A JACOBI–LIKE METHOD FOR SOLVING ALGEBRAIC RICCATI EQUATIONS ON PARALLEL COMPUTERS

ANGELIKA BUNSE–GERSTNER * AND HEIKE FAßBENDER †

Abstract. An algorithm to solve continuous–time algebraic Riccati equations through the Hamiltonian Schur form is developed. It is an adaption for Hamiltonian matrices of an unsymmetric Jacobi method of Eberlein [15]. It uses unitary symplectic similarity transformations and preserves the Hamiltonian structure of the matrix. Each iteration step needs only local information about the current matrix, thus admitting efficient parallel implementations on certain parallel architectures. Convergence performance of the algorithm is compared with the Hamiltonian–Jacobi algorithm of Byers [12]. The numerical experiments suggest that the method presented here converges considerably faster for non–Hermitian Hamiltonian matrices than Byers' Hamiltonian–Jacobi algorithm. Besides that, numerical experiments suggest that for the method presented here the number of iterations needed for convergence can be predicted by a simple function of the matrix size.

Key words. Jacobi–like method, Hamiltonian matrix, eigenvalues, algebraic Riccati equations, parallel computation

AMS subject classifications. 65F15, 65Y05, 49N05, 93C05

1. Introduction.

The problem of solving the continuous–time algebraic Riccati equation

$$-XGX + XA + A^H X + Q = 0 \tag{1}$$

where $X, G, Q, A \in \mathbb{C}^{n\times n}$, $Q = Q^H$ and $G = G^H$, arises for instance in linear–quadratic optimal control problems. An example is a linear autonomous system whose state $x(t)$ is given by

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t)\\ x(0) &= x_0\end{aligned}$$

with $A \in \mathbb{C}^{n\times n}$ and $B \in \mathbb{C}^{n\times m}$. A typical problem in linear control theory is to compute an optimal feedback control

$$u(t) = Fx(t)\ ,\ \ F \in \mathbb{C}^{m\times n}$$

which minimizes the functional

$$J(u) = \int_0^\infty \left[x^H(t)Qx(t) + u^H(t)Ru(t)\right] dt$$

where

$$Q \in \mathbb{C}^{n\times n}, Q = Q^H \geq 0 \text{ and } R \in \mathbb{C}^{m\times m}, R = R^H > 0.$$

Let C be the full rank factorization of Q and assume that (A, B) is stabilizable and (C, A) is detectable. Under these assumptions, the algebraic Riccati equation

$$-XBR^{-1}B^H X + XA + A^H X + Q = 0 \tag{2}$$

* Corresponding author, Universität Bremen, Fachbereich 3 - Mathematik und Informatik, Postfach 330 440, 28334 Bremen, FRG, phone : (+)49-412-218-4205, fax : (+)49-412-218-4863, e-mail : angelika@mathematik.uni-bremen.de

† Universität Bremen, Fachbereich 3 - Mathematik und Informatik, Postfach 330 440, 28334 Bremen, FRG, phone : (+)49-412-218-4217, fax : (+)49-412-218-4863, e-mail : heike@mathematik.uni-bremen.de

has a unique Hermitian and positive semidefinite solution X and

$$u(t) := -R^{-1}B^H X x(t)$$

is the desired optimal control (see, e.g., [26, 32]). With $G = BR^{-1}B^H$, one obtains equation (1) from (2).

Suppose that $Y \in \mathbb{C}^{n\times n}$ is nonsingular and $Z \in \mathbb{C}^{n\times n}$ is such that the columns of $\begin{bmatrix} Y \\ Z \end{bmatrix}$ span an n-dimensional invariant subspace corresponding to the stable eigenvalues of the Hamiltonian matrix

$$H = \begin{bmatrix} A & G \\ Q & -A^H \end{bmatrix}, \tag{3}$$

i.e., corresponding to the eigenvalues of H with negative real part. Then $X = -ZY^{-1}$ is the stabilizing solution of (1), that is, X is the unique solution for which the eigenvalues of $A - GX$ are in the open left half plane (see, e.g., [28]).

Matrices of the form (3) are called *Hamiltonian.* They are characterized by $(JH)^H = JH$, where J is the $2n \times 2n$ matrix

$$J = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix}$$

and I is the $n \times n$ identity. The eigenvalues of a Hamiltonian matrix occur in pairs $\lambda, -\overline{\lambda}$. Since (A, B) is stabilizable and (C, A) is detectable, the matrix H has no purely imaginary eigenvalues. It is well known that a Hamiltonian matrix is invariant under symplectic similarity transformations. A matrix $S \in \mathbb{C}^{2n\times 2n}$ is *symplectic* if $S^H J S = J$.

Various algorithms have been proposed to solve equation (1), see, e.g., [1, 4, 5, 7, 8, 9, 10, 11, 18, 19, 29]. Unfortunately, these algorithms either apply only to a highly restricted class of Hamiltonian matrices or destroy the Hamiltonian structure or risk numerical instability by using non–unitary similarity transformations.

In 1981, Paige and Van Loan [33] proved that, if the eigenvalues of a Hamiltonian matrix H have nonzero real parts, then H has a Schur–like decomposition, the *Hamiltonian Schur decomposition*, i.e., there exists a unitary symplectic matrix

$$Q = \begin{bmatrix} Q_1 & Q_2 \\ -Q_2 & Q_1 \end{bmatrix}, \; Q_1, Q_2 \in \mathbb{C}^{n\times n}$$

such that

$$Q^H H Q = \begin{bmatrix} T & N \\ 0 & -T^H \end{bmatrix} \tag{4}$$

where $N = N^H \in \mathbb{C}^{n\times n}$, $T \in \mathbb{C}^{n\times n}$ is upper triangular, and the eigenvalues of T are the stable eigenvalues of H, that is

$$\lambda(T) = \{\beta \mid \beta \in \lambda(H), Re(\beta) < 0\}.$$

From (4) we obtain

$$H \begin{bmatrix} Q_1 \\ -Q_2 \end{bmatrix} = \begin{bmatrix} Q_1 \\ -Q_2 \end{bmatrix} T,$$

hence the desired nonnegative Hermitian solution of the Riccati equation (1) is given by $X = Q_2Q_1^{-1}$.

In 1990, Byers [12] proposed a symplectic Jacobi–like algorithm for the computation of the Hamiltonian Schur decomposition of a Hamiltonian matrix, the Hamiltonian-Jacobi algorithm. The algorithm maintains numerical stability by using a sequence of unitary symplectic similarity transformations. It requires $\mathcal{O}(n)$ computational time for a sweep when implemented on a mesh–connected $n \times n$ array processor system. However, the convergence can be very slow if the Hamiltonian matrix is not near to normality; and very often the method does not converge at all for problems of dimension greater than 20. Lin and You proposed in [30] a cubic acceleration method to improve a symplectic Jacobi–like algorithm for computing the Hamiltonian Schur decomposition of a Hamiltonian matrix and finding the positive semidefinite solution of the corresponding algebraic Riccati equation.

Byers' Hamiltonian–Jacobi algorithm is an adaption of a Jacobi–like method for non–Hermitian matrices proposed by Stewart [39]. This Jacobi–like algorithm computes the Schur decomposition of a general matrix. A different Jacobi–like method to solve this problem was proposed by Eberlein [15]. Both methods transform an arbitrary matrix successively to Schur form by similarity transformations with elementary Givens rotations. In each step an element of the current iterate, the *pivot element*, is annihilated. Zeros created in previous steps are not preserved. Stewart proposed to only use the elements of the lower subdiagonals as pivot elements, while Eberlein proposed to use all elements in the lower triangle. Further both methods differ in the choice of the rotation. Usually, in each step there are two possible rotations. Stewart always uses the one which corresponds to the largest rotation angle, while Eberlein chooses the smaller angle. Both methods have been adapted to compute the generalized Schur decomposition of an arbitrary complex matrix pencil. Charlier and van Dooren [13] adapted Stewart's method and were able to prove global convergence under a restriction on the rotation angles. In [6] Eberlein's method has been adapted to compute the generalized Schur decomposition of an arbitrary complex matrix pencil and the different methods have been compared. Although, so far no convergence proof for Eberlein's method has been found, it appears that the method of Eberlein converges much faster than the method of Stewart if the matrix under consideration is far from normal. Besides that, it seems that for the method of Eberlein, the number of iterations needed for convergence can be predicted by a simple function of the matrix size. For the method of Stewart no such observation was made. Such a relation is important for an implementation on a parallel computer, as one wants to perform a fixed amount of rotations to achieve convergence instead of determining the stopping criterium, as for the classical Jacobi method on serial machines, via the values of the elements in the lower left triangle of the matrix. This requires global knowledge of all matrix elements and is too expensive on most parallel architectures.

In this paper a Jacobi–like method for computing the Hamiltonian Schur form of a Hamiltonian matrix is developed analogous to the Jacobi–like method of Eberlein [15] for the computation of the Schur form of a general matrix (with which the reader is assumed to already be familiar). When used for matrices which are Hermitian and Hamiltonian, the algorithm is equivalent to the Kogbetliantz algorithm [27] for n–dimensional matrices; the Kogbetliantz algorithm is globally convergent (under mild assumptions) and converges asymptotically quadratic.

Section 2 describes the basic idea of the method and the connection to Kogbetliantz's algorithm. In each step of the algorithm a 4×4 Hamiltonian submatrix of the Hamiltonian matrix under consideration is transformed to (almost) Hamiltonian Schur form. In Section 3, we discuss how these 4×4 subproblems can be solved. Essentially, there are at most two different unitary symplectic transformations which transform a 4×4 Hamiltonian matrix into Hamiltonian Schur form. It is discussed which of these two transformations should be chosen. In Section 4 we sketch the possibilities of parallel implementations on a ring processor system and on a mesh–connected array of processors. Finally, in Section 5 we present numerical test results comparing our method with the method of Byers [12]. These examples demonstrate that our method converges much faster for problems which are not Hermitian. They also suggest that for the method proposed here the number of iterations needed for convergence can be predicted by a simple function of the matrix size. For the method of Byers no such observation was made. As pointed out before, such a relation is important for an implementation on a parallel computer.

There are still many issues to be considered before we can decide whether or not such a Jacobi–like method for the Hamiltonian eigenvalue problem is of practical value for use on parallel machines. For instance, the method as it stands now employs complex arithmetic, even if we start with a real Hamiltonian matrix. An important question is whether the method can be modified to allow only the use of real arithmetic. Another important question relates to the rate of convergence. In all numerical examples, the convergence seems to be between linear and quadratic. However, it has not been possible to prove this. The purpose of this paper is to provide insight and report experience about the performance of these Jacobi–like methods for the solution of Hamiltonian eigenvalue problems. Such knowledge is important because of their apparent suitability for certain parallel architectures.

2. Basic Considerations. An elementary step of the Jacobi method [22] for an $n \times n$ Hermitian matrix A consists in choosing a pair of indices (k,l), *the pivot pair*, and transforming the matrix A such that the (k,l) and the (l,k) entries are annihilated by a similarity transformation with a rotation U in the k,l–plane. U differs from the identity only in the four matrix elements $u_{kk} = u_{ll} = \cos\alpha$, $u_{kl} = -e^{-i\theta}\sin\alpha$, and $u_{lk} = e^{i\theta}\sin\alpha$. These four essential entries of U form a 2×2 unitary matrix, which diagonalizes the 2×2 submatrix of A consisting of the corresponding entries in the k,l–plane. Of the two angles α in $(-\pi/2, \pi/2]$, which solve this 2×2 diagonalization problem, the one smaller in modulus is chosen. Various Jacobi methods differ in the way the pivot indices are chosen. The parallel versions are always based on a *cyclic Jacobi method*, i.e., Jacobi methods in which all the indices below or above the diagonal are cyclically traversed in a special order. Such a traverse in which all possible index pairs are traversed exactly once is called a *sweep*. Convergence to a diagonal matrix can be proved under certain conditions and can be shown to be quadratic [21, 23, 24, 31, 37, 38, 41].

Similar to the Jacobi method [22], an elementary step of the Jacobi–like method of Eberlein [15] for a general $n \times n$ matrix A consists in choosing a pair of indices (k,l) and transforming the matrix A such that the (k,l) entry is annihilated by a similarity transformation with a rotation U in the k,l–plane. As before U differs from the identity only in the four matrix elements u_{kk}, u_{ll}, u_{kl} and u_{lk}. These four essential entries form a 2×2 unitary matrix which transforms the corresponding 2×2 submatrix of A to Schur form. Of the two possible transformations which solve this 2×2 problem the one closest to the identity is chosen. In each sweep all indices below the diagonal are traversed once. Convergence to Schur form has been observed, but has not yet been proven.

Thus both methods solve at each step a small subproblem of the same structure as the $n \times n$ problem, and annihilate all possible matrix elements below the diagonal once per sweep. Our idea here is to adapt this strategy for computing the Hamiltonian Schur form of a Hamiltonian matrix. Hence we propose that an elementary step of a Jacobi–like algorithm for computing the Hamiltonian Schur form consists in choosing a pair of pivot indices and transforming the corresponding Hamiltonian submatrix to Hamiltonian Schur form. For a structure–preserving Jacobi–like method for the Hamiltonian eigenproblem we have to restrict the choice of orthogonal/unitary transformations to those which are at the same time symplectic, as only those guarantee to preserve the Hamiltonian structure. It can be shown [33] that all $2n \times 2n$ unitary and symplectic matrices are of the form

$$\begin{bmatrix} U & V \\ -V & U \end{bmatrix}, \qquad U, V \in \mathbb{C}^{n\times n}, U^H U + V^H V = I.$$

Using Paige and Van Loan's result [33] we can use such unitary and symplectic matrices to transform the Hamiltonian submatrices to (almost) Hamiltonian Schur form. The smallest Hamiltonian subproblem of a Hamiltonian matrix H (3) is

$$\left[\begin{array}{c|c} a_{jj} & g_{jj} \\ \hline q_{jj} & -\overline{a}_{jj} \end{array}\right].$$

The transformation to Hamiltonian Schur form annihilates the element q_{jj}. Hence, only the diagonal elements of Q can be annihilated if 2×2 Hamiltonian subproblems would be used in a Jacobi–like method for computing the Hamiltonian Schur form of a Hamiltonian matrix. Convergence to Hamiltonian Schur form for such a procedure is very unlikely. Now consider a 4×4 Hamiltonian submatrix H_{ij} of H

$$H_{ij} = \left[\begin{array}{cc|cc} a_{ii} & a_{ij} & g_{ii} & \overline{g_{ji}} \\ a_{ji} & a_{jj} & g_{ji} & g_{jj} \\ \hline q_{ii} & \overline{q_{ji}} & -\overline{a}_{ii} & -\overline{a}_{ji} \\ q_{ji} & q_{jj} & -\overline{a}_{ij} & -\overline{a}_{jj} \end{array}\right] \qquad i, j \in \{1, \ldots, n\}\,,\ i \neq j$$

and assume for the moment that all such submatrices of H can be transformed to Hamiltonian Schur form. Choosing each pair of indices $(k, l), 1 \leq k, l \leq n$ once per sweep, all matrix elements below the diagonal of A and all elements of Q can be annihilated, if in each step we transform the 4×4 Hamiltonian submatrix corresponding to the current pair of indices to Hamiltonian Schur form. Thus we propose the algorithm given in Table 1. The ordering for the pairs of indices given there is one possibility. Any other ordering, as long as each pair is used once per sweep, can be used. We will come back to this question in Sections 4 and 5.

Unfortunately, the 4×4 Hamiltonian subproblems can have purely imaginary eigenvalues such that some subproblems can not be transformed to Hamiltonian Schur form. All that can be achieved in this case is:

```
Algorithm: Jacobi-like Method for Computing the Hamiltonian Schur form
repeat
    for $i = 1, \dots, n-1$
        for $j = i+1, \dots, n$
            compute $U_{ij}$ such that $U_{ij}^H H_{ij} U_{ij}$ has Hamiltonian Schur form
            (the eigenvalues of the upper $2 \times 2$ block of $U_{ij}^H H_{ij} U_{ij}$ are in
            the open left half plane)
            set $\widehat{U} := I$
                $\widehat{u}_{ii} := (U_{ij})_{11}$;          $\widehat{u}_{ji} := (U_{ij})_{21}$;
                $\widehat{u}_{ij} := (U_{ij})_{12}$;          $\widehat{u}_{jj} := (U_{ij})_{22}$;
                $\widehat{u}_{i,n+i} := (U_{ij})_{13}$;       $\widehat{u}_{j,n+i} := (U_{ij})_{23}$;
                $\widehat{u}_{i,n+j} := (U_{ij})_{14}$;       $\widehat{u}_{j,n+j} := (U_{ij})_{24}$;
                $\widehat{u}_{n+i,i} := (U_{ij})_{31}$;       $\widehat{u}_{n+j,i} := (U_{ij})_{41}$;
                $\widehat{u}_{n+i,j} := (U_{ij})_{32}$;       $\widehat{u}_{n+j,j} := (U_{ij})_{42}$;
                $\widehat{u}_{n+i,n+i} := (U_{ij})_{33}$;     $\widehat{u}_{n+j,n+i} := (U_{ij})_{43}$;
                $\widehat{u}_{n+i,n+j} := (U_{ij})_{34}$;     $\widehat{u}_{n+j,n+j} := (U_{ij})_{44}$;
            set $H := \widehat{U}^H H \widehat{U}$
until stopping criterium satisfied
```

TABLE 1

PROPOSITION 2.1 ([33],COROLLARY 3.2). *If $H \in \mathbb{C}^{2n\times 2n}$ is a Hamiltonian matrix, then there exists a unitary symplectic $U \in \mathbb{C}^{2n\times 2n}$ such that*

$$(5) \qquad U^H H U = \begin{array}{c} \left[\begin{array}{cccc} T_{11} & T_{12} & R_{11} & R_{12} \\ 0 & T_{22} & R_{21} & R_{22} \\ 0 & 0 & -T_{11}^H & 0 \\ 0 & K_{22} & -T_{12}^H & -T_{22}^H \end{array}\right] \begin{array}{l} \}p \\ \}q \\ \}p \\ \}q \end{array} \\ \begin{array}{cccc} \underbrace{\quad}_{p} & \underbrace{\quad}_{q} & \underbrace{\quad}_{p} & \underbrace{\quad}_{q} \end{array} \end{array} \quad , \quad p + q = n,$$

where T_{11} is upper triangular and $\begin{bmatrix} T_{22} & R_{22} \\ K_{22} & -T_{22}^H \end{bmatrix}$ is a Hamiltonian matrix with purely imaginary eigenvalues.

Thus, in the case that the 4×4 Hamiltonian subproblem has purely imaginary eigenvalues, we will try to achieve a form as close as possible to the Hamiltonian Schur form. That is, we want to choose a transformation such that in the *almost Hamiltonian form* (5) the norm of K_{22} is as small as possible.

A different motivation for the proposed algorithm can be given via the Kogbetliantz algorithm [27]. This algorithm is a generalization of the Jacobi method for solving the following problem: Let $C \in \mathbb{C}^{n\times n}$. Determine unitary matrices Z, $W \in \mathbb{C}^{n\times n}$ such that $\sum := WCZ^H$ is a diagonal matrix (upto scaling this is the SVD of C). An elementary step of the Kogbetliantz method consists in choosing a pair of indices (k,l), and transforming the matrix C such that the (k,l) and (l,k) entries are annihilated by a transformation with rotations W and Z in the k,l–plane. W and Z differ from the identity only in the four matrix elements with indices (k,k), (k,l), (l,k) and (l,l). Convergence can be proved under certain conditions and can be shown to be quadratic [17, 34].

The following proposition gives the connection between the transformation of a Hermitian and Hamiltonian matrix to Hamiltonian Schur form and a singular value decomposition.

PROPOSITION 2.2. *Let*

$$M = \begin{bmatrix} A & B \\ B & -A \end{bmatrix}$$

be a $2n \times 2n$ Hermitian and Hamiltonian matrix and let

$$S = \begin{bmatrix} U & V \\ -V & U \end{bmatrix}$$

be a $2n \times 2n$ unitary and symplectic matrix. Then the similarity transformation $\widetilde{M} := SMS^{-1}$ is equivalent to the transformation $\widetilde{C} := WCZ^H$ with $C = A + iB$, $W = U - iV$ and $Z = U + iV$.

Thus, if S is the unitary symplectic matrix which transforms the Hermitian and Hamiltonian matrix M to Hamiltonian Schur form, then $\widetilde{M} = \mathrm{diag}(\tilde{a}_{11}, \ldots, \tilde{a}_{nn}, -\tilde{a}_{11}, \ldots, -\tilde{a}_{nn})$ with $\tilde{a}_{jj} \leq 0$ for $j = 1, \ldots, n$, as $\widetilde{M} = SMS^H$ has to be Hermitian again. From Proposition 2.2 follows by comparing the real and imaginary part of $\widetilde{C}$ with the (1,1)- and (2,1)-block in $\widetilde{M}$ $\widetilde{C} = \mathrm{diag}(\tilde{a}_{11}, \ldots, \tilde{a}_{nn})$. On the other hand, if there are unitary matrices W and Z such that $\widetilde{C} = W(A + iB)Z^H = \mathrm{diag}(\tilde{a}_{11}, \ldots, \tilde{a}_{nn})$, $\tilde{a}_{jj} \leq 0$ for $j = 1, \ldots, n$, then Proposition 2.2 gives that $\widetilde{M}$ is in Hamiltonian Schur form.

Interpreting Kogbetliantz's algorithm with the help of Proposition 2.2 in terms of transforming the Hermitian and Hamiltonian matrix M one obtains a special case of the Jacobi–like method for computing the Hamiltonian Schur form given above. For details see [25]. Hence, in the case of Hermitian and Hamiltonian matrices, the algorithm proposed here is equivalent to Kogbetliantz's algorithm and its convergence properties are the same, if we choose the pivot index sequence and the rotation angles according to the demands of the convergence proofs [17, 34].

The Hamiltonian–Jacobi–iteration proposed by Byers [12] performs a sequence of unitary symplectic similarity transformations each of which decreases the norm of Q plus twice the norm of the lower triangle of A. This quantity will be denoted by

$$\sigma(H) = \sigma(\begin{bmatrix} A & G \\ Q & -A^H \end{bmatrix}) := \sum_{j=1}^{n} \left(\sum_{i=1}^{n} |q_{ij}|^2 + 2 \sum_{i=j+1}^{n} |a_{ij}|^2 \right). \tag{6}$$

Usually, $\sigma(H)$ converges to zero and H tends to Hamiltonian Schur form, but, as experiments show, often at an unacceptable slow rate. As will be discussed in the next section, our algorithm does not decrease $\sigma(H)$ in each step, but the numerical experiments show that only during the first few steps $\sigma(H)$ may increase slightly. Eventually $\sigma(H)$ decreases monotonically.

3. Solving 4×4 Hamiltonian Subproblems.

In each iteration step of our Jacobi–like method for computing the Hamiltonian Schur form, we have to compute the Hamiltonian Schur form of a 4×4 Hamiltonian matrix, or, if that is not possible, a form as close as possible to the Hamiltonian Schur form as in Proposition 2.1. Let

$$H = \begin{bmatrix} A & G \\ Q & -A^H \end{bmatrix} \in \mathbb{C}^{4 \times 4} \tag{7}$$

be a 4×4 Hamiltonian matrix with

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, Q = \begin{bmatrix} q_{11} & \overline{q_{21}} \\ q_{21} & q_{22} \end{bmatrix}, G = \begin{bmatrix} g_{11} & \overline{g_{21}} \\ g_{21} & g_{22} \end{bmatrix}, q_{11}, q_{22}, g_{11}, g_{22} \in \mathbb{R}$$

(note that $Q = Q^H$ and $G = G^H$). Depending on the eigenvalues of H a unitary symplectic transformation S can be computed such that $S^H H S = \widetilde{H}$ has

a) the desired Hamiltonian Schur form

$$\begin{bmatrix} \widetilde{a}_{11} & \widetilde{a}_{12} & \widetilde{g}_{11} & \overline{\widetilde{g}_{21}} \\ 0 & \widetilde{a}_{22} & \widetilde{g}_{21} & \widetilde{g}_{22} \\ 0 & 0 & -\overline{\widetilde{a}_{11}} & 0 \\ 0 & 0 & -\overline{\widetilde{a}_{12}} & -\overline{\widetilde{a}_{22}} \end{bmatrix} \tag{8}$$

with $\text{Re}(\widetilde{a}_{11}), \text{Re}(\widetilde{a}_{22}) < 0$ or

b) almost Hamiltonian Schur form

$$\begin{bmatrix} \widetilde{a}_{11} & \widetilde{a}_{12} & \widetilde{g}_{11} & \overline{\widetilde{g}_{21}} \\ 0 & \widetilde{a}_{22} & \widetilde{g}_{21} & \widetilde{g}_{22} \\ 0 & 0 & -\overline{\widetilde{a}_{11}} & 0 \\ 0 & \widetilde{q}_{22} & -\overline{\widetilde{a}_{12}} & -\overline{\widetilde{a}_{22}} \end{bmatrix} \tag{9}$$

with $\text{Re}(\widetilde{a}_{11}) < 0$ and $\begin{bmatrix} \widetilde{a}_{22} & \widetilde{g}_{22} \\ \widetilde{q}_{22} & -\overline{\widetilde{a}_{22}} \end{bmatrix}$ has purely imaginary eigenvalues or

c) "nothing" in terms of the Hamiltonian Schur form if H has only purely imaginary eigenvalues.

In case b) we want to choose a transformation such that in the almost Hamiltonian Schur form (9) the element $\widetilde{q}_{22}$ is as small as possible. In the case c) we could choose a transformation S to minimize the norm of the elements annihilated in a Hamiltonian Schur form, that is to choose S such that in $\widetilde{H} = S^H H S$ the quantity

$$\sqrt{2|\widetilde{a}_{21}|^2 + |\widetilde{q}_{11}|^2 + |\widetilde{q}_{21}|^2 + |\widetilde{q}_{12}|^2 + |\widetilde{q}_{22}|^2}$$

is minimized. As this minimization is very complex, we propose to choose $S = I$ instead. A different possibility is to choose S to introduce at least some zeros in $\widetilde{H} = S^H H S$, e.g., such that $\widetilde{q}_{21} = \widetilde{q}_{12} = 0$. Our numerical tests showed that using $S = I$ is a reasonable choice as long as there are not too many 4×4 subproblems with only purely imaginary eigenvalues. Otherwise convergence will be slow (or, if all subproblems have only purely imaginary eigenvalues, there will be no convergence at all). It is not yet clear what is the best choice of S in that case.

In the following we will discuss how the transformation matrix S can be computed in case a) where H has only eigenvalues with nonzero real part or b) where H has one pair of eigenvalues with nonzero real part and one pair of purely imaginary eigenvalues.

In [33] Paige and Van Loan give a constructive proof of the existence of the (almost) Hamiltonian Schur form (8), resp. (9). In this construction, first an eigenvalue λ_1 of H with $\text{Re}(\lambda_1) < 0$ and the corresponding eigenvector $[y^T, z^T]^T \in \mathbb{C}^4, y, z \in \mathbb{C}^2$ is chosen, i.e.

$$H \begin{bmatrix} y \\ z \end{bmatrix} = \lambda_1 \begin{bmatrix} y \\ z \end{bmatrix}.$$

Next a unitary symplectic matrix S_1 is computed such that

$$S_1^H \begin{bmatrix} y \\ z \end{bmatrix} = \alpha e_1 \ , \ \alpha \neq 0$$

where $e_1 = [1,0,0,0]^T$. From the equation $(S_1^H H S_1)e_1 = \lambda_1 e_1$, it is seen that

$$S_1^H H S_1 = \begin{bmatrix} \lambda_1 & * & * & * \\ 0 & a & * & g \\ 0 & 0 & -\overline{\lambda}_1 & 0 \\ 0 & q & * & -\overline{a} \end{bmatrix}$$

with $a \in \mathbb{C}$ and $q, g \in \mathbb{R}$. The eigenvalues of H are λ_1 and $-\overline{\lambda}_1$ together with the eigenvalues of the Hamiltonian matrix H' defined by

$$H' = \begin{bmatrix} a & g \\ q & -\overline{a} \end{bmatrix}.$$

In the case that H' has eigenvalues with nonzero real part, we propose to compute a unitary symplectic transformation S' such that

$$(S')^H H' S' = \begin{bmatrix} \lambda_2 & * \\ 0 & -\overline{\lambda}_2 \end{bmatrix}$$

with $\mathrm{Re}(\lambda_2) < 0$. In the case that H' has purely imaginary eigenvalues, a unitary symplectic transformation S' is computed such that the $(2,1)$–element in $(S')^H H' S'$ is minimal. In either case, if we choose

$$S = S_1 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & s'_{11} & 0 & s'_{12} \\ 0 & 0 & 1 & 0 \\ 0 & -s'_{12} & 0 & s'_{11} \end{bmatrix}$$

where s'_{11}, s'_{12} are the $(1,1),(1,2)$–elements of S', then S is unitary symplectic and $S^H H S$ has the desired form (8) or (9) depending on the eigenvalues of H.

This algorithm requires a–priori knowledge of the eigenvalues and eigenvectors of the 4×4 Hamiltonian matrix H (7). In our MATLAB[1] implementation, they are computed using the routine 'eig'. For an algorithm to compute S_1 directly, see algorithm 3 in [33]. In the last step we have to compute S'. This is discussed in [12], [30].

The algorithm outlined above starts by choosing an eigenvalue λ_1 of H with $\mathrm{Re}(\lambda_1) < 0$. If H has two eigenvalues λ_1, λ_2 with $\mathrm{Re}(\lambda_1) < 0$ and $\mathrm{Re}(\lambda_2) < 0$, then either λ_1 or λ_2 can be chosen. Choosing λ_1 yields the Hamiltonian Schur form

$$\begin{bmatrix} \lambda_1 & a & * & * \\ 0 & \lambda_2 & * & * \\ 0 & 0 & -\overline{\lambda}_1 & 0 \\ 0 & 0 & -\overline{a} & -\overline{\lambda}_2 \end{bmatrix} = \widetilde{H}_1 = (S^1)^H H S^1,$$

[1] MATLAB is a registered trademark of The MathWorks,Inc.

choosing λ_2 yields

$$\begin{bmatrix} \lambda_2 & b & * & * \\ 0 & \lambda_1 & * & * \\ 0 & 0 & -\overline{\lambda}_2 & 0 \\ 0 & 0 & -\overline{b} & -\overline{\lambda}_1 \end{bmatrix} = \widetilde{H}_2 = (S^2)^H H S^2.$$

S^2 can be obtained from S^1 by multiplication with a unitary matrix

$$T = \begin{bmatrix} Z & 0 \\ 0 & Z \end{bmatrix} \tag{10}$$

where Z is chosen such that

$$Z^H \begin{bmatrix} \lambda_1 & a \\ 0 & \lambda_2 \end{bmatrix} Z = \begin{bmatrix} \lambda_2 & * \\ 0 & \lambda_1 \end{bmatrix}.$$

Which of the two possible transformations should we use in our algorithm? In order to answer this question, consider the effect of one step of our algorithm to the Hamiltonian matrix H: only rows and columns $k, l, n+k$ and $n+l$ are changed, if the pivot pair chosen is $(k,l), 1 \leq k < l \leq n$.

For the moment, let H be a 10×10 Hamiltonian matrix. Then for the pivot indices $(1,3)$ the new iterate changes its entries as follows:

$$\left[\begin{array}{ccccc|ccccc}
\mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} \\
\mathcal{E} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathrm{x} \\
\mathcal{E} & \mathcal{E} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{X} \\
\mathcal{E} & \mathrm{e} & \mathcal{E} & \mathrm{x} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathrm{x} \\
\mathcal{E} & \mathrm{e} & \mathcal{E} & \mathrm{e} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathrm{x} \\
\hline
\mathcal{E} & \mathcal{E} & \mathcal{E} & \mathcal{E} & \mathcal{E} & \mathcal{X} & \mathcal{E} & \mathcal{E} & \mathcal{E} & \mathcal{E} \\
\mathcal{E} & \mathrm{e} & \mathcal{E} & \mathrm{e} & \mathrm{e} & \mathcal{X} & \mathrm{x} & \mathcal{E} & \mathrm{e} & \mathrm{e} \\
\mathcal{E} & \mathcal{E} & \mathcal{E} & \mathcal{E} & \mathcal{E} & \mathcal{X} & \mathcal{X} & \mathcal{X} & \mathcal{E} & \mathcal{E} \\
\mathcal{E} & \mathrm{e} & \mathcal{E} & \mathrm{e} & \mathrm{e} & \mathcal{X} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathrm{e} \\
\mathcal{E} & \mathrm{e} & \mathcal{E} & \mathrm{e} & \mathrm{e} & \mathcal{X} & \mathrm{x} & \mathcal{X} & \mathrm{x} & \mathrm{x}
\end{array}\right]$$

Here e, $\mathcal{E}$ denote entries in the lower triangular part of A and entries in the matrix Q, thus e, $\mathcal{E}$ denotes entries which should be zero at the end of our iteration process. The caligraphic letters $\mathcal{E}, \mathcal{X}$ indicate entries which are altered in this step; they are constructed as a combination of entries from the lower <u>and</u> upper triangle of A and from Q <u>and</u> G.

If we have already achieved an iterate H which has only very small entries in the lower triangular part of A and in Q, then we would not want to increase them again by combining them with possibly big entries from the upper triangle of A and from G.

Denote the nontrivial 4×4 submatrix of the transformation matrix S by

$$\begin{bmatrix} U & V \\ -V & U \end{bmatrix} \quad \text{with } U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \quad \text{and } V = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix}.$$

Consider, e.g., the pivot index pair $(1,3)$. For the elements which change in this step and which should remain very small, one obtains

element(s)	computation
$(2,1)$	$\epsilon\overline{u}_{11}+x\overline{u}_{12}+x\overline{v}_{11}+x\overline{v}_{12}$
$(3,2)$	$xu_{21}+\epsilon u_{22}+\epsilon v_{21}+\epsilon v_{22}$
$(7,1)$	$\epsilon\overline{u}_{11}+\epsilon\overline{u}_{12}+x\overline{v}_{11}+\epsilon\overline{v}_{12}$
$(7,3)$	$\epsilon\overline{u}_{21}+\epsilon\overline{u}_{22}+x\overline{v}_{21}+\epsilon\overline{v}_{22}$
$(7,8)$	$-\epsilon\overline{v}_{21}-\epsilon\overline{v}_{22}+x\overline{u}_{21}+\epsilon\overline{u}_{22}$
$(4,3),(5,3),(9,3),(10,3)$	$\epsilon\overline{u}_{21}+\epsilon\overline{u}_{22}+x\overline{v}_{21}+x\overline{v}_{22}$
$(4,1),(5,1),(9,1),(10,1)$	$\epsilon\overline{u}_{11}+\epsilon\overline{u}_{12}+x\overline{v}_{11}+x\overline{v}_{12}$
$(6,2),(6,4),(6,5),(6,7),(6,9),(6,10)$	$-xv_{11}-xv_{12}+\epsilon u_{11}+\epsilon u_{12}$
$(8,2),(8,4),(8,5),(8,7),(8,9),(8,10)$	$-xv_{21}-\epsilon v_{22}+\epsilon u_{21}+\epsilon u_{22}$.

As usual ϵ denotes very small entries, x arbitrary ones. The newly computed elements should be small again, thus the coefficients of the x–elements should be small. These factors are u_{12}, u_{21} and v_{ij} for $i,j=1,2$. Therefore, one strategy in choosing the best transformation for our purpose is to choose the transformation S which has the least weight on the off–diagonal elements, that is a transformation S such that

$$|u_{12}|^2+|u_{21}|^2+||V||_F^2=min.$$

Such a transformation will be called *inner transformation.* This choice is a direct adaption of Eberlein's choice of rotation in her nonsymmetric Jacobi–like method [15]. In each step of that algorithm, a 2×2 matrix is transformed to Schur form by a 2×2 elementary rotation. In general, there are two possibilities to choose this rotation. Eberlein suggests to choose the rotation which is closest to identity.

From this discussion it is obvious that the algorithm will not necessarily decrease $\sigma(H)$ (6), the norm of Q plus twice the norm of the lower triangle of A.

From (10) we obtain, that if S^1 transforms H to Hamiltonian Schur form such that λ_1 appears in position $(1,1)$, then S^1T transforms H to Hamiltonian Schur form such that λ_2 appears in position $(1,1)$. Denote S^1 and T by

$$S^1=\begin{bmatrix} U^1 & V^1\\ -V^1 & U^1\end{bmatrix} \text{ and } T=\begin{bmatrix} Z & 0\\ 0 & Z\end{bmatrix},\ Z \text{ unitary.}$$

Let

$$S^1T=\begin{bmatrix} U^2 & V^2\\ -V^2 & U^2\end{bmatrix}=\begin{bmatrix} U^1Z & V^1Z\\ -V^1Z & U^1Z\end{bmatrix}.$$

We want to choose the transformation for which the norm of the off–diagonal elements is minimal. That is, we want to choose S^1 if

$$|u_{12}^1|^2+|u_{21}^1|^2+||V^1||_F^2\le|u_{12}^2|^2+|u_{21}^2|^2+||V^2||_F^2$$

and S^1T otherwise. As Z is an unitary matrix, $||V^1||_F^2=||V^1Z||_F^2=||V^2||_F^2$ and the above inequality reduces to

$$|u_{12}^1|^2+|u_{21}^1|^2\le|u_{12}^2|^2+|u_{21}^2|^2. \qquad (11)$$

Hence, in order to decide which transformation to use, there is no need to compute S^1T, it suffices to compute u_{12}^2 and u_{21}^2. The remaining elements of S^1T will be computed only if necessary.

As discussed in Section 2, for Hermitian and Hamiltonian matrices

$$\begin{bmatrix} A & B \\ B & -A \end{bmatrix}, \; A = A^H, B = B^H$$

our Jacobi–like method for computing the Hamiltonian Schur form is equivalent to Kogbetliantz's Jacobi–like algorithm for computing the SVD of $C = A + iB$.

Paige and Van Dooren [34] proved that the cyclic Kogbetliantz algorithm ultimately converges quadratically in the absence of close singular values. In particular, they showed that the parameters of the transformation have to be chosen such that

$$\sqrt{2\left(|u_{12}|^2 + |v_{12}|^2\right)} \leq \frac{2\kappa}{\tau} \tag{12}$$

where

$$\kappa = \sqrt{|\hat{a}_{12} + i\hat{b}_{12}|^2 + |\hat{a}_{21} + i\hat{b}_{21}|^2}$$

$$\tau = \left| |\hat{a}_{11} + i\hat{b}_{11}| - |\hat{a}_{22} + i\hat{b}_{22}| \right|.$$

Here we denote the 4×4 Hermitian subproblem to be solved in every step by

$$\left[\begin{array}{cc|cc} \hat{a}_{11} & \hat{a}_{12} & \hat{b}_{11} & \hat{b}_{12} \\ \overline{\hat{a}_{12}} & \hat{a}_{22} & \overline{\hat{b}_{12}} & \hat{b}_{22} \\ \hline \hat{b}_{11} & \hat{b}_{12} & -\hat{a}_{11} & -\overline{\hat{a}_{12}} \\ \overline{\hat{b}_{12}} & \hat{b}_{22} & -\hat{a}_{12} & -\hat{a}_{22} \end{array}\right]$$

and the corresponding transformation by

$$S = \begin{bmatrix} U & V \\ -V & U \end{bmatrix}$$

as before. In the case of near convergence, κ will be very small and from (12) we obtain that $|u_{12}|$ and $|v_{12}|$ have to be very small as well (if H has no multiple eigenvalues). The strategy (11) of choosing the transformation S does not guarantee condition (12). One possible generalization of condition (12) to the general non–Hermitian case is

$$\sqrt{2\left(|u_{12}|^2 + |v_{12}|^2\right)} \leq \frac{\hat{\kappa}}{\hat{\tau}}$$

where

$$\hat{\kappa} = \sqrt{2\left(|a_{21}|^2 + |q_{21}|^2\right)},$$

$$\hat{\tau} = \left| \, |a_{11} + iq_{11}| - |a_{22} + iq_{22}| \, \right|.$$

In the Hermitian and Hamiltonian case, these equations are equivalent to those in (12). In the case of near convergence, that is, if the current iterate H is close to Hamiltonian Schur form, $\hat{\kappa}$ will be small and $|u_{12}|$ and $|v_{12}|$ have to be small as well (if H has no multiple eigenvalues). Thus a different strategy in choosing the transformation S is to use the above criterion.

In our implementation we only used (11) to decide which rotation to use. For an implementation that uses the Kogbetliantz strategy see [25].

4. Parallel Implementation.

The Jacobi–like method for computing the Hamiltonian Schur form from Section 2 allows parallel implementation following almost exactly the lines of those for the symmetric Jacobi method [3, 14, 15, 16, 35, 36, 40]. We will give a brief description of a possible implementation for a loop multiple processor system and for a mesh–connected grid of processors as introduced by Brent and Luk [3] for the Jacobi method.

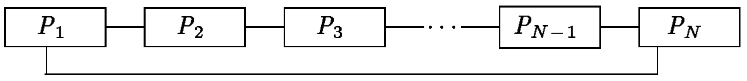
In each iteration step of the Jacobi–like method for computing the Hamiltonian Schur form only 16 matrix elements are needed to compute the next transformation $\widehat{U}_{ij}$ and only 4 rows and columns are involved; at most 6 matrix elements are annihilated. It is possible to modify the algorithm for parallel computations such that more than 6 elements are eliminated in one step. For example, eliminating the elements $a_{21}, q_{11}, q_{12}, q_{21}, q_{22}$ affects only the rows and columns $1, 2, n+1$ and $n+2$. In particular the elements in A, Q and G with the indices $(3,3), (3,4), (4,3)$ and $(4,4)$ are not affected. Therefore it is possible to compute the transformation $\widehat{U}_{34}$ which eliminates the elements $a_{43}, q_{33}, q_{34}, q_{43}, q_{44}$ (if possible) at the same time as the transformation $\widehat{U}_{12}$ which eliminates the elements $a_{21}, q_{11}, q_{12}, q_{21}, q_{22}$. Thus, if the transformation $\widehat{U}$ is chosen as $\widehat{U} = \widehat{U}_{12}\widehat{U}_{34}$, then $\widehat{U}^H H \widehat{U}$ would have zero elements in positions $(2,1), (4,3), (n+1,1), (n+2,1), (n+1,2), (n+2,2), (n+3,3), (n+4,3), (n+3,4), (n+4,4), (n+1,n+2)$ and $(n+3,n+4)$. But it is not possible to compute $\widehat{U}_{12}$ and $\widehat{U}_{13}$ at the same time because they both affect the first row and column of H. Thus, choosing the pivot indices appropriately, several transformations $\widehat{U}_{ij}$ can be computed simultaneously. To do so, one chooses a set of transformations to be computed simultaneously so that each transformation annihilates different matrix elements without changing the values of matrix elements annihilated by other transformations in the same set. For a $2n \times 2n$ Hamiltonian matrix, sets consisting of $\lfloor \frac{n}{2} \rfloor$ such transformations can be found ($\lfloor \frac{n}{2} \rfloor = n/2$ for n even, $= (n-1)/2$ for n odd). That is, at most $6 \cdot \lfloor \frac{n}{2} \rfloor$ matrix elements can be annihilated in parallel.

Hence, for an optimal implementation (see [36]) the pivot index ordering must be altered such that in each step of the method $N = \lfloor \frac{n}{2} \rfloor$ transformations can actually be computed and performed simultaneously and each pivot index pair occurs exactly once in such a parallel sweep. Thus the pivot ordering has to be rearranged such that the pivot indices can be blocked into $n-1$ (n for n odd) index pairs

$$(13) \qquad (k_1^j, l_1^j), \ldots, (k_N^j, l_N^j) \qquad k_s^j < l_s^j, \quad j = 1, \ldots, n-1 \quad (n \text{ for } n \text{ odd})$$

where each index occurs at most once in the jth block, and each pivot index pair (k,l), $k \neq l$, occurs exactly once in this sequence. The selection of the particular pattern is driven by the implementation of the algorithm on a specific architecture.

We first consider a loop of N processors which can communicate with their direct neighbors



.

Such a ring architecture has been studied for symmetric Jacobi methods, for instance, in [14, 40]. Here we follow [14] where an optimal implementation (in the sense above) is given with least communication between the processors. For the following we assume that n is even. Otherwise we append one zero row and column to each block of H and skip every transformation U_{ij} which would involve these zero row and column.

In the beginning each processor P_i holds a pivot index pair (k_i, l_i) and the corresponding four columns of H (that is the columns $k_i, l_i, n+k_i, n+l_i$). Each processor P_i computes the transformation $U_{k_i l_i}$ which transforms the 4×4 matrix $H_{k_i l_i}$ corresponding to the pivot index pair (k_i, l_i) to (almost) Hamiltonian Schur form. The next step is to transform the matrix H

$$H' = U_N^H U_{N-1}^H \dots U_1^H H U_1 U_2 \dots U_N \tag{14}$$

where $U_j = U_{k_j l_j}$. This transformation can not be performed trivially since the columns of the matrix H are distributed around the loop of processors. Each of the multiplications by matrices to the right of H in (14) affects only four columns of H. Since the processor that computed U_i currently holds the columns affected by U_i, the matrix $HU_1U_2 \dots U_N$ can be formed immediately with no interprocessor communication. Each of the multiplications to the left of H in (14) mixes four rows together. Each processor must therefore obtain descriptions of all of the transformations in order to update the rows of the four columns it currently holds. To avoid this expensive communication, a one–sided version of our algorithm can be used (for one–sided Jacobi methods see, e.g. [16]), i.e. we only carry out the column modification by multiplying $HU_1U_2 \dots U_N$. In the beginning the processors hold a pivot index pair, the corresponding four columns of H, and the corresponding columns of $U = I$. Now in each step the transformations are right–applied to H and U. After the jth step, processor P_s, holding columns $k_s^{j+1}, l_s^{j+1}, n+k_s^{j+1}$ and $n+l_s^{j+1}$ of the current H and U, can compute the 4×4 subproblem which determines the desired transformation, by forming the inner products of columns $k_s^{j+1}, l_s^{j+1}, n+k_s^{j+1}$ and $n+l_s^{j+1}$ of U with those of H. Then the transformation is computed and the columns in P_s are updated. No communication is necessary, once P_s has obtained its columns for the $(j+1)$st step.

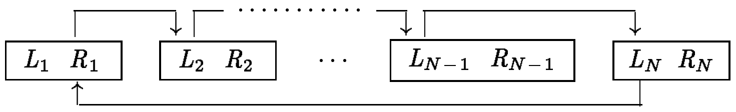
Note that in such a one-sided version of our algorithm the matrix $HU_1 \dots U_N$ is no longer Hamiltonian as we do not carry out the row modifications. But the 4×4 subproblems which have to be formed by inner products in each step are Hamiltonian again.

There are several suggestions in the literature on how to generate the $\frac{n(n-1)}{2}$ pairs of pivot indices to compute as many transformations simultaneously as possible. One suggestion was given by Brent and Luk [3]. In $(n-1)$ steps all $\frac{n(n-1)}{2}$ pivot index pairs are generated, so that in each step $N = \lfloor \frac{n}{2} \rfloor$ transformations can be computed. In step n we get the initial pivot index pairing in the initial ordering of the processors. During each step a processor must send and receive twice. In [40] sending and receiving twice is avoided. In n steps all $\frac{n(n-1)}{2}$ pivot index pairs are generated, one processor is idle every other step.

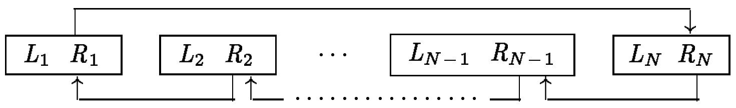
It is preferable to have one send and one receive per step and to keep all processors occupied, i.e. to return to the initial pivot index pairs after $(n-1)$ steps. Eberlein gives in [14] the following scheme, which has the desired properties. We assume that each processor P_s has registers L_s and R_s. Initially, L_s will hold columns $2s-1$ and $n+2s-1$ and R_s columns $2s$ and $n+2s$ of H and U, i.e.

$$(L_s, R_s) \quad \leftarrow \quad (2s-1, 2s) \quad \text{for } s = 1, \dots, N. \tag{15}$$

After the computation of the $(j-1)$st step the columns are exchanged according to

(16)


for j even

(17)


for j odd

For $n = 8$ with the initial distribution (15) the interchange (16), (17) results in the following migration of pivot index pairs corresponding to the registers.

step				
1	1 2	3 4	5 6	7 8
2	1 7	2 4	3 6	5 8
3	1 4	2 6	3 8	5 7
4	1 5	4 6	2 8	3 7
5	1 6	4 8	2 7	3 5
6	1 3	6 8	4 7	2 5
7	1 8	6 7	4 5	2 3
8	1 2	8 7	6 5	4 3

step				
9	1 7	8 5	6 3	4 2
10	1 4	7 5	8 3	6 2
11	1 5	7 3	8 2	6 4
12	1 6	5 3	7 2	8 4
13	1 3	5 2	7 4	8 6
14	1 8	3 2	5 4	7 6
15	1 2	3 4	5 6	7 8

The example shows that the pivot index sequence does not quite satisfy (13). After one sweep, that is after $n-1$ parallel steps we are in step n back to the initial pairing of indices, but they correspond to different processors now. Moreover, except for the first pair, k_s is now assigned to an R and ℓ_s to an L, i.e. their order is reversed. This holds true for the next steps, up to step $2n-1$, where we again are back to the initial pairing, but now in the initial ordering of processors. If an index pair (k_s^j, l_s^j), $k_s^j < l_s^j$, is assigned to a processor in reversed order, i.e.,

$$(L_s, R_s) \leftarrow (l_s^j, k_s^j)$$

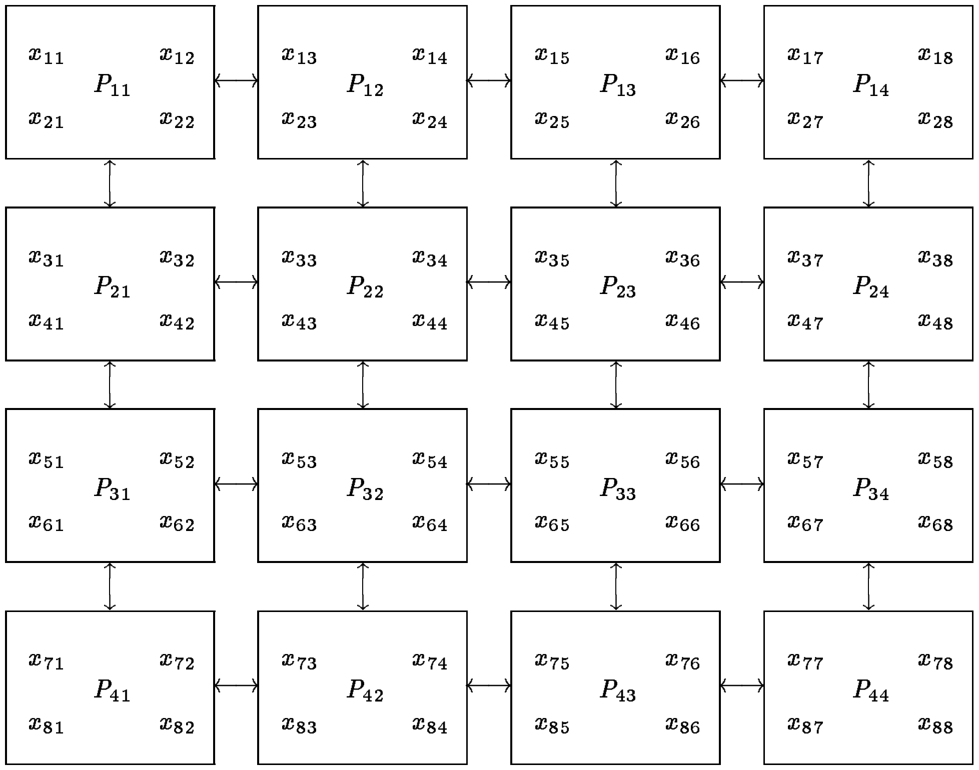
then the processor's task is slightly different; the 4×4 problem $H_{k_s^j l_s^j}$ has to be solved (not $H_{l_s^j k_s^j}$). This corresponds to a permutation of the columns, but the same operations are performed. If the program is terminated after an odd number of sweeps, a final permutation of H and Q is necessary. If it is terminated after an even number of sweeps no such permutation is necessary. With these minor modifications the pivot index sequence created according to (16), (17) can be shown to satisfy (13).

In each step of this parallel process $\mathcal{O}(n)$ operations are necessary to alter the columns in each processor. Thus one sweep needs $\mathcal{O}(n^2)$ time steps and assuming that the number of sweeps necessary for convergence is $c \cdot n$ for some constant c (see Section 5), on the whole $\mathcal{O}(n^3)$ time steps are needed.

The time for altering the columns of the iterates can only be reduced by an order of magnitude, if $\mathcal{O}(n^2)$ processors would be employed. Brent and Luk [3] suggested such a scheme for the symmetric Jacobi method. They considered a square array of $\frac{n}{2} \times \frac{n}{2}$ processors P_{ij} with horizontal and vertical communication connections. The ideas used in [3] apply to our algorithm as well (see also [15]). Each processor P_{kl} has registers L_{kl} and R_{kl} as before. Initially P_{kl} holds four 2×2 submatrices

$$\begin{bmatrix} x_{2k-1,2l-1} & x_{2k-1,2l} \\ x_{2k,2l-1} & x_{2k,2l} \end{bmatrix}$$

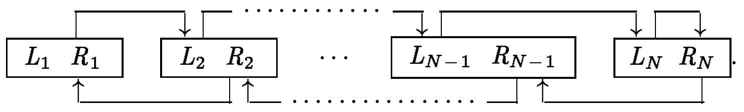
where x stands for entries from A, Q, G and $-A^H$. The $x_{2k-1,2l-1}$ and $x_{2k,2l-1}$ are kept in registers L_{kl} and $x_{2k-1,2l}$ and $x_{2k,2l}$ in R_{kl}. As before, a zero row and column is added for odd n. Here it is advisable to add them in front. We illustrate the initial state for an $n = 8$ example:


x_{11} x_{12} P_{11} x_{21} x_{22}
x_{13} x_{14} P_{12} x_{23} x_{24}
x_{15} x_{16} P_{13} x_{25} x_{26}
x_{17} x_{18} P_{14} x_{27} x_{28}
x_{31} x_{32} P_{21} x_{41} x_{42}
x_{33} x_{34} P_{22} x_{43} x_{44}
x_{35} x_{36} P_{23} x_{45} x_{46}
x_{37} x_{38} P_{24} x_{47} x_{48}
x_{51} x_{52} P_{31} x_{61} x_{62}
x_{53} x_{54} P_{32} x_{63} x_{64}
x_{55} x_{56} P_{33} x_{65} x_{66}
x_{57} x_{58} P_{34} x_{67} x_{68}
x_{71} x_{72} P_{41} x_{81} x_{82}
x_{73} x_{74} P_{42} x_{83} x_{84}
x_{75} x_{76} P_{43} x_{85} x_{86}
x_{77} x_{78} P_{44} x_{87} x_{88}

The diagonal processors P_{jj} act differently from the off–diagonal processor P_{ij} $(i \neq j)$. Each time step the diagonal processors P_{jj} compute a transformation Q_{kl} by solving the 4×4 subproblems H_{kl} which they are holding. For the initial distribution this corresponds to the $\frac{n}{2}$ pivot index pairs

$$(1,2)\,,\ (3,4)\,,\ (5,6)\,,\ldots,\ (n-1,n)\,. \tag{18}$$

For now let us assume that from each diagonal processor the computed transformations are passed through the processors in the same row and column, respectively, in constant time. After having received its transformations all processors perform the corresponding update on the 4×4 subproblem, which they currently hold, simultaneously. After all operations are completed, the processors interchange their data and proceed in the same way in order to eliminate other matrix elements. As before, we will use an interchange that creates a pivot index sequence which can be shown to satisfy (13) [3]. It corresponds to the following rule of interchange of indices (as before $N = \frac{n}{2}$)

(19) 

.

The data is interchanged according to (19) among the processors in each row and among the processors in each column, such that the N diagonal processors always contain the 2×2 submatrices of A, Q, G and $-A^H$ corresponding to the pivot index pairs. For $n = 8$ with the initial distribution (18) the interchange (19) results in the following migration of pivot index pairs:

step				
0	1 2	3 4	5 6	7 8
1	1 4	2 6	3 8	5 7
2	1 6	4 8	2 7	3 5
3	1 8	6 7	4 5	2 3
4	1 7	8 5	6 3	4 2
5	1 5	7 3	8 2	6 4
6	1 3	5 2	7 4	8 6
7	1 2	3 4	5 6	7 8

.

Note that here again pivot index pairs occur in reversed order, e.g., in step 4 of the example above we get the pair $(8, 5)$. For the 4×4 grid of processors that means that in step 4 processor P_{22} holds the 2×2 submatrix in the $(5, 8)$ plane in reversed order, i.e.,

$$\boxed{\begin{array}{ccc} x_{88} & & x_{85} \\ & P_{22} & \\ x_{58} & & x_{55} \end{array}} \; .$$

The task for processor P_{22} is as before to compute a transformation which eliminates $a_{85}, q_{55}, q_{58}, q_{85}$ and q_{88}, although these matrix entries are now in different registers as before.

After $n - 1$ steps the initial data distribution returns with this interchanging of data. One such parallel step of a sweep now only needs $\mathcal{O}(1)$ time steps. Even if communication costs are also considered, one sweep will only need $\mathcal{O}(n)$ time steps if the steps are nested as described in [3]. Thus, on the whole, $\mathcal{O}(n^2)$ time steps are necessary to compute the Hamiltonian Schur form, if we assume that $c \cdot n$ sweeps are needed.

Instead of (18) we could start with any other pairing of pivot indices satisfying the condition (13) for $j = 1$. This corresponds to permuting the initial matrix and running the method on this permuted matrix. The special pivot index ordering chosen might influence the speed of convergence.

5. Numerical Experiments.

The Jacobi–like algorithm for computing the Hamiltonian Schur form of a Hamiltonian matrix as discussed in Sections 2 and 3 was implemented in MATLAB and run on a 486 PC. As a stopping criterium we choose

$$\sigma(H)/||H||_F < tol$$

where $\sigma(H)$ is given in (6) and *tol* is chosen as $\sqrt{macheps}$. In general, the algorithm proposed here converges for values of *tol* of the order of *macheps*. The convergence rate is faster than linear and appears to be ultimately quadratic, so that at most 8 sweeps were necessary to reduce $\sigma(H)/||H||_F$ from $\sqrt{macheps}$ to $scalar * macheps$

in the examples studied here. Convergence of Byers' Hamiltonian Jacobi algorithm is extremly slow. Even when *tol* is chosen relatively big, convergence is not achieved in a reasonable time. Hence, in order to be able to run a vast number of test in a reasonable time period *tol* is chosen as $\sqrt{macheps}$.

The experiments presented here will illustrate the typical behavior of the proposed Jacobi–like algorithm. One set of tests was run using randomly generated Hamiltonian matrices with no purely imaginary eigenvalues, a second set of tests using matrices from the benchmark collection [2]. Our observations have been :

- The method did always converge; except for Hamiltonian matrices H for which every subproblem H_{ij} has only purely imaginary eigenvalues (modifying our algorithm such that in this case instead of the transformation $S = I$ a random unitary symplectic matrix S is chosen, usually resolved that problem).
- During the iteration on a randomly generated Hamiltonian matrix usually only a small number of 4×4 subproblems with only purely imaginary eigenvalues appeared. This did not seem to effect the convergence rate.
- Too many 4×4 subproblems with only purely imaginary eigenvalues does effect the convergence rate (upto no convergence if all subproblems have only purely imaginary eigenvalues). It is not yet clear what is the best choice for the transformation S in that case.
- The convergence rate is not monotone. But once the elements of Q and the lower triangle of A tend to small values, that is once a form close to Hamiltonian Schur form is achieved, the convergence rate is strictly monotone.
- Once a form close to Hamiltonian Schur form is achieved, the 4×4 subproblems have no purely imaginary eigenvalues.
- The convergence rate depends on the properties of the Hamiltonian matrix. If the matrix is Hermitian, then the convergence is fast, while for defective matrices the convergence is slow but always much faster than Byers' Hamiltonian Jacobi algorithm.
- The convergence is linear in the beginning of the process and approaches quadratic convergence at the end.
- The number of sweeps needed for convergence seems to be a linear function of the matrix size.
- The pivot index ordering has almost no effect on the convergence rate.

If the Hamiltonian matrix has purely imaginary eigenvalues (with odd multiplicity), convergence to almost Hamiltonian Schur form (5) can be observed.

The left graph in Fig. 1 displays a typical convergence behavior of a randomly generated 20×20 Hamiltonian matrix using Byers' Hamiltonian Jacobi algorithm and the method proposed in Section 2 with cyclic–by–row pivot ordering. The relative error $\sigma(H)/||H||_F$ is displayed after every sweep. The dotted line corresponds to Byers algorithm, the solid line corresponds to the method proposed here. It can be seen that for the latter method, the convergence appears to be monotonic and ultimately it is faster than linear. The convergence is not monotonic in the beginning as can be seen from the enlargement of the graph's curve in the right graph in Fig. 1. Here the relative error is displayed after every transformation for the first 5 sweeps. Especially at the beginning of a new sweep, the relative error increases slightly. But after a few sweeps, no increase can be observed, the convergence becomes monotonic.

As Byers' algorithm decreases $\sigma(H)$ in every step, its convergence is monotonic, but as can be seen, convergence is very slow. We did not include the entire convergence record, as after 250 sweeps the relative error was reduce to only 10^{-2}. Typically the

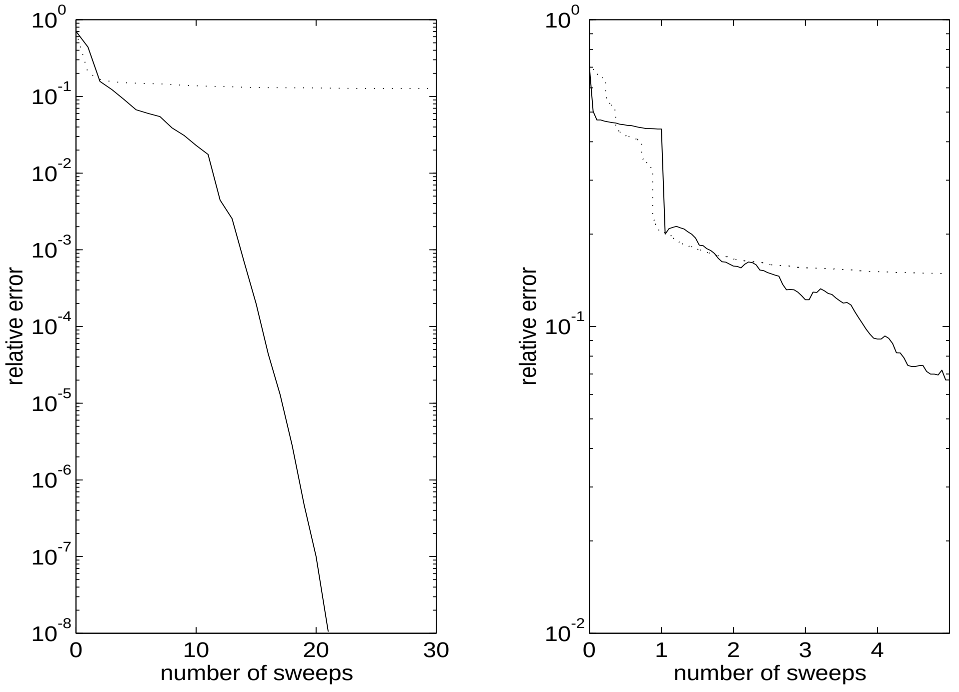


FIG. 1. *Typical convergence behavior, '···' Byers' algorithm, '-' algorithm proposed here*

relative error decreases significantly during the first few sweeps (see the right graph in Fig. 1). Then the reduction slows down till it becomes almost insignificant.

For a fair comparison of the method proposed here and Byers' method a rough estimate of the flop count is useful. Byers' algorithm computes $2n^2$ transformations per sweep, the method proposed in Section 2 computes $(n-1)n/2$ transformations. Due to the simpler structure of the transformations used in Byers' algorithm and the Hamiltonian structure of the matrix to be updated it takes only $32n^3 - 16n$ flops for computing the $2n^2$ similarity transformations. The computation of the $(n-1)n/2$ similarity transformations in the method proposed here takes $16n^3 - 8n^2 - 8n$ flops. Hence the amount of work for performing the similarity transformations per sweep in Byers' algorithm is twice as much as for the method proposed here. But the computation of each transformation in the method proposed here is about 16 times more expensive than the computation of a transformation in Byers' algorithm. A detailed flop count for the computation of a transformation as proposed in Section 3 can not be given exactly since the actual amount of work needed heavily depends on the eigenvalues of the 4×4 Hamiltonian submatrix. Moreover, we have to employ an iterative method in order to compute the eigenvalues of each 4×4 submatrix. Overall, we obtain that one sweep of Byers' algorithm is up to two times more expensive than a sweep of the method proposed here.

One set of tests was run by randomly generating 50 Hermitian and Hamiltonian matrices for dimensions $n = 3, 4, ..., 9, 10, 15, 20, 25, 30$. Fig. 2 shows the average number of sweeps necessary for convergence for the different dimensions using the method

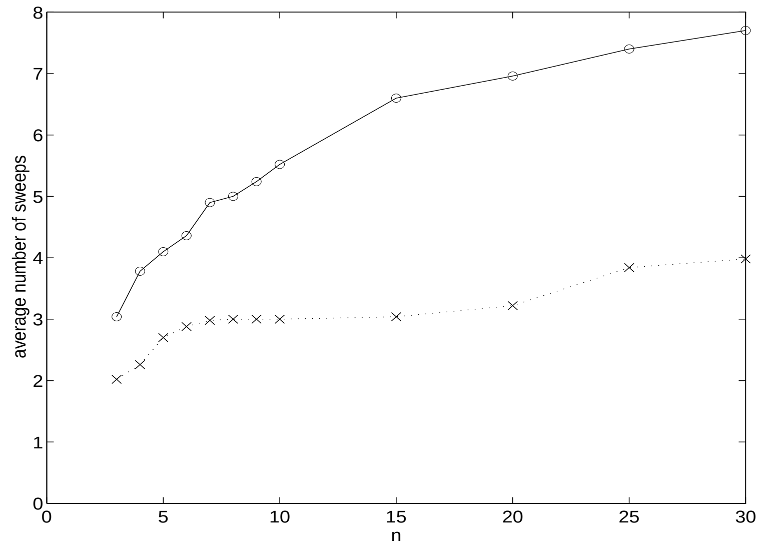


FIG. 2. *50 random Hermitian and Hamiltonian matrices of different dimensions, '· · ·' Byers' algorithm, '-' algorithm proposed here*

proposed here (again cyclic–by–row pivot ordering) and using Byers' Hamiltonian–Jacobi algorithm. As before the dotted line stands for Byers' algorithm, the solid line for the method proposed here. Both methods converge quite rapidly, Byers' algorithm being faster.

Fig. 3 shows a corresponding graph for randomly generated (non-Hermitian) Hamiltonian matrices for the method proposed here (again 50 matrices of the same dimensions as before). We omitted the curve for Byers' Hamiltonian–Jacobi algorithm, as a table including it was unreadable. The convergence for most examples was too slow, e.g., for $n = 3$ only 28 examples converged in a reasonable time (number of sweeps < 8), for the other 22 examples no convergence was achieved after 150 sweeps. Fig. 3 indicates a relation between the number of sweeps necessary for convergence and the dimension of the matrices. As pointed out before, such a relation is important for an implementation on a parallel computer, as one wants to perform a fixed amount of rotations to achieve convergence instead of determining the stopping criterium, as for the classical Jacobi method on serial machines, via the values of the elements in the lower left triangle of the matrix. This requires global knowledge of all matrix elements and is too expensive on most parallel architectures.

We got similar results for different pivot orderings. Fig. 4 shows the absolute frequency of the number of sweeps necessary to terminate the algorithm proposed here for 50 randomly generated 10×10 matrices using different pivot orderings. The bar plot to the left reports the results for the cyclic–by–row pivot ordering, the plot in the middle the results for an implementation on a mesh of processors, the plot to the right the results for an implementation on a ring. For the implementation on a mesh or ring of processors, we used the pivot orderings (16), (17), resp. (19) as discussed in Section 4. Note that for the pivot ordering given by (16), (17) we always perform an even number of sweeps to obtain the columns of the matrix in the correct order.

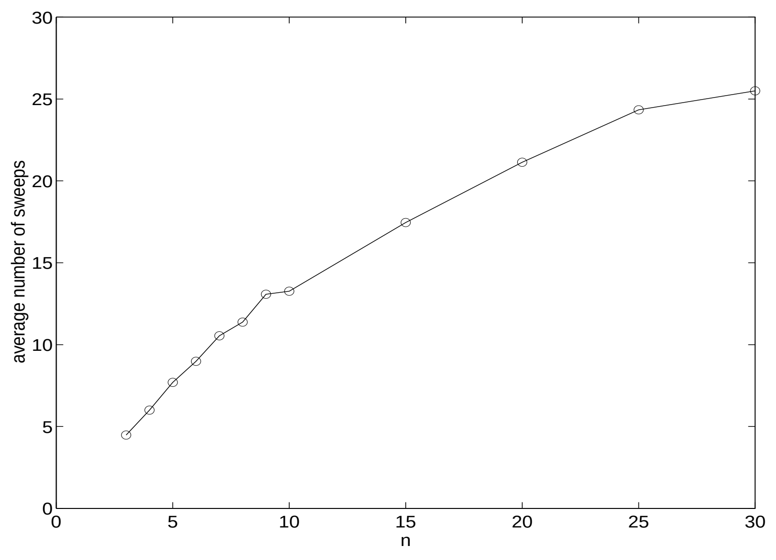


FIG. 3. *50 random Hamiltonian matrices of different dimensions*

Obviously, all three different pivot orderings show a similar normal distribution–like behavior, which one would like to see for random matrices. For larger dimensions the number of sweeps necessary to achieve convergence increases for all orderings. The corresponding tables remain qualitatively the same. For Byers' algorithm an almost normal distribution of the absolute frequency of the number of sweeps necessary for convergence was not observed. In a corresponding graph the frequencies spread out almost uniformly between 6 and 250 sweeps.

For matrices far from normality, the convergence is much slower than the one observed for the randomly generated matrices. A similar behavior was already observed by Greenstadt [20] and Eberlein [15] for their Jacobi–like algorithms to compute the Schur form of a general matrix. See also [6]. As this observation is not new, we refrain from giving an example.

Further tests with matrices from the benchmark collection [2] were run. Convergence was achieved for all test matrices. Despite for example 5, an example with $n = 9$ which took 50 sweeps for convergence, the observed convergence rate was quite fast for all examples.

6. Concluding Remarks.

We presented a Jacobi–like method for the computation of a Hamiltonian Schur decomposition of a Hamiltonian matrix, which generalizes the Jacobi–like method for unsymmetric matrices by Eberlein [15]. We indicated that our method allows parallel implementation on a ring of $\frac{n}{2}$ processors and on an $\frac{n}{2} \times \frac{n}{2}$ square array mesh–connected processor in complete analogy to the case of arbitrary unsymmetric matrices. Numerical experiments were run to compare the convergence of the method proposed here with Byers' Hamiltonian–Jacobi algorithm [12], which is a generalization of the unsymmetric Jacobi–like method by Stewart [39]. We found that the performance of the method proposed here on the experimental data was much superior to the performance of Byers' Hamiltonian–Jacobi algorithm for Hamiltonian matrices which are not Hermitian. The results given here clearly demonstrate the

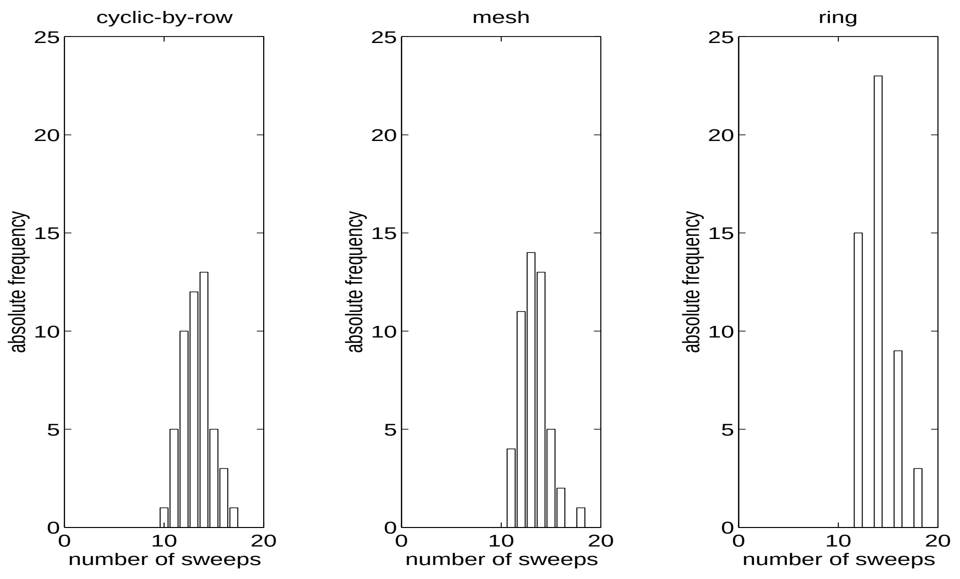


FIG. 4. *absolute frequency of the number of sweeps necessary for convergence, different pivot orderings*

significantly different behavior of the two approaches to develop a Jacobi–like method for Hamiltonian matrices. It might thus lead to a direction for further development and research on parallelizable methods to solve the Hamiltonian eigenproblem.

Acknowledgment Part of this work was done while the second author visited the University of California Santa Barbara. The second author would like to thank Alan Laub for making the visit possible. Further, the authors would like to thank Ralph Byers for providing his MATLAB implementation of the Hamiltonian–Jacobi algorithm.

REFERENCES

[1] G.Ammar, P.Benner, V.Mehrmann. *A multishift algorithm for the numerical solution of algebraic Riccati equations.* ETNA, 1(1993),33–48

[2] P.Benner, A.J.Laub, V.Mehrmann. *A collection of benchmark exemples for the numerical solution of algebraic Riccati equations I: continuous–time case.* Tech. Report SPC 95_22, Technische Universität Chemnitz-Zwickau, Germany,1995

[3] R.P.Brent, F.T.Luk. *The solution of singular–value and symmetric eigenvalue problems on multiprocessor arrays.* SIAM J.Sci.Stat.Comput., 6(1985),69–84

[4] A.Bunse–Gerstner. *Symplectic QR–like methods.* Habilitationsschrift Universität Bielefeld, 1986

[5] A.Bunse–Gerstner,R.Byers,V.Mehrmann. *Numerical methods for algebraic Riccati equations.* Proc.workshop on the Riccati equation in control, systems and signals (Como,Italy), 1989,107–116

[6] A.Bunse–Gerstner,H.Faßbender. *On the generalized Schur decomposition of a matrix pencil for parallel computation.* SIAM J.Sci.Stat.Comput., vol 12,no 4(1991),911–939

[7] A.Bunse–Gerstner,V.Mehrmann. *A symplectic QR-like algorithm for the solution of the real algebraic Riccati equation.* IEEE Trans.Comp., 31(1986),1104–1113

[8] A.Bunse–Gerstner,V.Mehrmann,D.Watkins. *An SR algorithm for Hamiltonian matrices based on Gaussian elimination.* Methods of OR, 58(1989),339–358

[9] R.Byers. *Hamiltonian and symplectic algorithm for the algebraic Riccati equation.*

Dept.Comp.Sci., Ithaka, NY, 1983

[10] R.Byers. *A Hamiltonian QR algorithm.* SIAM Sci.Stat., 7(1986),212–229

[11] R.Byers. *Solving the algebraic Riccati equation with the matrix sign function.* Lin.Alg.Appl., 85(1987),267–279

[12] R.Byers. *A Hamiltonian Jacobi algorithm.* IEEE Trans.Aut.Control, 35(1990),566–570

[13] J.–P.Charlier,P.Van Dooren. *A Jacobi–like algorithm for computing the generalized Schur form of a regular pencil.* JCAM, Vol 27 pp. 17-36, Sept. 1989

[14] P.J.Eberlein. *On using the Jacobi method on the hypercube.* Tech.Report 86–23, Department of Computer Science, State University of New York at Buffalo, Buffalo NY, 1986

[15] P.J.Eberlein. *On the Schur decomposition of a matrix for parallel computation.* IEEE Trans.Comp., 36(1987),167–174

[16] P.J.Eberlein. *On one–sided Jacobi methods for parallel computation.* SIAM J.Algebraic Discrete Methods, 8(1987),790–796

[17] G.E.Forsythe,P.Henrici. *The cyclic Jacobi method for computing the principal values of a complex matrix.* Trans.Amer.Math.Soc., 94(1960),1–23

[18] J.D.Gardiner,A.J.Laub. *A generalization of the matrix–sign–function solution for algebraic Riccati equations.* Int.Control, 44(1986),823–832

[19] J.D.Gardiner,A.J.Laub. *Parallel algorithms for algebraic Riccati equations.* Int.Control, 54(1991),1317–1333

[20] J.Greenstadt. *A method for finding roots of arbitrary matrices.* Math. Tables Aids Comput., 9(1955),47–52

[21] P.Henrici. *On the speed of convergence of cyclic and quasi–cyclic Jacobi methods for computing eigenvalues of Hermitian matrices.* J.Soc.Indust.Appl.Math., 6(1958),144–162

[22] G.G.J.Jacobi. *Über ein leichtes Verfahren, die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen.* J.Reine Angew.Math., 30(1846),51–95

[23] H.P.M.van Kempen. *On the convergence of the classical Jacobi method for real symmetric matrices with non–distinct eigenvalues.* Numer.Math., 9(1966),11–18

[24] H.P.M.van Kempen. *On the quadratic convergence of the special cyclic Jacobi method.* Numer.Math., 9(1966),19–22

[25] F. Klapper. *Ein Jacobi-ähnliches Verfahren zur Lösung der algebraischen Riccati-Gleichung auf Parallelrechnern.* Master's thesis, Fakultät für Mathematik, Universität Bielefeld, Germany, 1989

[26] H.W.Knobloch,H.Kwakernaak. *Lineare Kontrolltheorie.* Springer Berlin, 1985

[27] E.G.Kogbetliantz. *Solutions of linear equations by diagonalization of coeffients matrix.* Quart.Appl.Math., 13(1955),123–132

[28] P.Lancaster,L.Rodman. *Algebraic Riccati equations.* Oxford University Press, 1995

[29] A.J.Laub. *A Schur method for solving algebraic Riccati equations.* IEEE Trans.Comput., 24(1979),913–921

[30] W.W.Lin,S.S.You. *A symplectic acceleration method for the solution of the algebraic Riccati equation on a parallel computer.* Lin.Alg.Appl., 188/189(1993),437–464

[31] F.T.Luk,H.Park. *A proof of convergence for two parallel Jacobi SVD algorithms.* IEEE Trans.Comput., 38(1989),806–811

[32] V.Mehrmann. *The autonomous linear quadratic control problem, theory and numerical solution.* Lecture notes in control and information sciences 163, Springer Heidelberg 1991

[33] C.C.Paige,C.F.Van Loan. *A Schur decomposition for Hamiltonian matrices.* Lin.Alg.Appl., 14(1981),11–32

[34] C.Paige,P.Van Dooren. *On the quadratic convergence of Kogbetliantz's algorithm for computing the singular value decomposition.* Lin.Alg.Appl., 77(1986),301–313

[35] P.P.M.de Rijk. *A one–sided Jacobi algorithm for computing the singular value decomposition on a vector computer.* SIAM J.Sci.Stat.Comput., 10(1989),359–371

[36] A.H.Sameh. *On Jacobi and Jacobi–like algorithms for a parallel computer.* Math.Comp., 25(1971),579–590

[37] A.Schönhage. *Zur Konvergenz des Jacobi–Verfahrens.* Numer.Math., 3(1961),374–380

[38] A.Schönhage. *Zur quadratischen Konvergenz des Jacobi–Verfahrens.* Numer.Math., 5(1964),410–412

[39] G.W.Stewart. *A Jacobi–like algorithm for computing the Schur decomposition of a non–Hermitian matrix.* SIAM J.Sci.Stat.Comput., 6(1985),853–864

[40] R.A.Whiteside,N.S.Ostlund,R.G.Hibbard. *A parallel Jacobi diagonalization algorithm for a loop multiprocessor system.* IEEE Trans.Comput., 33(1984),409–413

[41] J.H.Wilkinson. *Note on the quadratic convergence of the cyclic Jacobi process.* Numer.Math., 4(1962),296–300