A Fast Eigenvalue Algorithm for Hankel Matrices

Franklin T. Luk[a] and Sanzheng Qiao[b]

[a]Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York 12180 USA

[b]Department of Computing and Software
McMaster University
Hamilton, Ontario L8S 4L7 Canada

ABSTRACT

We present an $O(n^2 \log n)$ algorithm for finding all the eigenvalues of an $n \times n$ complex Hankel matrix.

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:

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

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[2] 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 and requires at most $O(n^2)$ floating-point operations per Newton iteration and heuristically $O(\log n)$ iterations. Building on their work[2] Trench[5] 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 to find all the eigenvalues of an $n \times n$ Hankel matrix.

Our paper is organized as follows. How to exploit complex-symmetry is presented in Section 2, and how to construct complex-orthogonal rotations 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 an illustrative numerical example in Section 7.

Supported by the Natural Sciences and Engineering Research Council of Canada under grant OGP0046301

2. COMPLEX SYMMETRY

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

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

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,

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

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

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

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

$$J = WDW^T,$$

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-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 structures, we use the complex-orthogonal rotations in the QR iterations.

3. COMPLEX-ORTHOGONAL ROTATIONS

A basic operation in solving eigenvalue problems is the introduction of zeros into 2×1 vectors using rotations. In this paper, we use a 2×2 complex-orthogonal transformation which would reduce to the regular Givens rotation if all the given data were real. From its definition in (4), we derive the general form of a 2×2 complex-orthogonal matrix as

$$G = \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} c & s \\ s & -c \end{pmatrix}, \tag{6}$$

where $c^2 + s^2 = 1$. Here, we choose the nonsymmetric version in (6) since it coincides with the Givens rotation in the real case. 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[4] and Vandevoorde[7].

ALGORITHM 1 (COMPLEX-ORTHOGONAL ROTATION). *Given a complex vector* $\mathbf{x}$ *of (7), this algorithm computes the parameters* c *and* s *for the nonsymmetric 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
    $t = x_1/x_2$; $s = 1/\sqrt{1+t^2}$; $c = t \cdot s$;
end if.

4. FAST HANKEL MATRIX-VECTOR MULTIPLICATION

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 additional notations. Let

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

and

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

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:

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

that is,

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

The simplicity of equation (9) explains why, when we defined the vectors $\mathbf{h}$ and $\mathbf{t}$ we started with the $(1,1)$ element of H but with the $(1,n)$ element of T.

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

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

where

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

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

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

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 $T(\mathbf{h})$.

Given an n-element vector:

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

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:

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

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[6] (FFT); namely,

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

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

ALGORITHM 2 (FAST HANKEL MATRIX-VECTOR MULTIPLICATION). *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):

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

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

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

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

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

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

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

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 $\text{ifft}(\text{fft}(\widehat{\mathbf{c}}). * \text{fft}(\widehat{\mathbf{w}}))$ equals $30n\log(n) + O(n)$ flops. Thus, Algorithm 2 becomes superior to general matrix-vector multiplication when $n \geq 16$.

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

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

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., $\mathbf{q}_i^T\mathbf{q}_j = \delta_{ij}$, we get

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

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

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

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 $H = QJQ^T$, where J is a complex-symmetric tridiagonal matrix as shown in (15).*

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

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

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

We get a complex-orthogonal decomposition[1] of K:

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

where $R = (\mathbf{e}_1 \quad J\mathbf{e}_1 \quad J^2\mathbf{e}_1 \quad \dots \quad J^{n-1}\mathbf{e}_1)$. 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 $k = 1, 2, \dots, n$. Cullum and Willoughby[1] 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. 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. However, we rarely get an exact zero β_k in practice.

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[3] 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[1] and Vandevoorde[7] for theoretical results on convergence. Basically, if a complex-symmetric tridiagonal matrix J is nonsingular, irreducible, nondefective, and 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 since this matrix cannot be triangularized by a complex-orthogonal matrix. If the matrix Q is not desired, this algorithm requires $O(n)$ flops.

ALGORITHM 4 (COMPLEX-SYMMETRIC QR STEP). *Given an $n \times n$ complex-symmetric tridiagonal matrix J, this algorithm overwrites J with $Q^T J Q$ where Q is a product of complex-orthogonal matrices so that $Q^T(J - \mu I)$ is upper triangular and μ is the eigenvalue of the trailing 2×2 principal submatrix of J that is closer to $J(n, n)$.*

Initialize $Q = I$;
Find the eigenvalue μ of $J(n-1:n, n-1:n)$ that is closer to $J(n, n)$;
Set $x_1 = J(1,1) - \mu$; $x_2 = J(2,1)$;
for $k = 1 : n-1$
    Find a complex-orthogonal matrix G_k^T (applying Algorithm 1) to annihilate x_2 using x_1;
    $J = G_k^T J G_k$;
    $Q = Q G_k$;
    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 a numerical example.

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:

$$J = \begin{array}{c} \\ p \\ n-p-q \\ q \end{array} \begin{array}{c} \begin{array}{ccc} p & n-p-q & q \end{array} \\ \begin{pmatrix} J_{11} & 0 & 0 \\ 0 & J_{22} & 0 \\ 0 & 0 & J_{33} \end{pmatrix} \end{array},$$

   where p is minimized and q is maximized so that J_{33} is diagonal and J_{22} remains unreduced.
   (b) If $q < n-1$, apply Algorithm 4 to J_{22}.

Example. 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	$1.291-0.121i$	$0.099-0.063i$	10^{-6}
4	$9.637-8.712i$	$0.016-0.004i$	10^{-15}
5	$0.506-1.417i$	$-0.014+0.006i$	converge
6	$-0.454+0.089i$	10^{-8}	
7	$0.356-0.102i$	converge	
8	converge		

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}.

REFERENCES

1. 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.
2. 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.
3. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd Ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
4. F. T. Luk and S. Qiao, "Using complex-orthogonal transformations to diagonalize a complex symmetric matrix," *Advanced Signal Processing: Algorithms, Architectures, and Implementations VII*, F. T. Luk, Ed., Proc. SPIE 3162, pp. 418–425, 1997.
5. 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.
6. C. F. Van Loan, *Computational Frameworks for the Fast Fourier Transform,* SIAM, Philadelphia, PA, 1992.
7. 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.