

Routines for the diagonalization of complex matrices

T. Hahn^a

^aMax-Planck-Institut für Physik
Föhringer Ring 6, D-80805 Munich, Germany

MPP-2006-85
PACS: 02.10.Ud, 02.10.Yn, 02.60.Dc

Jacobi-type iterative algorithms for the eigenvalue decomposition, singular value decomposition, and Takagi factorization of complex matrices are presented. They are implemented as compact 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

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

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 $A \in \mathbb{C}^{m \times n}$,

where $m \geq n$ is assumed (for $m < n$, substitute A^T for A in the following):

$$\begin{aligned} V^* A W^\dagger &= \text{diag}(\sigma_1, \dots, \sigma_n), \\ V &\in \mathbb{C}^{m \times m}, \quad W^{-1} = W^\dagger \in \mathbb{C}^{n \times n}, \quad \sigma_i \in \mathbb{R}. \end{aligned} \quad (2)$$

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 $A = A^T \in \mathbb{C}^{n \times n}$,

$$\begin{aligned} U^* A U^\dagger &= \text{diag}(\sigma_1, \dots, \sigma_n), \\ U^{-1} &= U^\dagger, \quad \sigma_i \geq 0. \end{aligned} \quad (3)$$

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

$$V^T \text{diag}(\sigma_1, \sigma_2) W = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}^T \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \quad (5)$$

which can indeed be scaled to yield

$$U^T \text{diag}(\sigma_1, \sigma_2) U = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{pmatrix}^T \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{i}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{pmatrix}. \quad (6)$$

But consider the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (7)$$

which has the SVD

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}^T \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (8)$$

whereas its Takagi factorization is

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

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 6–10 sweeps, which for an $n \times n$ matrix translates into $(12-20)n^3$ multiply-add operations to obtain the eigenvalues only and $(18-30)n^3$ operations including the eigenvectors [5]. 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.

4. The 2×2 Formulas

4.1. Eigenvalue decomposition

Using the ansatz

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

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

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

$$\sigma_2 = A_{11} - \frac{1}{t_2} A_{21} = A_{22} - t_2 A_{12}. \quad (12)$$

Solving for t_1 and t_2 yields

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

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

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

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

$$\begin{aligned} \sigma_1 &= A_{11} + \delta, & \delta &= \frac{A_{12}A_{21}}{\Delta + D}. \end{aligned} \quad (16)$$

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

$$c_1 = c_2 = \frac{1}{\sqrt{1 + t_1 t_2}}, \quad (17)$$

which guarantees a unit determinant.

4.2. Takagi Factorization

Substituting the unitary ansatz

$$U = \begin{pmatrix} c & t c e^{i\varphi} \\ -t c e^{-i\varphi} & c \end{pmatrix}, \quad c, t \in \mathbb{R}, \quad (18)$$

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

$$\tilde{\sigma}_1 = e^{i\varphi} \sigma_1, \quad \tilde{A}_{11} = e^{i\varphi} A_{11}, \quad (19)$$

$$\tilde{\sigma}_2 = e^{-i\varphi} \sigma_2, \quad \tilde{A}_{22} = e^{-i\varphi} A_{22}, \quad (20)$$

we arrive at

$$\tilde{\sigma}_1 = \tilde{A}_{11} + t A_{12} = \tilde{A}_{22} + \frac{1}{t} A_{12}, \quad (21)$$

$$\tilde{\sigma}_2 = \tilde{A}_{11} - \frac{1}{t} A_{12} = \tilde{A}_{22} - t A_{12}. \quad (22)$$

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

$$t = \frac{A_{12}}{\tilde{\Delta} + \tilde{D}}, \quad (23)$$

$$\tilde{\Delta} = \frac{1}{2}(\tilde{A}_{11} - \tilde{A}_{22}), \quad (24)$$

$$\tilde{D} = \pm \sqrt{\tilde{\Delta}^2 + A_{12}^2}. \quad (25)$$

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

$$\sigma_1 = A_{11} + t A_{12} e^{-i\varphi}, \quad (26)$$

$$\sigma_2 = A_{22} - t A_{12} e^{i\varphi}. \quad (27)$$

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 $e^{i\varphi}$ and its conjugate appear in $\tilde{\Delta}$, we try the ansatz

$$e^{i\varphi} = \alpha A_{12} + \beta A_{12}^* \quad (28)$$

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

$$\begin{aligned} \tilde{\Delta} &\propto (\alpha A_{11} - \beta^* A_{22}) A_{12} + \\ &(\beta A_{11} - \alpha^* A_{22}) A_{12}^* \end{aligned} \quad (29)$$

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

$$e^{i\varphi} = \frac{A_{11}^* A_{12} + A_{22} A_{12}^*}{|A_{11}^* A_{12} + A_{22} A_{12}^*|}. \quad (30)$$

5. Singular Value Decomposition

Even though there exists an algorithm known as ‘One-Sided Jacobi’ [8] to perform an SVD, this method is not employed here as there is a simpler method with the same numerical characteristics.

Squaring the defining Eq. (2) gives

$$W A^\dagger V^T V^* A W^\dagger = \text{diag}(\sigma_1^2, \dots, \sigma_n^2). \quad (31)$$

As V has orthonormal rows, the σ_i^2 and the right-transformation matrix W can be found from an eigenvalue decomposition of the Hermitian matrix $A^\dagger A$. The left-transformation matrix V is obtained from

$$V_{ik} = \frac{1}{\sigma_i} (A W^\dagger)_{ki}, \quad \sigma_i \neq 0. \quad (32)$$

Eq. (2) makes no statement about the vectors V_{ik} corresponding to $\sigma_i = 0$. For this subspace we choose a basis which is orthonormal with respect to the vectors for non-zero σ_i . Specifically, we apply Gram–Schmidt to $V_{ik} = \delta_{ij}$, where j counts from m downward. Values of j for which the norm of the resulting vector becomes too close to zero are skipped.

6. Installation

The Diag package can be downloaded from the URL <http://www.feynarts.de/diag>. After unpacking the tar file, the library is built with

```
./configure
make
```

To compile also the Mathematica executable, one needs to issue “`make all`” instead of just “`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 routines in the Diag library allocate space for intermediate results according to a preprocessor variable `MAXDIM`, defined in `config.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.

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 $A(m,n)$. 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 = do not sort,
1 = sort into ascending order,
-1 = 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 `Eigensystem`, 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(n)
```

with the arguments

- **m, n** (input), the dimensions of **A**, $m \geq n$.
- **A** (input), the $m \times n$ matrix of which the SVD is sought.
- **d** (output), the singular values.
- **V** (output), the left-transformation matrix ($n \times m$).
- **W** (output), the right-transformation matrix ($n \times n$).
- **sort** (input), the sorting flag with values as above.

8. Description of the C Routines

The C version consists merely of an include file `CDiag.h` which sets up the correct interfacing code for using the Fortran routines. In particular the usual problem of transposition¹ between Fortran and C is taken care of.

The arguments are otherwise as for Fortran. To treat complex numbers uniformly in C and C++, `CDiag.h` introduces the new `double_complex` type which is equivalent to `complex<double>` in C++ and to `struct { double re, im; }` 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

¹C uses row-major storage for matrices, whereas Fortran uses column-major storage, i.e. a matrix is a vector of row vectors in C, and a vector of column vectors in Fortran.

```
double_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, (double_complex *)A, 3, d,
    (double_complex *)V, 5,
    (double_complex *)W, 3, 0);
```

Compilation should be done using the included `fcc` script, i.e. by replacing the C compiler with `fcc`, for example

```
fcc -Iinclude myprogram.c -llib -ldiag
```

The `fcc` script is configured and installed during the build process and automatically adds the necessary flags for linking with Fortran code.

9. The Mathematica Interface

The Mathematica version may not seem as useful as the Fortran 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:

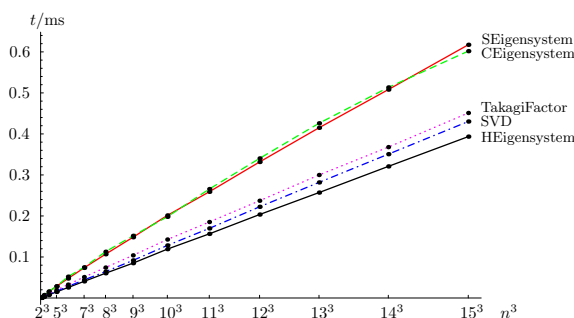
- `HEigensystem[A]` computes the eigenvalue decomposition $\{d, U\}$ of the Hermitian matrix **A** such that $U.A == \text{DiagonalMatrix}[d].U$.
- `SEigensystem[A]` computes the eigenvalue decomposition $\{d, U\}$ of the symmetric matrix **A** such that $U.A == \text{DiagonalMatrix}[d].U$.
- `CEigensystem[A]` computes the eigenvalue decomposition $\{d, U\}$ of the general matrix **A** such that $U.A == \text{DiagonalMatrix}[d].U$.
- `TakagiFactor[A]` computes the Takagi factorization $\{d, U\}$ of the symmetric matrix **A** such that
 $\text{Conjugate}[U].A == \text{DiagonalMatrix}[d].U$.

- `SVD[A]` computes the singular value decomposition $\{V, d, W\}$ of the matrix A such that `Conjugate[V].A == DiagonalMatrix[d].W`.

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 2.6 GHz. Each point is actually the average from diagonalizing 10^6 random matrices, to reduce quantization effects in the time measurement.

11. Summary

The Diag package contains Fortran subroutines for the eigenvalue decomposition, singular value decomposition, and Takagi factorization of a complex matrix. The Fortran library is supplemented by interfacing code to access the routines from C/C++ and 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

TH thanks the National Center for Theoretical Studies, Hsinchu, Taiwan, for warm hospitality during the time this work was carried out.

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. J.C. Nash, *Computer J.* **18** (1973) 74.