Electronic Transactions on Numerical Analysis.
Volume 44, pp. 327–341, 2015.
Copyright © 2015, Kent State University.
ISSN 1068–9613.

ETNA
Kent State University
http://etna.math.kent.edu

FAST AND STABLE UNITARY QR ALGORITHM*

JARED L. AURENTZ†, THOMAS MACH§, RAF VANDEBRIL§, AND DAVID S. WATKINS‡

Abstract. A fast Fortran implementation of a variant of Gragg's unitary Hessenberg QR algorithm is presented. It is proved, moreover, that all QR- and QZ-like algorithms for the unitary eigenvalue problems are equivalent. The algorithm is backward stable. Numerical experiments are presented that confirm the backward stability and compare the speed and accuracy of this algorithm with other methods.

Key words. eigenvalue, unitary matrix, Francis's QR algorithm, core transformations rotators

AMS subject classifications. 65F15, 65H17, 15A18, 15B10

1. Introduction.

This project began with a request from Alan Edelman [14] for fast code to compute the eigenvalues of a unitary matrix. We were able to fulfill this request by taking our unitary-plus-rank-one code [4] and removing the hard parts. At the same time we searched the web to find out what is already publicly available in Fortran or some other compiled language. To our surprise, even though many papers on the unitary eigenvalue problem have been written in the past thirty years (see Section 3), we found only one item, the divide-and-conquer code of Ammar, Reichel, and Sorensen [2]. Because of this surprising shortage of publicly available software, we decided to publish our codes.

We use Francis's implicitly shifted QR algorithm, which is the most popular method for solving dense medium-sized eigenvalue problems. A unitary upper Hessenberg matrix can be described by $\mathcal{O}(n)$ parameters. Using this representation of the matrix, one can implement the algorithm in a complexity of $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$. This was first done by Gragg [21] in 1986, and since then a number of improvements and variants have been suggested. For example, instead of operating on the upper Hessenberg form, one can operate on the CMV form or some other twisted factorization [35, 36], or one can transform the matrix to an odd-even pencil and apply a QZ algorithm to the pencil [5]. In this paper we prove that all of these variants are equivalent, that is, they are just different ways of looking at the same algorithm.

We conclude the paper with numerical experiments that demonstrate the speed and accuracy of our code. We compare against standard LAPACK 3.5.0 codes, which are $\mathcal{O}(n^3)$, and the divide-and-conquer code [2]. Our code is much faster than the LAPACK codes and just as accurate. For most of the classes of test problems that we considered, the divide-and-conquer code is faster than ours, but ours is much more accurate.

*Received November 18, 2014. Accepted May 21, 2015. Published online on June 17, 2015. Recommended by C. Jagels. The research was partially supported by the Research Council KU Leuven, projects CREA-13-012 Can Unconventional Eigenvalue Algorithms Supersede the State of the Art, OT/11/055 Spectral Properties of Perturbed Normal Matrices and their Applications, CoE EF/05/006 Optimization in Engineering (OPTEC), and fellowship F+/13/020 Exploiting Unconventional QR Algorithms for Fast and Accurate Computations of Roots of Polynomials by the Fund for Scientific Research–Flanders (Belgium) project G034212N Reestablishing Smoothness for Matrix Manifold Optimization via Resolution of Singularities by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office, Belgian Network DYSCO (Dynamical Systems, Control, and Optimization) and by the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 291068. The views expressed in this article are not those of the ERC or the European Commission, and the European Union is not liable for any use that may be made of the information contained here.

†Mathematical Institute, University of Oxford, Andrew Wiles Building, Woodstock Road, OX2 6GG Oxford, UK (aurentz@maths.ox.ac.uk).

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

§Department of Computer Science, KU Leuven, Celestijnenlaan 200A, 3001 Leuven (Heverlee), Belgium ({thomas.mach,raf.vandebril}@cs.kuleuven.be).

The paper is organized as follows. Section 2 lists a few applications of the unitary eigenvalue problem, and Section 3 provides an overview of previous work. Section 4 introduces the parametrization and establishes the notation and terminology we will use. In Section 5 we describe our version of the algorithm. In Section 6 we look at some of the variants and prove that they are all equivalent. In Section 7 we present our numerical experiments.

The software associated with this paper has been packaged into a Fortran 90 library called eiscor and can be found at https://github.com/eiscor/eiscor.

2. Some applications.

Killip and Nenciu [26] describe an ensemble of $n \times n$ random unitary matrices whose eigenvalues are distributed according to the Gibbs distribution for n particles of the Coulomb gas on the unit circle. Edelman [14] wanted code to use for tests involving large numbers of random matrices with large n from this ensemble.

Gauss-Szegő quadrature formulas [20, 22, 39] are formulas of maximal degree for estimating integrals with respect to measures with support on the unit circle. The sample points are the eigenvalues of a certain unitary matrix, and the weights are the squares of the absolute values of the first components of the eigenvectors.

Pisarenko frequency estimates [30] can be computed by solving a unitary eigenvalue problem as described by Cybenko [9].

3. Overview.

A unitary Hessenberg matrix A is efficiently stored as a product of n unitary factors

$$A = C_1 \cdots C_{n-1} C_n,$$

where each C_i differs from the identity matrix only in the rows/columns i and $i+1$. Gragg [21] developed the first unitary Hessenberg QR algorithm, updating the factors directly in each QR step. Convergence had been proved earlier by Eberlein and Huang [13]. Wang and Gragg [37,38] analyzed the relationship between convergence and shift choice. M. Stewart [32] proved the original algorithm to be numerically unstable. Improvements and a proof of stability are due to Gragg [23] and Stewart [33]. David and Watkins [10] presented a multishift version.

Ammar, Gragg, and Reichel [1] presented an approach for orthogonal matrices that computes the singular values of the matrix's real and imaginary parts in order to get eigenvalues with accurate real and imaginary parts. Rutishauser [15] deduced a method relying on the LU decomposition of the orthogonal matrix.

An approach that applies a QZ algorithm to a unitary pencil was described by Bunse-Gerstner and Elsner [5], and convergence as a particular case of a more general setting was proved later on by Vandebril and Watkins [35]. A bisection method, relying on a Sturm sequence, was proposed by Bunse-Gerstner and He [6].

Divide-and-conquer approaches were developed originally by Gragg and Reichel [24] and Ammar, Reichel, and Sorenson [2, 3], and later by Gu et al. [25].

Unitary matrices, their rank properties, and eigenvalue methods were analyzed in great generality by Delvaux and Van Barel [11, 12]. An alternative interpretation of Gragg's algorithm [21] linking the transmission of shifts to the updating of the matrix factorization is included.

The algorithm presented by Gemignani [18] differs significantly from the other methods. A Möbius transformation is used to convert the unitary matrix to Hermitian diagonal-plus-semiseparable form, after which the eigenvalues of this matrix are computed.

4. Unitary matrix factorization and properties.

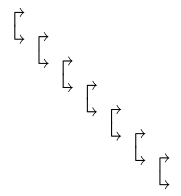
We assume the matrix $A \in \mathbb{C}^{n \times n}$ to be of unitary Hessenberg form already [11,19,34], thus $A_{i,j} = 0$ for all $i > j+1$. Each unitary Hessenberg matrix can be factored into a product of n unitary matrices $A = C_1 C_2 \cdots C_{n-1} C_n$, where each C_i is equal to the identity except in the 2×2 submatrix $(i : i+1, i : i+1)$. Thus

ETNA
Kent State University
http://etna.math.kent.edu

C_n differs from the identity only in the (n, n) position. Optionally we can absorb C_n into C_{n-1} so that the product has only $n - 1$ factors: $A = C_1C_2 \cdots C_{n-1}$. We call the matrices C_i *core transformations*. The subscript i refers to the position of the active part, and it follows that core transformations C_i and C_j commute whenever $|i - j| > 1$.

We will use a pictorial description to assist in the understanding of the algorithm. A core transformation will be represented as with the tiny arrows pinpointing the active part. Our unitary Hessenberg matrix A is now factored as a *descending sequence* of core transformations $(n = 8)$

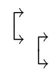
$$A = C_1C_2 \cdots C_{n-1} =$$



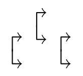
.

There are two operations, the *turnover* and *fusion*, required to describe the algorithm. The fusion stands for uniting two core transformations acting on the same rows by forming their product. A 3×3 unitary matrix can always be factored in two different ways as the product of three core transformations; by computing the QL and QR decomposition, we obtain the following equalities

$$U_1V_2W_1 =$$



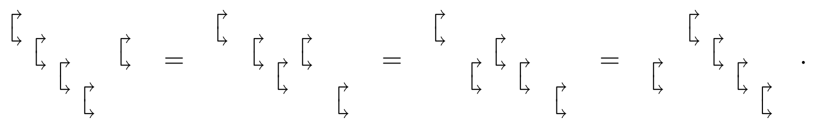
$$= \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} =$$



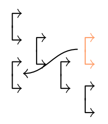
$$= U_2V_1W_2.$$

A *turnover* is the transition from left to right, or vice versa, thereby changing the active parts of the core transformations without changing their product.

We can use a turnover to pass a core transformation through a descending sequence as follows:

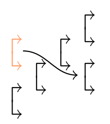


For this operation we will use the shorthand



.

The arrow indicates that a core transformation disappears from the right (the core transformation is in light orange), and a new one appears on the left. Exactly two of the core transformations from the descending sequence participate in the turnover. We will also use an analogous operation



.

which passes a core transformation from left to right through an ascending sequence.

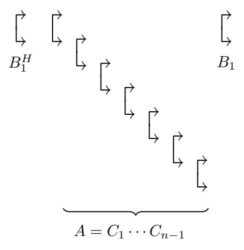
ETNA
Kent State University
http://etna.math.kent.edu

5. Unitary QR algorithm. We use Francis's implicitly shifted QR algorithm [16, 17]. Each iteration consists of three steps: choose a suitable shift to enhance the convergence, apply a similarity transformation to create a bulge, and chase the bulge until it disappears from the bottom of the matrix. Repeated iterations lead to deflations, and eventually we find all of the eigenvalues as explained in numerous textbooks, e.g., [40].

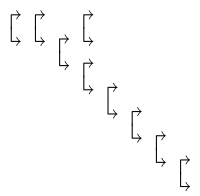
Choice of shifts. The shift μ is used to accelerate convergence. A good shift is one that approximates an eigenvalue well. Since all eigenvalues of A have absolute value 1, it seems reasonable to choose a shift that lies on the unit circle. We use a projected or unimodular Wilkinson shift. Therefore we pick the eigenvalue $\hat{\mu}$ of the trailing 2×2 submatrix $A_{k-1:k,k-1:k}$ that is closer to $A_{k,k}$, with k the largest index of the undeflated part of A. The shift μ is the projection $\hat{\mu}/\left|\hat{\mu}\right|$ of $\hat{\mu}$ on the unit circle. The projected or unimodular Wilkinson shift strategy was investigated by Wang and Gragg in [38]. They proved global convergence and showed by numerical experiments that the projected Wilkinson shift is superior to the standard Wilkinson shift without projection and to the shift given in [13].

Initial similarity transformation: bulge generation. The initial similarity transformation introduces the bulge perturbing the Hessenberg structure. For a suitable shift μ, let $x = (A - \mu I)e_1$, with e_1 the first standard basis vector. As A is Hessenberg, x has only two nonzero elements. The core transformation B_1 acting on rows one and two such that $B_1^H x = e_1$ determines the first similarity transformation $B_1^H A B_1$. Pictorially we get

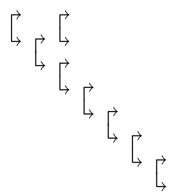
$$B_1^H A B_1 = B_1^H\, C_1 C_2 \cdots C_{n-1}\, B_1 =$$



$$=$$



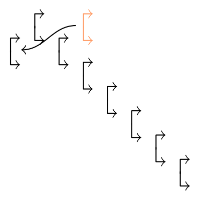
$$=$$



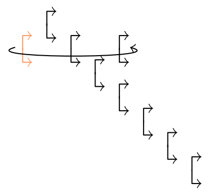
,

where we fused the two leftmost core transformations $B_1^H C_1 = \tilde{C}_1$, and only B_1, *the bulge*, disturbs the descending sequence of core transformations.

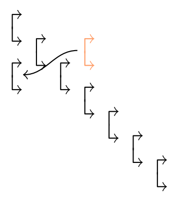
Bulge chasing. Chasing the bulge consists of performing two operations repeatedly until the bottom of the matrix is reached: first execute a turnover moving the bulge to the left and second do a similarity transformation bringing the bulge back to the right. Pictorially the first two chasing steps look as follows; the black arrows indicate the bulge's motion.

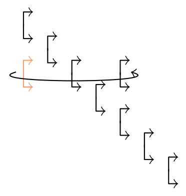


Turnover

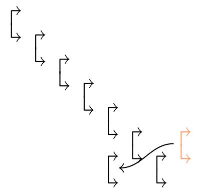


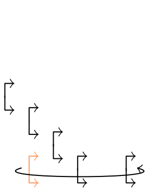
Similarity

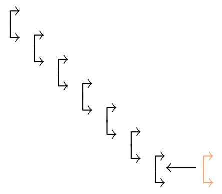

Turnover


Similarity

This process is repeated until the bulge reaches the bottom, where it is absorbed via a fusion. The next pictures show the final turnover, similarity, and fusion.


Turnover


Similarity


Fusion

Deflation. After several iterations, one or more subdiagonal entries will become negligibly small. In our factored form, this is signaled by a core transformation tending to become diagonal. A detailed discussion can be found in Mach and Vandebril [28].

Further details. So far we have given a high-level description of the algorithm leaving out certain details. For example, we have not specified how we implement turnovers.

Our code is derived from our implementation of a fast QR algorithm for companion matrices [4], which can be applied to unitary-plus-rank-one matrices in general. In [4] we store the unitary-plus-rank-one upper Hessenberg matrix A in QR-decomposed form: $A = QR$, where Q is unitary and R is upper triangular. In addition, Q is upper Hessenberg and R is unitary-plus-rank-one. The novel aspect of [4] is the efficient and stable storage scheme for R.

In our current setting, A is unitary, so in the QR decomposition we have $Q = A$ and $R = I$. To get our unitary code from the unitary-plus-rank-one code, we just took out R. This is what we meant when we referred to "removing the hard parts" in the introduction.

See [4, Section 6] for the implementation of the turnover and other fine points. The algorithm that we have described here is a single-shift algorithm. We also have a double-shift algorithm for real orthogonal matrices that operates in real arithmetic. In the interest of brevity, we have not described it here. It is detailed in [4], and the real orthogonal version can be obtained from that description by ignoring the "R part".

Data storage. In the implementation we choose the unitary core transformations to be rotators of the form $\left[\begin{smallmatrix} c & -s \\ s & \bar{c} \end{smallmatrix}\right]$ with $|c|^2 + s^2 = 1$. This is different from [21], where reflectors $\left[\begin{smallmatrix} c & s \\ s & -\bar{c} \end{smallmatrix}\right]$ were used. This choice does not mean that rotators are superior to reflectors; we believe that both are equally suited. We just had to choose one, and we stuck with the rotators we had used before.

In the real double shift (RDS) code, c and s stay real throughout the iterations. Thus roughly $2n$ reals are stored. In the complex single shift (CSS) code, we represent the rotators by real variables for the real part of c, the imaginary part of c, and for s, which is always kept real. To keep s real we have to add a diagonal matrix D containing phase factors, thus $A = C_1 C_2 \cdots C_{n-1} D$. Thus in the CSS code roughly $5n$ reals are stored.

Computing eigenvectors. The focus of this project is on eigenvalues, not eigenvectors. However, our implementation does provide an option to compute the eigenvectors. We emphasize that this part of the code is not at all novel and certainly not fast. It computes the eigenvectors by forming the product of all similarity transformations applied to A. Thus $\mathcal{O}(n^2)$ rotations are multiplied together, which costs $\mathcal{O}(n^3)$ flops. We use the eigenvectors for the residual computations in the numerical experiments in the next section. These serve as a check on the backward stability and accuracy of the eigenvalue computations.

Our eigenvector routines would be useful to anyone who wants a complete set of eigenvectors that is orthonormal to working precision and is not concerned with the high computational cost. If just a few eigenvectors are wanted or numerical orthogonality is not a concern, then inverse iteration is superior. We may provide an $\mathcal{O}(n^2)$ eigenvector routine in the future.

Gauss-Szegő quadrature formulas. Our codes can be used to compute Gauss-Szegő quadrature formulas [20, 22, 39] for estimating integrals with respect to measures on the unit circle. The sample points are the eigenvalues of a certain unitary matrix, and the weights are the squares of the absolute values of the first components of the eigenvectors. Thus the weights can be obtained by accumulating just the first row of the eigenvector matrix, and this can be done in $O(n^2)$ time. In practice the weight computation increases the computing time by some 60% to 70%.

6. Twisted QR algorithms and pencil methods.

In Section 4, the unitary matrix A was assumed to be of Hessenberg form. By executing some similarity transformations on the factored form, one can easily realize any factorization $U^H AU = C_{p_1} C_{p_2} \cdots C_{p_{n-1}}$, where p is a permutation of $[1, 2, \ldots, n-1]$, thereby not changing the core transformations only their relative positions. The core transformations are said to form a *twisted pattern*. Direct similarity transformations to any twisted shape also exist [34], and the most well-known twisted ordering is likely the CMV ordering (assume n even) $C_1 C_3 C_5 \cdots C_{n-1} \cdot C_2 C_4 C_6 \cdots C_{n-2}$ [7,8,31,39].

Eigenvalues of such a twisted pattern can be obtained via *twisted* QR algorithms, where the convergence is determined by rational functions instead of polynomials as in the classical QR case [34, 36]. In general these are distinct algorithms, but in the unitary case they all collapse to the same thing: there is no benefit of considering a particular pattern: all (twisted) QR algorithms are independent of p and provide identical outcomes as we will prove in this section.

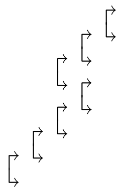
6.1. The unitary twisted QR algorithm.

The shift computation and deflation procedures are the same as in the Hessenberg case, so we discuss only the initial similarity transformation and the bulge chase.

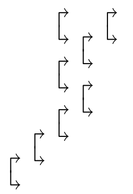
In [36] we developed twisted QR steps of arbitrary degree. Here we will restrict our attention to single (degree-one) steps and rely on the fact that one step of degree m is equivalent to m steps of degree one.

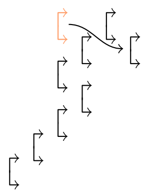
Initial similarity transformation. The form of the initial similarity transformation is determined by the relative positions of C_1 and C_2. If C_1 is positioned to the left of C_2, the initial core transformation is computed as in the Hessenberg case, that is, B_1^H transforms the second entry of $x = (A - \mu I)e_1 = (C_1 - \mu I)e_1$ to zero. If C_1 is located to the right of C_2, we need to zero the second element of $x = (I - \mu A^{-1})e_1 = (I - \mu A^H)e_1 = (I - \mu C_1^H)e_1$ (see [36] for more details). One easily checks that the core transformation $B_1^H C_1$ will do the job. In either case we get the same outcome. In the first case, $C_1 C_2$ is transformed to $B_1^H C_1 C_2 B_1$, while in the second case, $C_2 C_1$ is transformed to $B_1^H C_1 C_2 B_1$. After a fusion this becomes a product of three core transformations in a vee shaped configuration. We emphasize that the two different ways of initializing the iteration result in exactly the same three core transformations. The first step of the bulge chase will be to turn them over.

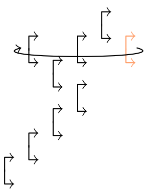
Bulge chasing. The bulge chase consists again of repeatedly applying a turnover followed by a similarity. But now the bulge can show up on either the left or the right. The bulge will be the core transformation B_{j+1} that is not sandwiched between two core transformations $\hat{C}_j$ and C_{j+2}, where $\hat{C}_j$ results from the turnover and C_{j+2} is part of the original factorization. As a consequence one can bring, via unitary similarity, the bulge to the other side, either right or left. The flow of the algorithm is illustrated below for $n = 8$ on the pattern $C_7C_6C_5C_3C_4C_2C_1$. Pictorially this product looks like

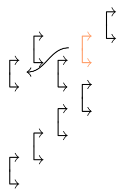
$A =$  .

An initial similarity and fusion are performed. Then a turnover is executed creating a bulge on the right. A unitary similarity transformation moves the bulge back to the left. The next turnover creates a bulge on the left.

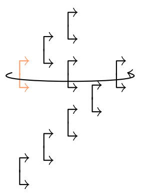

After initial similarity

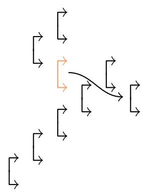

Turnover

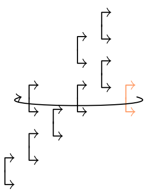

Similarity


Turnover

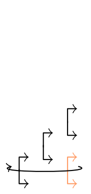
Another similarity transformation brings the bulge back to the right and the process is continued.

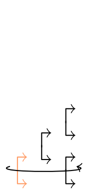

Similarity


Turnover


Similarity

Comparing the pattern of the intermediate factorizations with the original pattern, one can observe that the pattern moves upward [34]. Once we get to the bottom, we have the freedom to position the final core transformation to the left or to the right of the current pattern.


Similarity to the left

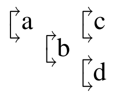

Similarity to the right

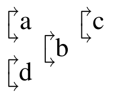
In the general matrix setting, this choice matters and can have a significant impact on the convergence rate [36]. However, in the unitary case it makes no difference as we now show.

6.2. Equivalence of twisted QR steps. Suppose the result of a QR step on the factored Hessenberg matrix equals $\hat{C}_1\hat{C}_2\cdots\hat{C}_{n-1}$. Then we will prove that the result of a twisted QR step executed on an arbitrary pattern equals $\hat{C}_{p_1}\hat{C}_{p_2}\cdots\hat{C}_{p_{n-1}}$, that is, the factors will be identical. Only the pattern could differ. This proves that in the unitary case, the pattern is of no importance: any (twisted) QR step, with identical shift, always results in identical core transformations.

We will prove by induction that the turnover executed in each step always involves the same three core transformations no matter what pattern of core transformations was considered originally. To initiate the induction, recall the discussion of the initial similarity transformation. There we noted that after the first fusion, the three core transformations at the top of the configuration are always the same regardless of the original ordering. These are the core transformations that participate in the first turnover operation.

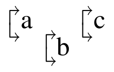
Now we do the induction step. Studying our example above, we find that at each step we have one of two configurations, either

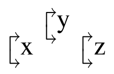


or


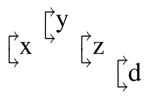
.

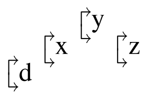
In either case we do the turnover



$\Rightarrow$


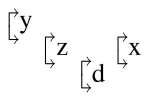
resulting in

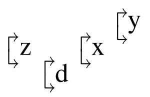


or


.

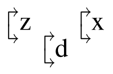
In the left case we move x from the left to the right, and in the right case, we move z from right to left, resulting in



or


.

In both cases the next turnover will involve



.

We note also that the transformation y will not participate in any subsequent turnovers in this iteration. It will become one of the core transformations in the representation of the next iterate, and it is the same in both cases. This completes the induction.

After the final turnover and similarity, a single fusion concludes the chasing step. It is easy to check that the flexibility allowing to execute the fusion left or right has no effect on the contents of the final bottom core transformation.

6.3. QZ methods on unitary pencils. Let $A = C_1 \cdots C_{n-1}$ be a factorization into core transformations of a unitary Hessenberg matrix. The eigenvalues of A can be computed as eigenvalues from any pencil (U, V^H), where U is a unitary matrix constructed from some core transformations C_i and V by the remaining core transformations in any ordering [35]. For any such pencil there is a twisted QZ algorithm to compute the eigenvalues. In the unitary case, the twisted QZ algorithm on (U, V^H) is identical to a twisted QR algorithm on the product UV. Since all twisted QR methods are identical, we can conclude that all twisted QZ algorithms are essentially the same and no different from the Hessenberg QR algorithm in the unitary case. The algorithm of Bunse-Gerstner and Elsner [5] falls into this category.

7. Numerical experiments.

The computations were executed on an Intel Core i5-3570 CPU running at 3.40 GHz with 8 GB of memory. GFortran 4.6.3 was used to compile the Fortran codes. We compared our codes with LAPACK 3.5.0's ZHSEQR and DHSEQR codes and Ammar, Reichel, and Sorensen's unitary divide-and-conquer (D&C) code [2].

Our algorithm is backward stable. Therefore it should always produce results that have a backward error that is a modest multiple of the machine precision. Since the eigenvalues of a unitary matrix are perfectly conditioned, such a tiny backward error guarantees that the computed eigenvalues are accurate to (nearly) machine precision. The following experiments verify these expectations.

For most of our test matrices the exact eigenvalues are not known. Therefore we use the maximum residual

$$\max_i \| Av_i - \lambda_i v_i \|_2, \tag{7.1}$$

with (λ_i, v_i) the computed eigenpairs, as our measure of accuracy. This is a backward error, and if it is small, it guarantees an equally small error in the eigenvalues. In one example, Example 7.5, the eigenvalues are known exactly. In that example, and only in that example, we used the maximum error in the computed eigenvalues as our measure of accuracy.

In the examples below, the times shown are for computing eigenvalues only. We also computed eigenvectors in order to compute the residuals (7.1) and check our backward stability claim. The unitary divide-and-conquer code also has the capability of computing eigenvectors. That code is fast, much faster than ours if also eigenvectors are computed, but it is also much less accurate.

We test both the complex single shift code and the real double shift code. For the double shift code we make the examples real. Since divide-and-conquer is only available in a complex version, we use that version in both the real and complex tests.

EXAMPLE 7.1. This example was taken from Nguyen, Nguyen, and Vu [29, Figure 1]. The rotators in the unitary Hessenberg's factorization are determined by fixed numbers m and p for all rotators as

$$C_k(k:k+1,k) = \begin{bmatrix} (-1)^{k-1} p \sqrt{1 - m^{\frac{2}{\beta k}}} \\ \sqrt{m^{\frac{2}{\beta k}}} \end{bmatrix} \text{ and } C_n(n,n) = (-1)^{n-1},$$

where m is drawn from a uniform distribution in $(0, 1)$ and p is taken randomly (uniformly distributed) on the unit circle. Runtime and accuracy are displayed in Figure 7.1 (single shift code) and in Figure 7.4 (double shift code). Our code is about three times faster than D&C

and many times faster than LAPACK at large dimensions. Our accuracy is comparable to that of LAPACK and much better than that of D&C.

EXAMPLE 7.2 (Type I). This example appeared in Ammar et al. [2], Gu et al. [25], and Gemignani [18, Type I]. The rotations are defined for varying m and p drawn from a uniform distribution in $(0, 1)$ and on the unit circle, respectively,

$$C_k(k:k+1,k) = \begin{bmatrix} (-1)^{k-1}mp \\ \sqrt{1-m^2} \end{bmatrix} \text{ and } C_n(n,n) = 1. \tag{7.2}$$

The results are presented in Figure 7.2 (single shift code) and in Figure 7.5 (double shift code). On this class of problems, D&C is fastest for large problems and exhibits better than $\mathcal{O}(n^2)$ performance. However, it is not very accurate. The same trend is also observed in the next two examples.

EXAMPLE 7.3 (Type II). This example originates from Gemignani [18, Type II]. The rotations are real, and the angle θ is drawn from a uniform distribution in $(0, \pi)$

$$C_k(k:k+1,k) = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \text{ and } C_n(n,n) = 1.$$

In addition all rotations are normalized to ensure that the matrix is unitary. By choosing n odd and C_n the identity, one eigenvalue is forced to be -1. The results are qualitatively similar to those of Example 7.2. We therefore choose not to display them.

EXAMPLE 7.4 (Type III). This example can be found in Gu et al. [25, Type III] and Gemignani [18, Type III] and is a block version of type I. All four blocks use the same matrix of type I. These blocks are then coupled by a rotation (7.2) with small sine; we choose $m = 0.001$ and p randomly on the unit circle. The results are qualitatively similar to those of Example 7.2. We therefore choose not to display them. In this class of problems we encountered one large real example for which our code failed. It hit the iteration limit before it was able to resolve a cluster of sixteen eigenvalues within 10^{-15} of -1.

EXAMPLE 7.5 (Known eigenvalues). The unitary Hessenberg matrices are obtained from solving an inverse eigenvalue problem with the software from Mach, Van Barel, and Vandebril [27], where the eigenvalues are uniformly distributed on the unit circle. Our measure of error is the forward error, i.e., the maximum error in the computed eigenvalues. The results are shown in Figure 7.3 (single shift code) and in Figure 7.6 (double shift code). In the complex case this method of matrix generation is not sufficiently accurate for matrices of dimension larger than 100, so only small dimensions are shown.

8. Conclusions.

A fast and backward stable implementation of the implicitly shifted QR algorithm for computing eigenvalues of unitary matrices was presented. Moreover, we proved that all fast QR- and QZ-like methods for solving the unitary eigenvalue problem are essentially the same.

Acknowledgment. We thank Alan Edelman, Massachusetts Institute of Technology, for requesting a fast unitary eigenvalue solver. We also thank Nick Trefethen, University of Oxford, and the referees for their constructive remarks, which helped us to improve the paper.

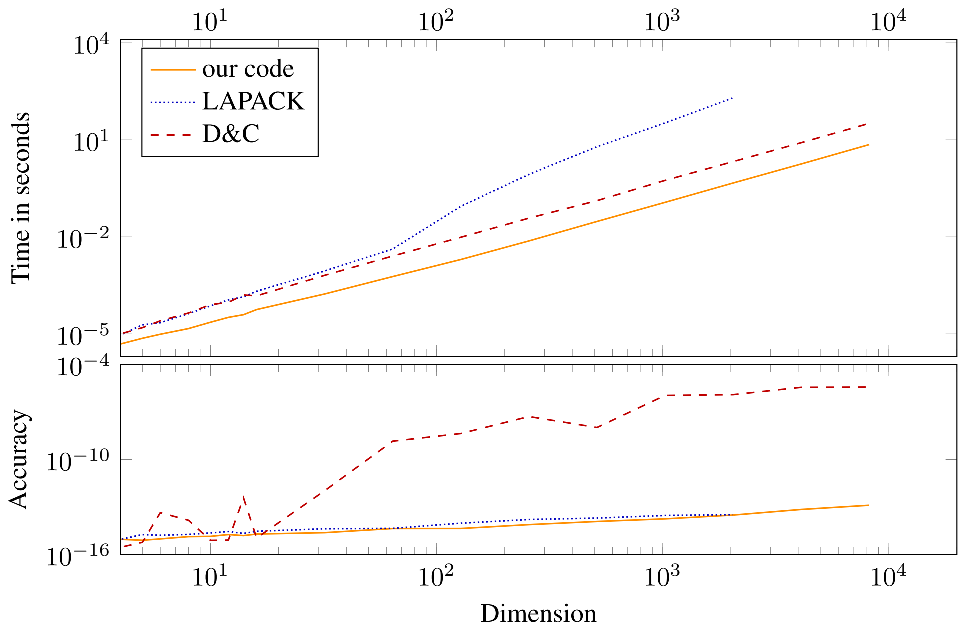


FIG. 7.1. *Runtime and accuracy for Example 7.1*, $\beta = 2$.

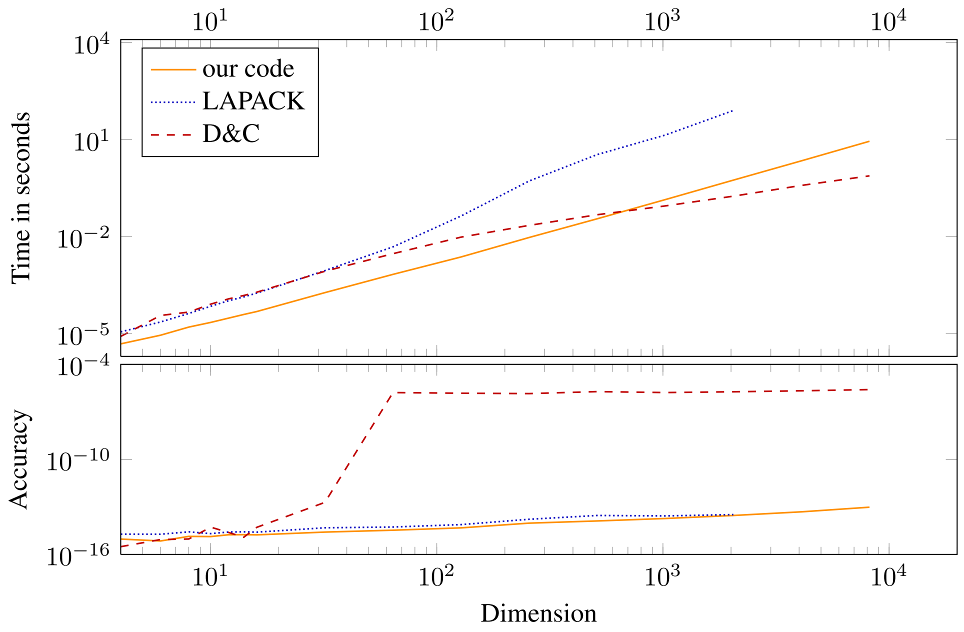


FIG. 7.2. *Runtime and accuracy for Example 7.2.*

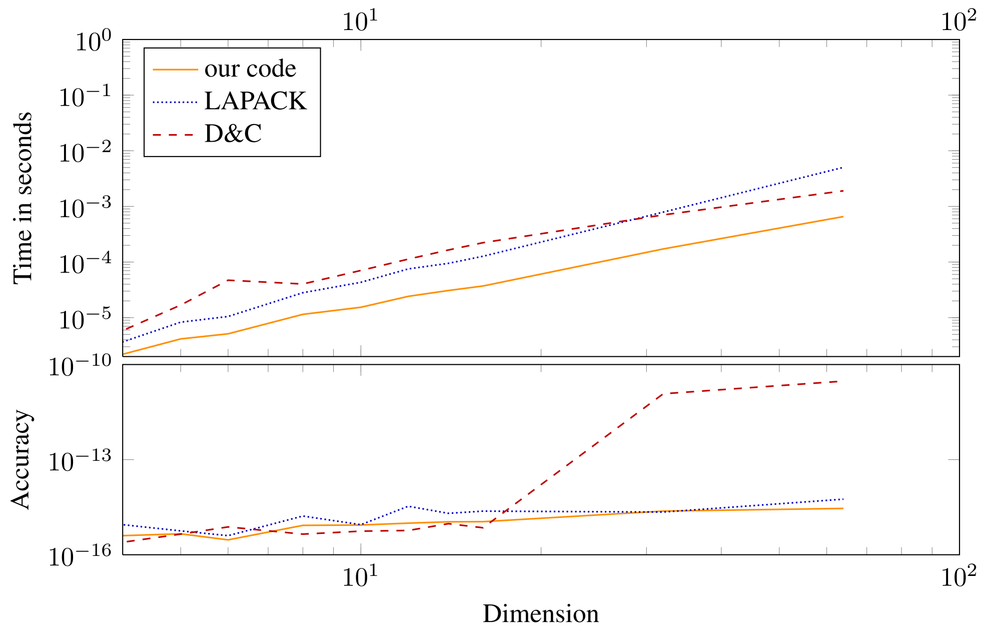


FIG. 7.3. *Runtime and accuracy for Example 7.5.*

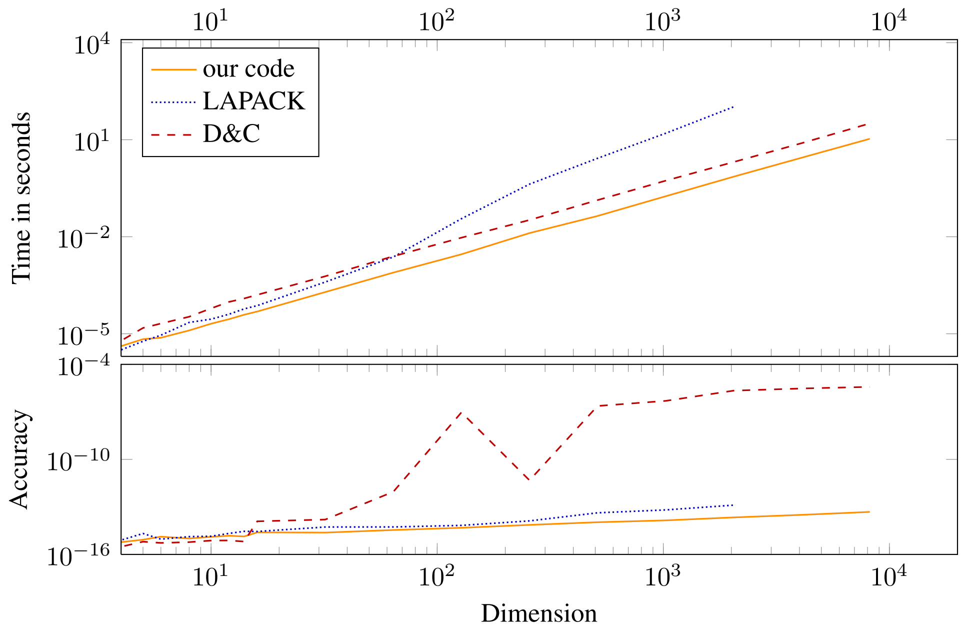


FIG. 7.4. *Runtime and accuracy for Example 7.1, $\beta = 2$, double shift code.*

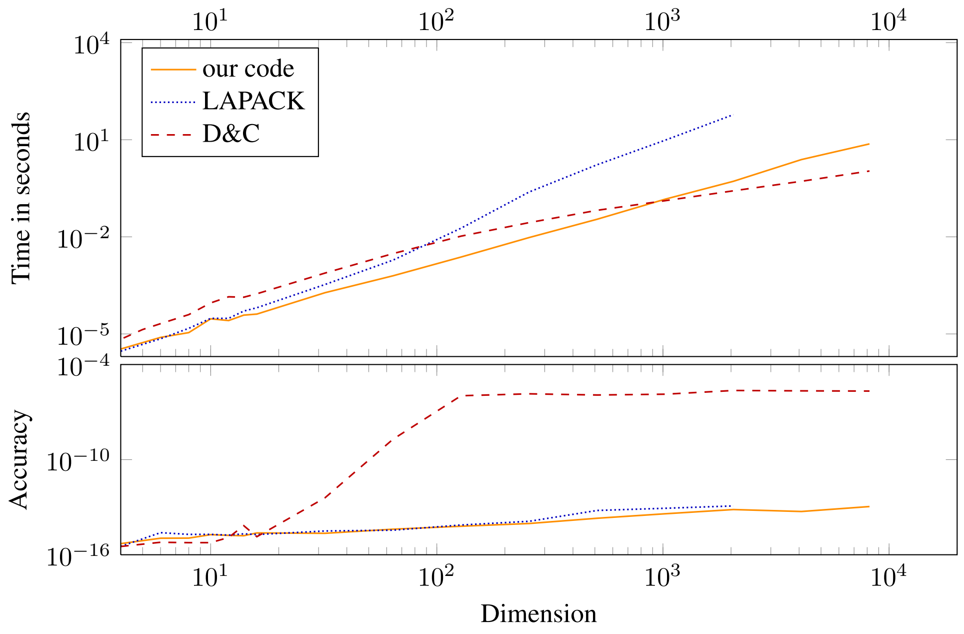


FIG. 7.5. *Runtime and accuracy for Example 7.2, double shift code.*

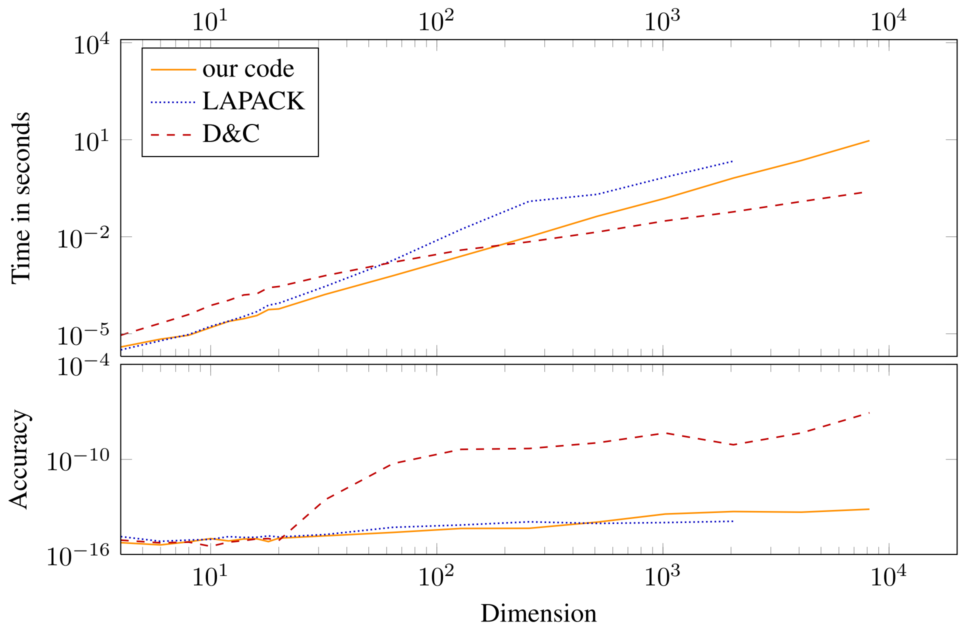


FIG. 7.6. *Runtime and accuracy for Example 7.5, double shift code.*

REFERENCES

[1] G. S. AMMAR, W. B. GRAGG, AND L. REICHEL, *On the eigenproblem for orthogonal matrices*, in Proceedings of the 25th IEEE Conference on Decision & Control, IEEE Conference Proceedings, Los Alamitos, 1986, pp. 1963–1966.

[2] G. S. AMMAR, L. REICHEL, AND D. C. SORENSEN, *An implementation of a divide and conquer algorithm for the unitary eigenproblem*, ACM Trans. Math. Software, 18 (1992), pp. 292–307.

[3] ———, *Corrigendum: Algorithm 730: An implementation of a divide and conquer algorithm for the unitary eigenproblem*, ACM Trans. Math. Software, 20 (1994), p. 161.

[4] J. L. AURENTZ, T. MACH, R. VANDEBRIL, AND D. S. WATKINS, *Fast and backward stable computation of roots of polynomials*, SIAM J. Matrix Anal. Appl., to appear, 2015.

[5] 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.

[6] A. BUNSE-GERSTNER AND C. HE, *On a Sturm sequence of polynomials for unitary Hessenberg matrices*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1043–1055.

[7] M. J. CANTERO, L. MORAL, AND L. VELAZQUEZ, *Five-diagonal matrices and zeros of orthogonal polynomials on the unit circle*, Linear Algebra Appl., 362 (2003), pp. 29–56.

[8] R. CRUZ-BARROSO AND S. DELVAUX, *Orthogonal Laurent polynomials on the unit circle and snake-shaped matrix factorizations*, J. Approx. Theory, 161 (2009), pp. 65–87.

[9] G. CYBENKO, *Computing Pisarenko frequency estimates*, in Proceedings of the 1984 Conference on Information Systems and Sciences, Princeton University, Princeton, 1985, pp. 587–591.

[10] R. J. A. DAVID AND D. S. WATKINS, *Efficient implementation of the multishift QR algorithm for the unitary eigenvalue problem*, SIAM J. Matrix Anal. Appl., 28 (2007), pp. 623–633.

[11] S. DELVAUX AND M. VAN BAREL, *Eigenvalue computation for unitary rank structured matrices*, J. Comput. Appl. Math., 213 (2008), pp. 268–287.

[12] ———, *Unitary rank structured matrices*, J. Comput. Appl. Math., 215 (2008), pp. 268–287.

[13] P. J. EBERLEIN AND C. P. HUANG, *Global convergence of the QR algorithm for unitary matrices with some results for normal matrices*, SIAM J. Numer. Anal., 12 (1975), pp. 97–104.

[14] A. EDELMAN, Private communication, June 2014.

[15] S. M. FALLAT, M. FIEDLER, AND T. L. MARKHAM, *Generalized oscillatory matrices*, Linear Algebra Appl., 359 (2003), pp. 79–90.

[16] J. G. F. FRANCIS, *The QR transformation: a unitary analogue to the LR transformation. I.*, Comput. J., 4 (1961), pp. 265–271.

[17] ———, *The QR transformation. II.*, Comput. J., 4 (1962), pp. 332–345.

[18] L. GEMIGNANI, *A unitary Hessenberg QR-based algorithm via semiseparable matrices*, J. Comput. Appl. Math., 184 (2005), pp. 505–517.

[19] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, 4th ed., Johns Hopkins University Press, Baltimore, 2013.

[20] W. B. GRAGG, *Positive definite Toeplitz matrices, the Hessenberg process for isometric operators, and Gaussian quadrature on the unit circle*, in Numerical Methods of Linear Algebra, E. S. Nikolaev, ed., Moscow University Press, Moscow, 1982, pp. 16–32.

[21] ———, *The QR algorithm for unitary Hessenberg matrices*, J. Comput. Appl. Math., 16 (1986), pp. 1–8.

[22] ———, *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.

[23] ———, *Stabilization of the uhqr-algorithm*, in Advances in Computational Mathematics, Proc. of the Int. Symposium on Computational Mathematics, Z. Chen, Y. Li, C. A. Micchelli, and Y. Xu, eds., Lecture Notes in Pure and Applied Mathematics, 202, Dekker, New York, 1999, pp. 139–154.

[24] W. B. GRAGG AND L. REICHEL, *A divide and conquer method for unitary and orthogonal eigenproblems*, Numer. Math., 57 (1990), pp. 695–718.

[25] M. GU, R. GUZZO, X.-B. CHI, AND X.-Q. CAO, *A stable divide and conquer algorithm for the unitary eigenproblem*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 385–404.

[26] R. KILLIP AND I. NENCIU, *Matrix models for circular ensembles*, Int. Math. Res. Notices, 2004 (2004), pp. 2665–2701.

[27] T. MACH, M. VAN BAREL, AND R. VANDEBRIL, *Inverse eigenvalue problems linked to rational Arnoldi, and rational nonsymmetric Lanczos*, J. Comput. Appl. Math., 272 (2014), pp. 377–398.

[28] T. MACH AND R. VANDEBRIL, *On deflations in extended QR algorithms*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 559–579.

[29] H. NGUYEN, O. NGUYEN, AND V. VU, *On the number of real roots of random polynomials*, Preprint on arXiv, 2014. http://arxiv.org/abs/1402.4628

[30] V. F. PISARENKO, *The retrieval of harmonics from a covariance function*, Geophys. J. Roy. Astron. Soc., 33 (1973), pp. 347–366.

[31] B. SIMON, *CMV matrices: five years after*, J. Comput. Appl. Math., 208 (2007), pp. 120–154.

[32] M. STEWART, *Stability properties of several variants of the unitary Hessenberg QR-algorithm*, in Structured Matrices in Mathematics, Computer Science and Engineering II, V. Olshevsky, ed., vol. 281 of Contemporary Mathematics, American Mathematical Society, Providence, 2001, pp. 57–72.
[33] ———, *An error analysis of a unitary Hessenberg QR algorithm*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 40–67.
[34] R. VANDEBRIL, *Chasing bulges or rotations? A metamorphosis of the QR-algorithm*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 217–247.
[35] R. VANDEBRIL AND D. S. WATKINS, *An extension of the QZ algorithm beyond the Hessenberg-upper triangular pencil*, Electron. Trans. Numer. Anal., 40 (2012), pp. 17–35.
http://etna.mcs.kent.edu/volumes/2011-2020/vol40/abstract.php?vol=40&pages=17-35
[36] ———, *A generalization of the multishift QR algorithm*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 759–779.
[37] T. L. WANG AND W. B. GRAGG, *Convergence of the shifted QR algorithm, for unitary Hessenberg matrices*, Math. Comp., 71 (2002), pp. 1473–1496.
[38] ———, *Convergence of the unitary QR algorithm with unimodular Wilkinson shift*, Math. Comp., 72 (2003), pp. 375–385.
[39] D. S. WATKINS, *Some perspectives on the eigenvalue problem*, SIAM Rev., 35 (1993), pp. 430–471.
[40] ———, *Fundamentals of Matrix Computations*, 3rd ed., Wiley, Hoboken, 2010.