Algorithm 696
An Inverse Rayleigh Iteration for Complex Band Matrices

GÉZA SCHRAUF
Deutsche Airbus GmbH, Bremen

A FORTRAN 77 implementation of a generalized inverse Rayleigh iteration procedure for the calculation of eigenvalues and left and right eigenvectors of a complex band matrix is described. The core of the procedure is an algorithm that solves systems with the original matrix and with the adjoint matrix by calculating only one LU-decomposition.

Categories and Subject Descriptors: F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*computation on matrices*; G.1.0 [**Numerical Analysis**]: General—*numerical algorithms*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*eigenvalues, linear systems* (*direct and iterative methods*), *sparse and very large systems*

General Terms: Algorithms

Additional Key Words and Phrases: Banded matrices, inverse Rayleigh iteration

The algorithm was developed in order to determine the amplification rates of instability waves in compressible boundary layers [5]. The problem can be reduced to eigenvalue calculations of complex, banded, and asymmetric matrices.

An algorithm for banded matrices is not contained in EISPACK [3]. The computing time for an eigenvalue calculation of an N-dimensional, square matrix is essentially proportional to N^3. However, for a band matrix with a bandwidth of M elements, $M \ll N$, the computing time can be reduced to M^2N when the band structure is taken into account.

It seems that an inverse Rayleigh iteration could be constructed using LINPACK [1] routines. However, after the eigenvalue shift has been applied to the matrix, the linear system to be solved could be exactly singular, and LINPACK would stop at that point.

The whole program conforms to standard FORTRAN 77. Because the standard does not contain complex double-precision numbers, we replace complex by real arithmetic. The elements of the real and the imaginary part of the matrix are stored in the two arrays AR and AI as illustrated by the

Author's address: Deutsche Airbus GmbH, Postfach 107845, 2800 Bremen 1, Germany.

following example:

$$
\begin{array}{cccccccccccc}
* & * & a_{r,31} & a_{r,41} & a_{r,51} & 0 & 0 & 0 & 0 & 0 & 0 & \\
 & * & a_{r,22} & a_{r,32} & a_{r,42} & a_{r,52} & 0 & 0 & 0 & 0 & 0 & \\
 & & a_{r,13} & a_{r,23} & a_{r,33} & a_{r,43} & a_{r,53} & 0 & 0 & 0 & 0 & \\
 & & 0 & a_{r,14} & a_{r,24} & a_{r,34} & a_{r,44} & a_{r,54} & 0 & 0 & 0 & \\
 & & 0 & 0 & a_{r,15} & a_{r,25} & a_{r,35} & a_{r,45} & a_{r,55} & 0 & 0 & \\
 & & 0 & 0 & 0 & a_{r,16} & a_{r,26} & a_{r,36} & a_{r,46} & a_{r,56} & 0 & \\
 & & 0 & 0 & 0 & 0 & a_{r,17} & a_{r,27} & a_{r,37} & a_{r,47} & a_{r,57} & \\
 & & 0 & 0 & 0 & 0 & 0 & a_{r,18} & a_{r,28} & a_{r,38} & a_{r,48} & * \\
 & & 0 & 0 & 0 & 0 & 0 & 0 & a_{r,19} & a_{r,29} & a_{r,39} & * & *
\end{array}
$$

The first index counts the position of the element in the band, and the second one indicates the row. Only the band is stored, and the user has to supply zeros for the positions marked with an asterisk, so that the array AR for the real part of the matrix contains the following elements:

$$
\begin{array}{ccccc}
0 & 0 & a_{r,31} & a_{r,41} & a_{r,51} \\
0 & a_{r,22} & a_{r,32} & a_{r,42} & a_{r,52} \\
a_{r,13} & a_{r,23} & a_{r,33} & a_{r,43} & a_{r,53} \\
a_{r,14} & a_{r,24} & a_{r,34} & a_{r,44} & a_{r,54} \\
a_{r,15} & a_{r,25} & a_{r,35} & a_{r,45} & a_{r,55} \\
a_{r,16} & a_{r,26} & a_{r,36} & a_{r,46} & a_{r,56} \\
a_{r,17} & a_{r,27} & a_{r,37} & a_{r,47} & a_{r,57} \\
a_{r,18} & a_{r,28} & a_{r,38} & a_{r,48} & 0 \\
a_{r,19} & a_{r,29} & a_{r,39} & 0 & 0
\end{array}
$$

The imaginary part of the matrix is stored analogously in the array AI. This storage arrangement is taken from Schrauf [4]. It has the advantage that the elements of the rows are stored with consecutive addresses so that the program vectorizes efficiently.

The calling sequence is the following:

```
CALL GIRI(AR, AI, AWR, AWI, M, N, XLR, XLI, JP, UR, UI, VR, VI,
·         INIT, WR, WI, OMEGAR, OMEGAI, EPS, NSIMPL, LAST,
·         IERROR, INFO, IUNIT)
```

The arguments are the following.

AR, AI	DOUBLE PRECISION, input. AR(M, N), AI(M, N) contain the real and imaginary parts of the matrix. Their values remain unchanged.
AWR, AWI	DOUBLE PRECISION. AWR(M, N), AWI(M, N) are temporary arrays used to store real and imaginary parts of the upper triangular matrix of the LU-decomposition of $A - \omega I$.
M	INTEGER, input, number of elements in each row of the band. M must be an odd number.
N	INTEGER, input, number of equations.
XLR, XLI	DOUBLE PRECISION. XLR((M − 1)/2, N), XLI((M − 1)/2, N) are

	temporary arrays used to store real and imaginary parts of the pseudo-lower-triangular matrix of the LU-decomposition of $A - \omega I$.
JP	INTEGER, JP(N) is a temporary array used to store the pivoting information.
UR, UI	DOUBLE PRECISION. UR(N), UI(N) contain real and imaginary parts of initial approximation for the right eigenvector as input (used if INIT = 1) and the real and imaginary part of the computed right eigenvector as output.
VR, VI	DOUBLE PRECISION. VR(N), VI(N) contain real and imaginary parts of initial approximation for the left eigenvector as input (used if INIT = 1) and the real and imaginary part of the computed left eigenvector as output.
INIT	INTEGER, input. INIT = 0: No initial vectors are provided. The vectors $UR = UI = VR = VI = (1, \ldots, 1)$ are used as initial vectors for the iterative computation of the right and left eigenvectors. INIT = 1: Initial approximations for the right and left eigenvectors are provided. Their real and imaginary parts are stored in the arrays UR, UI, VR, VI.
WR, WI	DOUBLE PRECISION. WR(N), WI(N) are work arrays.
OMEGAR, OMEGAI	DOUBLE PRECISION. OMEGAR and OMEGAI contain an initial approximation of the real and imaginary parts of the eigenvalue as input and the real and imaginary parts of the computed eigenvalue as output.
EPS	DOUBLE PRECISION, input. Absolute error tolerance for the computed eigenvalue: The iteration is terminated if the absolute value of the computed increment for the eigenvalue is lower than EPS. An appropriate value for EPS is, for example, the square root of MACHEP, which is the smallest DOUBLE PRECISION constant for which $1 + MACHEP > 1$ holds.
NSIMPL	INTEGER, input. Number of simplified iteration steps without calculating a new LU-decomposition. The recommended value for NSIMPL is 2.
LAST	INTEGER, input. Number of the last iteration step that is followed by NSIMPL-simplified iteration steps. The recommended value for LAST is 1.
IERROR	INTEGER, output, error code: IERROR = 0: No errors occurred. IERROR = 1: M is not odd. IERROR = 2: The iteration did not converge.
INFO	INTEGER, input. INFO = 0: Convergence information is not displayed. INFO = 1: Convergence information is displayed.
IUNIT	INTEGER, input. Unit number on that the convergence information is written.

APPENDIX A

Organization of the LU-Decomposition

The core of the method is an algorithm that computes solutions of the two systems

$$Ax = b \tag{1}$$

$$A^*y = c \tag{2}$$

with only one LU-decomposition of A. In order to explain how we solve the second system with the adjoint matrix, we have to review the Gauss algorithm used to solve the first system. In the lth step of the Gauss algorithm we interchange the lth and the jp(l)-th row of the intermediate matrix A_{l-1} and also the elements of the right-hand side b_{l-1}. This operation can be expressed with the help of the permutation matrix

$$P_l = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & 0 & \cdots & 1 & & & & \\ & & & & 1 & & & & & \\ & & & \vdots & \ddots & \vdots & & & & \\ & & & & & 1 & & & & \\ & & & 1 & \cdots & 0 & & & & \\ & & & & & & 1 & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & 1 & \end{pmatrix}$$

as

$$\tilde{A}_l = P_l A_{l-1}, \quad \tilde{b}_l = P_l b_{l-1}.$$

Afterwards, we eliminate the subdiagonals of $\tilde{A}_l$ by adding suitable multiples x_{il} of the lth row of $\tilde{A}_l$ to the m_0 following rows and by changing the right-hand side accordingly. In other words,

$$A_l = G_l \tilde{A}_l, \quad b_l = G_l \tilde{b}_l,$$

where the matrix G_l is defined as

$$G_l = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & g_{1l} & 1 & & \\ & & \vdots & & \ddots & \\ & & g_{m_0 l} & & & 1 \end{pmatrix}.$$

After $n - 1$ steps, we obtain the upper triangular matrix

$$U = \left\{ \prod_{l=1}^{n-1} G_l P_l \right\} A \tag{3}$$

and the modified right-hand side

$$r = \left\{ \prod_{l=1}^{n-1} G_l P_l \right\} b. \tag{4}$$

We do not need to calculate the lower triangular matrix L of the decomposition. It suffices to have the elements of the matrices G_l that are stored in the

array XL to perform the operation,

$$r = L^{-1}b. \tag{5}$$

Comparing (4) and (5), we have

$$L^{-1} = \prod_{l=1}^{n-1} G_l P_l. \tag{6}$$

We see now that we can easily perform the operation

$$(L^*)^{-1} = (L^{-1})^* = \prod_{l=1}^{n-1} P_l G^*_l \tag{7}$$

that is required to solve (2). The lth step of solving the system $L^*y = z$ consists of computing

$$\tilde{z}_l = G^*_l z_{l-1} \Leftrightarrow \tilde{z}^j_l = z^j_{l-1} + \sum_{\iota=1}^{m_0} \bar{g}_{\iota l} z^{\iota}_{l-1},$$

where $\bar{g}_{\iota l}$ is the complex conjugate of $g_{\iota l}$, first, and subsequently, of calculating

$$z_l = P_l \tilde{z}_{l-1},$$

i.e., of interchanging the lth and the jp(l)-th element of $\tilde{z}_l$.
It remains to show how to solve

$$U^*z = c. \tag{8}$$

The superdiagonals of U are stored in the arrays AR and AI as follows:

$a_{r,11}$	$a_{r,21}$	$a_{r,31}$	$a_{r,41}$	$a_{r,51}$
$a_{r,12}$	$a_{r,22}$	$a_{r,32}$	$a_{r,42}$	$a_{r,52}$
$a_{r,13}$	$a_{r,23}$	$a_{r,33}$	$a_{r,43}$	$a_{r,53}$
$a_{r,14}$	$a_{r,24}$	$a_{r,34}$	$a_{r,44}$	$a_{r,54}$
$a_{r,15}$	$a_{r,25}$	$a_{r,35}$	$a_{r,45}$	$a_{r,55}$
$a_{r,16}$	$a_{r,26}$	$a_{r,36}$	$a_{r,46}$	0
$a_{r,17}$	$a_{r,27}$	$a_{r,37}$	0	0
$a_{r,18}$	$a_{r,28}$	0	0	0
$a_{r,19}$	0	0	0	0

We could solve (8) as follows:

$$z_1 = \frac{1}{\bar{a}_{11}}$$

$$z_l = \frac{1}{\bar{a}_{1l}}\left[c_l - \sum_{\iota=2}^{\max\{l-1,\, m\}} \bar{a}_{i,\, l-\iota+1} z_{l+1-i}\right], \qquad \text{for} \quad l = 2, \ldots, n. \tag{9}$$

In order to be able to vectorize the calculation of formula (9), we shift the elements in the columns of AR, AI down

$$\tilde{a}_{\iota,j} \leftarrow a_{i,j-i+1} \qquad \text{for} \quad i = 2, \ldots, m, \text{ and } j = n, \ldots, 1.$$

Hence, we can compute z_l as

$$z_l = \frac{1}{\tilde{a}_{1l}}\left[c_l - \sum_{i=2}^{\max\{l-1,\,m\}} \tilde{\tilde{a}}_{i,\,l} z_{l+1-i}\right],$$

i.e., with a formula in which the second index of $\tilde{\tilde{a}}$ is independent of i.

THE GENERALIZED INVERSE RAYLEIGH ITERATION

If we know an approximation of an isolated eigenvalue λ of the complex matrix A, we can calculate λ by choosing two initial vectors u_0, v_0, for example, $u_0 = v_0 = (1, \ldots, 1)^T$, and use the following iteration scheme:

(1) Solve the two systems
$$(A - \lambda_{i-1} I)x = u_{i-1},$$
$$(A - \lambda_{i-1} I)^* y = v_{i-1}.$$

(2) Normalize the solution vectors
$$u_i = \|x\|^{-1} x,$$
$$v_i = \|y\|^{-1} y.$$

(3) Update λ
$$\lambda_i = \lambda_{i-1} + \frac{v_i^* A u_i}{v_i^* u_i}.$$

The quotient in (3) is a generalized Rayleigh quotient. Therefore, the scheme is called a generalized Rayleigh iteration. It can be shown [2] that the iteration converges cubically.

REFERENCES

1. Dongarra, J J., Moler, C. B., Bunch, J. R., and, Stewart, G. W. *LINPACK User's Guide.* SIAM, Philadelphia, 1979.
2. Ostrowsky, A M. On the convergence of the Rayleigh quotient iteration for the computation of characteristic roots and vectors. I, II, III, IV, V, VI. Arch. rat. *Mech. Anal 1* (1958), 233-241; *2* (1958), 423-428; *3* (1958) 325-340; *3* (1958) 341-347; *3* (1958) 472-481; *4* (1959) 153-165.
3. Smith, B. T., Boyle, J. M., Dongarra, J. J., Garbow, B. S , Ikebe, Y., Klema, V. C., and Moler, C. B. *Matrix Eigensystem Routines—EISPACK Guide* Springer Verlag, Berlin, 1976.
4. Schrauf, G. ALGORITHM 664. A Gauss algorithm to solve systems with large, banded matrices using random-access disk storage. *ACM Trans. Math Softw. 14* (1988), 257-260
5. Schrauf, G An efficient solver of the eigenvalue problem of the linear stability equations for three-dimensional, compressible boundary-layer flows. In *Strömungen mit Ablösung, 6.* DGLR-Bericht 88-05, 1988. DGLR Fach-Symposium 8.-10 Nov 1988, Braunschweig

Received January 1990; revised July 1990; accepted July 1990