stichting
mathematisch
centrum

MC

AFDELING NUMERIEKE WISKUNDE
(DEPARTMENT OF NUMERICAL MATHEMATICS)

NW 69/79 AUGUSTUS

M. LOUTER-NOOL

THE CALCULATION OF EIGENVECTORS AND INVARIANT SUBSPACES

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

1980 Mathematics subject classification: 65F15

The calculation of eigenvectors and invariant subspaces

by

M. Louter-Nool

ABSTRACT

This paper contains PASCAL procedures for the calculation of eigenvalues of a real possibly defective and/or derogatory matrix and the corresponding eigenvectors and invariant subspaces.

KEY WORDS & PHRASES: *real possibly defective and/or derogatory matrices, eigenvalues, eigenvectors, invariant subspaces*

Contents

1. INTRODUCTION

In this paper we consider the algebraic eigenvalue problem of real general matrices, especially of matrices having multiple eigenvalues.

If a matrix is defective, then there is no complete set of eigenvectors. Most published procedures fail to recognize a matrix to be defective and attempt to give the same number of eigenvectors as eigenvalues. In case of a defective matrix or one, that is close to a defective matrix, it is more satisfactory to determine invariant subspaces associated with groups of eigenvalues.

The vectors $v_1,v_2,\ldots,v_r$ are said to span an *invariant subspace* of A of order r, if $v_1,v_2,\ldots,v_r$ are linearly independent and $Av_i (i = 1,2,\ldots,r)$ lies in that same subspace. The vectors $v_1,v_2,\ldots,v_r$ are called principal vectors or 'grade vectors'. In GOLUB and WILKINSON [2] an algorithm is given to determine such vectors.

We have chosen the Schur-decomposition as a stable method for calculating eigenvalues.

The computed eigenvalues of defective and/or derogatory matrices can be widely distributed around the exact (multiple) eigenvalue. In practice, it is very difficult to recognize clusters of eigenvalues.

Suppose, we have found a cluster corresponding with eigenvalue λ of order r, then it would be convenient to find an invariant subspace of the same order by means of the method of Golub and Wilkinson. However, the number of computed grade vectors is restricted by a chosen tolerance in the algorithm of Golub and Wilkinson. When the order of the cluster does not fit the number of grade vectors that is found, two possibilities are left:

1. one could again calculate the grade vectors with another tolerance
2. the cluster was ill-chosen. One could repeat the calculation with another cluster size.

In section 5 we shall give some numerical results on the perturbations of multiple eigenvalues and on the interaction between the order of a cluster and the number of vectors that is found.

In section 2 a brief account is given on the Schur-decomposition. In section 3 we shall give an elucidation of the algorithm of Golub and Wilkinson.

Both the Schur-decomposition and the algorithm to determine the grade vectors are implemented in PASCAL. In section 4 we shall describe the PASCAL procedures with their parameters. Moreover, the sourcetexts are listed.

2. THE SCHUR-DECOMPOSITION

In this section we shall describe an algorithm for finding all eigenvalues of a real general matrix. In numerical analysis most of the methods for solving the eigenvalue problem for a matrix A of general form depend on the application of a series of similarity transformations, which converts A into a matrix of special form. If a matrix can be reduced to triangular form, then the diagonal elements of the triangular matrix are the eigenvalues of the original matrix. In this paper we shall consider real general matrices with eigenvalues, which may be complex. If A has complex eigenvalues and we reduce A to triangular form, then A will be converted into a complex matrix. Fortunately, complex eigenvalues always occur in conjugate pairs. Instead of reducing A to triangular form, it appears to be more convenient to reduce A to a real quasi-triangular form, in which the 2×2 blocks have conjugate eigenvalues. In STEWART [9], one can find the following theorem.

THEOREM 2.1. *Let* $A \in \mathbb{R}^{n\times n}$. *Then there exists an orthogonal matrix* U, *such that* U^TAU *is real quasi-triangular. Moreover,* U *may be chosen so that any* 2×2 *diagonal block of* U^TAU *has only complex eigenvalues (which must therefore be conjugates).*

The decomposition by unitary transformations to a real quasi-triangular form is associated with the name of Schur.

As a consequence of this theorem we have at our disposal a method to determine the eigenvalues of A without using complex arithmetic, even when A has complex eigenvalues.

The reduction of A consists of two main parts. We shall first reduce A to upper-Hessenberg form, i.e. to a matrix $A^{(1)}$, such that

$$a_{ij}^{(1)} = 0 \qquad (i > j+1).$$

The latter part of the reduction is performed by means of the QR algorithm.

In section 2.1 we shall give a brief account on the reduction to upper-Hessenberg form. In section 2.2 we shall elucidate the QR algorithm.

2.1. The reduction to upper-Hessenberg form

In the literature, several methods are known to reduce a matrix to upper-Hessenberg form (e.g. see WILKINSON [10]). We shall reduce a given matrix by unitary transformations necessary for the Schur-decomposition. This reduction is based upon pre- and postmultiplications by elementary reflectors, which are also known as elementary Hermitian matrices or as Householder matrices.

2.2. The QR algorithm

The QR algorithm with shifts of origin is described by the relations

$$\left.\begin{aligned} Q_s(A_s-k_sI) &= R_s \qquad (2.2.1)\\ A_{s+1} &= R_sQ_s^T+k_sI \qquad (2.2.2)\end{aligned}\right\}\quad s = 0,1,2,\ldots$$

where Q_s is orthogonal, R_s is upper triangular and k_s is a scalar, the shift of origin. From (2.2.1) and (2.2.2), it follows that

$$A_{s+1} = Q_sA_sQ_s^T. \qquad (2.2.3)$$

In this way, we produce a sequence of similar matrices and it turns out, that with a proper choice of the shift k_s, the off-diagonal elements in the last row of A_s can be made to approach zero very swiftly. When the QR algorithm is applied to a full matrix, it is computationally very expensive. However, when the QR algorithm is applied to an upper-Hessenberg matrix, the volume of work is far less. Moreover, if A_0 is in upper-Hessenberg form, then so are the subsequent matrices A_s.

The shift can be chosen in different ways. In order to achieve rapid convergence, it is essential that the shift should be close to an eigenvalue of A_0. Suppose A_0 has merely real eigenvalues. If we take k_s equal to a_{nn}, then the convergence is quadratical for a simple eigenvalue and linear for a multiple one. WILKINSON [10] suggested another shift-strategy, which, in practice, appears to give better results. This shift-strategy can be represented in the following way:

If $B^{(s)}$ is denoted by

$$B^{(s)} = \begin{bmatrix} a^{(s)}_{n-1n-1} & a^{(s)}_{n-1n} \\ a^{(s)}_{nn-1} & a^{(s)}_{nn} \end{bmatrix} \tag{2.2.4}$$

then k_s is chosen to be that eigenvalue of $B^{(s)}$, which is closest to $a^{(s)}_{nn}$.

However, even when A_0 is real, some of the eigenvalues may be complex. If $B^{(s)}$ in (2.2.4) has complex eigenvalues, then k_s should be complex and so A_{s+1}. In order to avoid complex arithmetic, which is very expensive, one can use a variant of the QR-algorithm, the so-called 'double' QR-algorithm.

This algorithm is based upon two successive single QR-iteration steps with shifts equal to the eigenvalues of the matrix $B^{(s)}$. For further details of this method we refer to STEWART [9].

Though the convergence of the double QR-algorithm is comparable with that of the single QR-algorithm, and, moreover, with only double QR-iteration steps, one could determine the eigenvalues of a real matrix with both real and complex eigenvalues, we apply a combination of both methods. This combination implies, that, if the eigenvalues of the 2×2 bottom righthand side corner are real, we perform a single QR-iteration step, otherwise a double QR-iteration step.

For further details see method and performance of subsection 4.2. We refer also to subsection 5.4.

3. AN ALGORITHM FOR DETERMINATION OF GRADE VECTORS

In GOLUB and WILKINSON [2] an algorithm is suggested for the determination of grade vectors. In this section we shall give some further reflections on this method.

Moreover, we have implemented this algorithm in PASCAL for real matrices having real eigenvalues (see subsection 4.4). However, this method for calculating grade vectors can also be applied to complex matrices.

3.1. Introduction

We shall first give a brief account to the theory of canonical forms and the associated vectors.

Let us consider the (complex) matrix A of order n. There exists a non-singular matrix X, such that

$$X^{-1}AX = J, \qquad AX = XJ, \tag{3.1.1}$$

where J is the Jordan canonical form (J.c.f.) of A. This J.c.f. is a block diagonal matrix with each diagonal block being an elementary Jordan block. Such a block associated with an eigenvalue λ_i of A will be denoted by $J_r(\lambda_i)$, where

$$J_r(\lambda_i) = \begin{bmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix}$$

is of order r. Apart from the ordering of blocks along the diagonal of J, the J.c.f. is unique. If A is a matrix of order 10 with only 2 distinct eigenvalues λ_1 and λ_2, and λ_1 is associated with one block of order 2 and one of order 3, while λ_2 is associated with one block of order 1 and one of order 4, its J.c.f. could be presented in the form

$$\begin{bmatrix} J_2(\lambda_1) & & & \\ & J_3(\lambda_1) & & \\ & & J_1(\lambda_2) & \\ & & & J_4(\lambda_2) \end{bmatrix}.$$

Let us consider the $J_3(\lambda_1)$ block, then we get from (3.1.1):

$$\begin{aligned} Ax_3 &= \lambda_1 x_3, & (A-\lambda_1 I)x_3 &= 0, \\ Ax_4 &= \lambda_1 x_4 + x_3, & (A-\lambda_1 I)x_4 &= x_3, \\ Ax_5 &= \lambda_1 x_5 + x_4, & (A-\lambda_1 I)x_5 &= x_4, \end{aligned} \tag{3.1.2}$$

where x_i is the i-th column vector of X.

The first of these relations implies, that x_3 is an eigenvector of A corresponding to λ_1. The remaining equations imply

$$(A-\lambda_1 I)^2 x_4 = 0, \qquad (A-\lambda_1 I)^3 x_5 = 0. \tag{3.1.3}$$

In general, vectors x_j which satisfy the relations

$$(A-\lambda_i I)^{p-1} x_j \neq 0 \quad \text{and} \quad (A-\lambda_i I)^p x_j = 0 \tag{3.1.4}$$

are called *vectors of grade* p. Moreover, if such vectors satisfy

$$(A-\lambda_i I)x_j = x_{j-1}, \tag{3.1.5}$$

then they are called *principal vectors*.

We shall give some more definitions, which should appear in the introduction.

A matrix is said to be *defective*, if the J.c.f. is not strictly diagonal.

A matrix is said to be *derogatory*, if there is at least one λ_i, which is associated with more than one diagonal block in the J.c.f. .

Let n_j be the number of vectors of grade j associated with some eigenvalue λ. Notice, that n_1 is the number of blocks associated with λ in the J.c.f.. In general, n_j is the number of those blocks, which are of dimensions not less than j. Consequently, we may conclude

$$n_j \geq n_{j+1} \tag{3.1.6}$$

for j = 1,2,...,s-1, while $n_{s+1} = 0$. Furthermore, if λ is an eigenvalue of multiplicity k, we have

$$k = \sum_{j=1}^{s} n_j. \tag{3.1.7}$$

In the next subsections, the vectors of grade j will be denoted by $x_1^{(j)}, x_2^{(j)}, \ldots, x_{n_j}^{(j)}$.

3.2. The algorithm for determination of grade vectors

Let A be a possibly defective and/or derogatory matrix of order n and let λ be an eigenvalue of A of multiplicity k. Consider matrix B, where

$$B = A - \lambda I.$$

Let us assume, that B possesses n_1, singular values equal to zero, then B is called to be of *nullity* n_1. This implies, that A has n_1 independent eigenvectors associated with λ. If $n_1 < k$, then A does not have a complete set of eigenvectors. In that case, we wish to determine the invariant subspace of order k spanned by the vectors of grade 1, grade 2 and those of higher grade.

The algorithm of Golub and Wilkinson describes a method for calculating these vectors. In the following subsections, we shall discuss some stages of the process, starting with the determination of the vectors of grade 1, the eigenvectors.

3.2.1. The determination of vectors of grade 1

In this subsection we shall describe the vectors of grade 1. The vectors are defined by

$$Bx = 0. \tag{3.2.1.1}$$

By means of the singular value decomposition (S.V.D) on B, we obtain

$$B = U \Sigma V^H, \tag{3.2.1.2}$$

U and V being unitary matrices of order n and Σ being a diagonal matrix. From (3.2.1.1) and (3.2.1.2), it follows, that

$$U^H Bx = \Sigma(V^H x) = \Sigma y. \tag{3.2.1.3}$$

The assumption, B being of nullity n_1, implies that Σ must contain n_1

diagonal elements equal to zero, supposed to be in the first n_1 positions. For this reason, the first n_1 components of y are arbitrary and we may take for y in turn each of the first n_1 column vectors of the identity matrix. Hence, we can choose as the eigenvectors the first n_1 column vectors of the matrix V. Therefore the vectors of grade 1 are orthogonal.

3.2.2. The determination of vectors of grade 2

Before we shall describe the second stage of the algorithm, we give the following lemma.

LEMMA 3.2.2. *If* x *is any vector of grade* s,

$$B^s x = 0, \qquad B^{s-1} x \neq 0$$

then Bx *is a vector of grade* s-1 *and hence lies in the subspace spanned by vectors of grade* s-1 *and possibly by vectors of lower grade.*

Hence, if we define

$$Bx = ([x_1^{(1)}|x_2^{(1)}|..|x_{n_1}^{(1)}])z, \qquad (3.2.2.1)$$

where z denotes $(\omega_1, \omega_2, \dots, \omega_{n_1})^T$, then Bx lies in the subspace spanned by the vectors of grade 1. Now, the problem is to find a vector z under the condition of x being a vector of grade 2.

Premultiplication of (3.2.2.1) by U^H of (3.2.1.2) gives

$$U^H Bx = ([U^H x_1^{(1)}|U^H x_2^{(1)}|..|U^H x_{n_1}^{(1)}])z. \qquad (3.2.2.2)$$

If we denote $R_1 = [U^H x_1^{(1)}|U^H x_2^{(1)}|..|U^H x_{n_1}^{(1)}]$, then (3.2.2.2) becomes

$$U^H Bx = R_1 z. \qquad (3.2.2.3)$$

However, from (3.2.1.2), we know

$$U^H Bx = \Sigma(V^H x) = \Sigma y. \qquad (3.2.2.4)$$

Since the first n_1 elements of Σ are equal to zero, z must be a vector, such that

$$R_1 z = p = (0,\dots,0,\pi_{n_1+1},\dots,\pi_n)^T. \tag{3.2.2.5}$$

However, we still have to compute such a vector.

Suppose, we can construct a nonsingular matrix Z_1 of order n_1, such that

$$R_1 Z_1 = \begin{array}{c} \\ \left[\begin{array}{cc} \begin{array}{c} n_2 \\ 0..0 \\ \vdots \quad \vdots \\ 0..0 \\ x..x \\ \vdots \quad \vdots \\ x..x \end{array} & \begin{array}{c} n_1-n_2 \\ x..x \\ \vdots \quad \vdots \\ x..x \\ x..x \\ \vdots \quad \vdots \\ x..x \end{array} \end{array}\right] \end{array} \begin{array}{l} n_1 \\ \\ \\ n-n_1 \end{array} = [p_1|\dots|p_{n_2}|p_{n_2+1}|\dots|p_{n_1}] = P_1 \tag{3.2.2.6}$$

then the first n_2 column vectors of Z_1 satisfy (3.2.2.5). Solving

$$\Sigma y_i = p_i (i = 1,2,\dots,n_2) \quad \text{and} \quad x_i = V y_i, \tag{3.2.2.7}$$

we obtain the grade vectors $x_1^{(2)}, x_2^{(2)}, \dots, x_{n_2}^{(2)}$. Obviously, the first n_1 positions of y_i are again arbitrary. Taking the elements zero, then the vectors of grade 2 are orthogonal to the eigenvectors.

The nonsingular matrix Z_1 can be conveniently computed by means of an S.V.D-composition. The matrix R_1 is partitioned into

$$\begin{bmatrix} R_1^{(1)} \\ \hline R_1^{(2)} \end{bmatrix} \tag{3.2.2.8}$$

where $R_1^{(1)}$ being the upper matrix of R_1 of order n_1 and $R_1^{(2)}$ being the remaining matrix. An S.V.D. on $R_1^{(1)}$ yields

$$R_1^{(1)} = U_1 \Sigma_1 V_1^H. \tag{3.2.2.9}$$

Taking $Z_1 = V_1$, we may write

$$R_1 Z_1 = \left[\begin{array}{c} U_1\Sigma_1 \\ \hdashline R_1^{(2)}V_1 \end{array}\right] = P_1. \qquad (3.2.2.10)$$

If $R_1^{(1)}$ is of nullity n_2, the number of vectors of grade 2, according to the J.c.f. of A, as defined in subsection 3.1, we shall find all vectors of grade 2 in the way we have outlined above.

<u>THEOREM 3.2.2</u>. *The nullity of* $R_1^{(1)}$ *is equal to* n_2.

<u>PROOF</u>. Let m be the nullity of $R_1^{(1)}$ and suppose $m < n_2$. This means, we have not determined all independent vectors of grade 2. Then there will be at least one vector z, linearly independent of the first m column vectors of Z_1, such that $R_1 z = 0$. However, this is in contradiction with the nullity of $R_1^{(1)}$, so necessarily m must be greater than or equal to n_2.

Suppose, $m > n_2$. In that case, we did find more vectors, than there should be accordingly to the J.c.f.. Then the vectors found of grade 1 and 2 are linearly dependent. However, we can prove these vectors to be linearly independent. Let $x_1^{(1)},\dots,x_{n_1}^{(1)},x_1^{(2)},\dots,x_m^{(2)}$ be linearly dependent. Then

$$\sum_{i=1}^{n_1} \alpha_i x_i^{(1)} + \sum_{i=1}^{m} \beta_i x_i^{(2)} = \vec{0} \quad \text{for some } \alpha_i\text{'s and } \beta_i\text{'s} \neq 0. \qquad (3.2.2.11)$$

Premultiplying (3.2.2.11) by $U^H B$ yields

$$\sum_{i=1}^{n_1} \alpha_i U^H(Bx_i^{(1)}) + \sum_{i=1}^{m} \beta_i (U^H B x_i^{(2)}) = \vec{0} \qquad (3.2.2.12)$$

$$\Rightarrow \sum_{i=1}^{n_1} \alpha_i U^H \vec{0} + \sum_{i=1}^{m} \beta_i p_i = \sum_{i=1}^{m} \beta_i p_i = \vec{0} \qquad (3.2.2.13)$$

Since the vectors $p_i (i = 1,\dots,m)$ are linearly independent all β_i's should be zero. Moreover, the vectors $x_1^{(1)},\dots,x_{n_1}^{(1)}$ are linearly independent, (see subsection 3.2.2), so all α_i's should be zero too in relation (3.2.2.11).

This implies m is equal to n_2.

3.2.3. The determination of vectors of grade 3 and of higher grade

Analogous to relation (3.2.2.1), we define

$$Bx = ([x_1^{(2)}|..|x_{n_2}^{(2)}|x_1^{(1)}|..|x_{n_1}^{(1)}])z, \tag{3.2.3.1}$$

so Bx lies in the subspace spanned by vectors of grade 1 and grade 2. Premultiplication by the matrix U^H of (3.2.1.2) and extension of the vector z to a nonsingular matrix Z of order n_1+n_2 yields

$$U^H BX = ([U^H x_1^{(2)}|..|U^H x_{n_2}^{(2)}|U^H x_1^{(1)}|..|U^H x_{n_1}^{(1)}])Z. \tag{3.2.3.2}$$

If we take $Z = [^{I}\ _{Z_1}]\,W$, where w is a nonsingular matrix of order n_1+n_2 and where $Z_1 = V_1$ (see (3.2.2.9)), the relation (3.2.3.2) reduces to

$$U^H BX = ([U^H x_1^{(2)}|..|U^H x_{n_2}^{(2)}|p_1|..|p_{n_2}|p_{n_2+1}|..|p_{n_1}])W. \tag{3.2.3.3}$$

The vectors $p_1,p_2,...,p_{n_2}$ have already been dealt with and gave us the vectors $x_1^{(2)},x_2^{(2)},...,x_{n_2}^{(2)}$.

Suppose, z is a vector, such that (3.2.3.1) yields a vector of grade 3. If w_α is defined by

$$w_\alpha = \alpha_0 z + \sum_{i=1}^{n_2} \alpha_i e_{n_2+i}, \tag{3.2.3.4}$$

e_{n_2+i} being the (n_2+i)-th column vector of the identity matrix, then ω_α too is a vector, such that (3.2.3.1) yields a vector of grade 3, or, in case $\alpha_0 = 0$, one of lower grade.

It is evident, that the purpose of the algorithm is to find a set of independent grade vectors. In order to avoid the matrix W to contain more than one vector w_α, the column vectors $p_1,p_2,...,p_{n_2}$ are omitted. Equation (3.2.3.3) becomes

$$U^H BX = ([U^H x_1^{(2)}|..|U^H x_{n_2}^{(2)}|p_{n_2+1}|..|p_{n_1}])Z_2. \tag{3.2.3.5}$$

Notice, R_2 again is an n by n_1 matrix and consequently, Z_2 is of order n_1.

A partition of R_2 into $R_2^{(1)}$ and $R_2^{(2)}$ according to (3.2.2.8) and an S.V.D. on $R_2^{(1)}$

$$R_2^{(1)} = U_2 \Sigma_2 V_2^H, \tag{3.2.3.6}$$

gives us $Z_2 = V_2$. The vectors of grade 3 are obtained analogous to (3.2.2.7), where the vectors p_i are now the first n_3 column vetors of the matrix P_2.

In general, at the (s+1)-th stage, we replace the first n_s column vectors of the matrix

$$P_{s-1} = R_{s-1}Z_{s-1}, \tag{3.2.3.7}$$

which have given us the grade vectors $x_1^{(s)},x_2^{(s)},\dots,x_{n_s}^{(s)}$ by $U^H x_1^{(s)}$, $U^H x_2^{(s)},\dots,U^H x_{n_s}^{(s)}$. Notice, that the remaining vectors are linear combinations of vectors of grade s-1 and of lower grade. The so-obtained matrix R_s is partitioned into $R_s^{(1)}$ and $R_s^{(2)}$ and by means of an S.V.D. on $R_s^{(1)}$, we get the matrix $Z_s = V_s$. The algorithm is now complete. The process terminates when the nullity of $R_s^{(1)}$ is zero.

Obviously, at the (s+1)-th stage, the last n_1-n_s column vectors of $R_s^{(1)}$ (i.e. the upper parts of the remaining vectors of P_{s-1}) are linear independent. So, a column vector p of P_s with zeros in the first n_1 positions can only be produced by at least one of the vectors $U^H x_1^{(s)}, U^H x_2^{(s)},\dots,U^H x_{n_s}^{(s)}$. This implies, that we do produce vectors of the requisite grade s+1 and no vectors of lower grade.

Analogous to theorem 3.2.2., one can prove, the nullity of $R_s^{(1)}$ to be equal to n_{s+1}, the number of vectors of grade s+1 according to the J.c.f. .

As we have mentioned previously, all successive S.V.D!s are performed on a matrix of order n_1 and the process terminates when the nullity of $R_s^{(1)}$ is equal to zero. Notice, that in case $n_1 = 1$ (only one Jordan block is associated with λ), $R_s^{(1)}$ will be, at each stage, a 1×1 matrix, so, in that case, the process terminates, when the first element of $U^H x_1^{(s)}$ is nonzero.

Finally, we remark, that the vectors of grade 1 are orthogonal and by taking the first n_1 components of the subsequent solutions of $\Sigma y = p$ (see (3.2.2.7)) to be zero, all vectors of grade higher than one are orthogonal to those eigenvectors. But, thus is not true of any of the subsequent sets of vectors.

4. THE PROCEDURES

In this section we shall describe the PASCAL-procedures [5], which are an implementation of the algorithms of the sections 2 and 3. All procedures were tested on the Control Data Cyber 73/173. In section 4.1, the types, that are used, are listed. In the sections 4.2, 4.3 and 4.4 we shall give a brief description of the procedures hessenberg, psthesmat, schur and gradevec with their parameters. The sourcetexts of the procedures are listed at the end of each subsection.

4.1. The types

In this section we shall list the types, that are used in the procedures

```
inx1 = 1 .. nmax;
sub1 = 1 .. nmax1;(* nmax1 = nmax - 1*)
sub2 = 1 .. nmax2;(* nmax2 = nmax - 2*)
mat1 = array [inx1, inx1] of real;
vec1 = array [sub1] of real;
vec2 = array [sub2] of real;
complex = record re, im: real end;
vecc1 = array [inx1] of complex;
pelement = ^element;
element = record number: integer; next: pelement end;
recaux = record tol, correction: real;
                maxit, counts, countd: integer
         end
gradeaux = record tol, min, max: real end;
```

4.2. The procedures hessenberg and psthesmat

In this subsection we shall describe two procedures:

1. procedure hessenberg.

This procedure transforms a given matrix A to upper-Hessenberg form H by means of premultiplying and postmultiplying the given matrix by orthogonal matrices.

2. procedure psthesmat.
This procedure calculates the postmultiplication matrix U from the data generated in procedure hessenberg. The transpose of the postmultiplication matrix is equal to the premultiplication matrix. So, $H = U^T AU$.

The heading of the procedure hessenberg is:

```
procedure hessenberg (var a: mat1; n: inx1; var b:
   vec1; var pi: vec2);
```

The meaning of the formal parameters is:

```
var a: mat1;
       entry: the given matrix;
       exit : the upper triangular matrix (incl. diagonal)
              is a part of the upper-Hessenberg matrix; the
              strictly lower triangular matrix contains the data
              for procedure psthesmat;
n           : inx1;
              the order of the matrix; n ≤ nmax;
var b       : vec1;
              exit: the subdiagonal of the Hessenberg matrix;
var pi      : vec2;
              exit: this array contains data for procedure psthesmat.
```

The heading of procedure psthesmat is:

```
procedure psthesmat (var a, u: mat1; n: inx1;
   var pi: vec2);
```

The meaning of the formal parameters is:

```
var a: mat1;
       entry: the data concerning the postmultiplication
              matrix as generated by procedure hessenberg;
```

var u : mat1;
exit: the postmultiplication matrix;
n : inx1;
the order of the matrix; n ≤ nmax;
var pi: vec2;
entry: the array as generated by procedure hessenberg.

Both procedures make use of the following external functions/procedures from [6]:

function arreb: real;
function matmat(ℓ, u, i, j: inx1; var a,b: mat1): real;
function tammat(ℓ, u, i, j: inx1; var a,b: mat1): real;
procedure elmcol(ℓ, u, i, j: inx1; var a,b: mat1; s: real);
procedure elmrowvec(ℓ, u, i, j: inx1; var a,b: mat1; s: real);

Method and performance:

Let us assume a given matrix A of order n is stored in the array A[1..nmax, 1..nmax]. The procedure hessenberg reduces A to upper-Hessenberg form H by pre- and post- multiplying it with orthogonal matrices. The matrix H is overwritten on A with details on the transformations in the lower triangular matrix. The subdiagonal of H is stored in the array B[1..nmax1]. The array PI[1..nmax2] contains some further details on the transformations.

The procedure psthesmat calculates the postmultiplication matrix U from the information stored in the lower triangular matrix of A and the array PI.

The procedure hessenberg is an implementation of the algorithm described in [9]. However, here we skip a transformation, if the column on which our attention is focussed is already (approximately) in the desired form; i.e. if the maximum of the absolute values of the elements, that ought to be zero is smaller than a certain constant.

Source texts:

```
procedure hessenberg (var a: mat1; n: inx1; var b: vec1;
                      var pi: vec2);
   var k, k1, i, j: inx1;
       max, norm, rho, sigma: real;
begin (* hessenberg *)
   norm := 0; for i := 1 to n do
   for j := 1 to n do
   if abs (a[i,j]) > norm then norm := abs (a[i,j]);
   for k := 1 to n - 2 do
   begin k1 := k + 1; max := 0;
      for i := n downto k + 2 do
      if abs (a[i,k]) > max then max := abs (a[i,k]);
      if max < arreb * norm then
      begin pi[k] := 0; b[k] := a[k1,k] end
      else
      begin if abs (a[k1,k]) > max then max := abs (a[k1,k]);
         for i := k1 to n do a[i,k] := a[i,k]/max;
         sigma := sqrt (tammat (k1, n, k, k, a, a));
         if a[k1,k] < 0 then sigma := - sigma;
         a[k1,k] := a[k1,k] + sigma;
         pi[k] := sigma * a[k1,k];
         b[k] := - max * sigma;
         (* premultiply *)
         for j := k1 to n do
         elmcol (k1, n, j, k, a, a, -tammat (k1, n, k, j, a, a)
                                    /pi[k]);
         (* postmultiply *)
         for i := 1 to n do
         elmrowcol (k1, n, i, k, a, a, -matmat (k1, n, i, k, a, a)
                                        /pi[k])
      end
   end; b[n-1] := a[n,n-1]
end (* hessenberg *);
```

```
procedure psthesmat (var a, u: mat1; n: inx1; var pi: vec2);
   var k, k1, i, j: inx1;
begin (* psthesmat *)
   k1 := n - 1;
   u[n,n] := 1; u[k1,n] := 0; u[n,k1] := 0; u[k1,k1] := 1;
   for k := n - 2 downto 1 do
   begin if pi[k] <> 0 then
      for j := k1 to n do
      elmcol (k1, n, j, k, u, a,
              -tammat (k1, n, k, j, a, u) / pi[k]);
      for j := k1 to n do
      begin u[k,j] := 0; u[j,k] := 0 end;
      u[k,k] := 1; k1 := k
   end
end (* psthesmat *);
```

4.3. The function schur

In this subsection we shall describe the function schur. This function calculates the real and/or complex eigenvalues of a real upper-Hessenberg matrix.

The heading of the function schur is:

```
function schur (var a: mat1; n: inx1; var eig: vecc1;
   var aux: recaux): integer;
```

schur := the number of eigenvalues not calculated. So schur = 0 means, that the process is completed within the maximum allowed number of iterations.

The meaning of the formal parameters is:

var a: mat1;

entry: the upper-hessenberg matrix;

exit : the matrix is used as working space, so its contents is disturbed;

n   : inx1;

        the order of the matrix; n ≤ nmax;

<u>var</u> eig: vecc1;

        the eigenvalues of the matrix;

        if only n-k eigenvalues are calculated, then these values are stored in eig[k+1..n];

<u>var</u> aux: recaux;

        entry:

        aux.tol: real;

            the absolute tolerance for the QR-iteration;

        aux.correction: real;

            see method and performance (this subsection);

        aux.maxit: integer;

            the maximum allowed number of iterations;

        exit:

        aux.countd: integer;

            the number of QR-double iterations performed;

        aux.counts: integer;

            the number of QR-single iterations performed;

Function schur makes use of the following external procedures from [6]:

<u>procedure</u> rotcol (ℓ, u, i, j: inx1; <u>var</u> a: mat1; c, s: real);
<u>procedure</u> rotrow (ℓ, u, i, j: inx1; <u>var</u> a: mat1; c, s: real);

The local functions/procedures of function schur are:

1. <u>function</u> rot2 (<u>var</u> α,β: real): real;
   given the scalars α and β, this function returns the scalars γ and σ ($\gamma^2+\sigma^2 = 1$) and gives as a result the scalar ν, such that

$$\begin{pmatrix} \gamma & \sigma \\ -\sigma & \gamma \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \nu \\ 0 \end{pmatrix} .$$

2. <u>function</u> rot3 (<u>var</u> α, β, γ, π: real): real;
   given the vector $(\alpha,\beta,\gamma)^T$, this function returns the scalars v_1, v_2, v_3 and π and gives as a result the scalar ν, such that

$$(I-\pi^{-1}VV^T)\begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix}=\begin{pmatrix}-\nu\\ 0\\ 0\end{pmatrix},$$

where $V = (v_1,v_2,v_3)^T$ and $\pi = \frac{1}{2}.\ \|V\|_2^2$.

3. <u>procedure</u> qrsingle (ℓ, u: inx1; shift: real);
   this procedure performs one single QR-iteration step on the matrix a[ℓ..u,ℓ..u]. For information about the parameter shift, see method and performance (this subsection).
4. <u>procedure</u> qrdouble (ℓ, u: inx1; shift: real);
   this procedure performs one double QR-iteration step on the matrix a[ℓ..u,ℓ..u]. For information about the parameter shift, see method and performance (this subsection).
5. <u>function</u> deflation (u: inx1): inx1;
   this function gives as a result the row number of the first element on the subdiagonal (from u downto 1), which is smaller than aux.tol.

Method and performance:

The method used in function schur for calculating the eigenvalues of a real upper-Hessenberg matrix H of order n stored in the array A[1..nmax,1..nmax] is a combination of the QR-single and QR-double iteration. If the eigenvalues of the lower righthand 2 by 2 matrix (say B) of the considered principal submatrix H' of H are real or complex, then respectively a QR-single or QR-double iteration step is performed.

In the same way as described in [1, section 241], an iterate H is partitioned into 4 submatrices, if for some k, the absolute value of the k-th element of the subdiagonal is smaller than a certain tolerance.

Before an actual iteration step is applied on an iterate H', an explicit shift is subtracted from the diagonal elements of H'. The value of this shift equals the last diagonal element, here called h'_{kk}, of H' plus a small perturbation ε (see (4.3.1)). The QR-iteration step (i.e. either a QR-single or a QR-double step) is performed on the shifted matrix H'. The (explicit) shift of the QR-single iteration is chosen to be equal to the absolute smallest eigenvalue of $(B-(h'_{kk}+\varepsilon)I)$. The (implicit) shifts of the QR-double iteration are chosen to be equal to the eigenvalues of $(B-(h'_{kk}+\varepsilon)I)$.

Finally, $h'_{kk}+\varepsilon$ is added back to the diagonal elements.

The shift h'_{kk} is disturbed by

$$\varepsilon = \sqrt{\text{aux.correction} * H'[k,k-1] * H'[k-1,k-2]}$$

to attempt any unexpected nonconvergence of the iteration. A suitable choice for aux.correction is the square root of the machine precision.

The eigenvalues of submatrices of order 1 and 2 are calculated directly, so that the process is completed if successive partitionings have led to principal submatrices all of order 1 or 2.

Source text:

```
function schur (var a: mat1; n: inx1; var eig: vecc1;
                var aux: recaux): integer;
   var og, og1, bg, i, l: integer;
       corr, delta, det, discr, s: real;
   function rot2 (var c, s: real): real;
      var delta, max: real;
   begin max := abs (c);
      if abs (s) > max then max := abs (s);
      c := c / max; s := s / max;
      delta := sqrt (sqr (c) + sqr (s));
      c := c / delta;s := s / delta;
      rot2 := max * delta
   end (* rot2 *);

   function rot3 (var a, b, c, pi: real): real;
      var bb, cc, max, sigma: real;
   begin max := abs (a); bb := abs (b); cc := abs (c);
      if bb > max then max := bb; if cc > max then max := cc;
      a := a / max; b := b / max; c := c / max;
      sigma := sqrt (sqr (a) + sqr (b) + sqr (c));
      if a < 0 then sigma := - sigma;
      a := a + sigma; pi := a * sigma;
      rot3 := max * sigma
   end (* rot3 *);
```

```
procedure qrsingle (l, u: inx1; shift: real);
    var k, k1: integer;
        c1, s1, c, s, rot: real;
begin a[l,l] := a[l,l] - shift;
   for k := l to u-1 do
   begin k1 := k+1; a[k1,k1] := a[k1,k1] - shift;
      c := a[k,k]; s := a[k1,k];
      rot := rot2 (c,s);
      if k > l then
      begin a[k,k-1] := rot * s1; rot := rot * c1 end;
      a[k,k] := rot;
      rotrow (k1, u, k, k1, a, c, s);
      rotcol (l, k, k, k1, a, c, s);
      a[k,k] := a[k,k] + shift;
      c1 := c; s1 := s
   end;
   a[u,u-1] := a[u,u] * s1; a[u,u] := a[u,u] * c1 + shift;
   with aux do counts := counts + 1
end (* qrsingle *);

procedure qrdouble (l, u: inx1; shift: real);
   var a1, a2, a3, pi, rot, tau: real;
       j, k, k1, k2, k3, k4, l1, u1, min: integer;
begin l1 := l + 1; u1 := u - 1;
   for k := l - 1 to u - 3 do
   begin k1 := k+1; k2 := k+2; k3 := k+3; k4 := k+4;
      if k > = l then
      begin a1 := a[k1,k]; a2 := a[k2,k]; a3 := a[k3,k] end else
      begin
         a[l,l] := a[l,l] - shift;
         a[l1,l1] := a[l1,l1] - shift;
         a2 := a[u1,u1] - shift;
         a3 :=-a[u,u1] * a[u1,u];
         a1 := (a[l,l] * (a[l,l] - a2) + a3) / a[l1,l] + a[l,l1];
         a2 := a[l,l] + a[l1,l1] - a2;
         a3 := a[l+2,l1]
      end;
      rot := rot3 (a1, a2, a3, pi);
      if k < 1 then a[k3,k1] := 0 else a[k1,k] := -rot;
      a[k3,k3] := a[k3,k3] - shift;

      (* premultiply *)
      for j := k1 to u do
      begin
         tau := (a1 * a[k1,j] + a2 * a[k2,j] + a3 * a[k3,j]) / pi;
         a[k1,j] := a[k1,j] - tau * a1;
         a[k2,j] := a[k2,j] - tau * a2;
         a[k3,j] := a[k3,j] - tau * a3
      end;

      (* postmultiply *)
      if k4 > u then min := u else
      begin min := k4; a[k4,k1] := 0; a[k4,k2] := 0 end;
      for j := 1 to min do
      begin
         tau := (a1 * a[j,k1] + a2 * a[j,k2] + a3 * a[j,k3]) / pi;
         a[j,k1] := a[j,k1] - tau * a1;
         a[j,k2] := a[j,k2] - tau * a2;
         a[j,k3] := a[j,k3] - tau * a3
      end;
      a[k1,k1] := a[k1,k1] + shift
   end;

   a1 := a[u1,u-2]; a2 := a[u,u-2];
   a[u1,u-2] := rot2 (a1,a2);
   rotrow (u1, u, u1, u, a, a1, a2)
   rotcol (1, u, u1, u, a, a1, a2);
   a[u1,u1] := a[u1,u1] + shift; a[u,u] := a[u,u] + shift;
   with aux do countd := countd + 1
end (* qrdouble *);
```

```
   function deflation (u: inx1): inx1;
      var bg: inx1;
          b: boolean;
   begin bg := u; b := true;
      while b and (bg > 1) do
      if abs (a[bg,bg-1]) > aux.tol then
      bg := bg - 1 else
      begin b := false; a[bg,bg-1] := 0 end;
      deflation := bg
   end (* deflation *);

begin og := n; for i := 1 to n do eig[i].im := 0;
   with aux do
   begin countd := 0; counts := 0;
      while (og > 0) and ((counts + countd) < maxit) do
      begin bg := deflation (og); og1 := og - 1;
         l := og - bg;
         if l = 0 then
         begin eig[og].re := a[og,og]; og := og1 end
         else
         begin
            if l > 1 then
            corr := sqrt (correction * abs (a[og,og1]*a[og1,og-2]));
            delta := (a[og1,og1] - a[og,og]) / 2;
            det := a[og,og1] * a[og1,og];
            discr := sqr (delta) + det;
            if discr < 0 then
               if l = 1 then
               with eig[og] do
               begin re := delta + a[og,og];
                  im := sqrt (- discr);
                  eig[og1].re := re;
                  eig[og1].im :=-im;
                  og := og - 2
               end
```

```
                else qrdouble (bg, og, a[og,og] + corr)
             else
             begin if abs(delta) > = tol then
                begin delta := 1 / delta;
                   s := - delta * det / (sqrt(sqr(delta)*det+1)+1)
                end else
                if det < sqr(delta) then s := 0 else s := sqrt(det);
                if 1 = 1 then
                begin eig[og].re := a[og,og] + s;
                      eig[og1].re := a[og1,og1] - s;
                   og := og - 2
                end else
                qrsingle (bg, og, a[og,og] + s + corr)
             end
          end
       end
    end (* aux *); schur := og
end (* schur *);
```

4.4. The procedure gradevec

In this subsection we shall describe the procedure gradevec. By means of this procedure the grade vectors of a real defective and/or derogatory matrix A associated to a real eigenvalue λ are calculated.

The heading of the procedure gradevec is:

```
    procedure gradevec (var a: mat1; n, nmax: inx1; lambda:
       real; col: inx1; var list: pelement; var pv: mat1;
       var aux: gradeaux);
```

The meaning of the formal parameters is:

var a	: mat1;
	the given matrix;
n,max	: inx1;
	n is the order of the matrix; nmax is the length of the rows and columns in the calling program; nmax $\geq$ n;

```
lambda   : real;
           the given eigenvalue;
col      : inx1;
           the column number in which the first grade vector is
           stored (input);
var list: pelement;
           exit: a dynamic variable to a list of elements; an element
           is a record block with two fields: the first field of the
           first element contains the number of vectors of grade one
           (the eigenvectors) associated with lambda; the second field
           is a dynamic variable of type element to the next element,
           which contains the number of vectors of grade 2 and a dynamic
           variable to the next element etc.
           the dynamic variable of the last element of list has the
           value nil; if list = nil, no grade vectors have been found;
var pv   : mat1;
           exit: the grade vectors are stored in the array
           pv[1..n,col..col+(k-1)] where k equals the sum of the first
           fields of the elements of list;
var aux : gradeaux;
           entry:
           aux.tol: real;
              the absolute tolerance for the singular values;
              (see method and performance (this subsection));
           exit:
           aux.min: real;
              the minimum value of the singular values not
              neglected;
           aux.max: real;
              the maximum value of the singular values
              neglected;
```

Procedure gradevec makes use of the following external procedures from [6]:

```
function giant: real;
```

<u>function</u> matmat (ℓ, u, i, j: inxl; <u>var</u> a,b: matl): real;
<u>function</u> tammat (ℓ, u, i, j: inxl; <u>var</u> a,b: matl): real;
<u>procedure</u> ichcol (ℓ, u, i, j: inxl; <u>var</u> a: matl);
<u>procedure</u> plsvalr (<u>var</u> a: matl; n, m, ja, ju, jv: inxl;
   isw: integer; <u>var</u> wk, q: vecl; <u>var</u> u,v: matl);

The local functions/procedures of procedure gradevec are:

1. <u>procedure</u> ichval (<u>var</u> a,b: real);
   this procedure interchanges the values of a and b;
2. <u>function</u> svdsort (<u>var</u> a, u, v: matl; n: inxl; <u>var</u> sigma: vecl): integer;
   given the matrix a of order n; the S.V.D-composition on a, consisting of arrays u and v and sigma, is calculated. Furthermore, the singular values are sorted such that the singular values smaller than aux.tol are stored in the first positions of array sigma. Moreover, the column vectors of u and v are sorted correspondingly.

   svdsort := the number of singular values smaller than aux.tol.

   If svdsort > 0, then a new element is added to list.
3. <u>procedure</u> preuh (<u>var</u> r: matl);
   in this procedure the grade vectors, calculated in the previous step (already stored in pv) are premultiplied by u-transpose (see construction of R_s (3.2.3.7));
4. <u>procedure</u> pstv2 (<u>var</u> p,r: matl);
   the matrix r is postmultiplied by the matrix v2 (=V_2 in (3.2.3.6)) and stored in the matrix p. Moreover, the first n_i column vectors of p are divided by sigma;
5. <u>procedure</u> storepv (<u>var</u> p: matl);
   if storepv is called the first time, pv is filled with the first n_1 column vector of the matrix v1 (= V of (3.2.1.2)), otherwise, with the first n_i column vectors of p premultiplied by v1;
6. <u>procedure</u> supervisor (<u>var</u> p,r: matl);
   this is a recursive procedure calling itself alternatingly with actual parameters p,r and r,p. The procedure terminates when no more singular values smaller than aux.tol are obtained.

Method and performance:

The procedure gradevec calculates 'all' grade vectors associated to a particular real eigenvalue λ of a real defective and/or derogatory matrix A. The eigenvalues of A should have been found using some stable algorithm such as the QR-algorithm (e.g. by means of the function schur subsection 4.3)). However, when A is really defective, the computed eigenvalues λ_i may be arbitrary bad. One should find a cluster of computed eigenvalues around the exact eigenvalue. The problem is how to determine which eigenvalues belong to a certain cluster. If one could recognize a cluster of r eigenvalues close to $\bar{\lambda}$ (where $r\bar{\lambda} = \sum_r \lambda_i$) one should expect to find r grade vectors associated to $\bar{\lambda}$. However, the procedure gradevec terminates when no more singular values smaller than aux.tol are found. So, if gradevec finds s grade vectors and s $\neq$ r, one has to change the value of aux.tol. Some control is possible by means of aux.min and aux.max. When the quotient aux.min/aux.max is sufficiently great, there is a good reason to assume $\bar{\lambda}$ is very close to λ. Suppose s is still unequal to r, then it is to recommend to change the cluster format. In most cases, one will find another $\bar{\lambda}' = (s\bar{\lambda}' = \sum_s \lambda_i)$, which is very close to $\bar{\lambda}$. See also the numerical results of gradevec in subsection 5.6.

In figure 4.4.1 a flow chart of procedure gradevec is given.

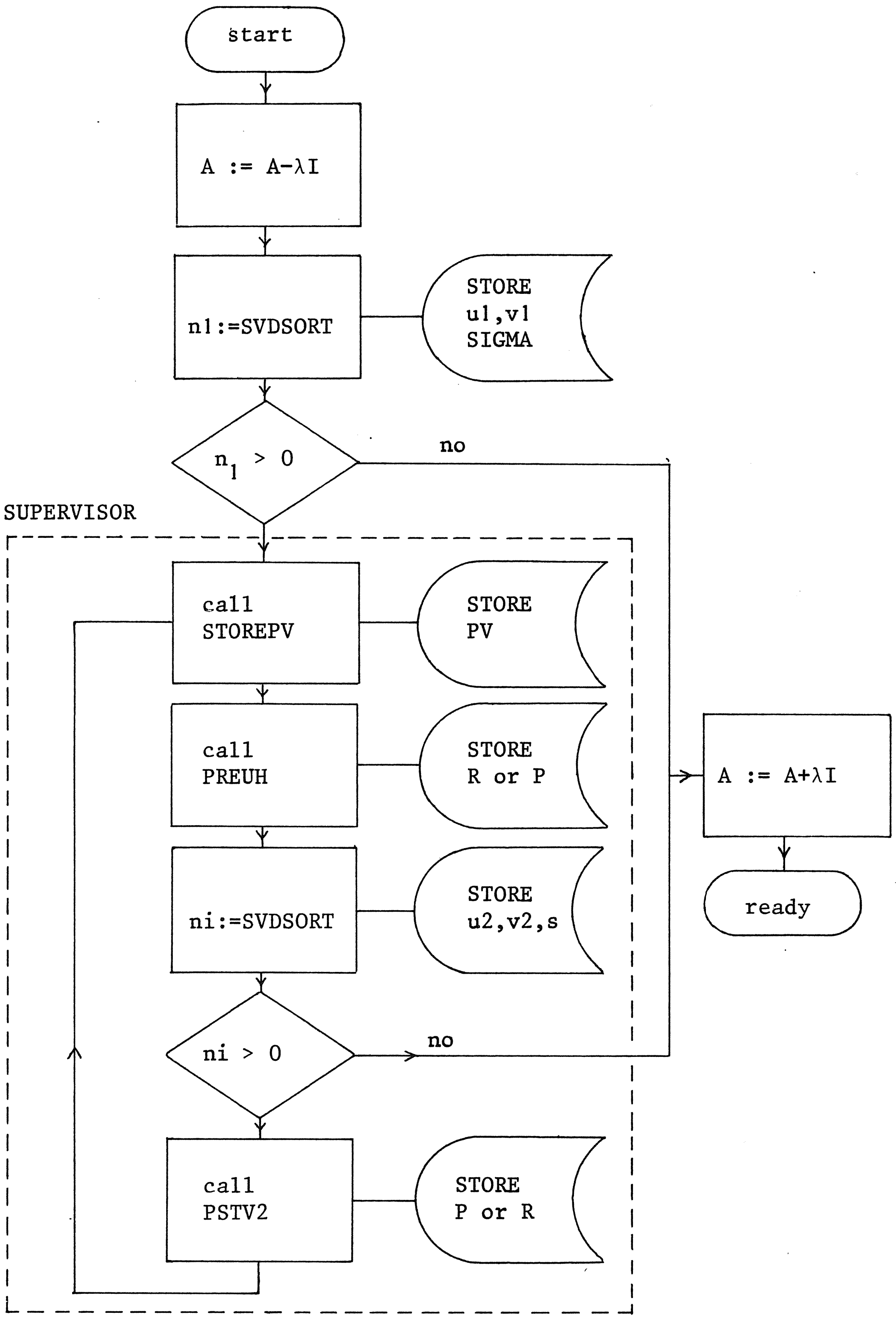

Figure 4.4. A flow-chart of procedure gradevec.

Source text:

```
procedure gradevec (var a: mat1; n, nmax: inx1; lambda: real;
                    col: inx1; var list: pelement; var pv: mat1;
                    var aux: gradeaux);

    var i, n1, ni, num: integer;
        u1, u2, v1, v2, p, r: mat1;
        sigma, s: vec3;
        pt: pelement;
        first: boolean;

  procedure ichval (var a, b: real);
    var s: real;
  begin s := a; a := b; b := s end;

  function svdsort (var a, u, v: mat1; no: inx1; var sigma: vec3):
    integer;
    label 1;
    var k, l: integer;
        wk: vec3;
        z: pelement;
  begin
    with aux do
    begin
      k := 1; l := no + 1;
      plsvalr (a, no, no, nmax, nmax, nmax, 1, wk, sigma, u, v);
      repeat if sigma[k] >= tol then
        begin
          repeat l := l - 1;
            if l = k then goto 1
          until sigma[l] < tol;
          ichcol (1, no, k, l, u);
          ichcol (1, no, k, l, v);
          ichval (sigma[k], sigma[l])
        end; k := k + 1
        until k = 1;
        1: if k > 1 then
        begin new (z); z^.number := k - 1;
           if first then list := z else pt^. next := z; pt := z
        end;
        for 1 := 1 to k - 1 do
        if sigma[1] > max then max := sigma[1];
        for 1 := k to no do
        if sigma[1] < min then min := sigma[1]
     end; svdsort := k - 1
  end (* svdsort *);

  procedure preuh (var r: mat1);
     var i, j: inx1;
         x: real;
  begin
     with aux do
     for j := 1 to ni do
     begin
        for i := 1 to n1 do
        begin x := tammat (1, n, i, j+num, u1, pv);
           if abs (x) < tol then
              if n1 = 1 then r[i,j] := x else r[i,j] := 0
           else r[i,j] := x
        end;
        for i := n1 + 1 to n do
        r[i,j] := tammat (1, n, i, j+num, u1, pv)
     end
  end (* preuh *);

  procedure pstv2 (var p,r: mat1);
     var i, j: inx1;
  begin
     for i := 1 to n1 do
     begin
        for j := 1 to ni do p[i,j] := 0;
```

```
      for j := ni + 1 to n1 do
      p[i,j] := s[j] * u2[i,j]
   end;
   for i := n1 + 1 to n do
   if ni <> n1 then
   begin
      for j := 1 to ni do
      p[i,j] := matmat (1, n1, i, j, r, v2) / sigma[i];
      for j := ni + 1 to n1 do
      p[i,j] := matmat (1, n1, i, j, r, v2)
   end else
   for j := 1 to n1 do p[i,j] := r[i,j] / sigma[i]
end (* pstv2 *);

procedure storepv (var p: mat1);
   var i, j: inx1;
begin
   if first then
   begin first := false;
      for i := 1 to n do
      for j := 1 to ni do
      pv[i,j+num] := v1[i,j]
   end else
   for i := 1 to n do
   for j := 1 to ni do
   pv[i,j+num] := matmat (1, n, i, j, v1, p)
end (* storepv *);

procedure supervisor (var p, r: mat1);
begin
   storepv (r);
   preuh (r);
   num := num + ni;
   ni := svdsort (r, u2, v2, n1, s);
   if ni > 0 then
   begin
```

```
            pstv2 (p,r);
            supervisor (r,p)
         end
      end (* supervisor *);

   begin (* gradevec *)
      list := nil; first := true;
      with aux do
      begin min := giant; max := 0 end;
      for i := 1 to n do a[i,i] := a[i,i] - lambda;
      n1 := svdsort (a, u1, v1, n, sigma);
      if n1 > 0 then
      begin
         num := col - 1; ni := n1;
         supervisor (p,r);
         pt^.next := nil
      end;
      for i := 1 to n do a[i,i] := a[i,i] + lambda
   end (* gradevec *);
```

5. NUMERICAL RESULTS

In this section we shall list some numerical results of the procedures discussed in section 4.

Though the function schur calculates both real and complex eigenvalues of a real general matrix, we have merely tested matrices with real eigenvalues, especially matrices with real multiple eigenvalues.

One of the most difficult practical problems is to recognize clusters of eigenvalues. For this reason, we shall list the minimum number of correct digits of the computed eigenvalues of defective and/or derogatory test matrices. These values could be an indication for the cluster tolerance.

Furthermore, we have compared the number of QR-single and QR-double iterations performed to calculate all eigenvalues with the number of iterations performed if only QR-double iterations are used, to see if it is worthwhile to include single QR steps.

Finally, some notes on the results of the procedure gradevec are given, especially on the interaction between the determination of the clusters and the number of grade vectors associated with the mean eigenvalue of such a cluster.

5.1. The test matrices

Unfortunately in the literature, only a few examples are known of matrices with multiple eigenvalues. In GREGORY and KARNEY [3], some methods are discussed to generate test matrices. We have created a collection of test matrices in the following way:

Let J be an n-th order matrix in J.c.f.. Premultiplying J by a matrix X and postmultiplying it by the inverse of X, we obtain a matrix A of order n, such that

$$A = XJX^{-1}.$$

Hence, in particular J is the J.c.f. of A.

For X we have chosen the matrix defined by $X = [x_{ij}]$, where

$$x_{ij} = x_{ji} = n+1-i, \qquad \text{if } i \geq j.$$

The inverse of X is defined by $X^{-1} = [x_{ij}]$, where

$$x_{11} = 1, \quad x_{ii} = 2, \qquad (i = 2,\dots,n)$$

$$x_{ii-1} = x_{i-1i} = 1, \qquad (i = 2,\dots,n)$$

$$x_{ij} = 0, \qquad \text{otherwise}$$

(see example 3.12 of GREGORY and KARNEY [3]).

In our tests we selected matrices from the following five classes:

I matrices of order k with only one multiple eigenvalue λ_1 and the non-linear divisor $(\lambda-\lambda_1)^k$

II matrices of order 10 with only one multiple eigenvalue λ_1, one non-linear divisor $(\lambda-\lambda_1)^k$ and (10-k) linear divisors $(\lambda-\lambda_1)$.

III matrices of order 10 with two multiple eigenvalues λ_1 and λ_2, and, two corresponding nonlinear divisors $(\lambda-\lambda_1)^k$ and $(\lambda-\lambda_2)^{10-k}$

IV other matrices of order 10 with one or more multiple eigenvalues and one or more linear and nonlinear divisors

V some test matrices of GREGORY and KARNEY [3].

5.2. Sorting the eigenvalues

The function schur calculates the eigenvalues of a real matrix, which may be both complex or real. However, these eigenvalues are not sorted. In RUHE [8], a method is described to sort the eigenvalues. The following process is suggested (for complex eigenvalues):

$$\lambda'_n = \lambda_n, \tag{5.2.1}$$

$$|\lambda'_{k-1} - \lambda_k| = \min_{j<k} |\lambda'_j - \lambda_k| \qquad (k = n,\ldots,2).$$

Each time, when $\lambda'_k \neq \lambda_k$ the eigenvalues are interchanged. After the sorting possibly close eigenvalues will appear together, since each time it is sure that the next eigenvalue chosen is the one closest to the already sorted eigenvalues. We use another process, however.

Because of the restriction of procedure gradevec, which can only be used in case of a real eigenvalue, we are only interested in clusters of real eigenvalues. However, if one calculates the real eigenvalues of a matrix by means of the function schur, it may happen, that one obtains a pair of complex conjugate eigenvalues with a small imaginary part. So, one has to convert such a complex pair into a double real one.

A method, that can be used, is to take the norm of the eigenvalue. Another method is to neglect the small imaginary part. We prefer the latter, since from numerical results, it appears, that the computed trace of matrix of order n with only one multiple eigenvalue λ is exactly equal to n times λ. For this reason, we shall first convert the complex eigenvalue into a real one by neglecting the imaginary parts, and subsequently sort the eigenvalues, starting with the smallest one.

In view of this the function schur does not contain a sorting process.

5.3. The perturbation of eigenvalues

After the eigenvalues are calculated and sorted, we wish to recognize clusters. The exact eigenvalues of our test matrices are known, so, it is not difficult to recognize which ones belong together. However, in practice, the eigenvalues are unknown, so, one needs a satisfactory criterion to decide, which eigenvalues form a cluster.

In WILKINSON [10, pp.72-81] a perturbation theory based on Gershgorin's theorem is discussed. Wilkinson distinguishes five main cases. The first two cases relate to non-defective matrices. The third case concerns the perturbation of a simple eigenvalue of a defective matrix. Wilkinson proves, that the presence of a non-linear divisor makes no essential difference to the behaviour of a simple eigenvalue.

In relation with the last two main cases, dealing with the perturbation of multiple eigenvalues of defective matrices, Wilkinson derives the following theorem:

THEOREM 5.3. *Let the elementary divisors corresponding to* λ_1 *be*

$$(\lambda_1-\lambda)^{r_1}, (\lambda_1-\lambda)^{r_2}, \ldots, (\lambda_1-\lambda)^{r_s}$$

where

$$r_1 \geq r_2 \geq \ldots \geq r_s \quad \text{and} \quad \sum_s r_i = t.$$

If the matrix elements are perturbed by values in magnitude $\leq \varepsilon$, *then for sufficiently small* ε *their lies at least one* λ *in a disc centre* λ_1 *and radius* $K_1\ \varepsilon^{s/t}$ *for some value of* K_1. *On the other hand all the corresponding perturbed eigenvalues lie in a disc centre* λ_1 *of radius* $K_2\ \varepsilon^{1/r}$ *for some* K_2.

As a consequence of this theorem we may say, that the presence of a nonlinear divisor $(\lambda_1-\lambda)^r$ effects the behaviour of an eigenvalue corresponding with a linear divisor $(\lambda_1-\lambda)$, in contrary with a simple eigenvalue. Moreover, we may conclude, that the maximum perturbation of an eigenvalue of a matrix of class I of order k, is of the same order, i.e. $\varepsilon^{1/k}$, as the maximum perturbation of an eigenvalue of a matrix of class II with a nonlinear divisor of order k.

In table 5.3.1 we list the computed maximum perturbation of the eigenvalues of class I, II and III (calculated by means of the function schur, where aux.tol = 1E-10). In the first column the order of the nonlinear divisor is given. The perturbation is represented by the minimum number of significant digits; i.e. max $- {}^{10}\log|\lambda_1 - \lambda_i'|$, where the λ_i''s are the computed eigenvalues. For λ_1 the value 2 is chosen. In case of a matrix of class III $\lambda_1 = 2$ and $\lambda_2 = 3$.

k \ class	I	II	III $\lambda_1 = 2$	III $\lambda_2 = 3$
10	1.99	1.99	1.99	——
9	2.29	2.01	2.12	exact
8	2.39	2.32	2.34	exact
7	3.54	3.49	2.83	4.69
6	4.68	3.60	3.68	3.53
5	6.38	5.08	4.93	2.65
4	6.47	6.07	10.13	2.49
3	exact	5.69	exact	2.00
2	exact	.	5.77	1.87
1			exact	2.13

Table 5.3.1

From table 5.3.1, it appears, that in case of a nonlinear divisor of order 10, at least a tolerance of order 1E-1 is required to recognize only one cluster with an eigenvalue of multiplicity 10.

Some of the results have been compared with the results from procedure peigrf, i.e. a PASCAL-FORTRAN interface procedure from NUMPAS [6], which calls the subroutine eigrf of the IMSL-library [4]. The perturbations of the eigenvalues calculated by means of the function schur appear to be smaller, but not in the order of a whole significant digit.

GOLUB and WILKINSON [2] suggested, that the computed eigenvalues are probably not the best values to use. If, for example, a matrix A has a well-defined J.c.f. and there is just one block $J_r(\lambda_1)$ associated with λ_1, one will expect the computed λ_i' to include a set of r values which, though not particularly close to λ_1 will be such that their sum is very close to $r\lambda_1$.

It appears that (though the results seem to be rather bad) in all tested cases, the mean value of a well-chosen cluster, is exactly equal to the exact eigenvalue.

5.4. The number of QR-iterations performed

During the test phase of the function schur, it was possible to calculate the eigenvalues using only QR-double iterations or using a combination of QR-single and QR-double iterations (as described in section 4.3). Though a QR-double iteration is mathematically equal to two QR-single iterations, the number of operations is not twice as much; one QR-single iteration takes $4n^2$ operations and a QR-double iteration $5n^2$. For this reason, the number of operations has been taken into account at the decision, which method is preferable.

The results were rather surprising. In some cases the total number of iterations of the combination method is considerably smaller than the number of iterations, when only QR-double iterations are used. However, the reverse also happens in a few cases.

In spite of this surprising behaviour, the combination method seems to be preferable. The average number of iterations was about four per eigenvalue. It is worth mentioning, that both methods give rise to about the same accuracy of the eigenvalues.

5.5. Notes on the procedure gradevec

As is mentioned previously, the procedure gradevec calculates the grade vectors corresponding with one particular eigenvalue. Moreover, by means of this procedure, one can determine the J.c.f. of a matrix.

Since we know the J.c.f. of all matrices used (see section 5.1) we could easily verify the results.

The process to calculate the grade vectors terminates when no more singular values are found smaller than an absolute tolerance. The problem is to choose a suitable value for this tolerance.

First of all, we have examined all singular values that have been calculated during the process. In case of an exact eigenvalue, it is not difficult to decide, which singular values may be regarded as zero. By means of the values of aux.max and aux.min, we observed, that no singular values were found in the interval [1E-12,1E-4].

If the eigenvalue is not very close to the exact eigenvalue, the distinction between singular values that may be regarded as zero, and those, that may not, is not obvious at all.

The number of vectors found strongly depends on the chosen tolerance. One way to decide how many vectors belong to one particular eigenvalue is to compare the order of the cluster associated with that eigenvalue. However, it may well be possible that the cluster size is wrong. For this reason, the process does not terminate when enough vectors, corresponding with the cluster format, are found. If the quotient aux.min/aux.max is sufficiently large, there is a good reason to assume that the number of vectors found corresponds with the J.c.f; moreover, that particular eigenvalue is then very close to an exact eigenvalue.

5.6. Numerical results

In this section we list results of two numerical examples. The first one has been used earlier in the literature, the second one is a matrix of class IV formed accordingly to the description of section 5.1.

Example I.
This matrix is discussed in GREGORY and KARNEY [3]. It has a well-defined Jordan form with a 5-fold eigenvalue 2.0 with $n_1 = 2$, $n_2 = 2$ and $n_3 = 1$ (for the meaning of n_i see section 3.1) and a 4-fold eigenvalue 3.0 with $n_1 = 2$ and $n_2 = 2$ and finally a single eigenvalue 1.0. The eigenvalues are calculated by means of the function schur with aux.tol = aux.correction = 1E-10. After 7 QR-double and 16 QR-single iteration steps, we obtained all eigenvalues. The values listed here are already sorted and the small imaginary parts (maximum = 3.85E-7) were neglected. The second column contains the eigenvlaues as listed in RUHE [8].

schur	Ruhe
3.000 000 817 784 9 E+0	1.000 000 0 + 0.000 000 0i
3.000 000 000 000 0 E+0	2.999 844 9 + 0.000 000 0i
3.000 000 000 000 0 E+0	3.000 155 1 + 0.000 000 0i
2.999 999 182 214 6 E+0	3.000 155 1 + 0.000 000 0i
2.000 000 208 962 2 E+0	2.999 844 9 + 0.000 000 0i
2.000 000 000 000 0 E+0	1.999 009 1 - 0.000 082 6i
2.000 000 000 000 0 E+0	2.000 330 0 + 0.000 764 4i
1.999 999 999 999 9 E+0	2.000 660 9 - 0.000 681 9i
1.999 999 791 037 9 E+0	2.000 000 0 + 0.000 043 9i
1.000 000 000 000 2 E+0	2.000 000 0 - 0.000 043 9i

The eigenvalues of the second column are calculated by means of a QR-algorithm with complex arithmetic and sorted according to (5.2.1).

With a cluster tolerance of 1E-5, we recognize 3 clusters. In table 5.6.1 the results from gradevec (aux.tol = 1E-10) are listed. In the first column the mean value of the clusters (schur) is given.

eigenvalue	n_1	n_2	n_3	aux.min	aux.max	$\frac{\text{aux.min}}{\text{aux.max}}$
1.000 000 000 000 2	1	0	0	3.604E-2	2.525E-14	1.427E+12
2.000 000 000 000 0	2	2	1	9.637E-2	4.429E-14	2.175E+12
2.999 999 999 999 9	2	2	0	1.592E-2	1.206E-14	1.320E+12

Table 5.6.1

Example II.

We have formed this matrix of class V accordingly to the description of section 5.1. It has a 10-fold eigenvalue 2.0 and elementary divisors $(2-\lambda)^7$, $(2-\lambda)^2$ and $(2-\lambda)$; this implies $n_1 = 3$, $n_2 = 2$, $n_3 = n_4 = n_5 = n_6 = n_7 = 1$.

Again the eigenvalues were calculated by means of the function schur (aux.tol = aux.correction = 1E-10):

1. 2.001 452 022 470 0 E+0
2. 2.001 452 022 470 0 E+0
3. 2.000 000 842 937 1 E+0
4. 2.000 000 000 000 3 E+0
5. 1.999 999 999 999 8 E+0
6. 1.999 999 999 999 8 E+0
7. 1.999 999 157 063 0 E+0
8. 1.999 445 010 646 6 E+0
9. 1.999 445 010 646 6 E+0
10. 1.998 205 933 766 3 E+0

The maximum neglected imaginary part is 1.71E-3. According to table 5.3.1 we need at least a cluster tolerance of 1E-2 to recognize only one cluster with an eigenvalue of multiplicity 10. However, in this case we choose the cluster tolerance to be 1E-5. So, we obtain 4 clusters. It is clear, that the mean value of each cluster is now not exactly equal to the eigenvalue 2.0. In table 5.6.2 the results of procedure gradevec with aux. tol = 1E-10 are listed.

cluster	order	λ_1	n_1	n_2	n_3	n_4	n_5	n_6	n_7	aux.min	aux.max	$\frac{\text{aux.min}}{\text{aux.max}}$
I	1	1.9982059337663	1	1	1	1	0	0	0	4.213E-9	7.544E-12	5.585E+2
II	2	1.9994450106466	1	1	1	0	0	0	0	2.410E-10	9.367E-14	2.573E+3
III	5	2.0000000000000	3	2	1	1	1	1	1	2.072E-2	3.810E-14	5.438E+11
IV	2	2.0014520224700	1	1	1	1	0	0	0	2.180E-9	3.291E-12	6.624E+2

Table 5.6.2

It can be easily verified, that with another value of aux.tol we obtain more or less vectors. For example, if we take aux.tol equal to 1E-8, we obtain at least one extra grade vector in case of the clusters I, II, and IV, while cluster III seems to be less sensitive for changes of aux.tol.

Suppose, the number of vectors is restricted by order of the cluster, then we obtain 6 vectors of grade 1 and 4 of grade 2. Moreover, these vectors are approximately linearly dependent.

From both examples, it appears, that if the quotient aux.min/aux.max is sufficiently large (in order of magnitude of about 1E+10), the procedure gradevec yields exactly the correct number of grade vectors accordingly to the J.c.f.. Moreover, if this quotient is not large enough, as in example II in case of clusters I, II and IV, the number of vectors found does not fit the cluster size. However, since the quotient aux.min/aux.max of cluster III is sufficiently large, we may conclude, that the mean value of this cluster is very close to an exact eigenvalue and by means of procedure gradevec, one obtains the correct number of grade vectors.

Instead of changing the parameter aux.tol of procedure gradevec, it appears to be better, to change the cluster size. In case of a cluster of order 10, the mean value of this cluster is exactly equal to the mean value of cluster III. The grade vectors obtained corresponding with this value are the desired vectors.

ACKNOWLEDGEMENTS

The author would like to thank Prof.Dr. T.J. Dekker and Drs. W. Hoffman for their critical and inspiring remarks.

REFERENCES

[1] DEKKER, T.J. & W. HOFFMAN [1969], *Algol 60 procedures in numerical algebra. Part 2,* MC tract 23, Mathematisch Centrum, Amsterdam.

[2] GOLUB, G.H. & J.H. WILKINSON [1976], *Ill conditioned eigensystems and the computation of the J.c.f.,* SIAM Review, Vol. 18, No.4.

[3] GREGORY, R.T. & D.L. KARNEY [1969], *A collection of matrices for testing computational algorithms,* Wiley-interscience.

[4] I.M.S.L., *International mathematical and statistical libraries,* Library 3, Reference Manual.

[5] JENSEN, K. & N. WIRTH [1974], *PASCAL User manual and Report,* Springer-Verlag.

[6] NUMPAS, *A library of numerical procedures in PASCAL,* Mathematical Centre, Amsterdam.

[7] RUHE, A. [1970], *Properties of a matrix with a very ill-conditioned eigenproblem,* Numer. Math. 15, pp.57-60.

[8] RUHE, A. [1970], *An algorithm for numerical determination of the structure of a general matrix,* BIT 10, pp.196-216.

[9] STEWART, G.W. [1973], *Introduction to matrix computations,* Academic Press.

[10] WILKINSON, J.H. [1965], *The algebraic eigenvalue problem,* Clarendon Press.

[11] WILKINSON, J.H. [1972], *Note on matrices with a very ill-conditioned eigenproblem,* Numer. Math. 19, 176-178.