SIAM REVIEW
Vol. 47, No. 1, pp. 3–40

Product Eigenvalue Problems*

David S. Watkins†

Abstract. Many eigenvalue problems are most naturally viewed as product eigenvalue problems. The eigenvalues of a matrix A are wanted, but A is not given explicitly. Instead it is presented as a product of several factors: $A = A_k A_{k-1} \cdots A_1$. Usually more accurate results are obtained by working with the factors rather than forming A explicitly. For example, if we want eigenvalues/vectors of $B^T B$, it is better to work directly with B and not compute the product. The intent of this paper is to demonstrate that the product eigenvalue problem is a powerful unifying concept. Diverse examples of eigenvalue problems are discussed and formulated as product eigenvalue problems. For all but a couple of these examples it is shown that the standard algorithms for solving them are instances of a generic GR algorithm applied to a related cyclic matrix.

Key words. eigenvalue, QR algorithm, GR algorithm, generalized eigenvalue problem, SVD, symmetric, skew symmetric, positive definite, totally positive, pseudosymmetric, Hamiltonian, symplectic, unitary

AMS subject classifications. 65F15, 15A18

DOI. 10.1137/S0036144504443110

1. Introduction.

There are many situations in which one wishes to find some or all of the eigenvalues of a matrix. Moreover, in many cases the matrix is not given explicitly but rather as a product of two or more matrices:

$$A = A_k A_{k-1} \cdots A_1.$$

The challenge is to compute the eigenvalues of A by operating on the factors A_1, $A_2, \ldots, A_k$. This is an old problem, and there are several instances that are widely known.

Perhaps the best known example is the singular value decomposition (SVD). Every matrix $B \in \mathbb{R}^{n\times m}$ can be decomposed as

$$B = U\Sigma V^T,$$

where $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ are orthogonal, and $\Sigma \in \mathbb{R}^{n\times m}$ is diagonal:

$$\Sigma = \left[\begin{array}{cccc|c} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \ddots & & \\ & & & \sigma_r & \\ \hline & & & & 0 \\ & & & & & \ddots \end{array}\right], \qquad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0.$$

*Received by the editors April 17, 2004; accepted for publication (in revised form) July 14, 2004; published electronically February 1, 2005.
http://www.siam.org/journals/sirev/47-1/44311.html

†Department of Mathematics, Washington State University, Pullman, WA 99164-3113 (watkins@math.wsu.edu).

This decomposition exposes completely the structure of B and is therefore useful in many contexts. Once we have computed the SVD, we immediately have the spectral decompositions

$$BB^T = U(\Sigma\Sigma^T)U^T \quad \text{and} \quad B^TB = V(\Sigma^T\Sigma)V^T$$

implicitly. Thus the SVD of B gives the complete eigensystems of BB^T and B^TB without forming these products explicitly. From the standpoint of accuracy, this is the right way to compute these eigensystems, since information about the smaller eigenvalues are lost when BB^T and B^TB are computed in floating-point arithmetic.

Example. A product eigenvalue problem of this type arises in principal components analysis [59]. Suppose we have a sample of 1000 individuals, and for each individual we have a list of some 50 characteristics such as height, left index finger length, length of tail, and so on. All of these numbers can be stored in a 1000×50 matrix having a row for each individual and a column for each characteristic. Suppose we take the average of each characteristic and subtract it from the appropriate column to obtain a matrix $\hat{B}$ whose columns have mean zero. Then let $B = \rho\hat{B}$, where $\rho = 1/\sqrt{n-1}$. The product B^TB is then a symmetric matrix whose (i, j) entry is the covariance of the ith and jth characteristics. The objective of principal components analysis is to identify some simple combinations of the characteristics that account for the bulk of the variance of the sample. This can be accomplished by looking at eigenvectors of B^TB. The eigenvector associated with the largest eigenvalue λ_1 gives the relative weights of the combination of characteristics that has the greatest variance. The value of this greatest variance is λ_1. The eigenvector associated with the second largest eigenvalue λ_2 gives the weights of a second combination of characteristics, uncorrelated with the first combination, that accounts for the greatest remaining variation. Its variance is λ_2. The third, fourth, and fifth most important combinations can be found by looking at the third, fourth, and fifth eigenvectors, and so on.

It is now recognized that the most accurate way to compute the principal components is to compute the SVD $B = U\Sigma V^T$. The desired eigenvectors are the columns of V, and the associated variances are σ_1^2, σ_2^2, ..., the squares of the singular values. In this way, the eigenvalue problem for B^TB is solved without even forming the product B^TB.[1]

Example. It is easy to generate small problems in MATLAB that illustrate the advantage of working with B instead of B^TB. As an extreme case consider the MATLAB code

```
format long e
randn('state',123);
P = randn(3); [P,R] = qr(P);
Q = randn(2); [Q,R] = qr(Q);
S = diag([1, 1.23456789e-10]);
B = P(:,1:2)*S*Q';
Eigenvalues = eig(B'*B)
Singular_Values = svd(B)
```

This generates a 3×2 matrix with random singular vectors and specified singular values $\sigma_1 = 1$ and $\sigma_2 = 1.23456789 \times 10^{-10}$. It then computes the eigenvalues of B^TB

[1] Since only V and Σ are needed, a good deal of computational effort is spared by not computing the 1000×1000 orthogonal matrix U. This keeps the computing cost within reason.

by the command `eig`. These should be the squares of the singular values. Finally it computes the singular values of B using the `svd` command. Here's the output:

```
Eigenvalues =

    -2.775557561562891e-17
     9.999999999999998e-01

Singular_Values =

     9.999999999999999e-01
     1.234567850536871e-10
```

The large eigenvalue is correct to full (IEEE standard [5, 77] double) precision, but the tiny one is completely off; it's not even positive. This is not the fault of the `eig` command. By the time B^TB had been computed, the damage was done. The `svd` computation came out much better. The large singular value is correct to full precision and the tiny one to eight decimal places.

Let us now return to the general problem $A = A_kA_{k-1}\cdots A_1$. It often happens that some of the A_j are given in inverse form. That is, we might have, for example, $A = A_2B_2^{-1}A_1B_1^{-1}$, where A_1, A_2, B_1, and B_2 are given. We wish to compute eigenvalues of A without forming the inverses.

Many important eigenvalue problems present themselves most naturally as generalized eigenvalue problems $(A - \lambda B)v = 0$. This is the simplest instance of an eigenvalue problem involving an inverse, as it is equivalent to the eigenvalue problem for AB^{-1} or $B^{-1}A$. However, it is best to solve the problem without forming either of these products or even inverting B. Again accuracy can be lost through the inversion of B or formation of the product. Sometimes B does not have an inverse.

Example. If one wants to compute the fundamental vibrational modes of an elastic structure such as a building or an airplane wing, one often carries out a finite element analysis [91, 106]. The structure is approximated by a discrete assemblage of elements, and the vibrational modes are then found as solutions to a generalized eigenvalue problem $(A - \lambda B)v = 0$, where A and B are the finite-element stiffness and mass matrices, respectively.

Example. Here is a small MATLAB example in which a superior outcome is obtained by keeping A and B separate.

```
format long e
randn('state',123);
M = randn(2); [M,R] = qr(M);
N = randn(2); [N,R] = qr(N);
A = [ 2 1; 0 1e-8];  B = [ 1 1; 0 1e-8];
A = M*A*N; B = M*B*N;
Smart_Result = eig(A,B)
Other_Result = eig(A*inv(B))
```

This code produces a 2×2 matrix pair (A, B) for which the eigenvalues are $\lambda_1 = 2$ and $\lambda_2 = 1$. B is ill conditioned, as is A. The eigenvalues are computed two different ways. The command `eig(A,B)` uses the QZ algorithm [76], which works with A and B separately to compute the eigenvalues. The command `eig(A*inv(B))` obviously computes AB^{-1} and then finds the eigenvalues. Here are the results:

```
Smart_Result =

     1.999999999475219e+00
     1.000000000557143e+00

Other_Result =

     1.997208963148296e+00
     1.002791036386043e+00
```

We see that the QZ computation got the eigenvalues correct to about ten decimal places. The fact that it did not do better can be attributed to the ill conditioning of the eigenvalues. As we see, the computation that formed AB^{-1} explicitly did much worse.

In addition to the SVD problem and the generalized eigenvalue problem, there are many other examples of product eigenvalue problems. A rich variety will be discussed in this paper. Most of these problems are normally considered standard eigenvalue problems for which the matrix has some special structure. We shall see that in many cases we can profit by reformulating the structured problem as a product eigenvalue problem. The structures considered in this paper include skew-symmetric, symmetric, totally nonnegative, pseudosymmetric, Hamiltonian, symplectic, and unitary. Applications of the symmetric eigenvalue problem are numerous and well known. The pseudosymmetric eigenvalue problem is quite general. Indeed almost any matrix can be transformed to pseudosymmetric tridiagonal form by a similarity transformation. The nonsymmetric Lanczos process [69] generates a pseudosymmetric tridiagonal matrix whose eigenvalues must then be computed. The Hamiltonian eigenvalue problem arises in several contexts, including linear-quadratic optimal control problems [17, 26, 68, 70, 71, 74], determination of corner singularities in anisotropic elastic structures [6, 75], and stability of gyroscopic systems [67]. The symplectic eigenvalue problem arises in discrete-time linear-quadratic control problems [58, 68, 74, 79]. A unitary eigenvalue problem must be solved in order to determine the points and weights for a Gauss–Szegö quadrature rule [51, 53]. Gauss–Szegö rules are numerical integration formulas of optimal degree with respect to measures with support on the unit circle.

We will consider the solution of product eigenvalue problems by the famous QR algorithm and its close relatives, which are collectively known as GR algorithms. Historically, the first algorithms of this type were the quotient-difference and LR algorithms of Rutishauser [83, 84, 85] developed in the mid- to late 1950s. These were soon followed by the QR algorithm of Francis [37, 38] and Kublanovskaya [66]. More than 40 years later, the QR algorithm is still the most important method for computing the eigenvalues of small to medium-sized matrices (up to about 1000×1000, or perhaps a bit larger).[2] The QR codes that are used in practice for small matrices differ hardly at all from the form given by Francis [37, 38]. For larger matrices, newer innovations are beginning to have an impact [19, 20, 57]. See also [7, 29, 97, 98]. By the way, the problem of computing the eigenvalues of a single, explicitly given matrix $A \in \mathbb{C}^{n\times n}$ also falls within the scope of this paper. A is a product of k matrices, where $k = 1$.

[2]For matrices much larger than this, completely different methods are recommended, as it becomes essential to exploit sparseness [8, 72, 87, 88, 89]. However, even in this realm GR algorithms have a role to play. Many sparse eigenvalue solvers make repeated use of GR algorithms to solve small subproblems that result from projections onto low-dimensional subspaces.

Over time a number of extensions of the QR algorithm for a variety of product eigenvalue problems were developed. First there was Golub and Kahan's 1965 QR algorithm for the SVD [46, 47]. Then, by 1973, Moler and Stewart [76] had developed the QZ algorithm for the generalized eigenvalue problem. This effects QR iterations on AB^{-1} and $B^{-1}A$ implicitly, without ever forming these products or inverting B. Shortly thereafter, Van Loan [92] presented a generalization to products of four matrices that encompasses the standard problem, the SVD problem, the generalized eigenvalue problem, and the problem $(A^TA - \lambda B^TB)v = 0$ as special cases. Later Hench and Laub [56] and Bojanczyk, Golub, and Van Dooren [18] developed extensions of the QR algorithm to products of large numbers of matrices. This last development is called the *periodic QZ algorithm*.

Each of these steps extended a bit further the scope of the QR algorithm, or so it appeared. Now recent work of Kressner [65, 64] showed that each of these extensions is not really an extension; it is just an instance of the standard QR algorithm, if we view the product eigenvalue problem in an appropriate light. Adopting this viewpoint and generalizing it very slightly, we will find that a rich variety of algorithms for solving diverse problems flows from the GR algorithms for the standard eigenvalue problem.

2. The GR Algorithm for a Single Matrix. We begin with the single matrix case: find the eigenvalues of a matrix $A \in \mathbb{C}^{n\times n}$. It is a consequence of Galois theory [55] that all algorithms for computing eigenvalues of matrices larger than 4×4 are iterative. The generic GR algorithm is an iteration that repeatedly performs similarity transformations to move the matrix toward upper triangular form. Since similarity transformations preserve eigenvalues, and the eigenvalues of an upper triangular matrix are evident, the GR algorithm ultimately delivers the eigenvalues.

A single GR iteration has the following form: Pick a function f and compute $f(A)$. Then compute a decomposition $f(A) = GR$, where G is nonsingular and R is upper triangular. Finally, use G to effect a similarity transformation: $\hat{A} = G^{-1}AG$. In summary,

$$f(A) = GR, \qquad \hat{A} = G^{-1}AG. \tag{1}$$

$\hat{A}$ is our new iterate. It has the same eigenvalues as A, and, if f and G are chosen well, it will be closer to upper triangular than A was.

This looks like a time-consuming operation. Even if f is something so simple as a quadratic polynomial, the computation of $f(A)$ will require a matrix-matrix multiply, which is not cheap if A is large. The GR decomposition can be expensive as well, and so can the similarity transformation. For now, suffice it to say that there are ways of doing the operations economically. More will be said on this score later.

The function f is usually taken to be a polynomial, although sometimes a rational function is used. If f is a polynomial of degree m, say

$$f(A) = (A - \mu_1 I)(A - \mu_2 I)\cdots(A - \mu_m I),$$

then we speak of a GR iteration of *degree* m. The complex numbers $\mu_1, \ldots, \mu_m$ are called the *shifts* for the GR iteration. For rapid convergence it is best to choose shifts that are good approximations to eigenvalues of A. As we proceed with the iterations, we get progressively better estimates of eigenvalues. It follows that we should update our choice of shifts now and then. In practice, new shifts are chosen on each iteration. For discussions of shift selection and convergence, see [103] and [100], for example.

Usually the convergence to triangular form does not take place in a uniform manner. Typically we will find after some iterations that the current iterate satisfies

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \approx \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}.$$

That is, there is a $k \times (n-k)$ block, for some k, that is close enough to zero (on the same order of magnitude as the roundoff errors) that we can declare it to be zero for practical purposes. Then subsequent iterations can operate on the submatrices A_{11} and A_{22} separately. Typically A_{22} is a small matrix, and finding its eigenvalues λ_1, $\ldots$, λ_k is a relatively trivial task. We speak of *deflating out* the eigenvalues λ_1, $\ldots$, λ_k. Subsequent iterations operate on A_{11}.

There are many ways to do the decomposition $f(A) = GR$, since all we require in general is that G be nonsingular and R be upper triangular. However, we often put additional requirements on G. If we take G to be unitary, then we call it Q, and the GR iteration is then called a QR iteration. If we take G to be unit lower triangular (resp., symplectic), we call it L (resp., S) and refer to an LR (resp., SR) iteration.

Often the choice of f and G is determined partly by the *principle of structure preservation:* If A has any significant structure, then we should design a GR iteration that preserves the structure. Observation of this principle usually results in algorithms that are superior in speed and accuracy. For example, if A is Hermitian ($A^* = A$), we would like $\hat{A}$ also to be Hermitian. This can be achieved by taking G to be unitary, that is, by using the QR algorithm.

As another example, suppose A is real. This is a significant structure and should be preserved. Since A may well have some complex eigenvalues, it might appear that we would be forced out into the complex plane, for if we want to approximate complex eigenvalues well, we must use complex shifts. The remedy is simple: For each complex shift μ we use, we must also use the complex conjugate $\overline{\mu}$. This guarantees that $f(A)$ stays real. Then in the decomposition $f(A) = GR$, we can take G and R to be real to guarantee that we stay within the real field. These iterations can never approach upper triangular form, but they can approach quasi-triangular form, in which each complex conjugate pair of eigenvalues is delivered as a real 2×2 block. Thus each complex conjugate pair of eigenvalues is deflated together.

3. A GR Iteration for a Product of Matrices. Now consider a product

$$A = A_k A_{k-1} \cdots A_1 \in \mathbb{C}^{n \times n}.$$

In most of our applications, all of the factors A_i are square, but they do not have to be. A is closely related to the cyclic matrix

$$C = \begin{bmatrix} & & & & A_k \\ A_1 & & & & \\ & A_2 & & & \\ & & \ddots & & \\ & & & A_{k-1} & \end{bmatrix}.$$

The following theorem is well known[3] and easy to prove.

[3]This dates back at least to Varga's classic book [94], in which an equivalent result is used in the analysis of the convergence of the successive-overrelaxation (SOR) method.

THEOREM 3.1. *The nonzero complex number λ is an eigenvalue of A if and only if its kth roots $\lambda^{1/k}$, $\lambda^{1/k}\omega$, $\lambda^{1/k}\omega^2$, ..., $\lambda^{1/k}\omega^{k-1}$ are all eigenvalues of C.*

Here $\lambda^{1/k}$ denotes any one of the kth roots of λ, and $\omega = e^{2\pi i/k}$. There are also simple relationships between the eigenvectors, which we will not state. The theorem is easily proved by writing down the eigenvector equation $Cx = \tau x$ and noting the relationships that it implies. Notice that if

$$x = \begin{bmatrix} x_1^T & x_2^T & \cdots & x_{k-1}^T & x_k^T \end{bmatrix}^T$$

is an eigenvector of C associated with τ, and α is a kth root of unity, then

$$x(\alpha) = \begin{bmatrix} \alpha^{k-1}x_1^T & \alpha^{k-2}x_2^T & \cdots & \alpha x_{k-1}^T & x_k^T \end{bmatrix}^T$$

is an eigenvector of C associated with $\tau\alpha$.

The study of product GR iterations is most easily undertaken by considering a generic GR iteration on the cyclic matrix C. What sort of polynomial f should we use to drive the iteration? The cyclic structure of C implies that if τ is an eigenvalue, then so are $\tau\omega$, $\tau\omega^2$, ..., $\tau\omega^{k-1}$. If we wish to preserve the cyclic structure, we must seek to extract all of these eigenvalues simultaneously. This means that if we use a shift μ (approximating τ, say), we must also use $\mu\omega$, ..., $\mu\omega^{k-1}$ as shifts. Thus our driving polynomial f must have a factor

$$(z-\mu)(z-\mu\omega)(z-\mu\omega^2)\cdots(z-\mu\omega^{k-1}) = z^k - \mu^k.$$

If we wish to apply shifts μ_1, ..., μ_m, along with the associated $\mu_i\omega^j$, we should take

$$f(z) = (z^k - \mu_1^k)(z^k - \mu_2^k)\cdots(z^k - \mu_m^k).$$

Thus the principle of structure preservation dictates that $f(C)$ should be a polynomial in C^k: $f(C) = p(C^k)$.

Clearly C^k has the block-diagonal form

$$C^k = \begin{bmatrix} A_kA_{k-1}\cdots A_1 & & & \\ & A_1A_k\cdots A_2 & & \\ & & \ddots & \\ & & & A_{k-1}\cdots A_1A_k \end{bmatrix},$$

so $f(C)$ is also block diagonal.

Let us consider the case $k = 3$ for illustration. It is convenient to modify the notation slightly: In place of A_1, A_2, and A_3 we write A_{21}, A_{32}, and A_{13}, respectively. Then

$$C = \begin{bmatrix} & & A_{13} \\ A_{21} & & \\ & A_{32} & \end{bmatrix}$$

and

$$C^3 = \begin{bmatrix} A_{13}A_{32}A_{21} & & \\ & A_{21}A_{13}A_{32} & \\ & & A_{32}A_{21}A_{13} \end{bmatrix},$$

so

$$f(C) = p(C^3) = \begin{bmatrix} p(A_{13}A_{32}A_{21}) & & \\ & p(A_{21}A_{13}A_{32}) & \\ & & p(A_{32}A_{21}A_{13}) \end{bmatrix}.$$

To effect an iteration of the generic GR algorithm we must now obtain a decomposition $f(C) = GR$. The obvious way to do this is to decompose the blocks separately and assemble the result. Say $p(A_{13}A_{32}A_{21}) = G_1R_1$, $p(A_{21}A_{13}A_{32}) = G_2R_2$, and $p(A_{32}A_{21}A_{13}) = G_3R_3$. Then $f(C) = p(C^k) = GR$, where

$$G = \begin{bmatrix} G_1 & & \\ & G_2 & \\ & & G_3 \end{bmatrix} \quad \text{and} \quad R = \begin{bmatrix} R_1 & & \\ & R_2 & \\ & & R_3 \end{bmatrix}.$$

The GR iteration is completed by a similarity transformation $\hat{C} = G^{-1}CG$. Clearly

$$\hat{C} = \begin{bmatrix} & & \hat{A}_{13} \\ \hat{A}_{21} & & \\ & \hat{A}_{32} & \end{bmatrix} = \begin{bmatrix} & & G_1^{-1}A_{13}G_3 \\ G_2^{-1}A_{21}G_1 & & \\ & G_3^{-1}A_{32}G_2 & \end{bmatrix}. \tag{2}$$

The cyclic structure has been preserved.

Now consider what has happened to the product $A = A_{13}A_{32}A_{21}$. We easily check that $\hat{A}_{13}\hat{A}_{32}\hat{A}_{21} = G_1^{-1}A_{13}A_{32}A_{21}G_1$. The equations

$$p(A_{13}A_{32}A_{21}) = G_1R_1, \qquad \hat{A}_{13}\hat{A}_{32}\hat{A}_{21} = G_1^{-1}A_{13}A_{32}A_{21}G_1$$

together imply that a generic GR iteration driven by p has been effected on the product $A_{13}A_{32}A_{21}$. Similarly we have

$$p(A_{21}A_{13}A_{32}) = G_2R_2, \qquad \hat{A}_{21}\hat{A}_{13}\hat{A}_{32} = G_2^{-1}A_{21}A_{13}A_{32}G_2$$

and

$$p(A_{32}A_{21}A_{13}) = G_3R_3, \qquad \hat{A}_{32}\hat{A}_{21}\hat{A}_{13} = G_3^{-1}A_{32}A_{21}A_{13}G_3,$$

so the GR iteration on C implicitly effects GR iterations on the products $A_{13}A_{32}A_{21}$, $A_{21}A_{13}A_{32}$, and $A_{32}A_{21}A_{13}$ simultaneously. These are iterations of degree m with shifts $\mu_1^k, \ldots, \mu_m^k$, whereas the iteration on C is of degree mk with shifts $\mu_i\omega^j$, $i = 1, \ldots, m$; $j = 0, \ldots, k-1$.

This looks like a lot of work; it seems to require that we compute

$$p(A_{k,k-1}\cdots A_{21}A_{1k}), \quad p(A_{21}A_{1k}\cdots A_{32}),$$

and so on. Fortunately there is a way around all this work. All that is needed is the first column of one of these matrices, and this can be computed with relative ease. We never form any of the products. We work on the factors separately, carrying out transforms of the form

$$\hat{A}_{j+1,j} = G_{j+1}^{-1}A_{j+1,j}G_j, \tag{3}$$

as indicated in (2). If any of the factors is presented in inverse form, we perform the equivalent transformation on the inverse. That is, if we actually have in hand $B_{j,j+1} = A_{j+1,j}^{-1}$, instead of $A_{j+1,j}$, we do the transformation

$$\hat{B}_{j,j+1} = G_j^{-1}B_{j,j+1}G_{j+1}$$

instead of (3).

4. Efficient Implementation of GR Iterations.

Reduction to Hessenberg Form. We return to the case of a single matrix A. Efficient implementation of GR iterations requires a preprocessing step that transforms A to a condensed form, usually upper Hessenberg form. A is *upper Hessenberg* if $a_{ij} = 0$ whenever $i > j + 1$. This means that A is almost upper triangular:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ & a_{32} & a_{33} & a_{34} & a_{35} \\ & & a_{43} & a_{44} & a_{45} \\ & & & a_{54} & a_{55} \end{bmatrix}.$$

The transformation to upper Hessenberg form can be achieved by stages. Starting from a general matrix $A = B_0$, we apply a transformation G_1 of simple form such that $G_1^{-1}B_0$ has the desired zeros in the first column. G_1 can be a Householder transformation or a Gaussian elimination transformation, for example. The form of G_1 is

$$\begin{bmatrix} 1 & \\ & \hat{G}_1 \end{bmatrix},$$

so the first row of B_0 is untouched by G_1^{-1}. Then when we complete the similarity transformation to form $B_1 = G_1^{-1}B_0G_1$, the first column is untouched, and the newly created zeros remain intact. The second transformation $G_2 = \mathrm{diag}\{1, 1, \hat{G}_2\}$ creates the desired zeros in the second column of $G_2^{-1}B_1$. When we complete the similarity transformation $B_2 = G_2^{-1}B_1G_2$, the zeros remain intact. Continuing this way, we have, after $n - 2$ steps, an upper Hessenberg matrix

$$B = G_{n-2}^{-1} \cdots G_2^{-1}G_1^{-1}AG_1G_2 \cdots G_{n-2}.$$

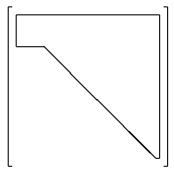
Chasing the Bulge. Let us assume that our matrix, now called A again, is in upper Hessenberg form. Then the iterations of the GR algorithm preserve that form, and the iterations can be performed implicitly by an efficient procedure known as bulge chasing. The key to effecting a GR iteration is to get the first column of the transforming matrix right. In the GR iteration (1) the transforming matrix G is determined by the equation $f(A) = GR$, where R is upper triangular. It follows that the first column of G is proportional to the first column of $f(A)$, so we just need the first column of $f(A)$ to get the iteration started. Since A is upper Hessenberg, $f(A)$ also has a simple form. Assuming f is a polynomial of degree m, then $f(A)$ will be m-Hessenberg; that is, it is zero below the mth subdiagonal. In particular, only the first $m + 1$ entries of the first column can be nonzero. The computation of the first column is simple, assuming $m \ll n$:

$$x = f(A)e_1 = (A - \mu_1 I)(A - \mu_2 I) \cdots (A - \mu_m I)e_1,$$

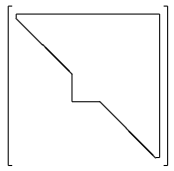
where e_1 is the vector with a one in the first position and zeros elsewhere. The computation of x can be effected as a series of m matrix-vector products, each of which is inexpensive and involves only a small submatrix in the upper left-hand corner of A. The computation of $(A - \mu_m I)e_1$ involves only the first column of $A - \mu_m I$, and the result has only two nonzero entries; the computation of $(A - \mu_{m-1}I)(A - \mu_m I)e_1$

involves only the first two columns of $A-\mu_{m-1}I$, and the result has only three nonzero entries, and so on. An alternative computation is given in [30].

Once we have the first column of $f(A)$, we initiate the implicit GR iteration by building a transformation $G_0 = \mathrm{diag}\{\hat{G}_0, I_{n-m-1}\}$ whose first column is proportional to the first column of $f(A)$. $\hat{G}_0$ can be a Householder or Gauss transform, for example. With G_0 in hand, we perform a similarity transformation $B_0 = G_0^{-1}AG_0$, which disturbs the upper Hessenberg form. The transformation $A \to G_0^{-1}A$ acts on the first $m+1$ rows, and the transformation $G_0^{-1}A \to B_0$ acts on the first $m+1$ columns. The result has the form

$B_0 =$  ,

where all entries outside of the outlined region are zero. There is a bulge in the Hessenberg form, extending down to row $m+2$. The remainder of the implicit GR iteration consists of returning the matrix to upper Hessenberg form by the method that we have already outlined above. Thus a G_1 is built such that the first column of $G_1^{-1}B_0$ is restored to upper Hessenberg form. Since only the entries $(3,1)$, ..., $(m+2,1)$ need to be set to zero, G_1 can be built so that G_1^{-1} acts on rows 2, ..., $m+2$ only. Then the transformation $G_1^{-1}B_0 \to B_1 = G_1^{-1}B_0G_1$ acts on columns 2, ..., $m+2$. It leaves the newly created zeros intact, but it adds a new row to the bulge. Therefore the bulge doesn't shrink, it gets moved. Each subsequent transformation removes one column from the bulge and adds a row, chasing the bulge down and to the right. In mid-chase we have

$B_k =$  .

Eventually the bulge is chased off the bottom of the matrix, and Hessenberg form is restored. This completes the implicit GR step. The new iterate is $\hat{A} = B_{n-2} = G^{-1}AG$, where $G = G_0G_1\cdots G_{n-2}$. Because of the form of G_1, ..., G_{n-2}, the first column of G is the same as the first column of G_0 and is therefore proportional to the first column of $f(A)$. Since the first column of G is right, we have effected a GR iteration. The general justification for this claim is given in [102]. In the QR case the usual justification invokes the implicit-Q theorem [49, 100].

5. Implicit Product *GR* Iterations.

Reduction to Condensed Form. Now consider a product $A = A_k\cdots A_1$. If we wish to effect product GR iterations efficiently, we must transform the factors A_i to a condensed form of some sort. Notice that if we make A_k upper Hessenberg and all other A_i upper triangular, then the product A will be upper Hessenberg. We will indicate how such a transformation can be effected [56] in the case $k=3$. Again it is

convenient to work with the cyclic matrix

$$C = \begin{bmatrix} & & A_{13} \\ A_{21} & & \\ & A_{32} & \end{bmatrix},$$

which we can write schematically as

$$\begin{bmatrix} & & & & & & & & x & x & x & x \\ & & & & & & & & x & x & x & x \\ & & & & & & & & x & x & x & x \\ & & & & & & & & x & x & x & x \\ z & z & z & z & & & & & & & & \\ z & z & z & z & & & & & & & & \\ z & z & z & z & & & & & & & & \\ z & z & z & z & & & & & & & & \\ & & & & y & y & y & y & & & & \\ & & & & y & y & y & y & & & & \\ & & & & y & y & y & y & & & & \\ & & & & y & y & y & y & & & & \end{bmatrix} \tag{4}$$

in the case where the factors are 4×4. The first step of the reduction applies an elimination matrix on the left to zero out the first column of A_{21} (the z-matrix) below the main diagonal. To complete a similarity transformation on C, we apply the inverse of the elimination matrix on the right. Since the left transformation acted only on the second block row of C, the inverse acts only on the second block column. That is, it affects the y-matrix only. The second step of the reduction applies an elimination matrix on the left to zero out the first column of the y-matrix below the main diagonal. Completing the similarity transformation on C, we apply the inverse transformation on the right. This affects only the x-matrix. The next step creates zeros in the first column of the x-matrix. But now we have to be cautious and zero out only the entries below the subdiagonal. That is, we apply a transformation on the left that operates on rows 2, 3, and 4 and creates zeros in positions $(3,1)$ and $(4,1)$ of the x-matrix. Now when we complete the similarity transformation we operate only on columns 2, 3, and 4. This affects only the z-matrix, and it does not disturb the zeros that were previously created there, because it does not touch the first column. At this point the matrix has the form

$$\begin{bmatrix} & & & & & & & & x & x & x & x \\ & & & & & & & & x & x & x & x \\ & & & & & & & & 0 & x & x & x \\ & & & & & & & & 0 & x & x & x \\ z & z & z & z & & & & & & & & \\ 0 & z & z & z & & & & & & & & \\ 0 & z & z & z & & & & & & & & \\ 0 & z & z & z & & & & & & & & \\ & & & & y & y & y & y & & & & \\ & & & & 0 & y & y & y & & & & \\ & & & & 0 & y & y & y & & & & \\ & & & & 0 & y & y & y & & & & \end{bmatrix}.$$

We now continue the reduction by making another round of the matrix, creating zeros in the second column of the z-matrix, the y-matrix, and the x-matrix in turn. At

each step our choice of how many zeros to create is just modest enough to guarantee that we do not destroy any of the zeros that we had created previously. After $n-1$ times around the matrix, we have reduced all of the blocks to triangular form, except for A_k, which is upper Hessenberg. The condensed C matrix looks like

(5)
$$\begin{bmatrix}
 & & & & & & & & x & x & x & x \\
 & & & & & & & & x & x & x & x \\
 & & & & & & & & & x & x & x \\
 & & & & & & & & & & x & x \\
z & z & z & z & & & & & & & & \\
 & z & z & z & & & & & & & & \\
 & & z & z & & & & & & & & \\
 & & & z & & & & & & & & \\
 & & & & y & y & y & y & & & & \\
 & & & & & y & y & y & & & & \\
 & & & & & & y & y & & & & \\
 & & & & & & & y & & & &
\end{bmatrix}$$

in our special case.

This procedure looks like a nice generalization of the standard reduction to Hessenberg form outlined in section 4, but Kressner [65] has shown that it is actually just an instance of the standard reduction. To see this, consider what happens when we do a perfect shuffle of the rows and columns of (4). That is, we reorder the rows and columns in the order 1, 5, 9, 2, 6, 10, 3, ... The result is

(6)
$$\left[\begin{array}{ccc|ccc|ccc|ccc}
 & & x & & & x & & & x & & & x \\
z & & & z & & & z & & & z & & \\
 & y & & & y & & & y & & & y & \\
\hline
 & & x & & & x & & & x & & & x \\
z & & & z & & & z & & & z & & \\
 & y & & & y & & & y & & & y & \\
\hline
 & & x & & & x & & & x & & & x \\
z & & & z & & & z & & & z & & \\
 & y & & & y & & & y & & & y & \\
\hline
 & & x & & & x & & & x & & & x \\
z & & & z & & & z & & & z & & \\
 & y & & & y & & & y & & & y &
\end{array}\right].$$

Now think about applying the standard reduction to upper Hessenberg form to this matrix. First we zero out the first column below the subdiagonal. Since there are only a few nonzeros in that column, the transformation needs only to act on rows 2, 5, 8, and 11, making zeros in positions $(5,1)$, $(8,1)$, and $(11,1)$. This transformation affects only the elements labeled z. The similarity transformation is completed by an operation on columns 2, 5, 8, and 11. This touches only elements labeled y. The second step is to zero out the second column below the subdiagonal. Since there are only a few nonzeros in this column, the transformation acts only on rows 3, 6, 9, and 12, and it touches only entries labeled y. The similarity transformation is completed by an operation on columns 3, 6, 9, and 12, which touches only entries labeled x. Continuing in this manner, we reduce the matrix to upper Hessenberg form, ending

with

$$
\left[\begin{array}{ccc|ccc|ccc|ccc}
 & & x & & & x & & & x & & & x \\
z & & & z & & & z & & & z & & \\
 & y & & & y & & & y & & & y & \\
\hline
 & & x & & & x & & & x & & & x \\
 & & & z & & & z & & & z & & \\
 & & & & y & & & y & & & y & \\
\hline
 & & & & & x & & & x & & & x \\
 & & & & & & z & & & z & & \\
 & & & & & & & y & & & y & \\
\hline
 & & & & & & & & x & & & x \\
 & & & & & & & & & z & & \\
 & & & & & & & & & & y &
\end{array}\right]. \tag{7}
$$

A moment's thought reveals that this is just the shuffled version of (5). Thus (5) is just a disguised upper Hessenberg matrix. Moreover, one easily checks that the operations in the reduction of (4) to (5) are identical to the operations in the reduction of the shuffled matrix (6) to the Hessenberg form (7). Thus the reduction of (4) to (5) is really just a very special instance of the standard reduction to Hessenberg form.

The Product *GR* Step. Once we have the upper Hessenberg matrix (7), we can perform implicit GR steps by bulge chasing. It turns out to be easier to think about the unshuffled version (5). We will continue to use the case $k = 3$ for illustration, although what we have to say is valid for any positive integer k. Now let

$$
C = \begin{bmatrix} & & H_{13} \\ T_{21} & & \\ & T_{32} & \end{bmatrix}
$$

denote the cyclic "Hessenberg" matrix (5). We know from section 3 that if we want to execute a GR iteration on C with a shift μ, then we should also apply shifts $\mu\omega$ and $\mu\omega^2$ ($\omega = e^{2\pi i/3}$) at the same time to preserve the cyclic structure. In other words, the polynomial that drives the GR iteration should have the form $p(C^3)$. The degree of p is the number of triples of shifts, and the degree of the GR iteration on C is three times the degree of p. To keep the discussion simple, let us suppose that we are going to apply just one triple of shifts: μ, $\mu\omega$, $\mu\omega^2$. Thus we will perform a GR iteration of degree 3 on C. As we observed in section 3, this is equivalent to a GR iteration of degree 1 with shift μ^3 applied to the product matrices $H_{13}T_{32}T_{21}$, $T_{21}H_{13}T_{32}$, and $T_{32}T_{21}H_{13}$ simultaneously. We have $p(C^3) = C^3 - \mu^3 I$.

From section 4 we know that the key to performing an implicit GR iteration is to get the first column of the transformation matrix right, and this is proportional to the first column of $p(C^3)$. However, the observations of section 4 apply to Hessenberg matrices, so we should really be working with the shuffled version (7) instead of (5). But the shuffled and unshuffled versions (of C, C^3, $p(C)$, etc.) have the same first column, up to a permutation of the entries, so it is okay to work with the unshuffled version. Recall that

$$
C^3 = \begin{bmatrix} H_{13}T_{32}T_{21} & & \\ & T_{21}H_{13}T_{32} & \\ & & T_{32}T_{21}H_{13} \end{bmatrix},
$$

so the first column of $p(C^3) = C^3 - \mu^3 I$ is essentially the same as the first column of the shifted product $H_{13}T_{32}T_{21} - \mu^3 I$. This column has only two nonzero entries, and it is trivial to compute, since the two T matrices are upper triangular. Proceeding as in section 4, we build a transforming matrix G_0 whose first column is proportional to the first column of $p(C)$. We apply G_0^{-1} to C on the left, affecting only the first two rows. When we apply G_0 on the right, it affects the first two columns, creating a bulge, denoted by a plus sign, in the "z" part of the matrix:

$$\begin{bmatrix}
 & & & & & & & & x & x & x & x \\
 & & & & & & & & x & x & x & x \\
 & & & & & & & & & x & x & x \\
 & & & & & & & & & & x & x \\
z & z & z & z & & & & & & & & \\
+ & z & z & z & & & & & & & & \\
 & & z & z & & & & & & & & \\
 & & & z & & & & & & & & \\
 & & & & y & y & y & y & & & & \\
 & & & & & y & y & y & & & & \\
 & & & & & & y & y & & & & \\
 & & & & & & & y & & & &
\end{bmatrix}.$$

The rest of the GR iteration consists of returning the matrix to the Hessenberg-triangular form (5) by the algorithm outlined earlier in this section. The first transformation acts on rows 5 and 6, returning the bulge entry to zero. When the similarity transformation is completed by applying a transformation to columns 5 and 6, a new bulge is created in the y-matrix. The next transformation acts on rows 9 and 10, annihilating this bulge. Completing the similarity transformation, we create a new bulge in the x-matrix. Next we apply a transformation to rows 2 and 3, annihilating the new bulge. When we complete the similarity transformation, the bulge shows up in the $(3, 2)$ position of the x-matrix. We have now chased the bulge once around the cycle, and it has ended up down and over one position from where it began. Continuing this process, we chase the bulge around and around the matrix until it falls off the bottom.

We have described a cyclic GR step of degree 3. Iterations of degree 6, 9, or higher generally follow the same pattern, except that the bulge is bigger. This is called the periodic GR algorithm [56]. It appears to be a generalization of the standard GR algorithm, but the observation of Kressner [65] shows that it is really just a special case. The reader is invited to check how these operations look in the shuffled coordinates. One easily sees that the periodic GR algorithm is just an ordinary (though very special) bulge chase of degree 3 on the upper Hessenberg matrix (7).

6. The Case $k = 2$. We have used the case $k = 3$ for illustration. In some applications k is 3 or larger, but in most of our illustrations we will have $k = 2$. Therefore we pause to consider what the matrices look like in this case. We have $A = A_2A_1$. The corresponding cyclic matrix is

$$C = \begin{bmatrix} & A_2 \\ A_1 & \end{bmatrix},$$

which looks like

$$\left[\begin{array}{ccc|ccc} & & & x & x & x \\ & & & x & x & x \\ & & & x & x & x \\ \hline y & y & y & & & \\ y & y & y & & & \\ y & y & y & & & \end{array}\right].$$

The Hessenberg form is

$$\left[\begin{array}{ccc|ccc} & & & x & x & x \\ & & & x & x & x \\ & & & & x & x \\ \hline y & y & y & & & \\ & y & y & & & \\ & & y & & & \end{array}\right].$$

The shuffled versions are

$$\left[\begin{array}{ccc|ccc|ccc} & x & & & x & & & x & \\ y & & & y & & & y & & \\ \hline & x & & & x & & & x & \\ y & & & y & & & y & & \\ \hline & x & & & x & & & x & \\ y & & & y & & & y & & \end{array}\right] \quad \text{and} \quad \left[\begin{array}{ccc|ccc|ccc} & x & & & x & & & x & \\ y & & & y & & & y & & \\ \hline & x & & & x & & & x & \\ & & & y & & & y & & \\ \hline & & & & x & & & x & \\ & & & & & & y & & \end{array}\right].$$

In this case the GR iterations are always of even degree, with the shifts occurring in $\pm\mu$ pairs. A typical driving polynomial looks like

$$f(C) = p(C^2) = (C^2 - \mu_1^2 I)(C^2 - \mu_2^2 I)\cdots(C^2 - \mu_m^2 I).$$

7. The Generalized Eigenvalue Problem. We begin our discussion of special cases by considering the generalized eigenvalue problem $(A - \lambda B)v = 0$. This is equivalent to the eigenvalue problem for $B^{-1}A$ or AB^{-1}. The corresponding cyclic matrix is

$$C = \begin{bmatrix} & A \\ B^{-1} & \end{bmatrix}.$$

Operations on C entail operations on B^{-1}, but, as we have already said, we need not work with B^{-1} explicitly. If we need to effect, say, the transformation $G_2^{-1}B^{-1}G_1$, we will instead perform the equivalent transformation $G_1^{-1}BG_2$.

The presence of the inverse changes the details of the reduction to Hessenberg-triangular form. The basic idea is to triangularize B first (implying that B^{-1} is then also triangular), then gradually reduce A to upper Hessenberg form while defending the triangular form of B. For details see [76], [49], or [100], for example.

Once the condensed form has been reached, GR iterations can be effected by bulge chasing. Chasing a bulge in B^{-1} is equivalent to chasing a bulge in B, since B has a bulge of a certain size in a certain location if and only if B^{-1} has a bulge of the same size in the same location. If we run the double (resp., quadruple) shift QR algorithm on the cyclic matrix (working with B instead of B^{-1}), we get exactly the single (resp., double) shift QZ algorithm of Moler and Stewart [76].

A nice feature of this algorithm is that it works perfectly well even if B^{-1} does not exist. In this case there are infinite eigenvalues. The analyses in [95] and [99] show that the infinite eigenvalues emerge rapidly at the top of B and can be deflated out. Meanwhile the presence of infinite eigenvalues does not interfere with the functioning of the algorithm. An alternative is to deflate out the infinite eigenvalues before the QZ iterations are begun [49, section 7.7.5], [99].

Van Loan's Extension. In 1975 Van Loan [92] published the VZ algorithm, which computes the eigenvalues of a matrix of the form $AB(CD)^{-1}$. In the case when $B = C = D = I$, it reduces to the QR algorithm on the single matrix A. When $B = D = I$, it reduces to the QZ algorithm on the pencil $A - \lambda C$. When $C = D = I$ and $A = B^T$, it reduces to the Golub–Kahan QR algorithm for the SVD. When $A = B^T$ and $C = D^T$, it yields an algorithm for computing eigenvalues of pencils of the form $B^TB - \lambda D^TD$.

If we apply the QR iterations of degree 4 or 8 to the cyclic matrix

$$\begin{bmatrix} & & & A \\ C^{-1} & & & \\ & D^{-1} & & \\ & & B & \end{bmatrix},$$

we obtain the VZ algorithm. It is understood that the inverse matrices are handled as outlined for the generalized eigenvalue problem above.

Periodic Control Systems. Hench and Laub [56] and Bojanczyk, Golub, and Van Dooren [18] studied discrete-time periodic control systems that lead to eigenvalue problems of the form

$$E_k^{-1}F_k \cdots E_2^{-1}F_2E_1^{-1}F_1$$

and developed the periodic QZ algorithm for solving them. The product is a formal product only, as many of the E_i are normally singular matrices. Fortunately the periodic QZ algorithm functions just fine even when some of the E_i are not invertible. Kressner [63] studied carefully the implementational details of the periodic QZ algorithm.

The periodic QZ algorithm also plays a role in the stable numerical solution of Hamiltonian eigenvalue problems arising from continuous-time optimal control problems [12, 13].

8. The Singular Value Decomposition. Suppose we wish to compute the SVD of a matrix $B \in \mathbb{R}^{n\times m}$. This is equivalent to computing the eigensystems of B^TB and BB^T or the cyclic matrix

$$\begin{bmatrix} & B^T \\ B & \end{bmatrix}. \tag{8}$$

The Golub and Kahan computation [46, 47] begins by reducing B to bidiagonal form. The result of Kressner [65] implies that this is equivalent to reducing the shuffled version of (8) to upper Hessenberg form. If

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix},$$

then the shuffled version of (8) is

$$\left[\begin{array}{cc|cc|c} 0 & b_{11} & 0 & b_{21} & b_{31} \\ b_{11} & 0 & b_{12} & 0 & 0 \\ \hline 0 & b_{12} & 0 & b_{22} & b_{32} \\ b_{21} & 0 & b_{22} & 0 & 0 \\ \hline b_{31} & 0 & b_{32} & 0 & 0 \end{array}\right].$$

The shuffle is not quite perfect, because there is one row of B (resp., column of B^T) left over, but this does not matter. After the reduction (using orthogonal transformations only), the matrix is actually tridiagonal due to symmetry:

$$\left[\begin{array}{cc|cc|c} 0 & \alpha_1 & 0 & 0 & 0 \\ \alpha_1 & 0 & \beta_1 & 0 & 0 \\ \hline 0 & \beta_1 & 0 & \alpha_2 & 0 \\ 0 & 0 & \alpha_2 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \end{array}\right].$$

This is the shuffled version of

$$\left[\begin{array}{cc|ccc} & & \alpha_1 & 0 & 0 \\ & & \beta_1 & \alpha_2 & 0 \\ \hline \alpha_1 & \beta_1 & & & \\ 0 & \alpha_2 & & & \\ 0 & 0 & & & \end{array}\right]. \tag{9}$$

Once the reduction to bidiagonal form has been achieved, we might as well assume B is square, since its nontrivial part is square. There is a block of zeros that can be disregarded, either below or to the right of the diagonal part. For example, if B is 3×2, we can get rid of the bottom row of zeros. This is the same as dropping the last row and column of (9).

Let us now assume that B is square and bidiagonal. In the 3×3 case we have

$$B = \left[\begin{array}{ccc} \alpha_1 & \beta_1 & \\ & \alpha_2 & \beta_2 \\ & & \alpha_3 \end{array}\right].$$

We can assume without loss of generality that all of the α_i and β_i are nonnegative. If any of them are zero, we can immediately do a deflation [49]. We will assume, therefore, that all of the α_i and β_i are positive. If we embed B in the cyclic matrix (8) and shuffle the rows and columns, we obtain the irreducible, symmetric, tridiagonal matrix

$$\left[\begin{array}{ccc|ccc|ccc} & \alpha_1 & & & & & & & \\ \alpha_1 & & \beta_1 & & & & & & \\ \hline & \beta_1 & & & \alpha_2 & & & & \\ & & \alpha_2 & & & \beta_2 & & & \\ \hline & & & & \beta_2 & & & \alpha_3 & \\ & & & & & \alpha_3 & & & \end{array}\right]. \tag{10}$$

If we apply the double-shift QR algorithm to this matrix with paired shifts $\pm\mu$, we are doing exactly the Golub–Kahan QR algorithm for the SVD. In his 1968 paper [45] Golub stopped just short of making this observation.

Accurate Computation of Singular Values. Demmel and Kahan [27] made an important improvement to the algorithm. First they made the observation (which was already implicit in Kahan's 1966 technical report [60]) that the entries of the bidiagonal B determine the singular values to high relative accuracy. This means that if we make a small relative perturbation in each of the α_i and β_i in B, the relative perturbation in each of the singular values is comparably small. Thus if we change each entry of B in the fifteenth decimal place, this will cause a change in each singular value in about the fifteenth decimal place. This is true even for very tiny singular values. For example, if a singular value has magnitude about 10^{-60}, the magnitude of the perturbation will be about 10^{-75}. Thus there is the possibility of computing all singular values, even the very tiny ones, to high relative accuracy. Demmel and Kahan showed how to do this in practice: In the Golub–Kahan QR algorithm with a zero shift, the arithmetic can be reorganized so that the entire step is performed without any subtractions. All additions in the step involve only positive numbers. Since there is no possibility of cancellation, all quantities are computed to high relative accuracy. Therefore, the zero-shift QR algorithm can be used to extract the tiniest singular values to high relative accuracy. This will take only a few iterations. The convergence of the QR algorithm depends upon ratios of singular values σ_{j+1}/σ_j. If $\sigma_{j+1}, \ldots, \sigma_n$ are tiny and σ_j is not, the ratio σ_{j+1}/σ_j will be favorable. Once the tiny singular values have been extracted, the others can be found by the standard shifted algorithm.

We must state an important caveat here. This algorithm can compute singular values of a bidiagonal matrix to high relative accuracy, but it cannot do the same for a general matrix. If the matrix must first be reduced to bidiagonal form, the roundoff errors in the preliminary reduction will normally destroy the accuracy of the tiny singular values. The following MATLAB experiment confirms this:

```
format
B = toeplitz([1 0 0],[1 1234567 0])
format long e
Singular_Values = svd(B)
SV_product = prod(Singular_Values)
```

This produces a 3×3 bidiagonal matrix and computes its singular values. Here is the output:

```
B =

           1     1234567           0
           0           1     1234567
           0           0           1

Singular_Values =

     1.234567500000506e+06
     1.234566500000506e+06
     6.561009579066186e-13

SV_product =

     1.000000000000000e+00
```

Two of the singular values are large, and one is tiny. MATLAB uses the Demmel–Kahan variant of the Golub–Kahan QR algorithm, so it computes all of the singular values to high relative accuracy. This is confirmed by computing the product of the singular values, which equals the determinant of B. MATLAB gets the correct value 1 to full precision. This would not have happened if the tiny singular value had not been computed to high relative accuracy. Now look at what happens when MATLAB computes the singular values of B^T:

```
Singular_Values_2 = svd(B')
SV_product_2 = prod(Singular_Values_2)
```

MATLAB does not notice that B^T is the transpose of an upper bidiagonal matrix. Instead it simply applies the reduction algorithm to transform it to upper bidiagonal form. Then it applies the Demmel–Kahan procedure. Here is the output:

```
Singular_Values_2 =

     1.234567500000506e+06
     1.234566500000506e+06
     6.561418075532522e-13

SV_product_2 =

     1.000062261220840e+00
```

The large singular values are still computed to full accuracy, but the tiny one is not. It is correct to only about four decimal places. The product of the computed singular values is correspondingly correct to only about four places. This is a mild example. It is easy to build more extreme examples in which the tiny singular value is computed as 0. (Just make the matrix bigger.)

Zero-Diagonal Symmetric Problem. The tridiagonal matrix (10) is special; all of its main-diagonal entries are zero. Matrices of this type arise in certain applications, for example, the computation of points for Gaussian quadrature formulas [41, 42, 43, 44, 50]. Given a positive weight function $w(x)$ on an interval $[a, b]$, we seek a formula

$$\sum_{i=1}^{m} w_i f(t_i) \approx \int_a^b f(t)w(t)\,dt$$

for approximating integrals weighted by w. It is a familiar fact that the optimal sample points $t_1, \ldots, t_m$ for an m-point formula are the zeros of the mth orthogonal polynomial ψ_m with respect to the weighted inner product

$$\langle f, g\rangle = \int_a^b f(t)g(t)w(t)\,dt.$$

It is perhaps not so well known that $t_1, \ldots, t_m$ are also the eigenvalues of a certain symmetric, tridiagonal matrix that can be built from coefficients generated in the process of constructing the orthogonal polynomials [50]. Thus the quadrature points can be obtained by solving a symmetric, tridiagonal eigenvalue problem. The main diagonal entries of the tridiagonal matrix are of the form

$$\int_a^b t\psi_i(t)^2 w(t)\,dt, \qquad i = 0,\ 1, \ldots,\ m. \tag{11}$$

If the interval $[a, b]$ is symmetric ($a = -b$) and the weight function is even, then each of the integrals in (11) is zero. Thus the tridiagonal matrix has zeros on its main diagonal; it has the form exemplified by (10). Deshuffling the matrix, we find that its eigenvalue problem is equivalent to the singular value problem for a bidiagonal matrix B. For each singular value σ, the special tridiagonal matrix has eigenvalues $\pm\sigma$.

Obviously this observation can be extended to symmetric, tridiagonal matrices with constant main diagonal τ. A shift by τ puts us in the zero-diagonal case [96].

The Skew-Symmetric Eigenvalue Problem. The skew-symmetric eigenvalue problem was analyzed by Ward and Gray [96]. If we wish to compute the eigenvalues of a skew-symmetric matrix $A \in \mathbb{R}^{n\times n}$, we can begin by reducing it to upper Hessenberg form. The principle of preservation of structure dictates that we should use orthogonal transformation matrices to preserve skew symmetry. Thus the upper Hessenberg form is actually tridiagonal. Moreover, the main-diagonal entries are zero. In the 6×6 case, the matrix looks like this:

$$\left[\begin{array}{cc|cc|cc} & \alpha_1 & & & & \\ -\alpha_1 & & -\beta_1 & & & \\ \hline & \beta_1 & & \alpha_2 & & \\ & & -\alpha_2 & & -\beta_2 & \\ \hline & & & \beta_2 & & \alpha_3 \\ & & & & -\alpha_3 & \end{array}\right]. \tag{12}$$

We have labeled the entries with a bit of foresight and a peek at (10). Deshuffling this matrix, we obtain

$$\left[\begin{array}{ccc|ccc} & & & \alpha_1 & & \\ & & & \beta_1 & \alpha_2 & \\ & & & & \beta_2 & \alpha_3 \\ \hline -\alpha_1 & -\beta_1 & & & & \\ & -\alpha_2 & -\beta_2 & & & \\ & & -\alpha_3 & & & \end{array}\right],$$

and it becomes clear that the skew-symmetric tridiagonal eigenvalue problem is equivalent to the SVD problem for a bidiagonal matrix B. If we apply the double-shift QR algorithm to (12) with paired shifts $\pm i\mu$, we are doing exactly the Golub–Kahan QR algorithm with shift μ on B. For each singular value σ of B, the tridiagonal matrix has a complex conjugate pair of eigenvalues $\pm i\sigma$.

The SVD of a Product. Before leaving the SVD, we take a slight digression. Golub, Sølna, and Van Dooren [48] considered the product SVD problem: Find the SVD of $B = B_k B_{k-1} \cdots B_1$, given the factors $B_1, \ldots, B_k$.[4] This is not as straightforward as it might at first seem. To understand the difficulties, consider the case of a product of two matrices $B = B_2 B_1$. The SVD problem for $B_2 B_1$ is equivalent to the eigenvalue problems for the products $B_1^T B_2^T B_2 B_1$ and $B_2 B_1 B_1^T B_2^T$, so it would seem that we should build an algorithm based on the cyclic matrix

$$\begin{bmatrix} & & & B_1^T \\ B_1 & & & \\ & B_2 & & \\ & & B_2^T & \end{bmatrix}.$$

[4] For some of the factors, we might be given B_i^{-1} instead of B_i.

If we reduce this matrix to the Hessenberg-triangular form by the algorithm described in section 5 (using orthogonal matrices, as this is an SVD problem), we lose some of the important structure. We obtain

$$\begin{bmatrix} & & & H_4 \\ T_1 & & & \\ & T_2 & & \\ & & T_3 & \end{bmatrix},$$

where $H_4 \neq T_1^T$ and $T_3 \neq T_2^T$. The algorithm does not realize that certain of the blocks are related and that these relationships should be maintained. What is needed is a reduction to a form

$$\begin{bmatrix} & & & \hat{B}_1^T \\ \hat{B}_1 & & & \\ & \hat{B}_2 & & \\ & & \hat{B}_2^T & \end{bmatrix},$$

in which, say, $\hat{B}_1$ is bidiagonal and $\hat{B}_2$ is diagonal, making $\hat{B} = \hat{B}_2\hat{B}_1$ also bidiagonal. Such a reduction may be hard to come by.

Golub, Sølna, and Van Dooren took a different approach. They transformed $B = B_k \cdots B_1$ by an orthogonal equivalence transformation to a form $\hat{B} = \hat{B}_k \cdots \hat{B}_1$, in which each of the factors $\hat{B}_i$ is upper triangular, none of the $\hat{B}_i$ is bidiagonal, but the product $\hat{B}$ *is* bidiagonal. In floating-point arithmetic the computed $\hat{B}$ will not be bidiagonal. However, one can compute its main diagonal and superdiagonal and ignore the rest. The method works better than expected. See [48] for details.

9. The Symmetric Eigenvalue Problem. Given a symmetric, tridiagonal matrix A, we can make it positive definite by applying a sufficiently positive shift. Let us suppose that A is symmetric, tridiagonal, and positive definite:

$$A = \begin{bmatrix} \gamma_1 & \delta_1 & & & & \\ \delta_1 & \gamma_2 & \delta_2 & & & \\ & \delta_2 & \gamma_3 & \ddots & & \\ & & \ddots & \ddots & \delta_{n-1} & \\ & & & \delta_{n-1} & \gamma_n & \end{bmatrix}.$$

Without loss of generality we can assume that all of the δ_j are nonzero. In fact, we can even assume that they are positive, since they can be made positive by a diagonal orthogonal similarity transformation (entries on main diagonal are ± 1). The γ_j are positive as well.

In the Cholesky decomposition $A = B^T B$, B is bidiagonal,

$$B = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ & \alpha_2 & \beta_2 & & & \\ & & \alpha_3 & \ddots & & \\ & & & \ddots & \beta_{n-1} & \\ & & & & \alpha_n & \end{bmatrix}, \tag{13}$$

and the α_j and β_j are all positive. Clearly these $2n-1$ parameters encode the same information as the γ_j and δ_j do. However, in the realm of uncertain data and floating-point arithmetic, the entries of B somehow do a better job. The result of Demmel

and Kahan [27] mentioned previously guarantees that the entries of B determine the singular values of B, and hence the eigenvalues of A, to high relative accuracy. In contrast, the entries of A do not, as is demonstrated by the simple example

$$A = \begin{bmatrix} 1+\epsilon & 1 \\ 1 & 1+\epsilon \end{bmatrix},$$

where ϵ is tiny. The eigenvalues of A are $\lambda_1 = 2+\epsilon$ and $\lambda_2 = \epsilon$. If we double ϵ, we double the tiny eigenvalue λ_2. Doubling ϵ amounts to relatively small perturbations of $\gamma_1 = \gamma_2 = 1+\epsilon$. This causes a relatively huge (100%) change in λ_2.

Early in this paper we stated that if we want the eigenvalues of B^TB, we should definitely work with B itself, rather than computing the product B^TB explicitly, since information is lost when B^TB is computed in floating-point arithmetic. The result of Demmel and Kahan gives a more precise view of the situation. If B is given, we should definitely compute the singular values of B and infer the eigenvalues of A. If A is given, there can be no harm in computing B and working with B instead of A. If we work with A directly, there is no hope of computing its tiny eigenvalues (if there are any) to high relative accuracy.

Quotient-Difference (qd) Algorithm. Another way to deal with a positive-definite tridiagonal matrix is to perform a diagonal similarity transformation, deliberately breaking the symmetry, to transform A to

$$T = D^{-1}AD = \begin{bmatrix} \gamma_1 & 1 & & & \\ \delta_1^2 & \gamma_2 & 1 & & \\ & \delta_2^2 & \gamma_3 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & \delta_{n-1}^2 & \gamma_n \end{bmatrix}. \tag{14}$$

This matrix has a decomposition $T = LR$, where L is unit lower (or left) triangular and R is upper (or right) triangular:

$$L = \begin{bmatrix} 1 & & & & \\ l_1 & 1 & & & \\ & l_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & l_{n-1} & 1 \end{bmatrix}, \qquad R = \begin{bmatrix} r_1 & 1 & & & \\ & r_2 & 1 & & \\ & & r_3 & \ddots & \\ & & & \ddots & 1 \\ & & & & r_n \end{bmatrix}.$$

This is what is commonly called an LU decomposition. In the context of eigenvalue computations, the symbol R is often used instead of U. Once again we have a set of $2n-1$ parameters encoding the matrix.

Obviously the decomposition $T = LR$ must be closely related to the Cholesky decomposition $A = B^TB$. Indeed, if one looks for the relationship, one easily finds that $l_j = \beta_j^2$ for $j = 1, \ldots, n-1$, and $r_j = \alpha_j^2$ for $j = 1, \ldots, n$, where the α_j and β_j are as in (13). It follows that the l_j and r_j parameters determine the eigenvalues of T and A to high relative accuracy, just as the α_j and β_j do. It may therefore be possible to produce high-accuracy algorithms that operate on the entries of L and R.

Let us take the L and R and place them in the cyclic matrix

$$C = \begin{bmatrix} & L \\ R & \end{bmatrix}.$$

For illustration, the 4×4 case looks like this:

$$C = \left[\begin{array}{cccc|cccc} & & & & 1 & & & \\ & & & & l_1 & 1 & & \\ & & & & & l_2 & 1 & \\ & & & & & & l_3 & 1 \\ \hline r_1 & 1 & & & & & & \\ & r_2 & 1 & & & & & \\ & & r_3 & 1 & & & & \\ & & & r_4 & & & & \end{array}\right].$$

The shuffled version is

$$\left[\begin{array}{cc|cc|cc|cc} 0 & 1 & & & & & & \\ r_1 & 0 & 1 & & & & & \\ \hline & l_1 & 0 & 1 & & & & \\ & & r_2 & 0 & 1 & & & \\ \hline & & & l_2 & 0 & 1 & & \\ & & & & r_3 & 0 & 1 & \\ \hline & & & & & l_3 & 0 & 1 \\ & & & & & & r_4 & 0 \end{array}\right].$$

This special form is preserved by the LR algorithm. Of course, the degree of the LR iterations has to be even, with shifts in pairs $\pm\mu$. Let us consider a step of degree 2 with shifts ± 0. We will work with the unshuffled form. The iteration is set in motion by a transformation whose first column is proportional to the first column of

$$C^2 - 0^2 I = \left[\begin{array}{cc} LR & \\ & RL \end{array}\right],$$

which is $r_1 \left[\begin{array}{ccccc} 1 & l_1 & 0 & \cdots & 0 \end{array}\right]^T$. The LR algorithm performs similarity transformations $\tilde{C} \to \tilde{L}^{-1}\tilde{C}\tilde{L}$, where $\tilde{L}$ is a unit lower triangular matrix built up from Gaussian elimination transforms (with no pivoting). Thus our first transform will have the form

$$\tilde{L} = \left[\begin{array}{cc|c} 1 & & \\ l_1 & 1 & \\ \hline & & I_{n-2} \end{array}\right], \qquad \tilde{L}^{-1} = \left[\begin{array}{cc|c} 1 & & \\ -l_1 & 1 & \\ \hline & & I_{n-2} \end{array}\right].$$

The transformation $C \to \tilde{L}^{-1}C$ subtracts l_1 times the first row from the second row. The sole effect of this is to transform the entry l_1 to 0 in C. This zero is important; it is the consequence of using a zero shift. The transformation $\tilde{L}^{-1}C \to \tilde{L}^{-1}C\tilde{L}$, completing the similarity transformation, adds l_1 times the second column to the first column, creating a bulge to the left of r_2. In the case $n = 4$ the result looks like this:

$$\left[\begin{array}{cccc|cccc} & & & & 1 & & & \\ & & & & 0 & 1 & & \\ & & & & & l_2 & 1 & \\ & & & & & & l_3 & 1 \\ \hline \hat{r}_1 & 1 & & & & & & \\ b & r_2 & 1 & & & & & \\ & & r_3 & 1 & & & & \\ & & & r_4 & & & & \end{array}\right],$$

where $\hat{r}_1 = r_1 + l_1$, and $b = r_2 l_1$.

We pause to note the general rule of Gaussian elimination similarity transformations: If the left transformation subtracts m times row i from row j, then the right transformation (completing the similarity transformation) adds m times column j to column i.

The rest of the LR iteration consists of chasing the bulge from the matrix. Accordingly, the second transformation subtracts $b/\hat{r}_1$ times row $n+1$ from row $n+2$ to annihilate the bulge. It will soon become clear that a good name for this multiplier is $\hat{l}_1$. Thus we are subtracting $\hat{l}_1 = b/\hat{r}_1$ times row $n+1$ from row $n+2$. We then complete the similarity transformation by adding $\hat{l}_1$ times column $n+2$ to column $n+1$ to yield

$$\left[\begin{array}{ccccc|ccc} & & & & & 1 & & \\ & & & & & \hat{l}_1 & 1 & \\ & & & & & l_2\hat{l}_1 & l_2 & 1 \\ & & & & & & & l_3 \\ \hline \hat{r}_1 & 1 & & & & & & \\ 0 & \tilde{r}_2 & 1 & & & & & \\ & & r_3 & 1 & & & & \\ & & & r_4 & & & & \end{array}\right],$$

where $\tilde{r}_2 = r_2 - \hat{l}_1$. The entry $l_2\hat{l}_1$ is the new bulge. The subtraction in the formula for $\tilde{r}_2$ is unwelcome. Fortunately there is a second formula that requires only multiplication and division:

$$\tilde{r}_2 = r_2 - \hat{l}_1 = r_2 - \frac{r_2 l_1}{r_1 + l_1} = \frac{r_2 r_1}{\hat{r_1}}.$$

To chase the new bulge forward we must subtract the appropriate multiple of row 2 from row 3. Clearly the appropriate multiple is l_2. It is important that the submatrix

$$\left[\begin{array}{cc} \hat{l}_1 & 1 \\ l_2\hat{l}_1 & l_2 \end{array}\right]$$

has rank 1. Therefore, when we subtract l_2 times the second row from the third row, we zero out both the bulge and the entry l_2. We complete the similarity transformation by adding l_2 times column 3 to column 2. The result is

$$\left[\begin{array}{ccccc|ccc} & & & & & 1 & & \\ & & & & & \hat{l}_1 & 1 & \\ & & & & & 0 & 0 & 1 \\ & & & & & & & l_3 \\ \hline \hat{r}_1 & 1 & & & & & & \\ 0 & \hat{r}_2 & 1 & & & & & \\ & r_3 l_2 & r_3 & 1 & & & & \\ & & & r_4 & & & & \end{array}\right],$$

where $\hat{r}_2 = \tilde{r}_2 + l_2$. The next step works out just the same as the previous step: We subtract $\hat{l}_2 = r_3 l_2/\hat{r}_2$ times row $n+2$ from row $n+3$ to chase the bulge, and we add $\hat{l}_2$ times column $n+3$ to column $n+2$ to create a new bulge $l_3\hat{l}_2$ to the left of l_3. The step creates an intermediate quantity $\tilde{r}_3 = r_3 - \hat{l}_2$, which fortunately can be computed by the alternate formula $\tilde{r}_3 = r_3\tilde{r}_2/\hat{r}_2$. Also, the 2×2 submatrix in the

vicinity of the new bulge has rank 1, which guarantees that the transformation that chases the bulge further will also set the entry l_3 to zero. The algorithm continues in this fashion until the bulge has been chased off the bottom. The complete algorithm is quite succinct:

$$
\begin{array}{l}
\tilde{r}_1 \leftarrow r_1 \\
\text{for } j = 1, \ldots, n-1 \\
\quad \left[\begin{array}{l} \hat{r}_j \leftarrow \tilde{r}_j + l_j \\ \hat{l}_j \leftarrow l_j (r_{j+1}/\hat{r}_j) \\ \tilde{r}_{j+1} \leftarrow \tilde{r}_j (r_{j+1}/\hat{r}_j) \end{array} \right. \\
\hat{r}_n \leftarrow \tilde{r}_n
\end{array} \tag{15}
$$

As it turns out, this is not a new algorithm; it is the differential form of the quotient-difference (qd) algorithm of Rutishauser. This observation is certainly of historical interest, as the qd algorithm was the first GR algorithm ever [83, 84, 85]. The differential form of the algorithm appeared in [86], which was published posthumously. The algorithm is not just of historical interest. Inspecting (15), we see that all of the operations are either multiplications or divisions, except for one addition in the loop. This is a sum of two positive numbers; all of the quantities appearing in (15) are positive. Therefore, there is no possibility of cancellation, and all quantities are computed to high relative accuracy. Consequently the algorithm is able to compute eigenvalues to high relative accuracy.

Fernando and Parlett [34, 80] resurrected the differential qd algorithm and advocated its use. In related work, Parlett and Dhillon [28, 81] have shown how to compute eigenvectors efficiently and accurately. The name *holy grail* has been attached to this work. Although the name strikes this author as grandiose, these algorithms do appear to have a bright future.

Anyone who looks at the derivation of the zero-shift QR algorithm of Demmel and Kahan [27] will be struck by its resemblance to the differential qd algorithm, as presented here. Indeed, the two algorithms are closely related. An old result [105, p. 545] implies that one zero-shift QR step is equivalent to two zero-shift qd steps.

An advantage of the qd algorithm is that shifts can be introduced easily. To shift by μ, make the following two changes to (15): Replace the lines $\tilde{r}_1 \leftarrow r_1$ and $\tilde{r}_{j+1} \leftarrow \tilde{r}_j (r_{j+1}/\hat{r}_j)$ by $\tilde{r}_1 \leftarrow r_1 - \mu$ and $\tilde{r}_{j+1} \leftarrow \tilde{r}_j (r_{j+1}/\hat{r}_j) - \mu$, respectively. See [80] or [100, section 6.6]. As long as μ is smaller than the smallest eigenvalue (and here, if some eigenvalues have already been deflated, we do not need to consider them), high relative accuracy is guaranteed.

Since the L and R matrices are just rescalings of the matrix B of (13), it is clear that the differential qd algorithm can also be used for SVD calculations. The algorithm can also be used in the nonpositive case (i.e., shift larger than the smallest eigenvalue), and it can even be applied to nonsymmetrizable tridiagonal matrices ((14) with some of the δ_k^2 replaced by negative numbers), but in either case, negative values of r_k and l_k enter the computation, and the accuracy guarantee is lost. This does not necessarily imply that the algorithm will perform badly.

10. Totally Nonnegative Matrices. A matrix $A \in \mathbb{R}^{n \times n}$ is called *totally positive* (TP) if all of its minors of all sizes are positive. In other words, for any k, given any k rows and k columns of A, the corresponding $k \times k$ submatrix has a positive determinant. A matrix is *totally nonnegative* (TN) if all of its minors are nonnegative. TN and TP matrices arise in a variety of situations [23, 39, 61]. For example, the Vandermonde matrix of a positive increasing sequence is TP.

Every TN matrix has a decomposition into a product of bidiagonal matrices, the entries of which parametrize the family of TN matrices. We will not describe this decomposition in detail. In the 3×3 case it has the form

$$\begin{bmatrix} 1 & & \\ & 1 & \\ & a_2 & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ b_1 & 1 & \\ & b_2 & 1 \end{bmatrix} \begin{bmatrix} c_1 & & \\ & c_2 & \\ & & c_3 \end{bmatrix} \begin{bmatrix} 1 & d_1 & \\ & 1 & d_2 \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & e_2 \\ & & 1 \end{bmatrix}.$$

In the parametrization of a TP matrix all of the parameters $a_2, b_1, \ldots, e_2$ are positive, but in a TN matrix certain of them can be zero. An interesting example is the TP *Pascal matrix* [21, 31]

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{bmatrix},$$

which has the bidiagonal decomposition

$$\begin{bmatrix} 1 & & \\ & 1 & \\ & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ 1 & 1 & \\ & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & \\ & 1 & 1 \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & 1 \\ & & 1 \end{bmatrix}.$$

The bidiagonal decomposition has its origins in the papers [104] and [73]. See also [32, 36, 40].

Koev [62] has shown recently that the parameters of the bidiagonal factorization determine the eigenvalues and singular values of a nonsingular TN matrix to high relative accuracy. Koev has also developed algorithms that compute the eigenvalues and singular values to high relative accuracy, given that the matrix is presented in factored form. This is another example of a product eigenvalue problem. We want the eigenvalues (or singular values) of the product of the bidiagonal factors, but we work with the factors themselves, never forming the product. Koev's algorithm for the eigenvalues performs a similarity transformation to tridiagonal form by a sequence of Neville eliminations. These are Gaussian eliminations in which a multiple of one row is subtracted from the row immediately below it to create a zero in that row. The key to high accuracy is this: the Neville elimination in the TN matrix is equivalent to setting one parameter to zero in the factored form. Thus the elimination incurs no error (and it is practically free). The similarity transformation is completed by adding a multiple of one column to an adjacent column. This is not free, but it can be done with high relative accuracy while working with the factored form. See [62] for details. Once the matrix is in (factored) tridiagonal form, it can be symmetrized, and its eigenvalues can be computed to high relative accuracy by either of the two methods discussed in the previous section, the zero-shift QR algorithm [27] or the differential qd algorithm [34, 80, 86].

This is the first example of a class of (mostly) nonsymmetric matrices whose eigenvalues can be determined to high relative accuracy.

11. Pseudosymmetric Matrices. A real, tridiagonal matrix

$$A = \begin{bmatrix} \gamma_1 & \delta_1 & & & \\ \beta_1 & \gamma_2 & \delta_2 & & \\ & \beta_2 & \gamma_3 & \ddots & \\ & & \ddots & \ddots & \delta_{n-1} \\ & & & \beta_{n-1} & \gamma_n \end{bmatrix}$$

is called *pseudosymmetric* if $\beta_j = \pm\delta_j$ for $j = 1, \ldots, n-1$. Almost every real $n \times n$ matrix can be reduced by a similarity transformation to pseudosymmetric tridiagonal form in finitely many steps [11, 105]. However, the transformation cannot always be done stably. If the nonsymmetric Lanczos process [69] is applied in the usual way, it generates a pseudosymmetric tridiagonal matrix whose eigenvalues must then be computed. As we shall see, Hamiltonian and symplectic eigenvalue problems can be reduced to pseudosymmetric eigenvalue problems in a natural way.

Our objective is always to compute eigenvalues as accurately as possible. The algorithms to be discussed in this and the next two sections do a good job in the vast majority of cases, but there are no a priori stability or accuracy guarantees. There are, however, ways of checking a posteriori whether or not the results are any good.

One sees easily that a tridiagonal A is pseudosymmetric if and only if it can be expressed as a product

$$A = TD = \begin{bmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & b_{n-1} & a_n \end{bmatrix} \begin{bmatrix} d_1 & & & & \\ & d_2 & & & \\ & & d_3 & & \\ & & & \ddots & \\ & & & & d_n \end{bmatrix},$$

where D is a *signature matrix*; that is, it is diagonal with each d_j equal to 1 or -1. T is symmetric, and we can even make all the b_j nonnegative. Thus the pseudosymmetric eigenvalue problem can be treated as a product eigenvalue problem. Placing the factors T and D in a cyclic matrix, we have

$$C = \begin{bmatrix} & T \\ D & \end{bmatrix}. \tag{16}$$

Illustrating with the 3×3 case, we have

$$C = \left[\begin{array}{ccc|ccc} & & & a_1 & b_1 & \\ & & & b_1 & a_2 & b_2 \\ & & & & b_2 & a_3 \\ \hline d_1 & & & & & \\ & d_2 & & & & \\ & & d_3 & & & \end{array}\right].$$

The shuffled version is the Hessenberg matrix

$$\left[\begin{array}{cc|cc|cc} & a_1 & & b_1 & & \\ d_1 & & & & & \\ \hline & b_1 & & a_2 & & b_2 \\ & & d_2 & & & \\ \hline & & & b_2 & & a_3 \\ & & & & d_3 & \end{array}\right].$$

In the class of GR algorithms, the one that preserves pseudosymmetric structure is called HR [22, 24]. The H stands for *hyperbolic*. An HR iteration on $A = TD$ has the form

$$p(A) = HR, \qquad \hat{A} = H^{-1}AH,$$

where R is upper triangular and H has the special property

$$H^T D H = \hat{D},$$

$\hat{D}$ being another signature matrix. Then $\hat{A} = \hat{T}\hat{D}$, where $\hat{T}$ is tridiagonal and symmetric. Thus pseudosymmetric structure is preserved.

Equivalently we can do an HR iteration on the cyclic matrix C defined in (16). Here we are in the case $k = 2$, so the driving function $f(C)$ for the iteration must be a function of C^2, say $f(C) = p(C^2)$. Thus

$$f(C) = p(C^2) = \begin{bmatrix} p(TD) & \\ & p(DT) \end{bmatrix}. \tag{17}$$

The HR decomposition of $p(C^2)$ is built up from HR decompositions of $p(TD)$ and $p(DT)$. These are fortunately related. If $p(TD) = HR$ with $H^T D H = \hat{D}$, then $p(DT) = Dp(TD)D = DHRD = (DH\hat{D})(\hat{D}RD) = \tilde{H}\tilde{R}$, where $\tilde{H} = DH\hat{D} = H^{-T}$ satisfies $\tilde{H}^T D \tilde{H} = \hat{D}$, and $\tilde{R}$ is upper triangular. Here we have used repeatedly the fact that a signature matrix is equal to its inverse. Thus the HR decomposition of $f(C)$ looks like

$$\begin{bmatrix} p(TD) & \\ & p(DT) \end{bmatrix} = \begin{bmatrix} H & \\ & H^{-T} \end{bmatrix} \begin{bmatrix} R & \\ & \tilde{R} \end{bmatrix}. \tag{18}$$

Performing the similarity transformation with the H factor from this decomposition, we complete the HR iteration on C:

$$\begin{aligned} \hat{C} = \begin{bmatrix} & \hat{T} \\ \hat{D} & \end{bmatrix} &= \begin{bmatrix} H^{-1} & \\ & H^T \end{bmatrix} \begin{bmatrix} & T \\ D & \end{bmatrix} \begin{bmatrix} H & \\ & H^{-T} \end{bmatrix} \\ &= \begin{bmatrix} & H^{-1}TH^{-T} \\ H^T D H & \end{bmatrix}. \end{aligned} \tag{19}$$

This implicitly effects HR iterations on TD and DT simultaneously:

$$p(TD) = HR, \qquad \hat{T}\hat{D} = H^{-1}(TD)H$$

and

$$p(DT) = \tilde{H}\tilde{R}, \qquad \hat{D}\hat{T} = \tilde{H}^{-1}(DT)\tilde{H}.$$

In practice the HR iteration is implemented by bulge chasing, as described in section 5. Each elimination in the bulge chase is done by either an orthogonal transformation

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \qquad c^2 + s^2 = 1,$$

or a hyperbolic transformation

$$\begin{bmatrix} c & s \\ s & c \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} s & c \\ c & s \end{bmatrix}, \qquad c^2 - s^2 = 1.$$

An orthogonal (resp., hyperbolic) transformation is used if the two rows that the transformation affects correspond to entries of D that have the same (resp., opposite) sign. In this way, the signature matrix structure of D is preserved. The algorithm is quite straightforward.

12. The Hamiltonian Eigenvalue Problem. A matrix $\mathcal{H} \in \mathbb{R}^{2n\times 2n}$ is called *Hamiltonian* if it has the form

$$\mathcal{H} = \begin{bmatrix} A & K \\ N & -A^T \end{bmatrix}, \quad \text{where } K = K^T,\ N = N^T. \tag{20}$$

A more compact way of stating the definition makes use of the matrix

$$J = \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix}. \tag{21}$$

It is easy to check that $\mathcal{H}$ satisfies (20) if and only if $(J\mathcal{H})^T = J\mathcal{H}$.

Hamiltonian eigenvalue problems arise in a variety of settings, including linear-quadratic optimal control problems [17, 26, 68, 70, 71, 74], determination of corner singularities in anisotropic elastic structures [6, 75], and stability of gyroscopic systems [67]. Various algorithms of GR type for the Hamiltonian eigenvalue problem, as well as some background theory, are addressed in [3, 4, 13, 26, 78, 82, 93].

A matrix $\mathcal{S}$ is called *symplectic* if $\mathcal{S}^T J\mathcal{S} = J$, where J is as in (21). It is easy to check that if $\mathcal{H}$ is Hamiltonian, and $\mathcal{S}$ is symplectic, then $\mathcal{S}^{-1}\mathcal{H}\mathcal{S}$ is Hamiltonian. Thus the Hamiltonian structure will be respected by GR algorithms for which the transforming matrix is symplectic. These are the so-called SR algorithms [26].

In [26] Bunse-Gerstner and Mehrmann showed that every Hamiltonian matrix is similar by a symplectic similarity transformation to a matrix of the highly condensed form

$$\begin{bmatrix} E & T \\ D & -E \end{bmatrix}, \tag{22}$$

where E and D are diagonal and T is tridiagonal (and symmetric). There is some slack in the reduction algorithm: the main-diagonal entries of E can be chosen arbitrarily, while those of D must be nonzero but can be rescaled up or down. Bunse-Gerstner and Mehrmann advocated choosing the entries with an eye to making the transformation as stable as possible. This is a laudable goal. A more aggressive approach is to choose the entries to make the condensed form (22) as simple as possible. If one decides to do this, then the logical choice for E is the zero matrix. We can also scale D so that each of its main-diagonal entries is either $+1$ or -1; that is, it is a signature matrix. With these choices, the condensed form becomes

$$C = \begin{bmatrix} & T \\ D & \end{bmatrix}, \tag{23}$$

with T tridiagonal and symmetric and D a signature matrix. This is *exactly* the form of the cyclic matrix C (16) that arose in the discussion of pseudosymmetric matrices. Thus the Hamiltonian eigenvalue problem is equivalent to the pseudosymmetric product eigenvalue problem.

If we wish to apply the SR algorithm to the condensed C of (23) or to any Hamiltonian matrix, we must choose shifts in a way that respects Hamiltonian structure. The eigenvalues of a real Hamiltonian matrix possess the symmetry illustrated in Figure 1. If λ is an eigenvalue, then so are $-\lambda$, $\overline{\lambda}$, and $-\overline{\lambda}$. In particular, the eigenvalues occur in $\pm\lambda$ pairs. Thus the shifts should also be chosen in plus-minus pairs. In other words, the driving function for a Hamiltonian SR iteration on C should be a function of C^2: $f(C) = p(C^2)$, just as in (17). Now consider the HR decomposition (18). It is

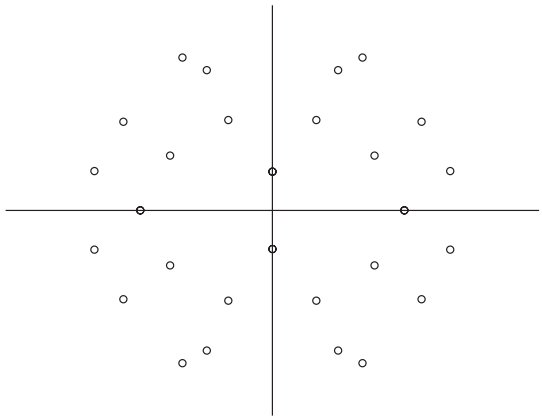


Fig. 1 *Hamiltonian eigenvalue symmetry.*

a simple matter to check that for any real nonsingular matrix V, the block-diagonal matrix

$$\begin{bmatrix} V & \\ & V^{-T} \end{bmatrix}$$

is symplectic. Therefore the HR decomposition (18) is also an SR decomposition, and the HR iteration (18), (19) is also an SR iteration on C. In conclusion, if the condensed form (23) for Hamiltonian matrices is used, then the Hamiltonian SR algorithm is identical to the HR algorithm for pseudosymmetric matrices. This observation was made previously in [15] and put to use in [101].

13. The Symplectic Eigenvalue Problem. As we stated above, a matrix $S \in \mathbb{R}^{2n\times 2n}$ is *symplectic* if $S^T J S = J$, where J is as in (21). The symplectic eigenvalue problem arises in discrete-time linear-quadratic control problems [58, 68, 74, 79]. Since the set of symplectic matrices forms a group, symplectic structure is preserved by similarity transformations by symplectic matrices. Therefore, the SR algorithms preserve symplectic structure. The details of applying SR algorithms to symplectic matrices have been discussed in [9, 10, 14, 16, 33, 35].

Banse and Bunse-Gerstner [9, 10] showed that every symplectic matrix can be reduced by a symplectic similarity transformation to a condensed *butterfly form*

$$B = \begin{bmatrix} E & T_1 \\ D & T_2 \end{bmatrix}, \tag{24}$$

where E and D are diagonal and T_1 and T_2 are tridiagonal. The submatrices are not uniquely determined; there is some slack in the choice of parameters. If one is after the simplest possible butterfly form, then one can choose $E = 0$, which immediately forces $T_1 = -D^{-1}$ by the symplectic property of B. The main-diagonal entries of D can be rescaled by arbitrary positive factors, so we can rescale it to make it a signature matrix. Suppose we do that. Then the symplectic property of B also implies that the matrix defined by $T = DT_2 = D^{-1}T_2$ is symmetric. With these choices, the symplectic butterfly matrix takes the form

$$B = \begin{bmatrix} 0 & -D \\ D & DT \end{bmatrix}, \tag{25}$$

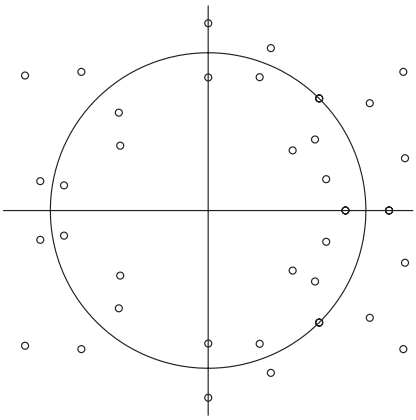


Fig. 2 *Symplectic eigenvalue symmetry.*

where D is a signature matrix (i.e., diagonal with main-diagonal entries ± 1) and T is symmetric and tridiagonal. Notice that the characteristics of D and T are the same as those possessed by the same symbols in the previous two sections.

If we want to apply the SR algorithm to the condensed symplectic matrix B of (25), how should we choose the shifts so that the symplectic structure is respected? The eigenvalues of a real symplectic matrix possess the symmetry structure illustrated in Figure 2. If λ is an eigenvalue, then so are λ^{-1}, $\overline{\lambda}$, and $\overline{\lambda}^{-1}$. In particular, the eigenvalues occur in λ, λ^{-1} pairs. Therefore we should choose shifts in μ, μ^{-1} pairs. An SR iteration on B has the form

$$f(B) = SR, \qquad \hat{B} = S^{-1}BS.$$

If the driving function f is a polynomial, then its factors should appear in pairs of the form $B - \mu I$, $B - \mu^{-1}I$. Multiplying these factors together, we obtain

$$B^2 - (\mu + \mu^{-1})B + I.$$

Notice that if we multiply this product by B^{-1}, we obtain an interesting result:

$$B^{-1}(B - \mu I)(B - \mu^{-1}I) = (B + B^{-1}) - (\mu + \mu^{-1})I,$$

which suggests that it might be advantageous to take f to be a rational function of B instead of a polynomial. If we want to apply shifts $\mu_1, \ldots, \mu_k$ and their inverses, we can take $f(B) = p(B + B^{-1})$, where p is the kth-degree polynomial whose zeros are $\mu_1 + \mu_1^{-1}, \ldots, \mu_k + \mu_k^{-1}$.

This idea relies on the accessibility of B^{-1}. Fortunately it is a simple matter to invert any symplectic matrix; the equation $S^T J S = J$ implies $S^{-1} = -JS^T J$. You can easily check that

$$B^{-1} = \begin{bmatrix} TD & D \\ -D & 0 \end{bmatrix}.$$

This has the happy consequence that

$$B + B^{-1} = \begin{bmatrix} TD & \\ & DT \end{bmatrix}$$

and

$$f(B) = p(B + B^{-1}) = \begin{bmatrix} p(TD) & \\ & p(DT) \end{bmatrix}.$$

Compare this with (17). In order to do an iteration of the SR algorithm, we need an SR decomposition of $f(B)$. However, as we already observed in the previous section, the HR decomposition (18) is also an SR decomposition, because the matrix

$$S = \begin{bmatrix} H & \\ & H^{-T} \end{bmatrix}$$

is symplectic. We complete an SR iteration by performing a similarity transformation with this matrix:

$$\begin{aligned} \hat{B} = S^{-1}BS &= \begin{bmatrix} H^{-1} & \\ & H^T \end{bmatrix} \begin{bmatrix} 0 & -D \\ D & DT \end{bmatrix} \begin{bmatrix} H & \\ & H^{-T} \end{bmatrix} \\ &= \begin{bmatrix} 0 & -H^{-1}DH^{-T} \\ H^TDH & (H^TDH)(H^{-1}TH^{-T}) \end{bmatrix} \\ &= \begin{bmatrix} 0 & -\hat{D} \\ \hat{D} & \hat{D}\hat{T} \end{bmatrix}, \end{aligned}$$

where

$$\hat{T} = H^{-1}TH^{-T} \quad \text{and} \quad \hat{D} = H^TDH.$$

In conclusion, if the condensed form (25) is used, the symplectic SR algorithm is equivalent to the pseudosymmetric HR algorithm. This observation was made previously in [15] and put to use in [101].

14. The Unitary Eigenvalue Problem. As our final example of an eigenvalue problem that is best handled in product form, we briefly consider the unitary eigenvalue problem. Gauss–Szegö quadrature rules are numerical integration formulas of optimal degree for measures with support on the unit circle. Computation of the points and weights for a Gauss–Szegö rule require the computation of the eigensystem of a unitary upper Hessenberg matrix [51, 53].

Every unitary proper upper Hessenberg matrix $U \in \mathbb{C}^{n\times n}$ can be expressed as a product

$$U = G_1G_2 \cdots G_{n-1}G_n, \tag{26}$$

where each G_k has a very simple form:

$$G_k = \begin{bmatrix} I_{k-1} & & & \\ & -\gamma_k & \sigma_k & \\ & \sigma_k & \overline{\gamma}_k & \\ & & & I_{n-k-1} \end{bmatrix}, \qquad \sigma_k > 0,\ |\gamma_k|^2 + \sigma_k^2 = 1,$$

for $k = 1, \ldots, n-1$, and

$$G_n = \begin{bmatrix} I_{n-1} & \\ & -\gamma_n \end{bmatrix}, \qquad |\gamma_n| = 1.$$

The *Schur parameters* γ_k and *complementary Schur parameters* σ_k are uniquely determined. This factorization allows us to express the unitary Hessenberg matrix in terms of the n complex parameters γ_k and the $n-1$ real parameters σ_k, which is much less than the $O(n^2)$ matrix entries in the conventional representation of U. Certainly we would prefer to work with the factored form. In principle we can get rid of the parameters σ_k as well; they are redundant, since

$$\sigma_k = \sqrt{1 - |\gamma_k|^2}.$$

However, in the face of error, this formula is unable to produce an accurate value for σ_k when $|\gamma_k|$ is close to 1, because cancellation occurs when $|\gamma_k|^2$ is subtracted from 1. It is therefore necessary to keep the σ_k in practice.

The unitary factorization (26) does not fit the pattern established in section 5, in which one of the factors is upper Hessenberg and the others are upper triangular. Therefore the methodology outlined in this paper cannot be applied directly to this factorization. Several alternative approaches have been developed. Gragg [52] showed how to effect iterations of the QR algorithm in terms of the Schur parameters. That algorithm turned out to be unstable, but M. Stewart [90] showed how to stabilize it. Ammar, Gragg, and Reichel [1, 2, 54] proposed two other methods for the real, orthogonal eigenvalue problem. One method breaks the problem into two half-size SVD problems, and the other uses a divide-and-conquer approach.

Other approaches are based on an *odd-even* factorization. In [1] it was shown that the unitary matrix U (26) is unitarily similar to

$$V = (G_1 G_3 G_5 \cdots)(G_2 G_4 G_6 \cdots).$$

If one lets $H_o = G_1 G_3 G_5 \cdots$ and $H_e = G_2 G_4 G_6 \cdots$, then

$$V = H_o H_e, \tag{27}$$

and each of the factors is block diagonal:

$$H_o = \begin{bmatrix} -\gamma_1 & \sigma_1 & & & & \\ \sigma_1 & \overline{\gamma}_1 & & & & \\ & & -\gamma_3 & \sigma_3 & & \\ & & \sigma_3 & \overline{\gamma}_3 & & \\ & & & & \ddots & \end{bmatrix}$$

and

$$H_e = \begin{bmatrix} 1 & & & & & \\ & -\gamma_2 & \sigma_2 & & & \\ & \sigma_2 & \overline{\gamma}_2 & & & \\ & & & -\gamma_4 & \sigma_4 & \\ & & & \sigma_4 & \overline{\gamma}_4 & \\ & & & & & \ddots \end{bmatrix}.$$

The factorization (27) also does not fit the pattern established in section 5, since neither of the factors in (27) is upper triangular. Bunse-Gerstner and Elsner [25] reformulated the product (27) as a generalized eigenvalue problem

$$H_o - \lambda H_e^{-1}$$

and developed a backward stable QZ algorithm for this "odd-even" pencil.

15. Concluding Remarks. A wide variety of algorithms for computing eigenvalues of products of matrices can be viewed as GR algorithms on an associated cyclic matrix or its shuffled version [65]. These include the QZ algorithm [76] for the generalized eigenvalue problem and its extensions [18, 56, 92], the Golub–Kahan QR algorithm [46] and its high-accuracy improvement [27] for the SVD, the QR algorithm for the skew-symmetric and zero-diagonal symmetric eigenvalue problems [96], and the high-accuracy differential quotient-difference algorithm [34, 80, 86] for the symmetric, positive-definite eigenvalue problem or the SVD. The totally nonnegative eigenvalue problem can be reduced (with high relative accuracy) to a positive definite, symmetric, tridiagonal eigenvalue problem, which can then be solved accurately by methods just mentioned [62]. The pseudosymmetric eigenvalue problem can be viewed as a product eigenvalue problem, which can then be solved by an HR algorithm on a cyclic matrix

$$C = \begin{bmatrix} & T \\ D & \end{bmatrix}.$$

This version of the HR algorithm is also an SR algorithm and can be used to solve the Hamiltonian and symplectic eigenvalue problems, once they have been reduced to suitable condensed forms [15]. The unitary eigenvalue problem is also best viewed as a product eigenvalue problem. However, the two known useful factorizations of unitary matrices do not fit the same mold as the other factorizations considered in this paper.

REFERENCES

[1] G. Ammar, W. Gragg, and L. Reichel, *On the eigenproblem for orthogonal matrices*, in Proceedings of the 25th IEEE Conference on Decision and Control, Athens, Greece, 1986, pp. 1963–1966.

[2] G. Ammar, W. Gragg, and L. Reichel, *Determination of Pisarenko frequency estimates as eigenvalues of an orthogonal matrix*, in Advanced Algorithms and Architectures for Signal Processing II, F. T. Luk, ed., Proc. SPIE 826, SPIE, Bellingham, WA, 1987, pp. 143–145.

[3] G. S. Ammar, P. Benner, and V. Mehrmann, *A multishift algorithm for the numerical solution of algebraic Riccati equations*, Electron. Trans. Numer. Anal., 1 (1993), pp. 33–48.

[4] G. S. Ammar and V. Mehrmann, *On Hamiltonian and symplectic Hessenberg forms*, Linear Algebra Appl., 149 (1991), pp. 55–72.

[5] ANSI/IEEE, *IEEE Standard for Binary Floating Point Arithmetic*, Std. 754-1985 ed., IEEE, New York, 1985.

[6] T. Apel, V. Mehrmann, and D. Watkins, *Structured eigenvalue methods for the computation of corner singularities in 3D anisotropic elastic structures*, Comput. Methods Appl. Mech. Engrg., 191 (2002), pp. 4459–4473.

[7] Z. Bai and J. Demmel, *On a block implementation of the Hessenberg multishift QR iteration*, Internat. J. High Speed Comput., 1 (1989), pp. 97–112.

[8] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., *Templates for the Solution of Algebraic Eigenvalue Problems*, SIAM, Philadelphia, 2000.

[9] G. Banse, *Symplektische Eigenwertverfahren zur Lösung zeitdiskreter optimaler Steuerungs probleme*, Ph.D. thesis, University of Bremen, Germany, 1995.

[10] G. Banse and A. Bunse-Gerstner, *A condensed form for the solution of the symplectic eigenvalue problem*, in Systems and Networks: Mathematical Theory and Applications, U. Helmke, R. Menniken, and J. Sauer, eds., Akademie Verlag, Berlin, 1994, pp. 613–616.

[11] F. L. Bauer, *Sequential reduction to tridiagonal form*, J. Soc. Indust. Appl. Math., 7 (1959), pp. 107–113.

[12] P. Benner, R. Byers, V. Mehrmann, and H. Xu, *Numerical Computation of Deflating Subspaces of Embedded Hamiltonian Pencils*, Tech. Rep. 99-15, SFB 393, Fakultät für Mathematik, TU Chemnitz, Chemnitz, Germany, 1999.

[13] P. Benner, R. Byers, V. Mehrmann, and H. Xu, *Numerical computation of deflating subspaces of skew-Hamiltonian/Hamiltonian pencils*, SIAM J. Matrix Anal. Appl., 24 (2002), pp. 165–190.

[14] P. Benner and H. Fassbender, *The symplectic eigenvalue problem, the butterfly form, the SR algorithm, and the Lanczos method*, Linear Algebra Appl., 275–276 (1998), pp. 19–47.

[15] P. Benner, H. Fassbender, and D. S. Watkins, *Two connections between the SR and HR eigenvalue algorithms*, Linear Algebra Appl., 272 (1998), pp. 17–32.

[16] P. Benner, H. Fassbender, and D. S. Watkins, *SR and SZ algorithms for the symplectic (butterfly) eigenproblem*, Linear Algebra Appl., 287 (1999), pp. 41–76.

[17] J. Bittanti, A. J. Laub, and J. C. Willems, eds., *The Riccati Equation*, Springer-Verlag, New York, 1991.

[18] A. Bojanczyk, G. H. Golub, and P. Van Dooren, *The periodic Schur decomposition; algorithm and applications*, in Advanced Signal Processing Algorithms, Architectures, and Implementations III, Proc. SPIE 1770, SPIE, Bellingham, WA, 1992, pp. 31–42.

[19] K. Braman, R. Byers, and R. Mathias, *The multishift QR algorithm. Part* I: *Maintaining well-focused shifts and level* 3 *performance*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 929–947.

[20] K. Braman, R. Byers, and R. Mathias, *The multishift QR algorithm. Part* II: *Aggressive early deflation*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 948–973.

[21] R. Brawer and M. Pirovino, *The linear algebra of the Pascal matrix*, Linear Algebra Appl., 174 (1992), pp. 13–23.

[22] M. A. Brebner and J. Grad, *Eigenvalues of $Ax = \lambda Bx$ for real symmetric matrices A and B computed by reduction to pseudosymmetric form and the HR process*, Linear Algebra Appl., 43 (1982), pp. 99–118.

[23] F. Brenti, *Combinatorics and total positivity*, J. Combin. Theory Ser. A, 71 (1995), pp. 175–218.

[24] A. Bunse-Gerstner, *An analysis of the HR algorithm for computing the eigenvalues of a matrix*, Linear Algebra Appl., 35 (1981), pp. 155–173.

[25] A. Bunse-Gerstner and L. Elsner, *Schur parameter pencils for the solution of the unitary eigenproblem*, Linear Algebra Appl., 154–156 (1991), pp. 741–778.

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

[27] J. Demmel and W. Kahan, *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 873–912.

[28] I. S. Dhillon, *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*, Ph.D. thesis, University of California, Berkeley, 1997.

[29] A. A. Dubrulle, *The Multishift QR Algorithm—Is It Worth the Trouble?*, Tech. Rep. G320-3558x, IBM Corp., Palo Alto, CA, 1991.

[30] A. A. Dubrulle and G. H. Golub, *A multishift QR iteration without computation of the shifts*, Numer. Algorithms, 7 (1994), pp. 173–181.

[31] A. Edelman and G. Strang, *Pascal matrices*, Amer. Math. Monthly, 111 (2004), pp. 189–197.

[32] S. M. Fallat, *Bidiagonal factorizations of totally nonnegative matrices*, Amer. Math. Monthly, 108 (2001), pp. 697–712.

[33] H. Fassbender, *Symplectic Methods for the Symplectic Eigenproblem*, Kluwer Academic/Plenum Publishers, New York, 2000.

[34] K. V. Fernando and B. N. Parlett, *Accurate singular values and differential qd algorithms*, Numer. Math., 67 (1994), pp. 191–229.

[35] U. Flaschka, V. Mehrmann, and D. Zywietz, *An analysis of structure preserving methods for symplectic eigenvalue problems*, RAIRO Automat.-Prod. Inform. Ind., 25 (1991), pp. 165–190.

[36] S. Fomin and A. Zelevinsky, *Total positivity: Tests and parametrizations*, Math. Intelligencer, 22 (2000), pp. 23–33.

[37] J. G. F. Francis, *The QR transformation: A unitary analogue to the LR transformation.* I, Comput. J., 4 (1961), pp. 265–272.

[38] J. G. F. Francis, *The QR transformation.* II, Comput. J., 4 (1961), pp. 332–345.

[39] M. Gasca and C. A. Micchelli, eds., *Total Positivity and Its Applications*, Kluwer Academic, Dordrecht, The Netherlands, 1996.

[40] M. Gasca and J. M. Peña, *On factorizations of totally positive matrices*, in Total Positivity and Its Applications, M. Gasca and C. A. Micchelli, eds., Kluwer Academic, Dordrecht, The Netherlands, 1996, pp. 109–130.

[41] W. Gautschi, *Construction of Gauss-Christoffel quadrature formulas*, Math. Comp., 22 (1968), pp. 251–270.
[42] W. Gautschi, *On generating Gaussian quadrature rules*, in Numerische Integration, G. Hämmerlin, ed., Birkhäuser Verlag, Basel, 1979, pp. 147–154.
[43] W. Gautschi, *Is the recurrence relation for orthogonal polynomials always stable?*, BIT, 33 (1993), pp. 277–284.
[44] W. Gautschi, *Orthogonal polynomials: Applications and computation*, Acta Numerica 1996, Cambridge University Press, Cambridge, UK, 1996, pp. 45–119.
[45] G. H. Golub, *Least squares, singular values, and matrix approximations*, Apl. Mat., 13 (1968), pp. 44–51.
[46] G. H. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. Soc. Indust. Appl. Math. Ser. B Numer. Anal., 2 (1965), pp. 205–224.
[47] G. H. Golub and C. Reinsch, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403–420.
[48] G. H. Golub, K. Sølna, and P. Van Dooren, *Computing the SVD of a general matrix product/quotient*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 1–19.
[49] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.
[50] G. H. Golub and J. H. Welsh, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.
[51] W. B. Gragg, *Positive definite Toeplitz matrices, the Arnoldi process for isometric operators, and Gaussian quadrature on the unit circle*, in Numerical Methods in Linear Algebra, E. S. Nikolaev, ed., Moscow University Press, 1982, pp. 16–23 (in Russian).
[52] W. B. Gragg, *The QR algorithm for unitary Hessenberg matrices*, J. Comput. Appl. Math., 16 (1986), pp. 1–8.
[53] W. B. Gragg, *Positive definite Toeplitz matrices, the Arnoldi process for isometric operators, and Gaussian quadrature on the unit circle*, J. Comput. Appl. Math., 46 (1993), pp. 183–198 (English translation of [51]).
[54] W. B. Gragg and L. Reichel, *A divide and conquer algorithm for the unitary eigenproblem*, in Proceedings of the Second Conference on Hypercube Multiprocessors, SIAM, Philadelphia, 1987, pp. 639–647.
[55] C. R. Hadlock, *Field Theory and Its Classical Problems*, The Carus Mathematical Monographs, Mathematical Association of America, Washington, DC, 1978.
[56] J. J. Hench and A. J. Laub, *Numerical solution of the discrete-time periodic Riccati equation*, ACM Trans. Automat. Control, 39 (1994), pp. 1197–1210.
[57] G. Henry, D. Watkins, and J. Dongarra, *A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures*, SIAM J. Sci. Comput., 24 (2002), pp. 284–311.
[58] D. Hinrichsen and N. K. Son, *Stability radii of linear discrete-time systems and symplectic pencils*, Int. J. Robust Nonlinear Control, 1 (1991), pp. 79–97.
[59] J. E. Jackson, *A User's Guide to Principal Components*, John Wiley and Sons, New York, 1991.
[60] W. Kahan, *Accurate Eigenvalues of a Symmetric Tridiagonal Matrix*, Tech. Rep. CS 41, Computer Science Department, Stanford University, Stanford, CA, 1966.
[61] S. Karlin, *Total Positivity*, Stanford University Press, Stanford, CA, 1968.
[62] P. Koev, *Accurate eigenvalues and SVDs of totally nonnegative matrices*, SIAM J. Matrix Anal. Appl., to appear.
[63] D. Kressner, *An efficient and reliable implementation of the periodic QZ algorithm*, in IFAC Workshop on Periodic Control Systems, 2001; available online from www.math.tu-berlin.de/~kressner/.
[64] D. Kressner, *Numerical Methods and Software for General and Structured Eigenvalue Problems*, Ph.D. thesis, TU Berlin, Institut für Mathematik, Berlin, Germany, 2004.
[65] D. Kressner, *The periodic QR algorithm is a disguised QR algorithm*, Linear Algebra Appl., 2004, to appear.
[66] V. N. Kublanovskaya, *On some algorithms for the solution of the complete eigenvalue problem*, USSR Comput. Math. and Math. Phys., 3 (1961), pp. 637–657.
[67] P. Lancaster, *Strongly stable gyroscopic systems*, Electron. J. Linear Algebra, 5 (1999), pp. 53–66.
[68] P. Lancaster and L. Rodman, *The Algebraic Riccati Equation*, Oxford University Press, Oxford, UK, 1995.
[69] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bur. Standards, 45 (1950), pp. 255–281.

[70] A. J. Laub, *A Schur method for solving algebraic Riccati equations*, IEEE Trans. Automat. Control, AC-24 (1979), pp. 913–921.
[71] A. J. Laub, *Invariant subspace methods for the numerical solution of Riccati equations*, in The Riccati Equation, J. Bittani, A. J. Laub, and J. C. Willems, eds., Springer-Verlag, Berlin, 1991, pp. 163–196.
[72] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998. ARPACK homepage, http://www.caam.rice.edu/software/ARPACK/index.html.
[73] C. Loewner, *On totally positive matrices*, Math. Z., 63 (1955), pp. 338–340.
[74] V. L. Mehrmann, *The Autonomous Linear Quadratic Control Problem*, Lecture Notes in Control and Inform. Sci. 163, Springer-Verlag, Berlin, 1991.
[75] V. Mehrmann and D. Watkins, *Structure-preserving methods for computing eigenpairs of large sparse skew-Hamiltonian/Hamiltonian pencils*, SIAM J. Sci. Comput., 22 (2001), pp. 1905–1925.
[76] C. B. Moler and G. W. Stewart, *An algorithm for generalized matrix eigenvalue problems*, SIAM J. Numer. Anal., 10 (1973), pp. 241–256.
[77] M. L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic*, SIAM, Philadelphia, 2001.
[78] C. C. Paige and C. Van Loan, *A Schur decomposition for Hamiltonian matrices*, Linear Algebra Appl., 41 (1981), pp. 11–32.
[79] T. Pappas, A. Laub, and N. Sandell, *On the numerical solution of the discrete-time algebraic Riccati equation*, IEEE Trans. Automat. Control, AC-25 (1980), pp. 631–641.
[80] B. N. Parlett, *The new qd algorithms*, Acta Numerica 1995, Cambridge University Press, Cambridge, UK, 1995, pp. 459–491.
[81] B. N. Parlett and I. S. Dhillon, *Relatively robust representations of symmetric tridiagonals*, Linear Algebra Appl., 309 (2000), pp. 121–151.
[82] A. C. Raines, III and D. S. Watkins, *A class of Hamiltonian-symplectic methods for solving the algebraic Riccati equation*, Linear Algebra Appl., 205/206 (1994), pp. 1045–1060.
[83] H. Rutishauser, *Der Quotienten-Differenzen-Algorithmus*, Z. Angew. Math. Phys., 5 (1954), pp. 233–251.
[84] H. Rutishauser, *Der Quotienten-Differenzen-Algorithmus*, Mitt. Inst. Angew. Math. ETH 7, Birkhäuser, Basel, 1957.
[85] H. Rutishauser, *Solution of eigenvalue problems with the LR-transformation*, Nat. Bur. Standards Appl. Math. Ser., 49 (1958), pp. 47–81.
[86] H. Rutishauser, *Lectures on Numerical Mathematics*, Birkhäuser, Basel, 1990. (Translation of *Vorlesungen über numerische Mathematik*, 1976.)
[87] G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi–Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.
[88] D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 357–385.
[89] D. Sorensen, *Numerical methods for large eigenvalue problems*, Acta Numerica 2002, Cambridge University Press, Cambridge, UK, 2002, pp. 519–584.
[90] M. Stewart, *An Error Analysis of a Unitary Hessenberg QR Algorithm*, Tech. Rep. TR-CS-98-11, Department of Computer Science, Australian National University, Canberra, 1998; available online from http://eprints.anu.edu.au/archive/00001557/.
[91] G. Strang and G. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.
[92] C. F. Van Loan, *A general matrix eigenvalue algorithm*, SIAM J. Numer. Anal., 12 (1975), pp. 819–834.
[93] C. F. Van Loan, *A symplectic method for approximating all the eigenvalues of a Hamiltonian matrix*, Linear Algebra Appl., 61 (1984), pp. 233–251.
[94] R. S. Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ, 1962.
[95] R. C. Ward, *The combination shift QZ algorithm*, SIAM J. Numer. Anal., 12 (1975), pp. 835–853.
[96] R. C. Ward and L. J. Gray, *Eigensystem computation for skew-symmetric and a class of symmetric matrices*, ACM Trans. Math. Software, 4 (1978), pp. 278–285.
[97] D. S. Watkins, *Shifting strategies for the parallel QR algorithm*, SIAM J. Sci. Comput., 15 (1994), pp. 953–958.
[98] D. S. Watkins, *The transmission of shifts and shift blurring in the QR algorithm*, Linear Algebra Appl., 241–243 (1996), pp. 877–896.

[99] D. S. Watkins, *Performance of the QZ algorithm in the presence of infinite eigenvalues*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 364–375.
[100] D. S. Watkins, *Fundamentals of Matrix Computations*, 2nd ed., John Wiley and Sons, New York, 2002.
[101] D. S. Watkins, *On Hamiltonian and symplectic Lanczos processes*, Linear Algebra Appl., 385 (2004), pp. 23–45.
[102] D. S. Watkins and L. Elsner, *Chasing algorithms for the eigenvalue problem*, SIAM J. Matrix Anal. Appl., 12 (1991), pp. 374–384.
[103] D. S. Watkins and L. Elsner, *Convergence of algorithms of decomposition type for the eigenvalue problem*, Linear Algebra Appl., 143 (1991), pp. 19–47.
[104] A. Whitney, *A reduction theorem for totally positive matrices*, J. Anal. Math., 2 (1952), pp. 88–92.
[105] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, UK, 1965.
[106] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method*, 5th ed., Butterworth-Heinemann, Boston, 2000.