A Fast Eigenvalue Algorithm for Hankel Matrices *

Franklin T. Luk
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180 USA

Sanzheng Qiao
Department of Computing and Software
McMaster University
Hamilton, Ontario L8S 4K1 Canada

Dedicated to Robert J. Plemmons on the occasion of his 60th birthday

Abstract

We present an algorithm that can find all the eigenvalues of an $n \times n$ complex Hankel matrix in $O(n^2 \log n)$ operations. Our scheme consists of an $O(n^2 \log n)$ Lanczos-type tridiagonalization procedure and an $O(n)$ QR-type diagonalization method.

Keywords: Hankel matrix, Toeplitz matrix, circulant matrix, fast Fourier transform, Lanczos tridiagonalization, eigenvalue decomposition, complex-symmetric matrix, complex-orthogonal transformations.

1 INTRODUCTION

The eigenvalue decomposition of a structured matrix has important applications in signal processing. Common occurring structures include an $n \times n$ Hankel matrix:

$$H = \begin{pmatrix} h_1 & h_2 & \dots & h_{n-1} & h_n \\ h_2 & h_3 & \dots & h_n & h_{n+1} \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ h_{n-1} & h_n & \dots & h_{2n-3} & h_{2n-2} \\ h_n & h_{n+1} & \dots & h_{2n-2} & h_{2n-1} \end{pmatrix}, \tag{1}$$

or an $n \times n$ Toeplitz matrix:

$$T = \begin{pmatrix} t_n & t_{n-1} & \dots & t_2 & t_1 \\ t_{n+1} & t_n & \dots & t_3 & t_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ t_{2n-2} & t_{2n-3} & \dots & t_n & t_{n-1} \\ t_{2n-1} & t_{2n-2} & \dots & t_{n+1} & t_n \end{pmatrix}. \tag{2}$$

*This work was partly supported by the Natural Sciences and Engineering Research Council of Canada under grant OGP0046301

There is extensive literature on inverting such matrices or solving such linear systems. However, efficient eigenvalue algorithms for structured matrices are still under development. Cybenko and Van Loan [3] proposed an algorithm for computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix. Their algorithm is based on the Levinson-Durbin Algorithm and Newton's method; it requires up to $O(n^2)$ floating-point operations per Newton iteration and heuristically $O(\log n)$ iterations. Building on their work [3], Trench [8] presented an algorithm for Hermitian Toeplitz matrices. His algorithm requires $O(n^2)$ operations per eigenvalue-eigenvector pair. In this paper, we study the eigenvalue problem of a Hankel matrix. Taking advantage of two properties, namely that a complex Hankel matrix is symmetric and that a permuted Hankel matrix can be embedded in a circulant matrix, we develop an $O(n^2 \log n)$ algorithm that can find all the eigenvalues of an $n \times n$ Hankel matrix. We should point out that our new method is a theoretical contribution; considerable work is required to develop a practical software. An error analysis of this algorithm can be found in [6].

Our paper is organized as follows. How to exploit complex-symmetry is presented in Section 2, and how to construct complex-orthogonal transformations in Section 3. An $O(n \log n)$ scheme for multiplying a Hankel matrix and a vector is described in Section 4, and an $O(n^2 \log n)$ Lanczos tridiagonalization process in Section 5. A QR procedure to diagonalize a complex-symmetric tridiagonal matrix is given in Section 6, followed by an overall computational procedure and two numerical examples in Section 7.

2 COMPLEX SYMMETRY

Our idea is to take advantage of the symmetry of a Hankel matrix. In general, an eigenvalue decomposition of H (assuming that it is nondefective) is given by

$$H = XDX^{-1} \tag{3}$$

where D is diagonal and X is nonsingular. Note that the following Hankel matrix is defective:

$$H = \begin{pmatrix} 2 & i \\ i & 0 \end{pmatrix}.$$

We will pick the matrix X to be *complex-orthogonal*; that is,

$$XX^{\mathrm{T}} = I. \tag{4}$$

So, $H = XDX^{\mathrm{T}}$. We apply a special Lanczos tridiagonalization to the Hankel matrix (assuming that the Lanczos process does not prematurely terminate):

$$H = QJQ^{\mathrm{T}}, \tag{5}$$

where Q is complex-orthogonal and J is complex-symmetric tridiagonal. Then we diagonalize J:

$$J = WDW^{\mathrm{T}},$$

where W is complex-orthogonal and D is diagonal. Thus, we get (3) with

$$X = QW.$$

The dominant cost of the Lanczos method is matrix-vector multiplication which in general takes $O(n^2)$ operations. We propose a fast $O(n \log n)$ Hankel matrix-vector multiplication algorithm. Thus we can tridiagonalize a Hankel matrix in $O(n^2 \log n)$ operations. The resulting tridiagonal matrix is complex-symmetric. In order to maintain its symmetric and tridiagonal structure, we use the complex-orthogonal transformations in the QR iteration.

3 COMPLEX-ORTHOGONAL TRANSFORMATIONS

A basic operation in solving eigenvalue problems is the introduction of zeros into 2×1 vectors using 2×2 transformations. From its definition in (4), we derive the general form of a 2×2 complex-orthogonal matrix as

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} c & s \\ s & -c \end{pmatrix},$$

where $c^2 + s^2 = 1$. Here, we choose the nonsymmetric version:

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}. \tag{6}$$

In the real case, the transformation G of (6) reduces to a Givens rotation. Given a complex 2-element vector

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \tag{7}$$

where $x_1^2 + x_2^2 \neq 0$, the following algorithm computes the nonsymmetric transformation G of (6) so that

$$G\mathbf{x} = \begin{pmatrix} \sqrt{x_1^2 + x_2^2} \\ 0 \end{pmatrix}. \tag{8}$$

For more details, see Luk and Qiao [5] and Vandevoorde [10].

Algorithm 1 (Complex-Orthogonal Transformation) *Given a complex vector* $\mathbf{x}$ *of (7), this algorithm computes the parameters* c *and* s *for the complex-orthogonal transformation* G *of (6), so that (8) holds.*

if $(|x_1| > |x_2|)$
    $t = x_2/x_1$; $c = 1/\sqrt{1+t^2}$; $s = t \cdot c$;
else
    $\tau = x_1/x_2$; $s = 1/\sqrt{1+\tau^2}$; $c = \tau \cdot s$;
end if.

This algorithm will be used in the complex-orthogonal diagonalization in Section 6.

4 FAST HANKEL MATRIX-VECTOR PRODUCT

In this section, we describe an $O(n \log n)$ algorithm for multiplying an $n \times n$ Hankel matrix into an n-element vector. We begin with some additional notations. Let

$$\mathbf{h} \equiv (h_1 \quad h_2 \quad h_3 \quad \ldots \quad h_{2n-1})^{\mathrm{T}}$$

and

$$\mathbf{t} \equiv (t_1 \quad t_2 \quad t_3 \quad \ldots \quad t_{2n-1})^{\mathrm{T}}$$

denote two $(2n-1)$-element vectors specifying the $n \times n$ Hankel matrix $H(\mathbf{h})$ and the $n \times n$ Toeplitz matrix $T(\mathbf{t})$ of equations (1) and (2), respectively.

First, we permute a Hankel matrix into a Toeplitz matrix. Let Π represent an $n \times n$ permutation matrix that reverses all columns of H in postmultiplication:

$$\Pi = \begin{pmatrix} 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & \ldots & 1 & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & 1 & \ldots & 0 & 0 \\ 1 & 0 & \ldots & 0 & 0 \end{pmatrix};$$

that is,

$$H(\mathbf{h})\,\Pi = T(\mathbf{h}). \tag{9}$$

The simplicity of equation (9) explains why we start with H_{11} (respectively T_{1n}) when we define the vector $\mathbf{h}$ (respectively $\mathbf{t}$).

Next, we embed the Toeplitz matrix $T(\mathbf{h})$ in a larger circulant matrix. Consider a $(2n-1) \times (2n-1)$ circulant matrix:

$$C = \begin{pmatrix} c_1 & c_{2n-1} & c_{2n-2} & \ldots & c_3 & c_2 \\ c_2 & c_1 & c_{2n-1} & \ldots & c_4 & c_3 \\ c_3 & c_2 & c_1 & \ldots & c_5 & c_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{2n-2} & c_{2n-3} & c_{2n-4} & \ldots & c_1 & c_{2n-1} \\ c_{2n-1} & c_{2n-2} & c_{2n-3} & \ldots & c_2 & c_1 \end{pmatrix} \equiv C(\mathbf{c}),$$

where

$$\mathbf{c} \equiv (c_1 \quad c_2 \quad c_3 \quad \ldots \quad c_{2n-1})^{\mathrm{T}}.$$

Note that $\mathbf{c}$ represents the first column of C. Consider a special choice of this vector:

$$\widehat{\mathbf{c}} = (h_n \quad h_{n+1} \quad h_{n+2} \quad \ldots \quad h_{2n-1} \quad h_1 \quad h_2 \quad \ldots \quad h_{n-1})^{\mathrm{T}}. \tag{10}$$

Then

$$C(\widehat{\mathbf{c}}) = \begin{pmatrix} h_n & h_{n-1} & h_{n-2} & \dots & h_1 & h_{2n-1} & h_{2n-2} & \dots & h_{n+1} \\ h_{n+1} & h_n & h_{n-1} & \dots & h_2 & h_1 & h_{2n-1} & \dots & h_{n+2} \\ h_{n+2} & h_{n+1} & h_n & \dots & h_3 & h_2 & h_1 & \dots & h_{n+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{2n-1} & h_{2n-2} & h_{2n-3} & \dots & h_n & h_{n-1} & h_{n-2} & \dots & h_1 \\ h_1 & h_{2n-1} & h_{2n-2} & \dots & h_{n+1} & h_n & h_{n-1} & \dots & h_2 \\ h_2 & h_1 & h_{2n-1} & \dots & h_{n+2} & h_{n+1} & h_n & \dots & h_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{n-1} & h_{n-2} & h_{n-3} & \dots & h_{2n-1} & h_{2n-2} & h_{2n-3} & \dots & h_n \end{pmatrix},$$

where the leading $n \times n$ principal submatrix is $T(\mathbf{h})$. This technique of embedding a Toeplitz matrix in a larger circulant matrix to achieve fast computation is widely used in preconditioning methods [1, 7].

Given an n-element vector:

$$\mathbf{w} = (\, w_1 \quad w_2 \quad w_3 \quad \dots \quad w_n \,)^{\mathrm{T}}, \tag{11}$$

we want to compute the matrix-vector product

$$\mathbf{p} = H\mathbf{w}. \tag{12}$$

We see that

$$\mathbf{p} = H(\mathbf{h})\mathbf{w} = T(\mathbf{h})(\Pi\mathbf{w}).$$

Let $\widehat{\mathbf{w}}$ denote a special $(2n-1)$-element vector:

$$\widehat{\mathbf{w}} = (\, w_n \quad w_{n-1} \quad \dots \quad w_1 \quad 0 \quad \dots \quad 0 \,)^{\mathrm{T}}, \tag{13}$$

which can be obtained from the n-vector $\Pi\mathbf{w}$ by appending it with $n-1$ zeros. So $\mathbf{p}$ is given by the first n elements of the product $\mathbf{y}$, defined by

$$\mathbf{y} \equiv C(\widehat{\mathbf{c}})\widehat{\mathbf{w}}.$$

This circulant matrix-vector multiplication can be efficiently computed via the Fast Fourier Transform [9] (FFT); namely,

$$C(\widehat{\mathbf{c}})\widehat{\mathbf{w}} = \mathrm{ifft}(\mathrm{fft}(\widehat{\mathbf{c}}) .* \mathrm{fft}(\widehat{\mathbf{w}}))$$

where $\mathrm{fft}(\mathbf{v})$ denotes a one-dimensional FFT of a vector $\mathbf{v}$, $\mathrm{ifft}(\mathbf{v})$ a one-dimensional inverse FFT of $\mathbf{v}$, and ".*" a componentwise multiplication of two vectors.

Algorithm 2 (Fast Hankel Matrix-Vector Product) *Given a vector* $\mathbf{w}$ *in (11) and a Hankel matrix* H *in (1), this algorithm computes the product vector* $\mathbf{p}$ *of (12) by using the Fast Fourier Transform.*

1. Define a $(2n-1)$-element vector $\widehat{\mathbf{c}}$ as in equation (10):

$$\widehat{\mathbf{c}} = (\, h_n \quad h_{n+1} \quad h_{n+2} \quad \ldots \quad h_{2n-1} \quad h_1 \quad h_2 \quad \ldots \quad h_{n-1} \,)^{\mathrm{T}}.$$

2. Define a $(2n-1)$-element vector $\widehat{\mathbf{w}}$ as in equation (13):

$$\widehat{\mathbf{w}} = (\, w_n \quad w_{n-1} \quad \ldots \quad w_1 \quad 0 \quad \ldots \quad 0 \,)^{\mathrm{T}}.$$

3. Compute a $(2n-1)$-element vector $\mathbf{y}$ by

$$\mathbf{y} = \mathrm{ifft}(\mathrm{fft}(\widehat{\mathbf{c}}) .* \mathrm{fft}(\widehat{\mathbf{w}})).$$

4. Let $\mathbf{y} = (\, y_1 \quad y_2 \quad \ldots \quad y_{2n-2} \quad y_{2n-1} \,)^{\mathrm{T}}$. Then the desired n-element product $\mathbf{p}$ of (12) is given by

$$\mathbf{p} = (\, y_1 \quad y_2 \quad \ldots \quad y_{n-1} \quad y_n \,)^{\mathrm{T}}.$$

How much work does this algorithm require? In general, a complex matrix-vector multiplication involves $8n^2$ real floating-point operations (flops) and an FFT of a vector of size n costs $5n\log(n)$ flops. In Algorithm 2, each of the two FFT requires $5(2n-1)\log(2n-1)$ flops, the pointwise multiplication $6(2n-1)$ flops, and the inverse FFT $5(2n-1)\log(2n-1)$ flops. The total cost of $\mathrm{ifft}(\mathrm{fft}(\widehat{\mathbf{c}}) .* \mathrm{fft}(\widehat{\mathbf{w}}))$ equals $30n\log(n)+O(n)$ flops. Thus, Algorithm 2 becomes superior to general matrix-vector multiplication when $n > 16$.

5 LANCZOS TRIDIAGONALIZATION

In this section, we derive a tridiagonalization method for H based on a Lanczos iterative process. Our goal is to find a complex-orthogonal matrix Q so that equation (5) holds. Note that

$$HQ = QJ. \tag{14}$$

Let

$$Q = (\, \mathbf{q}_1 \quad \mathbf{q}_2 \quad \mathbf{q}_3 \quad \ldots \quad \mathbf{q}_n \,)$$

and

$$J = \begin{pmatrix} \alpha_1 & \beta_1 & & & 0 \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_{n-1} \\ 0 & & & \beta_{n-1} & \alpha_n \end{pmatrix}. \tag{15}$$

Consider the kth column of both sides of (14). We have

$$H\mathbf{q}_k = \beta_{k-1}\mathbf{q}_{k-1} + \alpha_k\mathbf{q}_k + \beta_k\mathbf{q}_{k+1}, \tag{16}$$

where

$$\beta_0\mathbf{q}_0 = 0 \quad \text{and} \quad \beta_n\mathbf{q}_{n+1} = 0.$$

Since Q is complex-orthogonal, i.e., $\mathbf{q}_i^{\mathrm{T}}\mathbf{q}_j = \delta_{ij}$, we get

$$\alpha_k = \mathbf{q}_k^{\mathrm{T}} H \mathbf{q}_k.$$

Equation (16) implies that

$$\beta_k\mathbf{q}_{k+1} = H\mathbf{q}_k - \alpha_k\mathbf{q}_k - \beta_{k-1}\mathbf{q}_{k-1}.$$

Setting

$$\mathbf{r}_k = H\mathbf{q}_k - \alpha_k\mathbf{q}_k - \beta_{k-1}\mathbf{q}_{k-1},$$

we get

$$\beta_k = \sqrt{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k} \quad \text{and} \quad \mathbf{q}_{k+1} = (1/\beta_k)\mathbf{r}_k,$$

for $k < n$. We have thus derived a generic Lanczos tridiagonalization method. Note that we have used only the property that the matrix H is complex-symmetric.

Algorithm 3 (Lanczos Tridiagonalization) *Given an $n \times n$ complex-symmetric matrix H, this algorithm computes a complex-orthogonal matrix Q such that $H = QJQ^{\mathrm{T}}$, where J is a complex-symmetric tridiagonal matrix as shown in (15).*

Initialize $\mathbf{q}_1$ such that $\mathbf{q}_1^{\mathrm{T}}\mathbf{q}_1 = 1$;
Set $\mathbf{r}_0 = \mathbf{q}_1$; $\beta_0 = 1$; $\mathbf{q}_0 = 0$; $k = 0$;

while $(\beta_k \neq 0)$
  $\mathbf{q}_{k+1} = (1/\beta_k)\mathbf{r}_k$;
  $k = k + 1$;
  $\alpha_k = \mathbf{q}_k^{\mathrm{T}} H \mathbf{q}_k$;
  $\mathbf{r}_k = H\mathbf{q}_k - \alpha_k\mathbf{q}_k - \beta_{k-1}\mathbf{q}_{k-1}$;
  $\beta_k = \sqrt{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$;
end

If all $\beta_k \neq 0$, then Algorithm 3 runs until $k = n$. The dominant cost is the Hankel matrix-vector product $H\mathbf{q}_k$. Using Algorithm 2 to perform this task, we obtain an $O(n^2 \log n)$ tridiagonalization algorithm. Consider a Krylov matrix K, defined by

$$K = K(H, \mathbf{q}_1, n) \equiv (\, \mathbf{q}_1 \quad H\mathbf{q}_1 \quad H^2\mathbf{q}_1 \quad \ldots \quad H^{n-1}\mathbf{q}_1 \,).$$

We get a complex-orthogonal decomposition [2] of K:

$$K = QR, \tag{17}$$

where $R = (\, \mathbf{e}_1 \quad J\mathbf{e}_1 \quad J^2\mathbf{e}_1 \quad \ldots \quad J^{n-1}\mathbf{e}_1 \,)$. Checking the diagonal elements of the upper triangular matrix R, we find that they are nonzero if and only if $\beta_k \neq 0$, for $k = 1, 2, \ldots, n$. Cullum and Willoughby [2] show that the decomposition (17), if it exists, is essentially unique in the sense that if

$$H = Q_1 R_1 \quad \text{and} \quad H = Q_2 R_2$$

are two different complex-orthogonal decompositions of H, then

$$Q_2 = Q_1 S \quad \text{and} \quad R_2 = S R_1,$$

where S is a signature matrix, i.e., $S = \mathrm{diag}(\pm 1)$. Note that some nonsingular complex-symmetric matrix does not have a complex-orthogonal decomposition (17); an example is the following matrix:

$$\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}. \tag{18}$$

If Algorithm 3 stops at $k < n$, then we are stuck in an invariant subspace. It is possible that $\beta_k = 0$ even when $\mathbf{r}_k \neq 0$ for some $k < n$. However, we rarely get an exact zero β_k in practice. But a small β_k relative to $\|\mathbf{r}_k\|$ could create difficulties. See Luk and Qiao [6] for details.

Theorem 1 *If there exists a complex-orthogonal decomposition (17) of the Krylov matrix $K(H, \mathbf{q}_1, n)$ and if R is nonsingular, then Algorithm 3 runs until $k = n$.*

6 COMPLEX-ORTHOGONAL DIAGONALIZATION

In this section, we describe a QR-type algorithm for diagonalizing a complex-symmetric tridiagonal matrix. Basically, we use the implicit QR method with the Wilkinson shift [4] and replace all unitary transformations by complex-orthogonal transformations. However, it should be pointed out that this QR-type algorithm offers less guarantee for convergence than the standard QR method. See Cullum and Willoughby [2] and Vandevoorde [10] for theoretical results on convergence. Also, Cullum and Willoughby [2] present extensive numerical experiments. Basically, if a complex-symmetric tridiagonal matrix J is nonsingular, irreducible and nondefective, and if it has no eigenvalues equal in magnitude, then the following algorithm with all shifts equal to zero will converge. For example, the 2-by-2 complex-symmetric matrix in (18) is nonsingular, irreducible, and nondefective, but it has two eigenvalues, $1 \pm i$, that are equal in magnitude. The following algorithm fails because this matrix cannot be triangularized by a complex-orthogonal matrix. If the matrix Q is

not desired, this algorithm requires $O(n)$ flops. For simplicity, we denote by L the trailing (last) 2×2 principal submatrix of J:

$$L = \begin{pmatrix} J_{n-1,n-1} & J_{n-1,n} \\ J_{n,n-1} & J_{n,n} \end{pmatrix}$$

Algorithm 4 (Complex-Symmetric QR Step) *Given an $n \times n$ complex-symmetric tridiagonal matrix J, this algorithm overwrites J with $Q^{\mathrm{T}}JQ$ where Q is a product of complex-orthogonal matrices so that $Q^{\mathrm{T}}(J - \mu I)$ is upper triangular and μ is the eigenvalue of L that is closer to J_{nn}.*

Initialize $Q = I$;
Find the eigenvalue μ of L that is closer to J_{nn};
Set $x_1 = J_{11} - \mu$; $x_2 = J_{21}$;
for $k = 1 : n - 1$
    Find a complex-orthogonal matrix G_k^{T} (applying Algorithm 1)
    to annihilate x_2 using x_1;
    $J = G_k^{\mathrm{T}} J G_k$;
    $Q = Q G_k$;
    if $k < n - 1$
        $x_1 = J_{k+1,k}$; $x_2 = J_{k+2,k}$;
    end if
end for.

7 OVERALL ALGORITHM

We combine our algorithms into an $O(n^2 \log n)$ eigenvalue procedure for an $n \times n$ Hankel matrix, and conclude the paper with two numerical examples.

Algorithm 5 (Fast Hankel Eigenvalue Algorithm) *Given an $n \times n$ Hankel matrix H, this algorithm computes all its eigenvalues.*

1. Tridiagonalize H via Algorithm 3 (applying Algorithm 2 to find Hankel matrix-vector products). Let J denote the resultant complex-symmetric tridiagonal matrix.

2. Repeat until convergence

   (a) Set small subdiagonal elements in J to zero and partition J:

$$J = \begin{array}{c} \\ p \\ n-p-q \\ q \end{array} \begin{array}{c} \begin{array}{ccc} p & n-p-q & q \end{array} \\ \begin{pmatrix} \widehat{K} & 0 & 0 \\ 0 & \widehat{J} & 0 \\ 0 & 0 & \widehat{D} \end{pmatrix} \end{array},$$

where p is minimized and q is maximized so that $\widehat{D}$ is diagonal and $\widehat{J}$ remains unreduced.

(b) If $q < n-1$, apply Algorithm 4 to $\widehat{J}$.

Example 1. Apply Algorithm 3 to the Hankel matrix:

$$H = \begin{pmatrix} 0.900+0.783i & -0.538+0.524i & 0.214-0.087i & -0.028-0.963i \\ -0.538+0.524i & 0.214-0.087i & -0.028-0.963i & -0.111+0.476i \\ 0.214-0.087i & -0.028-0.963i & -0.111+0.476i & 0.231-0.648i \\ -0.028-0.963i & -0.111+0.476i & 0.231-0.648i & 0.584-0.189i \end{pmatrix}.$$

We get a tridiagonal matrix:

$$J = \begin{pmatrix} 0.267-0.584i & 0.208-0578i & 0 & 0 \\ 0.208-0.578i & 1.162+1.398i & 1.349-1.029i & 0 \\ 0 & 1.349-1.029i & -0.564-1.278i & 0.312-0.952i \\ 0 & 0 & 0.312-0.952i & 0.723+1.447i \end{pmatrix}.$$

The following table shows the subdiagonal elements of J during the execution of Algorithm 5:

Iteration	J_{21}	J_{32}	J_{43}
1	$0.465-0.098i$	$-0.761+0.428i$	$-0.040-0.630i$
2	$-0.649+0.115i$	$0.191+0.319i$	$-0.046+0.024i$
3	$1.291-0.121i$	$0.099-0.063i$	$O(10^{-6})$
4	$9.637-8.712i$	$0.016-0.004i$	$O(10^{-15})$
5	$0.506-1.417i$	$-0.014+0.006i$	converged
6	$-0.454+0.089i$	$O(10^{-8})$	
7	$0.356-0.102i$	converged	
8	converged		

The criterion for convergence is

$$|J_{i+1,i}| \leq \sqrt{2}\,(|J_{i,i}| + |J_{i+1,i+1}|)\epsilon_M$$

where ϵ_M is the machine precision. The computed eigenvalues are

$$H(\lambda) = \{0.9198-1.1431i,\ 0.099893-0.74375i,\ -0.24002+1.4976i,\ 0.80755+1.3762i\}.$$

Assuming that eigenvalues computed by the MATLAB function EIG() are fully accurate, we find that our errors are about 10^{-14}.

n	Algorithm 5	MATLAB EIG	Error
4	9,214	3,791	6.5×10^{-14}
8	47,888	38,181	1.8×10^{-14}
16	183,574	301,442	4.9×10^{-14}
32	763,808	2,443,679	2.3×10^{-7}
64	3,247,686	19,292,025	$1.2 \times 10^{+1}$

Table 1: Operation counts and computational errors.

Example 2. To illustrate the $O(n^2 \log n)$ behavior of our fast algorithm (versus the $O(n^3)$ requirement of the traditional approach), we chose $n \times n$ random complex Hankel matrices for $n = 4, 8, 16, 32, 64$. We generated these matrices by picking random vectors as their first columns and last rows. For accuracy and work comparison, we chose the MATLAB EIG function. We obtained the (real) floating-point operation counts using the MATLAB function FLOPS, and we assumed that the eigenvalues (call them λ_i) computed via EIG are fully accurate. Since the eigenvalues were complex, we ordered λ_i in non-increasing order of magnitude, i.e.,

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|.$$

We referred to the eigenvalues computed by Algorithm 5 as $\bar{\lambda}_i$, and ordered them in the same non-increasing fashion. We determined the errors of the computed eigenvalues via

$$\text{Error} = \sqrt{\sum_{i=1}^{n} \frac{|\lambda_i - \bar{\lambda}_i|^2}{|\lambda_i|^2}}.$$

Table 1 shows the operation counts and errors. As n doubles, the flop count for Algorithm 5 roughly quadruples and that for EIG increases by a factor of eight, agreeing with the theoretical predictions. Even for n as small as 16, Algorithm 5 requires only 60% of the flops of EIG. The accuracy of Algorithm 5 deteriorates rapidly as n increases, due probably to the loss of complex-orthogonality of the Lanczos vectors generated by Algorithm 3.

References

[1] R. H. Chan and M. K. Ng, "Conjugate gradient methods for Toeplitz systems," *SIAM Review*, Vol. 38, No. 3, 1996, pp. 427–482.

[2] J. K. Cullum and R. A. Willoughby, "A QL procedure for computing the eigenvalues of complex symmetric tridiagonal matrices," *SIAM J. Matrix Anal. Appl.*, Vol. 17, No. 1, 1996, pp. 83–109.

[3] G. Cybenko and C. F. Van Loan, "Computing the minimum eigenvalue of a symmetric positive definite Toeplitz matrix," *SIAM J. Sci. and Stat. Comput.*, 7, 1986, pp. 123–131.

[4] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 3rd Ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[5] F.T. Luk and S. Qiao, "Using complex-orthogonal transformations to diagonalize a complex symmetric matrix," in *Advanced Signal Processing Algorithms, Architectures, and Implementations VII*, F.T. Luk, Ed., Proc. SPIE 3162, pp. 418–425, 1997.

[6] F.T. Luk and S. Qiao, "Analysis of a Fast Hankel Eigenvalue Algorithm," in *Advanced Signal Processing Algorithms, Architectures, and Implementations IX*, F.T. Luk, Ed., Proc. SPIE 3807, pp. 324–333, 1999.

[7] M.K. Ng and R.J. Plemmons, "Fast recursive least squares adaptive filtering by fast Fourier transform-based conjugate gradient iterations," *SIAM J. Sci. Comput.*, Vol. 17, No. 4, 1996, pp. 920–941.

[8] W.F Trench, "Numerical solution of the eigenvalue problem for Hermitian Toeplitz matrices," *SIAM J. Matrix Anal. Appl.*, Vol. 10, No. 2, 1989, pp. 135–146.

[9] C.F. Van Loan, *Computational Frameworks for the Fast Fourier Transform,* SIAM, Philadelphia, PA, 1992.

[10] D. Vandevoorde, *A Fast Exponential Decomposition Algorithm and Its Applications to Structured Matrices*, Ph.D. Dissertation, Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY, 1996.