanishing off-diagonal elements — and choose Q_ℓ as if performing the first minor step of one QR iteration step (with shift)

for X. From (5) and (9) it can be seen that this means to choose Q_ℓ as a rotation in the plane $(\ell, \ell+1)$ with

$$\cos = \frac{d_\ell^2 - s}{\sqrt{(d_\ell^2 - s)^2 + (d_\ell e_{\ell+1})^2}}, \quad \sin = \frac{d_\ell e_{\ell+1}}{\sqrt{(d_\ell^2 - s)^2 + (d_\ell e_{\ell+1})^2}} \tag{11}$$

(or the negatives of these values), depending on the shift parameter s. Multiplying B with Q_ℓ from the right gives (schematically)

$$B'_{\ell+1} := BQ_\ell = B \cdot \begin{pmatrix} \cos & -\sin \\ \sin & \cos \end{pmatrix} = \begin{pmatrix} d'_\ell & * & & & \\ b'_{\ell+1} & * & e_{\ell+2} & & \\ & & * & * & \\ & & & * & * \\ & & & & * \end{pmatrix}. \tag{12}$$

$B'_{\ell+1}$ is no longer an upper bidiagonal matrix because of the bulge element $b'_{\ell+1} = \sin{\cdot} d_{\ell+1}$. The bulge element cannot vanish, since $\sin \neq 0$ and $d_{\ell+1} \neq 0$, see (10). To eliminate $b'_{\ell+1}$, multiply $B'_{\ell+1}$ from the left by a rotation P_ℓ^T in the plane $(\ell, \ell+1)$. Up to a phase factor, this rotation must be given by

$$\cos = \frac{d'_\ell}{\sqrt{(d'_\ell)^2 + (b'_{\ell+1})^2}}, \quad \sin = \frac{b'_{\ell+1}}{\sqrt{(d'_\ell)^2 + (b'_{\ell+1})^2}},$$

such that (schematically)

$$B_{\ell+1} := P_\ell^T \cdot B'_{\ell+1} = \begin{pmatrix} \cos & \sin \\ -\sin & \cos \end{pmatrix} \cdot B'_{\ell+1} = \begin{pmatrix} * & * & b_{\ell+1} & & \\ 0 & * & * & & \\ & & d_{\ell+2} & * & \\ & & & * & * \\ & & & & * \end{pmatrix}. \tag{13}$$

The bulge has moved and becomes $b_{\ell+1} = \sin{\cdot} e_{\ell+2}$. Since $b'_{\ell+1} \neq 0$, also $b_{\ell+1} \neq 0$. To eliminate $b_{\ell+1}$, multiply from the right with a rotation $\tilde{Q}_{\ell+1}$ like in (12), but now in the plane $(\ell+1, \ell+2)$. Because of $b_{\ell+1} \neq 0$, this rotation must have a component $\sin \neq 0$. The multiplication results in

$$B'_{\ell+2} := B_{\ell+1}\tilde{Q}_{\ell+1} = \begin{pmatrix} * & * & 0 & & & \\ & * & * & & & \\ & b'_{\ell+2} & * & e_{\ell+3} & & \\ & & & * & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

with $b'_{\ell+2} = \sin{\cdot} d_{\ell+2} \neq 0$. To eliminate the bulge $b'_{\ell+2}$, multiply from the left by a rotation $P_{\ell+2}^T$ in the plane $(\ell+1, \ell+2)$. Since $b'_{\ell+2} \neq 0$, this rotation once more is

uniquely determined (up to a phase factor) with $\sin \neq 0$. One gets

$$B_{\ell+2} := P_{\ell+1}^T \cdot B'_{\ell+2} = \begin{pmatrix} * & * & & & & \\ & * & * & b_{\ell+2} & & \\ & 0 & * & * & & \\ & & & d_{\ell+3} & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

with $b_{\ell+2} = \sin \cdot e_{\ell+3} \neq 0$. Continuing this way the bulge initially introduced by multiplication with Q_ℓ is chased down in knight's moves until it disappears. This is illustrated in Fig. 2, where the bulge at different moments in the computation is notated as x, x', x'', and x'''. All moves in north-eastern direction are effected by plane rotations from left and all moves in south-western direction are effected by plane rotations from right. The rotations $\tilde{Q}_{\ell+1}, \ldots, \tilde{Q}_{k-1}$ from the right and $P_\ell^T, \ldots, P_{k-1}^T$ from the left are all uniquely determined (up to phase factors) once Q_ℓ is chosen. In the end, with

$$\tilde{Q} := Q_\ell \cdot \tilde{Q}_{\ell+1} \cdots \tilde{Q}_{k-1} \quad \text{and} \quad P := P_\ell \cdot \ldots \cdot P_{k-1}$$

the matrix $P^T B \tilde{Q} =: \overline{B}$ is bidiagonal again. Therefore

$$(\overline{B})^T \overline{B} = \tilde{Q}^T B^T B \tilde{Q} = \tilde{Q}^T X \tilde{Q}$$

is a tridiagonal matrix again. The first rotation Q_ℓ was explicitly chosen as needed to perform a step of the QR iteration. It was seen in Section 3, that this and the fact that $\tilde{Q}^T X \tilde{Q}$ is tridiagonal are enough to conclude that $\tilde{Q}^T X \tilde{Q}$ is the result of a full QR step performed on X.

The conclusion is that one step of the QR iteration (with shift s) for $X = B^T B$ can be done implicitly as follows

(1) Depending on the chosen shift s set Q_ℓ as in (11), and multiply B from the right with Q_ℓ.

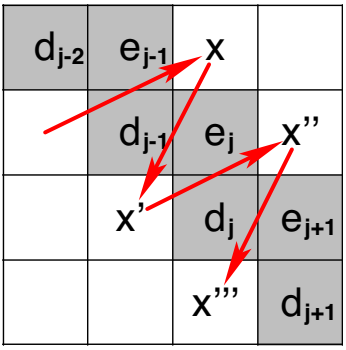

Fig. 2 Chasing the bulge

(2) Multiply with further rotations $\tilde{Q}_{\ell+1}, \ldots, \tilde{Q}_{k-1}$ and $P_\ell^T, \ldots, P_{k-1}^T$ to chase down the bulge until it disappears.

The above arguments depend on (10), but a violation of these conditions even is an advantage, since then savings are possible. In case $|e_k| \le tol$, $|d_k|$ is accepted as a singular value and k is decremented by 1. In case $|e_i| \le tol$ for some $\ell \le i < k$, e_i is neglected and the matrix B is split in two bidiagonal matrices which are diagonalized separately. The saving on $d_i = 0$ is almost as obvious: one can chase this zero to the right with plane rotations $\hat{P}_{i+1}^T, \ldots, \hat{P}_k^T$ from left in rows i and $j = i+1, \ldots, k$, so that the matrix B gets a zero row:

$$\begin{bmatrix} * & * & & & \\ & * & * & & \\ & & 0 & e_{i+1} & \\ & & & * & * \\ & & & & * \end{bmatrix} \xrightarrow{\hat{P}_{i+1}^T \cdot} \begin{bmatrix} * & * & & & \\ & * & * & & \\ & & 0 & 0 & x \\ & & & * & * \\ & & & & * \end{bmatrix} \xrightarrow{\hat{P}_{i+2}^T \cdot}$$

$$\cdots \xrightarrow{\hat{P}_{k-1}^T \cdot} \begin{bmatrix} * & * & & & \\ & * & * & & \\ & & 0 & \cdots & 0 \quad x' \\ & & & * & * \\ & & & & * \end{bmatrix} \xrightarrow{\hat{P}_k^T \cdot} \begin{bmatrix} * & * & & & \\ & * & * & & \\ & & 0 & \cdots & 0 \\ & & & * & * \\ & & & & * \end{bmatrix}.$$

Now B can be split in two bidiagonal matrices to be diagonalized separately. In case $|d_i| \le tol$, the effect is the following. Multiplication with $\hat{P}_{i+1}^T$ annihilates e_{i+1} and creates new elements x_{i+1} in position $(i, i+2)$ and η_{i+1} in position $(i+1, i)$. Multiplication with $\hat{P}_{i+2}^T$ annihilates x_{i+1} (in row i) and creates new elements x_{i+2} in position $(i, i+3)$ and η_{i+2} in position $(i+2, i)$. Multiplication with $\hat{P}_k^T$ annihilates x_{k-1} (in row i) and creates η_k in position (k, i). The result is a matrix of the form (all modified elements are overlined)

$$B = \begin{bmatrix} d_\ell & e_{\ell+1} & & & & & & & \\ & \ddots & \ddots & & & & & & \\ & & \ddots & \ddots & & & & & \\ & & & \ddots & \bar{e}_i & & & & \\ & & & & \bar{d}_i & 0 & & & \\ & & & & \eta_{i+1} & \bar{d}_{i+1} & \bar{e}_{i+2} & & \\ & & & & & \ddots & \ddots & \ddots & \\ & & & & & & \ddots & \ddots & \bar{e}_k \\ & & & & & & & & \bar{d}_k \end{bmatrix}.$$

By the orthogonality of plane rotations

$$\bar{d}_i^2 + \eta_{i+1}^2 + \ldots + \eta_k^2 = d_i^2 \le tol^2,$$

which shows that all elements $\bar{d}_i, \eta_{i+1}, \ldots, \eta_k$ have a magnitude less than or equal tol and can be neglected. B can again be split in two bidiagonal matrices. The case of a vanishing or negligible diagonal element d_i is called **cancellation**.

5 Choosing a shift

The way shifts are chosen is an essential and characteristic part of the SVD. The shift has only one purpose: to decrease $|e_k|$, the last off-diagonal entry in the current iteration matrix $B = B_{\ell,k}$ (again, the double index (ℓ, k) will be dropped). Remember, that a good shift is close to one of the *eigenvalues* of the matrix B^TB, which are the *squares* of the singular values. The non-trivial elements in the last two columns of B are

$$\begin{pmatrix} e_{k-1} & 0 \\ d_{k-1} & e_k \\ 0 & d_k \end{pmatrix} =: \begin{pmatrix} g & 0 \\ y & h \\ 0 & z \end{pmatrix}$$

One may assume that g, y, h and z are all non-zero, otherwise B would have been split. One of the simplest and most effective choices for the shift is the last diagonal element of B^TB, which is h^2+z^2, one of the **Wilkinson shifts**, also known as **Rayleigh shift**. An alternative, used in the original SVD of 1967, is to choose one eigenvalue of the last 2×2 diagonal block of B^TB. This choice was originally designed for the cases where one needs a pair of conjugate complex shifts. However, if one needs just *one* shift, it is mandatory to select the eigenvalue which is closer to h^2+z^2 because the other one could be far away. (Indeed this *had* been done in [3], but not been pointed out sufficiently.) Then the shift s comes from the quadratic

$$\det \begin{pmatrix} g^2+y^2-s & hy \\ hy & h^2+z^2-s \end{pmatrix} = 0.$$

We want the root s which is closer to h^2+z^2, i.e., the root $t := h^2+z^2-s$ closest to zero of the quadratic

$$t^2 + (g^2+y^2-h^2-z^2)\, t - h^2y^2 = 0.$$

It is good numerical practice to first compute the larger root t_1 and then the smaller one t_2 from $t_1t_2 = -h^2y^2$. As h and y are non-zero one can form

$$f := (g^2+y^2-h^2-z^2)/(2hy) = [(y-z)(y+z)+(g-h)(g+h)]/(2hy).$$

Let $w = \sqrt{f^2+1}$, then t has the roots $hy\,(-f \pm w)$. If $f > 0$, then $yh(-f-w)$ is the one with bigger modulus and $t = yh/(f+w)$ is the one with smaller modulus. If $f < 0$, then $yh(-f+w)$ is the one with bigger modulus and $yh/(f-w)$ is the one with smaller modulus. The shift to be chosen thus is

$$s = z^2+h^2-hy/(f+w) \quad \text{or} \quad s = z^2+h^2-hy/(f-w),$$

for $f > 0$ or $f < 0$, respectively. The only place where the shift enters into the computation of the implicit QR step is (11). Evidently the values for cos and sin in (11) do not change if $d_\ell^2 - s$ is replaced by $d_\ell - s/d_\ell$ and if $d_\ell e_{\ell+1}$ is replaced by $e_{\ell+1}$. This fact will be used in the program of Section 8.

For this second choice of shift (called shift strategy (b) in [5]), Wilkinson proved global convergence with a guaranteed quadratic convergence rate. Almost always convergence is cubic, as confirmed by the test cases examined in Section 9. One therefore expects $s \to z^2$ from $h \to 0$ and y bounded. After convergence, k is decremented until with enough QR steps $k = 0$ is reached.

6 Test for convergence

When $|e_k| \leq tol$, then $|d_k|$ is accepted as new singular value and k is decremented by 1. When $|e_i| \leq tol$ for some $i < k$, then the matrix B is split in two parts and the singular values of both parts are computed separately, as said above. The cancellation case $|d_i| \leq tol$ was considered in Section 4. It remains to choose the tolerance tol. Here, we repeat the proposal from [3] to choose

$$tol = \|B\|_\infty \cdot \epsilon_{\text{mach}} \tag{14}$$

with ϵ_{mach} the machine precision and B the initial bidiagonal matrix from (2).

7 Extended precision arithmetic

On the preceding pages all algorithmical aspects of computing the real matrices A, U, V, Σ and B have been described without mentioning their common data type. Now we are specific. We assume that a user keeps A stored as a variable in standard IEEE `double` floating point format and wants to get U, V, and Σ also as variables in `double` format. We call the corresponding variables *external* and use lower case letters to notate them. We perform the SVD computation inside a C function `SVD` in `long double` extended precision format. This function needs *internal* copies of data type `long double` of the external variables. The internal variables shall be denoted by upper case letters. We thus have

matrix	external variable	internal variable
A	`a`	`A`
U	`u`	`P` or `Q`
V	`v`	`Q` or `P`
Σ	`d`	`D`
B	–	`D`, `E`

where the dash in the last line means that B is only computed internally. Its diagonal is stored in `D` (which will be overwritten by the singular values on the diagonal of Σ) and its superdiagonal is stored in `E`. Explanations for the internal variables `P` and `Q` will be given in a moment.

When `SVD` is called, at first the C library function `malloc` is invoked to dynamically allocate memory for the internal variables. Then matrix A is copied from `a` to `A`, if $m \geq n$. If, however, $m < n$, then A^T is copied to `A`, as announced in Section 2.

With

$$ma := \max\{m, n\}, \quad mi := \min\{m, n\},$$

A can always be considered a $ma \times mi$ matrix. All rotations from left will be accumulated in P (of dimension $ma \times ma$) and all rotations from right will be accumulated in Q (of dimension $mi \times mi$). As seen in Section 2, at the end of the computation, P holds U^T and Q holds V, if $m \geq n$. If, however, $m < n$, then Q holds U and P holds V^T. Of course, P and Q must be initialized as identity matrices before the accumulations can start.

As already said in the introduction, on the Intel x86 architecture all arithmetic operations are done in 64 bit precision with full hardware speed. Also, the exponent of `long double` has 15 bits and thus four more bits than are provided for `double`. Therefore, there will be no problems with exponent overflow or underflow when squaring elements $a_{i,j}$ of A.

At the end of the computation in function SVD, the internal variables are rounded to `double` format and copied to the external variables a, u, v, and d. This is the moment to copy singular values in descending order from D to d. Columns / rows of P and Q must be copied in corresponding order. Finally the allocated memory for the internal variables must be freed (the 64 bit precision results get lost).

8 C program

An implementation of the SVD in the C programming language is available from the web page https://www.unibw.de/eit_mathematik/forschung/eprecsvd as the na59 package.

In the header file `SVD.h` one may switch on or off computations in extended double precision and choose between Rayleigh shifts (corresponding to shift strategy (a) in [5]) and Wilkinson shifts (shift strategy (b) in [5]).

The file `SVD.c` contains an implementation of the SVD algorithm. All constants of data type `long double` are marked by a suffix `W`. The `printf` statements are included for testing only and can be omitted. At the end of SVD a function `testSVD` is called, which is implemented in `SVDtestFunc.c`. Its purpose is to test whether the fundamental identities $U^T U = I_m$, $V^T V = I_n$ and $AV = U\Sigma$ (nearly) hold. The calling of this function can also be omitted, of course.

The file `SVDtest2.c` contains an implementation of all the test cases discussed in the following section.

9 Test results

Computations were carried out on an Intel processor i7-9850H, programs were compiled with the GNU compiler, version 10.2.0. The results obtained by computations in extended double precision arithmetic are noted as $\tilde{U}$, $\tilde{V}$ and $\tilde{\Sigma}$. These results were rounded to standard double precision format to give $\hat{U}$, $\hat{V}$ and $\hat{\Sigma}$. The singular values on the diagonal of $\hat{\Sigma}$ are noted as $\hat{\sigma}_k$.

When exact singular values are known, they are rounded to standard double precision format and noted as σ_k. In the cases where exact singular values are not known, numerical approximations were computed via an implementation of the SVD in 256-bit arithmetic, based on the GNU MPFR multiple precision library. The singular values obtained from a multiple precision computation were then rounded to standard double precision format and taken as substitutes for the unknown exact singular values. The abbreviation $\mu(A) = \max|a_{i,j}|$ for a matrix $A = (a_{i,j})$ is used below.

A *first example* is taken from [3]. It is the matrix

$$A = \begin{bmatrix} 22 & 10 & 2 & 3 & 7 \\ 14 & 7 & 10 & 0 & 8 \\ -1 & 13 & -1 & -11 & 3 \\ -3 & -2 & 13 & -2 & 4 \\ 9 & 8 & 1 & -2 & 4 \\ 9 & 1 & -7 & 5 & -1 \\ 2 & -6 & 6 & 5 & 1 \\ 4 & 5 & 0 & -2 & 2 \end{bmatrix} \tag{15}$$

with exactly known singular values

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

After a total of 6 QR iteration steps, the singular values $\hat{\sigma}_k$ from Table 1 were found.

A value of 0 in the second column of Table 1 means that both, exact and computed singular values, are identical after rounding to standard double precision format. The accuracy of the achieved decomposition is characterized by

$$\mu(\tilde{U}^T\tilde{U} - I_m) = 3.25_{10}{-}19, \quad \mu(\tilde{V}^T\tilde{V} - I_n) = 3.25_{10}{-}19$$

and

$$\mu(A\tilde{V} - \tilde{U}\tilde{\Sigma}) = 1.73_{10}{-}18.$$

These fundamental identities were checked *before* the results were rounded to standard double precision by calling the function `testSVD` from within function `SVD`.

A *second example* is the 10×7 Hilbert matrix A defined by

$$a_{i,j} = \frac{1}{i+j-1}, \quad i = 1, \ldots, 10, \quad j = 1, \ldots, 7. \tag{16}$$

Table 1 Singular values for example matrix (15)

$\hat{\sigma}_k$	$\lvert\hat{\sigma}_k - \sigma_k\rvert$
3.5327043465311391e+01	0
2.0000000000000000e+01	0
1.9595917942265423e+01	0
4.3368086899420177e−19	4.33681e−19
3.2526065174565133e−19	3.25261e−19

Table 2 Singular values for example matrix (16)

$\hat{\sigma}_k$	$\lvert\hat{\sigma}_k - \sigma_k\rvert$
$1.7034227893692420e+00$	0
$3.0386188435519512e-01$	0
$2.7332449735275176e-02$	0
$1.5763395495700157e-03$	$2.1684e-19$
$6.0439432539595851e-05$	$6.77626e-21$
$1.4835194053530775e-06$	$3.49401e-20$
$2.0211192654283165e-08$	$2.72738e-20$

In this case, exact singular values are not known and substitutes for them were obtained from a computation in 256-bit arithmetic, as described above. After a total of 8 QR iteration steps, the results given in Table 2 were found.

Concerning the fundamental identities, the results

$$\mu(\tilde{U}^T\tilde{U} - I_m) = 3.25_{10}-19, \quad \mu(\tilde{V}^T\tilde{V} - I_n) = 3.25_{10}-19$$

and

$$\mu(A\tilde{V} - \tilde{U}\tilde{\Sigma}) = 1.08_{10}-19$$

were obtained.

A *third example* is taken from [2]. Setting

$$B = \begin{bmatrix} 5 & -1 & -1 & 6 & 4 & 0 \\ -3 & 1 & 4 & -7 & -2 & -3 \\ 1 & 3 & -4 & 5 & 4 & 7 \\ 0 & 4 & -1 & 1 & 4 & 5 \\ 4 & 2 & 3 & 1 & 6 & -1 \\ 3 & -3 & -5 & 8 & 0 & 2 \\ 0 & -1 & -4 & 4 & -1 & 3 \\ -5 & 4 & -3 & -2 & -1 & 7 \\ 3 & 4 & -3 & 6 & 7 & 7 \end{bmatrix},$$

an 18×12 matrix is defined by

$$A = \begin{bmatrix} B & 2B \\ 3B & -B \end{bmatrix}. \tag{17}$$

Since this matrix has rank 6, $\sigma_7 = \ldots = \sigma_{12} = 0$. Substitutes for the exact singular values $\sigma_1, \ldots, \sigma_6$ of A were obtained from a computation in 256-bit arithmetic, as described above. After a total of 12 QR iteration steps, the results shown in Table 3 were found.

Again, the fundamental identities were checked, with the following results:

$$\mu(\tilde{U}^T\tilde{U} - I_m) = 5.42_{10}-19, \quad \mu(\tilde{V}^T\tilde{V} - I_n) = 6.51_{10}-19$$

and

$$\mu(A\tilde{V} - \tilde{U}\tilde{\Sigma}) = 1.39_{10}-17.$$

Table 3 Singular values for example matrix (17)

$\hat{\sigma}_k$	$\lvert\hat{\sigma}_k - \sigma_k\rvert$
7.2265903120085326e+01	0
4.9630339183086058e+01	0
4.4288698552845858e+01	0
3.6427417335191997e+01	0
3.0416324106579538e+01	0
2.5017401012828763e+01	0
2.0599841277224584e−18	2.05998e−18
1.8431436932253575e−18	1.84314e−18
1.8431436932253575e−18	1.84314e−18
1.8431436932253575e−18	1.84314e−18
1.3010426069826053e−18	1.30104e−18
1.0842021724855044e−18	1.08420e−18

A *fourth example* is taken from [3]. It is the 20×21 matrix defined by

$$a_{i,j} = \begin{cases} 0, & \text{if } i > j \\ 21 - i, & \text{if } i = j, \\ -1, & \text{if } i < j, \end{cases} \quad i = 1, \ldots, 20, \quad j = 1, \ldots, 21, \tag{18}$$

with exact singular values

$$\sigma_{21-k} = \sqrt{k(k+1)}, \quad k = 1, \ldots, 20.$$

The method converged after 39 QR iterations with the results shown in Table 4.

Table 4 Singular values for example matrix (18)

$\hat{\sigma}_1, \ldots, \hat{\sigma}_{10}$	$\hat{\sigma}_{11}, \ldots, \hat{\sigma}_{20}$
2.0493901531919196e+01	1.0488088481701515e+01
1.9493588689617926e+01	9.4868329805051381e+00
1.8493242008906929e+01	8.4852813742385695e+00
1.7492855684535900e+01	7.4833147735478827e+00
1.6492422502470642e+01	6.4807406984078604e+00
1.5491933384829668e+01	5.4772255750516612e+00
1.4491376746189438e+01	4.4721359549995796e+00
1.3490737563232042e+01	3.4641016151377544e+00
1.2489995996796797e+01	2.4494897427831779e+00
1.1489125293076057e+01	1.4142135623730951e+00

In this example, all values $|\sigma_k - \hat{\sigma}_k|$ were 0, i.e., computed and exact values were identical, when both rounded to standard double precision. The fundamental identities were checked with the following results:

$$\mu(\tilde{U}^T\tilde{U} - I_m) = 1.08_{10}{-}18, \quad \mu(\tilde{V}^T\tilde{V} - I_n) = 1.73_{10}{-}18$$

and

$$\mu(A\tilde{V} - \tilde{U}\tilde{\Sigma}) = 1.73_{10}{-}17.$$

A *fifth example* is again taken from [3]. It is the 30×30 matrix defined by

$$a_{i,j} = \begin{cases} 0, & \text{if } i > j \\ 1, & \text{if } i = j, \\ -1, & \text{if } i < j, \end{cases} \quad i = 1, \ldots, 30, \quad j = 1, \ldots, 30. \tag{19}$$

Substitutes for the exact singular values were computed numerically in multiple precision arithmetic. The method converged after 48 QR iterations. Again, all values $|\sigma_k - \hat{\sigma}_k|$ were 0, i.e., computed and exact values were identical, when both rounded to double precision. The fundamental identities were checked with the following results:

$$\mu(\tilde{U}^T\tilde{U} - I_m) = 9.76_{10}{-}19, \quad \mu(\tilde{V}^T\tilde{V} - I_n) = 6.51_{10}{-}19$$

and

$$\mu(A\tilde{V} - \tilde{U}\tilde{\Sigma}) = 3.47_{10}{-}18.$$

10 Conclusions

An implementation of the SVD in extended precision arithmetic substantially improves the accuracy of the computed results. This improvement comes at no loss in computational speed, when extended precision arithmetic is native, as for the x86 architecture. Only a minor programming effort is necessary to use the capabilities of extended precision arithmetic and the same program also runs in standard double precision arithmetic. So it is advantageous to use the updated program.

We have also given a full, elementary explanation of the algorithm of Golub and Reinsch. In [3], not all details were fully explained on an elementary level.

Acknowledgements Professor Christian Reinsch passed away on October 8, 2022. A year before his death he told me (M.R.) he was working on a revision of the SVD algorithm published in 1970 by Golub and himself. Because of his rapidly worsening eyesight he was no longer able to do the work all on his own. I offered my help and took over the programming, testing, and writing of this article. It was his wish that this work should be published under our joint authorship. I am honored by this proposal. I am very thankful for having met and worked with this outstanding mathematician to whom I owe very much.

Author contributions Both authors wrote and reviewed the first versions of the manuscript together. C.R. initiated the work and had the principal idea to use extended precision arithmetic. M.R. wrote the C program (based on the Algol version of C.R.), did the testing and added the explanation of the implicit QR algorithm. M.R. wrote and reviewed the final version of the manuscript and is responsible for all errors.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability No empirical data are used. The test cases discussed in Section 9 can be reproduced by the program which may be downloaded from https://www.unibw.de/eit_mathematik/forschung/eprecsvd.

Declarations

Ethical approval Not applicable

Competing interests The authors declare no competing interests.

References

1. Francis, J.: The QR transformation. A unitary analogue to the LR transformation. Comput. J. **4**, 265–271 (1961, 1962)
2. Gander, W.: The first algorithms to compute the SVD, https://people.inf.ethz.ch/gander/talks/Vortrag2022.pdf
3. Golub, G.H., Reinsch, C.: Singular value decomposition and least squares solutions. Numer. Math. **14**, 403–420 (1970)
4. Higham, N.J.: Acuracy and Stability of Numerical Algorithms, 2nd edn. SIAM (2002)
5. Wilkinson, J.H.: Global convergence of tridiagonal QR algorithm with origin shifts, Lin. Alg. Appl., 409–420 (1968)