Routines for the diagonalization of complex matrices

T. Hahn[a]

[a]Max-Planck-Institut für Physik
Föhringer Ring 6, D–80805 Munich, Germany

Jacobi-type iterative algorithms for the eigenvalue decomposition, singular value decomposition, and Takagi factorization of complex matrices are presented. They are implemented as compact C and Fortran 77 subroutines in a freely available library.

1. Introduction

This note describes a set of routines for the eigenvalue decomposition, singular value decomposition, and Takagi factorization of a complex matrix. Unlike many other implementations, the current ones are all based on the Jacobi algorithm, which makes the code very compact but suitable only for small to medium-sized problems.

Although distributed as a library, the routines are self-contained and can easily be taken out of the library and included in own code, removing yet another installation prerequisite. Owing to the small size of the routines (each about 3 kBytes source code) it is possible, in fact quite straightforward, to adapt the diagonalization routine to one's own conventions rather than vice versa.

2. Mathematical Background

2.1. Eigenvalue Decomposition

The eigenvalue decomposition of a nonsingular matrix $A \in \mathbb{C}^{n \times n}$ takes the form

$$U A U^{-1} = \mathrm{diag}(\sigma_1, \ldots, \sigma_n), \ \sigma_i \in \mathbb{C}. \quad (1)$$

The eigenvalues σ_i and transformation matrix U can be further characterized if A possesses certain properties:

- $A = A^\dagger$ (Hermitian): $U^{-1} = U^\dagger$, $\sigma_i \in \mathbb{R}$,

- $A = A^T$ (symmetric): $U^{-1} = U^T$.

2.2. Singular Value Decomposition

The singular value decomposition (SVD) can be applied to an arbitrary matrix $A \in \mathbb{C}^{m \times n}$:

$$V^* A W^\dagger = \mathrm{diag}(\sigma_1, \ldots, \sigma_{\bar{n}}), \quad (2)$$

$$V \in \mathbb{C}^{\bar{n} \times m}, \quad W^{-1} = W^\dagger \in \mathbb{C}^{n \times \bar{n}},$$
$$\bar{n} = \min(m, n), \quad \sigma_i \geqslant 0.$$

V consists of orthonormal row vectors, i.e. is also unitary for $m = n$.

2.3. Takagi Factorization

The Takagi factorization [1,2] is a less known diagonalization method for complex symmetric matrices $A = A^T \in \mathbb{C}^{n \times n}$,

$$U^* A U^\dagger = \mathrm{diag}(\sigma_1, \ldots, \sigma_n), \quad (3)$$

$$U^{-1} = U^\dagger, \quad \sigma_i \geqslant 0.$$

Although outwardly similar to the eigenvalue decomposition of a Hermitian matrix, it is really the special case of an SVD with $V = W^*$, as it applies even to singular matrices. Note also that the left and right factors, U^* and $U^\dagger$, are in general not inverses of each other.

One might think that the Takagi factorization is merely a scaled SVD. For example, the matrix

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad (4)$$

has the SVD

$$V^T \mathrm{diag}(\sigma_1, \sigma_2) W = \quad (5)$$

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}^T \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

which can indeed be scaled to yield

$$U^T \mathrm{diag}(\sigma_1, \sigma_2) U = \quad (6)$$

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{pmatrix}^T \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{pmatrix}.$$

But consider the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{7}$$

which has the SVD

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^T \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{8}$$

whereas its Takagi factorization is

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{pmatrix}^T \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{i}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{pmatrix}. \tag{9}$$

Although occurring less frequently than the eigenvalue decomposition, the Takagi factorization does have real applications in physics, e.g. in the diagonalization of mass matrices of Majorana fermions.

3. Jacobi Algorithm

The Jacobi algorithm [3] consists of iteratively applying a basic 2×2 diagonalization formula until the entire $n \times n$ matrix is diagonal. It works in several 'sweeps' until convergence is achieved. In each sweep it rotates away the non-zero off-diagonal elements using the 2×2 algorithm. Every such rotation of course creates other non-zero off-diagonal elements. It can be shown, however, that the sum of the absolute values of the off-diagonal elements is reduced in each sweep. More precisely, the Jacobi method has quadratic convergence [4].

Convergence is in most cases achieved in $5-10$ sweeps, which for an $n \times n$ matrix translates into $(10-20)n^3$ multiply–add operations to obtain the eigenvalues only and $(15-30)n^3$ operations including the eigenvectors ([5], cf. also Sect. 10). This compares with $\frac{2}{3}n^3 + 30n^2$ operations for the Householder/QL algorithm when just the eigenvalues are sought and $\frac{4}{3}n^3 + 3n^3$ when also the eigenvectors are needed.

For large n, the Jacobi algorithm is thus not the most efficient method. Nevertheless, for small to medium-sized problems the Jacobi method is a strong competitor, in particular as it has the following advantages:

- It is conceptually very simple and thus very compact.

- It delivers the eigenvectors at little extra cost.

- The diagonal values are accurate to machine precision and, in cases where this is mathematically meaningful, the vectors of the transformation matrix are always orthogonal, almost to machine precision.

For the various diagonalization problems discussed before, only the core 2×2 diagonalization formula changes, whereas the surrounding Jacobi algorithm stays essentially the same.

The famous Linear Algebra Handbook gives an explicit implementation of the Jacobi algorithm for real symmetric matrices [4], taking particular care to minimize roundoff errors through mathematically equivalent variants of the rotation formulas. The present routines are closely patterned on this procedure. For the Takagi factorization, the use of the Jacobi algorithm was first advocated in two conference papers [6,7] which give only few details, however. The two-sided Jacobi version of the SVD is used in [8]. Ref. [9] gives a more thorough overview of literature on the Jacobi method.

4. The 2×2 Formulas

4.1. Eigenvalue decomposition

Using the ansatz

$$U = \begin{pmatrix} c_1 & t_1 c_1 \\ -t_2 c_2 & c_2 \end{pmatrix} \tag{10}$$

the equation $UA = \mathrm{diag}(\sigma_1, \sigma_2)\,U$ becomes

$$\sigma_1 = A_{11} + t_1 A_{21} = A_{22} + \frac{1}{t_1} A_{12}\,, \tag{11}$$

$$\sigma_2 = A_{11} - \frac{1}{t_2} A_{21} = A_{22} - t_2 A_{12}\,. \tag{12}$$

Solving for t_1 and t_2 yields

$$t_1 = \frac{A_{12}}{\Delta + D}\,, \quad t_2 = \frac{A_{21}}{\Delta + D}\,, \tag{13}$$

$$\Delta = \frac{1}{2}(A_{11} - A_{22}), \tag{14}$$

$$D = \pm\sqrt{\Delta^2 + A_{12}A_{21}}. \tag{15}$$

For the numerical stability it is best to choose the sign of D which gives $t_{1,2}$ the larger denominator. This corresponds to taking the smaller rotation angle ($< \pi/4$). The diagonal values are

$$\sigma_1 = A_{11} + \delta, \qquad \delta = \frac{A_{12}A_{21}}{\Delta + D}. \tag{16}$$
$$\sigma_2 = A_{22} - \delta,$$

In order that U smoothly becomes unitary as A becomes Hermitian, we choose

$$c_1 = c_2 = \frac{1}{\sqrt{1 + t_1 t_2}}, \tag{17}$$

which guarantees a unit determinant.

4.2. Takagi Factorization

Substituting the unitary ansatz

$$U = \begin{pmatrix} c & t\,c\,\mathrm{e}^{\mathrm{i}\varphi} \\ -t\,c\,\mathrm{e}^{-\mathrm{i}\varphi} & c \end{pmatrix}, \tag{18}$$

$$c = \frac{1}{\sqrt{1 + t^2}}, \quad t \in \mathbb{R}, \tag{19}$$

into $U^*A = \mathrm{diag}(\sigma_1, \sigma_2)\,U$ and introducing

$$\tilde{\sigma}_1 = \mathrm{e}^{\mathrm{i}\varphi}\sigma_1, \qquad \tilde{A}_{11} = \mathrm{e}^{\mathrm{i}\varphi}A_{11}, \tag{20}$$

$$\tilde{\sigma}_2 = \mathrm{e}^{-\mathrm{i}\varphi}\sigma_2, \qquad \tilde{A}_{22} = \mathrm{e}^{-\mathrm{i}\varphi}A_{22}, \tag{21}$$

we arrive at

$$\tilde{\sigma}_1 = \tilde{A}_{11} + tA_{12} = \tilde{A}_{22} + \frac{1}{t}A_{12}, \tag{22}$$

$$\tilde{\sigma}_2 = \tilde{A}_{11} - \frac{1}{t}A_{12} = \tilde{A}_{22} - tA_{12}. \tag{23}$$

Comparing with Eqs. (11) and (12), the solution can be read off easily:

$$t = \frac{A_{12}}{\tilde{\Delta} + \tilde{D}}, \tag{24}$$

$$\tilde{\Delta} = \frac{1}{2}(\tilde{A}_{11} - \tilde{A}_{22}), \tag{25}$$

$$\tilde{D} = \pm\sqrt{\tilde{\Delta}^2 + A_{12}^2}. \tag{26}$$

Again it is best for numerical stability to choose the sign of $\tilde{D}$ which gives the larger denominator for t. The diagonal values become

$$\sigma_1 = A_{11} + t\,A_{12}\,\mathrm{e}^{-\mathrm{i}\varphi}, \tag{27}$$

$$\sigma_2 = A_{22} - t\,A_{12}\,\mathrm{e}^{\mathrm{i}\varphi}. \tag{28}$$

The assumption $t \in \mathbb{R}$ fixes the phase φ. It requires that A_{12} and $\tilde{\Delta}$ have the same phase, i.e. $\tilde{\Delta} = (\text{real number}) \cdot A_{12}$. Since both $\mathrm{e}^{\mathrm{i}\varphi}$ and its conjugate appear in $\tilde{\Delta}$, we try the ansatz

$$\mathrm{e}^{\mathrm{i}\varphi} = \alpha A_{12} + \beta A_{12}^* \tag{29}$$

and choose coefficients to make the A_{12}^* term in

$$\tilde{\Delta} \propto (\alpha A_{11} - \beta^* A_{22})A_{12} + (\beta A_{11} - \alpha^* A_{22})A_{12}^* \tag{30}$$

vanish. This is achieved by $\alpha = A_{11}^*$ and $\beta = A_{22}$ which also makes the coefficient of A_{12} real. Thus,

$$\mathrm{e}^{\mathrm{i}\varphi} = \frac{\kappa}{|\kappa|}, \quad \kappa = A_{11}^* A_{12} + A_{22}A_{12}^*. \tag{31}$$

For vanishing $\tilde{\Delta} \propto |A_{11}|^2 - |A_{22}|^2$, the reality condition is dispensable and α becomes a free parameter. The choice $\alpha = \tilde{\kappa}/A_{12}$, $\beta = \tilde{\kappa}^2\alpha^*$ guards against κ becoming zero:

$$\mathrm{e}^{\mathrm{i}\varphi} = \frac{\tilde{\kappa}}{|\tilde{\kappa}|}, \quad \tilde{\kappa} = \sqrt{\frac{A_{22}}{A_{11}}} \xrightarrow{A_{11} \to 0} 1. \tag{32}$$

The σ_i still have an arbitrary phase at this point, but this is irrelevant for the Jacobi rotations. The correct phase to ensure $\sigma_i \geqslant 0$ is adjusted at the end via $U_{ij} \to U_{ij}\sqrt{\sigma_i/|\sigma_i|}$, $\sigma_i \to |\sigma_i|$.

5. Singular Value Decomposition

We insert unitary parameterizations for the left and right transformation matrices $X = V, W$,

$$X = \begin{pmatrix} c_X & t_X\,c_X \\ -t_X^*\,c_X & c_X \end{pmatrix}, \tag{33}$$

$$c_X = \frac{1}{\sqrt{1 + |t_X|^2}}, \quad t_X \in \mathbb{C}, \tag{34}$$

into $V^*A = \mathrm{diag}(\sigma_1, \sigma_2)\,W$ and by eliminating $\sigma_{1,2}$ arrive at

$$A_{12} + A_{22}t_V^* = (A_{11} + A_{21}t_V^*)t_W, \tag{35}$$

$$A_{21} + A_{22} t_W^* = (A_{11} + A_{12} t_W^*) t_V . \qquad (36)$$

The solutions are evidently related through exchange of the off-diagonal elements. Explicitly,

$$t_V = \frac{M_V}{\Delta_V + D_V} , \qquad (37)$$

$$M_V = A_{11}^* A_{21} + A_{22} A_{12}^* , \qquad (38)$$

$$\Delta_V = \frac{1}{2}(|A_{11}|^2 - |A_{22}|^2 + \\ |A_{12}|^2 - |A_{21}|^2) , \qquad (39)$$

$$D_V = \pm\sqrt{\Delta_V^2 + |M_V|^2} , \qquad (40)$$

$$t_W = t_V|_{A_{12} \leftrightarrow A_{21}} . \qquad (41)$$

D_V and D_W share the same sign, which is chosen to yield the larger set of denominators for better numerical stability.

The denominators can become zero, too, which is the case when Eqs. (35) and (36) are redundant. We are free to choose $t_V = 0$ then and obtain $t_W = A_{12}/A_{11}$. If furthermore $A_{11} = 0$ ($= A_{22}$ here) we just swap the rows in A and V.

The singular values become

$$\sigma_1 = \frac{c_V}{c_W}(A_{11} + t_V^* A_{21}) , \qquad (42)$$

$$\sigma_2 = \frac{c_V}{c_W}(A_{22} - t_V A_{12}) , \qquad (43)$$

If $A \in \mathbb{C}^{m \times n}$ is not a square matrix, we consider two cases:

For $m > n$, we make A square by padding it with zero-columns at the right. For zero right column, $A_{12} = A_{22} = 0$, Eq. (43) guarantees that it is σ_2 that vanishes. That is, the above Jacobi rotation never moves a singular value into the zero-extended part of the matrix and the same is true for the possible row swap mentioned above. All singular values automatically end up as the *first* $\min(m,n)$ diagonal values of the Jacobi-rotated A thus.

For $m < n$, we apply this algorithm to A^T and at the end exchange V and W. This is the least involved solution, as A has to be copied to temporary storage for zero-extension anyway.

6. Installation

The Diag package can be downloaded from the URL http://feynarts.de/diag. After unpack-

ing the tar file, the library is built with

```
./configure
make
```

The generated files are installed into a platform-dependent directory with "`make install`" and at the end one can do a "`make clean`" to remove intermediate files.

The Fortran routines in the Diag library allocate space for intermediate results according to a preprocessor variable `MAXDIM`, defined in `diag-f.h`. This effectively limits the size of the input and output matrices but is necessary because Fortran 77 offers no dynamic memory allocation. Since the Jacobi algorithm is not particularly suited for large problems anyway, the default value of 16 should be sufficient for most purposes.

The Diag library by default arranges the transformation matrices as matrices of row eigenvectors (as in the conventions in Sect. 2). This can be changed to column vectors by configuring with

```
./configure --cols
```

The relations between input matrix A, the transformation matrices, and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ are then

$$
\begin{aligned}
AU &= U\Sigma & \texttt{\{H,S,C\}Eigensystem}, \\
AU^* &= U\Sigma & \texttt{TakagiFactor}, \\
AW^* &= V\Sigma & \texttt{SVD}.
\end{aligned}
$$

7. Description of the Fortran Routines

The general convention is that each matrix is followed by its leading dimension in the argument list, i.e. the m in $\texttt{A}(m,n)$. In this way it is possible to diagonalize submatrices with just a different invocation. Needless to add, the leading dimension must be at least as large as the corresponding matrix dimension.

7.1. Hermitian Eigenvalue Decomposition
Hermitian matrices are diagonalized with

```
subroutine HEigensystem(n, A,ldA,
                        d, U,ldU, sort)
integer n, ldA, ldU, sort
double complex A(ldA,n), U(ldU,n)
double precision d(n)
```

The arguments are as follows:

- n (input), the matrix dimension.

- A (input), the matrix to be diagonalized. Only the upper triangle of A needs to be filled and it is further assumed that the diagonal elements are real. Attention: the contents of A are not preserved.

- d (output), the eigenvalues.

- U (output), the transformation matrix.

- sort (input), a flag that determines sorting of the eigenvalues:

$$0 = \text{do not sort},$$
$$1 = \text{sort into ascending order},$$
$$-1 = \text{sort into descending order}.$$

The 'natural' (unsorted) order is determined by the choice of the smaller rotation angle in each Jacobi rotation.

7.2. Symmetric Eigenvalue Decomposition
The second special case is that of a complex symmetric matrix:

```
subroutine SEigensystem(n, A,ldA,
                           d, U,ldU, sort)
integer n, ldA, ldU, sort
double complex A(ldA,n), U(ldU,n)
double complex d(n)
```

The arguments have the same meaning as for HEigensystem, except that A's diagonal elements are not assumed real and sorting occurs with respect to the real part only.

7.3. General Eigenvalue Decomposition
The general case of the eigenvalue decomposition is implemented in

```
subroutine CEigensystem(n, A,ldA,
                           d, U,ldU, sort)
integer n, ldA, ldU, sort
double complex A(ldA,n), U(ldU,n)
double complex d(n)
```

The arguments are as before, except that A has to be filled completely.

7.4. Takagi Factorization
The Takagi factorization is invoked in almost the same way as SEigensystem:

```
subroutine TakagiFactor(n, A,ldA,
                           d, U,ldU, sort)
integer n, ldA, ldU, sort
double complex A(ldA,n), U(ldU,n)
double precision d(n)
```

The arguments are as for SEigensystem. Also here only the upper triangle of A needs to be filled.

7.5. Singular Value Decomposition
The SVD routine has the form

```
subroutine SVD(m, n, A,ldA,
                 d, V,ldV, W,ldW, sort)
integer m, n, ldA, ldV, ldW, sort
double complex A(ldA,n)
double complex V(ldV,m), W(ldW,n)
double precision d(min(m,n))
```

with the arguments

- m, n (input), the dimensions of A.

- A (input), the $m \times n$ matrix of which the SVD is sought.

- d (output), the singular values.

- V (output), the $\min(m,n) \times m$ left transformation matrix.

- W (output), the $\min(m,n) \times n$ right transformation matrix.

- sort (input), the sorting flag with values as above.

8. Description of the C Routines

The C routines are implemented and used exactly as their Fortran counterparts, except that the prototypes in CDiag.h should be included.

CDiag.h introduces the new Complex type to homogeneously treat complex numbers in C and C++. It is equivalent to double complex in C(99) and to complex<double> in C++.

C's syntax unfortunately does not allow the declaration of variable-size matrices as function

arguments, thus it is not possible for `CDiag.h` to set up the prototypes to directly accept C matrices without compiler warnings. To simplify matters, `CDiag.h` defines the macro `Matrix` which is used as in

```
Complex A[5][3], V[3][5], W[3][3];
double d[3];
...
SVD(5, 3, Matrix(A), d,
          Matrix(V), Matrix(W), 0);
```

This is equivalent to using an explicit cast and passing the leading dimension, i.e.

```
SVD(5, 3, (Complex *)A,3, d,
          (Complex *)V,5,
          (Complex *)W,3, 0);
```

9. The Mathematica Interface

The Mathematica version may not seem as useful as the library since Mathematica already has perfectly functional eigen- and singular value decompositions. The Takagi factorization is not available in Mathematica, however, and moreover the interface is ideal for trying out, interactively using, and testing the Diag routines.

The Mathematica executable is loaded with

```
Install["Diag"]
```

and makes the following functions available:

- `HEigensystem[A]` computes the eigenvalue decomposition {`d`,`U`} of the Hermitian matrix `A`.

- `SEigensystem[A]` computes the eigenvalue decomposition {`d`,`U`} of the symmetric matrix `A`.

- `CEigensystem[A]` computes the eigenvalue decomposition {`d`,`U`} of the general matrix `A`.

- `TakagiFactor[A]` computes the Takagi factorization {`d`,`U`} of the symmetric matrix `A`.

- `SVD[A]` computes the singular value decomposition {`V`,`d`,`W`} of the matrix `A`.

Note that these routines do not check whether the given matrix fulfills the requirements, e.g. whether it is indeed Hermitian.

10. Timings

The following plot shows the time for diagonalizing a random matrix of various dimensions. Note that the abscissa is divided in units of the dimension cubed; this accounts for the anticipated scaling behaviour of the Jacobi algorithm, hence the curves appear essentially linear.

The absolute time values should be taken for orientation only, as they necessarily reflect the CPU speed. For reference, the numbers used in the figure above were obtained on an AMD64-X2 CPU running at 3 GHz. Each point is actually the average from diagonalizing 10^6 random matrices, to reduce quantization effects in the time measurement.
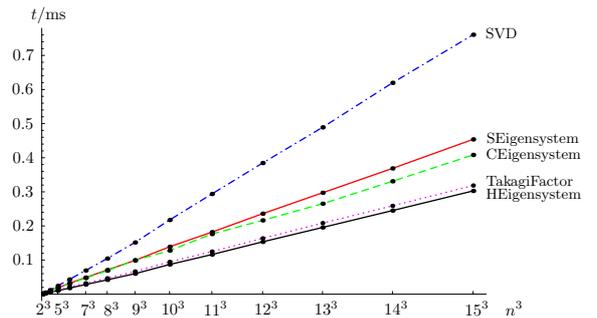
The average number of sweeps needed to diagonalize the 10^6 random matrices to machine precision is plotted in the next figure.
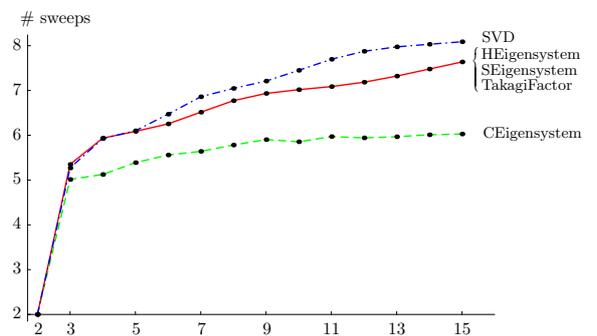
11. Summary

The Diag package contains C and Fortran subroutines for the eigenvalue decomposition, singular value decomposition, and Takagi factorization of a complex matrix. The library is supplemented

by interfacing code to access the routines from Mathematica.

The routines are based on the Jacobi algorithm. They are self-contained and quite compact, thus it should be straightforward to use them outside of the library. All routines are licensed under the LGPL.

Acknowledgements

REFERENCES

1. T. Takagi, *Japanese J. Math.* **1** (1927) 83.
2. R.A. Horn, C.A. Johnson, Matrix Analysis, Cambridge University Press, 1990, p. 201 f.
3. C.G.J. Jacobi, *Crelle J.* **30** (1846) 51.
4. H. Rutishauser, Contribution II/1, in: Handbook for Automatic Computation, ed. J.H. Wilkinson, C. Reinsch, Springer, 1971.
5. W.H. Press et al., Numerical Recipes in Fortran, 2nd ed., Cambridge University Press, 1992, Chapter 11.
6. L. De Lathauwer, B. De Moor, EUSIPCO 2002 conference proceedings, https://www.ensieta.fr/e3i2/intranet/Confs/Eusipco02/articles/paper323.html
7. X. Wang, S. Qiao, PDPTA 2002 conference proceedings Vol. I, http://www.dcss.mcmaster.ca/∼qiao/publications/pdpta02.ps.gz
8. R.P. Brent, F.T. Luk, C.F. van Loan, *J. of VLSI and Computer Systems* 3 (1983) 242. G.E. Forsythe, P. Henrici, *Trans. Amer. Math. Soc.* 94 (1960) 1.
9. G.H. Golub, C.F. van Loan, Matrix Computations, 3rd ed., The Johns Hopkins University Press, 1996, Sect. 8.4.