ABSTRACT

A DIVIDE-AND-CONQUER SPLIT SCHUR ALGORITHM

Steven J. Kifowit, Ph.D.
Department of Mathematical Sciences
Northern Illinois University, 2015
Gregory S. Ammar, Director

Positive definite Toeplitz systems of equations arise in a number of applications in pure and applied mathematics. Methods of solution specifically designed to exploit the symmetry of such linear systems have been studied in earnest since the late 1940's. By 1990, "superfast" methods, whose operation counts were asymptotically far lower than traditional methods, had been developed and implemented.

This dissertation concerns the superfast solution of Toeplitz systems. In particular, a new algorithm is described for solving the Yule-Walker equations associated with a Hermitian positive definite Toeplitz matrix. The new algorithm is based on a doubling procedure applied to the split Schur algorithm. This procedure computes the solution of the Yule-Walker equations by processing a family of split Levinson symmetric polynomials. The operation count for the new algorithm is among the lowest for all known direct methods for solving the Yule-Walker equations.

The foundations of the new superfast algorithm rest on the split Schur algorithm. A new derivation of the split Schur algorithm is also described in this work. That derivation

highlights the classical underpinnings of the split Schur algorithm and reveals its nature as a recursion on a certain class of functions.

The new superfast algorithm is rich in operations on symmetric polynomials. It derives its speed from fast Fourier transform techniques for polynomial multiplication and division. A number of new symmetry-exploiting FFT techniques are also contained in this work.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

AUGUST 2015

A DIVIDE-AND-CONQUER SPLIT SCHUR ALGORITHM

BY

STEVEN J. KIFOWIT

A DISSERTATION SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

DOCTOR OF PHILOSOPHY

DEPARTMENT OF MATHEMATICAL SCIENCES

Dissertation Director:
Gregory S. Ammar

ACKNOWLEDGEMENTS

First I would like to thank my adviser, Professor Gregory S. Ammar, for his guidance, encouragement, and wise counsel. His enduring patience was deeply appreciated; if the old cliché is true, he is indeed a virtuous man.

I would like to thank my committee members, Professors Zhuan Ye, Douglas Bowman, and Linda Sons, and my external examiner, Professor Michael Stewart, who were very generous with their time and expertise. Their thoughtful consideration of my work and their questions, comments, and suggestions were extremely valuable to me. In addition, I would like to express my appreciation to Professor Sons for being an inspiring mentor whose professionalism sets a standard that I strive to achieve.

I would like to acknowledge the support of my parents, James and Marilyn Kifowit, who did so much over the years to contribute to the completion of this work. I cannot thank them enough.

Finally, I owe a huge debt of gratitude to my loving wife, Stephanie, and my family for their everlasting support and encouragement. They truly have always been there for me!

DEDICATION

This work is dedicated to my father, James Kifowit, and my father-in-law, William Janak, who surely would have loved to celebrate its completion with me.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

Chapter

1 INTRODUCTION 1

1.1 Organization of this Dissertation 5

1.2 Notation. 7

2 REVIEW OF LITERATURE 10

2.1 Preliminaries 10

2.1.1 Definitions and Basic Facts 10

2.1.2 Circulant Matrices and FFTs. 18

2.2 Toeplitz Solvers. 28

2.2.1 Yule-Walker Equations 30

2.2.2 Gohberg-Semencul Formula 33

2.2.3 Szegő Polynomials and the Levinson-Durbin Algorithm 35

2.2.4 Schur's Algorithm and the Cholesky Factorization 39

2.2.5 The Generalized Schur Algorithm. 43

2.2.6 Conditioning of Toeplitz Matrices 48

3 SPLIT LEVINSON ALGORITHM. 50

3.1 History. 50

3.2 Split Levinson Symmetric Polynomials 51

3.3 The Algorithm 60

Chapter Page

3.4 Example . . . 66

4 SPLIT SCHUR ALGORITHM . . . 68

4.1 Carathéodory Functions . . . 69

4.2 Split Schur Functions . . . 70

4.3 The Algorithm . . . 77

4.4 Connections to Carathéodory Functions . . . 80

5 DIVIDE-AND-CONQUER SPLIT SCHUR ALGORITHM . . . 86

5.1 Main Ideas . . . 87

5.2 The Algorithm . . . 96

5.3 Example . . . 101

6 SUPERFAST IMPLEMENTATION OF THE DIVIDE-AND-CONQUER SPLIT SCHUR ALGORITHM . . . 103

6.1 Symmetric Vectors . . . 104

6.2 Zero-Padded Vectors . . . 108

6.2.1 DFTs of Zero-Padded Vectors . . . 110

6.2.2 Zero Padding $\dot{\mathfrak{u}}_m$ and $\dot{\mathfrak{q}}_m$ . . . 117

6.2.3 Zero Padding $\dot{\mathfrak{v}}_m$ and $\dot{\mathfrak{p}}_m$ . . . 119

6.2.4 Coefficient Vectors of p_m and q_m . . . 122

6.3 Split Schur Functions . . . 126

6.3.1 Computing $\mathfrak{h}_{m-1}^{(m)}$ . . . 129

6.3.2 Computing $\mathfrak{h}_m^{(m)}$ . . . 133

6.4 Compositions of Symmetric Polynomials . . . 136

6.4.1 Computing the DFT of $\dot{\mathfrak{u}}_{0,2m}$ . . . 137

6.4.2 Computing the DFT of $\dot{\mathfrak{v}}_{0,2m}$ . . . 139

Chapter Page

6.4.3 Computing the DFT of $\dot{\mathbf{p}}_{0,2m}$ 141

6.4.4 Computing the DFT of $\dot{\mathbf{q}}_{0,2m}$ 144

6.5 The Implementation 147

6.6 A Faster Implementation 154

7 NUMERICAL TESTS. 162

7.1 Test Matrix 1 164

7.2 Test Matrix 2 166

7.3 Test Matrix 3 168

7.4 Test Matrix 4 170

7.5 Discussion of Test Results 174

8 SUMMARY AND CONCLUSIONS 177

8.1 Contributions 177

8.2 Future Research 178

REFERENCES 181

LIST OF TABLES

Table Page

7.1 Errors in DCSSA (version A) associated with test matrix 1. 165

7.2 Errors in SLA associated with test matrix 1 166

7.3 Errors in DCSSA (version A) associated with test matrix 2. 167

7.4 Errors in DCSSA (version B) associated with test matrix 2. 167

7.5 Errors in DCSSA (version A) associated with test matrix 3 with $\theta = -0.5$ . . 169

7.6 Errors in qC-parameters, $\|\tilde{\zeta} - \zeta\|$, from DCSSA (versions A and B), SLA, and SSA (Test matrix 3 with $\theta = -0.5$) 170

7.7 Errors in DCSSA (version A) associated with test matrix 4. 173

7.8 Errors in qC-parameters, $\|\tilde{\zeta} - \zeta\|$, from DCSSA (versions A and B), SLA, and SSA (Test matrix 4) 173

7.9 Errors in *Maxima* DCSSA (version A) associated with test matrix 4 of order 1024. 174

CHAPTER 1

INTRODUCTION

In a variety of applications in pure and applied mathematics, one must solve a system of linear equations where the coefficient matrix is a Toeplitz matrix, i.e., a matrix whose elements are constant along each diagonal. Toeplitz systems originally arose in connection with the trigonometric moment problem, the Carathéodory coefficient problem, and the theory of polynomials orthogonal on the unit circle [1, 35, 53, 57]. Today, Toeplitz systems routinely arise in signal processing applications such as spectrum analysis [42], linear prediction [47], and filter design [36]. Other applications include statistical time-series analysis, the numerical solution of partial differential equations, and Padé approximation, to name but a few.

It has been known for some time that an $n \times n$ Hermitian positive definite Toeplitz system can be solved directly in far fewer arithmetic operations than required by traditional Gaussian elimination or Cholesky factorization. While these traditional methods have operation counts on the order of n^3, Levinson's algorithm [46], presented in 1947, is more economical by a factor of n. A streamlined version of Levinson's algorithm was presented by Durbin [32] in 1960 for the solution of the specific system of equations known as the Yule-Walker equations. The Levinson-Durbin algorithm requires roughly $2n^2$ operations, and it is arguably the most popular Toeplitz solver. Since its development, a number of other fast Toeplitz solvers have

been developed and analyzed (for example, see [9, 23, 40, 44, 58, 64]). For the purposes of this work, the most notable among these is the split Levinson algorithm of Delsarte and Genin [23].

Around 1980, a new class of direct Toeplitz solvers was introduced [5, 10, 13, 22, 48]. These algorithms typically make use of doubling strategies and are asymptotically faster than the Levinson-Durbin algorithm, requiring significantly fewer operations for sufficiently large systems. Notable among these is the generalized Schur algorithm of Ammar and Gragg [5]. When applied to real Toeplitz systems, it requires fewer than $8n(\log_2 n)^2$ arithmetic operations.

More recently, the preconditioned conjugate gradient method has been applied to Toeplitz systems [18]. For certain classes of Toeplitz matrices, this iterative technique can be used to obtain satisfactory results with an operation count on the order of $n \log_2 n$. Another recently proposed asymptotically fast Toeplitz solver [19] is based on transforming the Toeplitz matrix to a Cauchy-like matrix. The new Cauchy-like matrix is then approximated by a certain type of matrix with a special low-rank structure. Overall, this method has an operation count on the order of $n \log_2 n$, but in practice, it is not clear that it is faster than the direct superfast methods mentioned above. Although these methods lead to important classes of efficient Toeplitz solvers, they will not be considered here.

This dissertation is concerned with the direct, numerical solution of Hermitian positive definite Toeplitz systems. In particular, a new asymptotically-fast algorithm will be presented for the solution of the Yule-Walker equations of order n and the determination of

the Schur parameters associated with an $(n+1) \times (n+1)$ Toeplitz matrix. The following applications illustrate the types of problems solved by the new algorithm.

Suppose a continuous-time signal $s(t)$ is sampled at intervals of time T to obtain the discrete-time signal $s_n = s(nT)$. In the all-pole linear prediction problem [47, 49], it is assumed that s_n is a linear combination of past values and the input u_n:

$$s_n = -\sum_{k=1}^{p} a_k s_{n-k} + G u_n,$$

where the constant G is called the gain factor. If the input signal is completely unknown, as is the case in the analysis of electroencephalograms (EEGs) measured from spontaneous brain activity (for example, see [52]), the signal s_n is approximated by

$$\tilde{s}_n = -\sum_{k=1}^{p} a_k s_{n-k}.$$

The prediction error, or residual, r_n, is given by

$$r_n = s_n - \tilde{s}_n.$$

In the least-squares method, the prediction parameters a_k are obtained by minimizing the sum of the squares of the residuals, $E = \sum_n r_n^2$. By setting $\partial E / \partial a_j = 0$, one obtains the *normal equations*

$$\sum_{k=1}^{p} a_k \sum_n s_{n-k} s_{n-j} = -\sum_n s_n s_{n-j}, \quad 1 \le j \le p.$$

If s is a deterministic signal of infinite duration or a random, stationary (time-independent) signal, then the *autocorrelation function*

$$M(j-k) = \sum_n s_{n-k} s_{n-j}, \quad 1 \leq j, k \leq p$$

depends only on the difference $j-k$. The corresponding *autocorrelation matrix* $[M(j-k)]_{j,k=1}^{p}$ is a symmetric positive definite Toeplitz matrix, and the normal equations are the Yule-Walker equations. Fast solution methods for the Yule-Walker equations involve certain intermediate quantities called *Schur parameters*, which measure the partial correlations between s_n and s_{n+j}.

In transmission line theory, the Schur parameters are called *reflection coefficients*. They describe the reflected impulses of current at junction points in transmission lines [17]. The reflection coefficients are tied to the *central mass sequence* $\{\rho_1, \rho_2, \ldots, \rho_n\}$ of a symmetric positive definite Toeplitz matrix M_n:

$$\rho_k = \sup\{\rho : (M_k - \rho \Pi_k) \text{ is positive definite}\},$$

where M_k is the $k \times k$ leading submatrix of M_n and Π_k is the $k \times k$ matrix of 1's. Caflisch [17] showed that knowledge of the capacitance tapers of a transmission line is equivalent to knowledge of the reflection coefficients. This, in turn, is equivalent to the knowledge of a matrix central mass sequence. Dickinson [29] described how fast Toeplitz solvers could be used to obtain the central mass sequence of a Toeplitz matrix via its Schur parameters. In this

type of application, the solution of the Toeplitz system itself is not inherently interesting—the important quantities are the Schur parameters.

1.1 Organization of this Dissertation

This dissertation is organized as follows:

Chapter 2 contains background information and general results pertaining to the solution of Toeplitz systems. The chapter also includes a summary of basic facts concerning the discrete Fourier transform (DFT) and fast Fourier transform (FFT) algorithms. For the most part, FFT algorithms will be considered "black boxes," and their precise details will not be considered, though numerous references are given. Chapter 2 also includes descriptions of the fast and superfast direct Toeplitz solvers that have inspired the current work. Conspicuously absent from the chapter are the "split" Toeplitz solvers originally proposed by Delsarte and Genin [23, 24]. The split methods are described thoroughly in later chapters.

The split Levinson algorithm is derived in Chapter 3. The derivation is somewhat similar to that given by Krishna and Morgera [44], but it is unique in that it uses the output of the Levinson algorithm as it starting point. This approach makes the connections between the algorithms clear. The disadvantage, however, is that further efforts are required to make the algorithms independent of one another.

In the fourth chapter, the split Schur algorithm is derived. Using Schur's classical algorithm [51] and the split Levinson algorithm as starting points, the split Schur algorithm

is derived by splitting Schur functions into appropriate parts and then processing linear combinations of the parts. A new class of functions, called *quasi-Carathéodory functions*, is defined, and it is shown that the split Schur algorithm is a method for constructing a continued fraction representation for such a function.

In Chapter 5, a divide-and-conquer approach to the split Schur algorithm is proposed. A new Toeplitz solver is derived, which is based on a doubling strategy applied to a continued fraction representation of a certain quasi-Carathéodory function. The new algorithm is rich in polynomial multiplication and consequently can be implemented efficiently by using fast Fourier transform techniques.

An FFT-based implementation of the new divide-and-conquer split Schur algorithm is described in Chapter 6. This implementation applies to real symmetric positive definite Toeplitz matrices whose dimensions are powers of two. The algorithm processes a number of vectors with certain symmetry properties. In order to take advantage of FFT-based convolution techniques, these vectors must be padded with zeros, thus destroying their symmetries. A variety of techniques are developed in Chapter 6 for efficiently computing the DFTs of the zero-padded symmetric vectors. These techniques result in a real Toeplitz solver that requires $7n(\log_2 n)^2 + O(n \log n)$ real floating-point operations.

Chapter 7 contains the results of a number of numerical tests of the superfast implementation of the divide-and-conquer split Schur algorithm. These include tests of the accuracy of the new algorithm as well as accuracy comparisons of the new algorithm with other split algorithms.

The final chapter, Chapter 8, contains a brief summary of the main results described in this dissertation. It also contains a number of related problems that are proposed for future work.

1.2 Notation

Throughout this work, efforts have been made to use notation consistently. From this point forward, the following notation is used:

$\mathcal{D}$	open unit disk $\{z : \|z\| < 1\}$
Δ	machine precision
$\alpha_{\mathbb{R}}$, $\mu_{\mathbb{R}}$, $\delta_{\mathbb{R}}$	single real addition, multiplication, and division
$\alpha_{\mathbb{C}}$, $\mu_{\mathbb{C}}$, $\delta_{\mathbb{C}}$	single complex addition, multiplication, and division
ψ_j	For $j = 1, 2, \ldots, 7$, used to denote FFT flop counts
i	imaginary unit, $i = \sqrt{-1}$
ω	primitive nth root of unity, $\omega = e^{-2\pi i/n}$
n	positive integer (often the dim/deg of vectors, matrices, or polys)
A, B	general matrices
x, y	general vectors (i.e. column matrices)
a, b	general vectors, often used to denote the rhs of a linear system
j, k, ℓ, m	general integer indices
p, N	positive integers, usually an exponent

K	positive integer, used to denote the order of a matrix
s, t, z	general (real or complex) variables
f, g	general functions (often general polynomial functions)
H	general lower triangular matrix
L	general unit lower triangular matrix, or
	lower-triangular Toeplitz matrix operator
I (I_n)	$n \times n$ identity matrix
e_k	kth column of the identity matrix, $k = 1, 2, \ldots$
J (J_n)	$n \times n$ reversal matrix
P_n	$n \times n$ even/odd permutation matrix
$M = [m_{j-k}]$	Toeplitz matrix, generally indexed from 0
C	general circulant matrix or circulant matrix operator
Ω (Ω_n)	$n \times n$ Fourier matrix
W_k	$\mathrm{diag}([\omega^j]_{j=0}^{k-1})$
U	the unitary matrix $\Omega_n/\sqrt{n}$
R	the unit upper triangular matrix in $R^*MR = D$
r_k	kth column of the matrix R
D	general diagonal matrix, or
	diagonal matrix in the reverse Cholesky factorization $D = R^*MR$
δ_k	kth diagonal element of $D = R^*MR$ (i.e., kth prediction error)
σ_k	positive element of the rhs of the alternate YW equation ($\sigma_k = \delta_k$)

ρ_k	kth monic Szegő polynomial
γ_k	kth Schur parameter
T	used briefly as $T = LD$ in the fast Cholesky algorithm
ϕ, ϕ_k	Schur function
α_k, β_k	numerator and denominator of Schur function
τ_k	Möbius transformations in Schur/split Schur algorithm
T_k	composition of τ_j's
η_k, ξ_k	Schur polynomials in the generalized Schur algorithm
w_k	split Levinson symmetric polynomials in the split Levinson algorithm
ϖ_k	reversed split Levinson symmetric polynomials, $\overline{w}_k = \varpi_k$
λ_k, μ_k, ζ_k, ν_k	split Levinson/split Schur parameters
	(λ_k are called Jacobi parameters, ζ_k are called qC-parameters)
κ	used briefly to represent the product of Jacobi parameters
h_k	split Schur functions
$h_k^{(m)}$	split Schur function truncated to m terms
χ_k	quasi-Carathéodory functions from the split Schur algorithm
p_k, q_k, u_k, v_k	split Levinson polynomials in the DCSSA
θ_k	used briefly in the DCSSA to represent $\mu_{k+1}/\overline{\mu}_{k+1}$
Y, Z	used in proofs to represent DFTs of vectors x and y
O, E	used in proofs to denote arbitrary odd or even vectors
Ψ	used for flop count of the DCSSA

CHAPTER 2

REVIEW OF LITERATURE

The primary object of study is the linear system

$$Mx = b,$$

where the matrix M is a complex Toeplitz matrix. In particular, this work focuses on the direct, numerical solution of the Yule-Walker equations of order n associated with an $(n+1) \times (n+1)$ Hermitian positive definite Toeplitz matrix. In this chapter, fundamental results pertaining to the solution of this system of equations are reviewed.

2.1 Preliminaries

In this section, definitions, notation, and basic facts are laid out for subsequent use.

2.1.1 Definitions and Basic Facts

For any matrix B, B^T represents the *transpose* of B, $\overline{B}$ represents the *conjugate* of B, and B^* represents the *conjugate transpose* of B ($B^* = \overline{B}^T = \overline{B^T}$). Furthermore, if B is

nonsingular, B^{-1} represents the *inverse* of B. These operations have several important properties that will be used throughout:

- For any pair of matrices A and B for which AB is defined, $(AB)^T = B^T A^T$ and $(AB)^* = B^* A^*$.
- For nonsingular, square matrices A and B, $(B^{-1})^T = (B^T)^{-1}$, $(B^{-1})^* = (B^*)^{-1}$, and $(AB)^{-1} = B^{-1}A^{-1}$.

A matrix B is *symmetric* if $B = B^T$ and *Hermitian* if $B = B^*$.

Depending on the situation, the elements of an n-dimensional vector may be indexed from 1 to n, as in

$$x = [x_1, x_2, \ldots, x_n]^T = [x_j]_{j=1}^n,$$

or from 0 to $n-1$, as in

$$x = [x_0, x_1, \ldots, x_{n-1}]^T = [x_j]_{j=0}^{n-1}.$$

In either case, it should be clear from context which indexing scheme is being used.

Given the n-dimensional vector $x = [x_1, x_2, \ldots, x_n]^T$,

$$\mathrm{diag}(x) = \mathrm{diag}(x_1, x_2, \ldots, x_n)$$

represents the $n \times n$ *diagonal matrix* with the elements of x along the main diagonal and 0's elsewhere.

A Hermitian matrix $B \in \mathbb{C}^{n\times n}$ is *positive definite* if $x^*Bx > 0$ for every nonzero vector $x \in \mathbb{C}^n$ and *nonnegative definite* if the inequality is not strict. Positive definite matrices have the following important properties:

- All principal submatrices of a positive definite matrix are positive definite. (In particular, the diagonal elements are positive, real numbers.)
- Positive definite matrices are nonsingular, and their inverses are also positive definite.

Every positive definite, Hermitian matrix $B \in \mathbb{C}^{n\times n}$ has a unique *Cholesky factorization*:

$$B = HH^*,$$

where H is a lower-triangular matrix in $\mathbb{C}^{n\times n}$ whose main diagonal elements, $h_{k,k}$, are positive, real numbers. Letting $D = \text{diag}(h_{1,1}^2, h_{2,2}^2, \ldots, h_{n,n}^2)$, the Cholesky factorization can be rewritten

$$B = LDL^*,$$

where L is unit lower triangular.

The *identity matrix* of order n, I_n, is the $n \times n$ matrix with 1's along the main diagonal and 0's elsewhere. Whenever the size is irrelevant or clear from context, the subscript will be omitted and the appropriately-sized identity matrix will be denoted by I. The ith column of I is denoted by e_i. The columns may be indexed from $i = 1$ or $i = 0$ depending on the context.

The *reversal matrix*, J (or J_n), is the matrix obtained by reversing the columns (or rows) of I,

$$J = [e_n, e_{n-1}, \ldots, e_2, e_1].$$

Left multiplication of a matrix by J reverses the rows of the matrix, whereas right multiplication by J reverses the columns. Notice that $J = J^T = J^{-1}$.

The *even/odd permutation matrix* of order n, P_n, is defined for even n by

$$P_n = [e_1, e_3, \ldots, e_{n-1}, e_2, e_4, \ldots, e_n].$$

Equivalently, P_n is defined by its action on the n-dimensional vector $x = [x_j]_{j=0}^{n-1}$:

$$P_n^T x = \begin{bmatrix} x_e \\ \\ x_o \end{bmatrix},$$

where $x_e = [x_{2j}]_{j=0}^{n/2-1}$ and $x_o = [x_{2j+1}]_{j=0}^{n/2-1}$ are the *even-indexed and odd-indexed parts of* x, respectively. Left multiplication of a matrix by P_n^T results in the even/odd permutation of the rows of the matrix, whereas right multiplication by P_n results in the even/odd permutation of the columns.

A matrix $B \in \mathbb{C}^{n\times n}$ is *unitary* if $B^*B = I$.

A matrix $B \in \mathbb{C}^{n\times n}$ is said to be *persymmetric* if it is symmetric about its main cross (northeast-southwest) diagonal. That is to say that B is persymmetric if and only if $B =$

JB^TJ. An extremely important property, to be exploited throughout this work, is that the inverse of a nonsingular, persymmetric matrix is also persymmetric.

The matrix $M = M_{n+1} = [m_{j,k}]_{j,k=0}^{n} \in \mathbb{C}^{(n+1)\times(n+1)}$ is a *Toeplitz matrix* if

$$m_{j,k} = m_{j-k}; \quad j,k = 0,1,2,\ldots n.$$

Toeplitz matrices share the following important properties:

- The principal submatrices of a Toeplitz matrix are Toeplitz matrices.
- Toeplitz matrices are persymmetric.

Any Hermitian Toeplitz matrix has the following form:

$$M = M_{n+1} = \begin{bmatrix} m_0 & m_{-1} & m_{-2} & \cdots & m_{-n} \\ m_1 & m_0 & m_{-1} & \cdots & m_{1-n} \\ m_2 & m_1 & m_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & m_{-1} \\ m_n & m_{n-1} & \cdots & m_1 & m_0 \end{bmatrix},$$

where $m_{-j} = \overline{m}_j$. Furthermore if M is positive definite, then $m_0 > 0$ and the principal submatrices $M_k, k = 1,2,\ldots,n+1$, are Hermitian positive definite Toeplitz matrices.

The *eigenvalues* of the matrix $A \in \mathbb{C}^{n\times n}$ are the complex numbers λ such that

$$Ax = \lambda x.$$

An $n \times n$ matrix has n eigenvalues, counting multiplicity. Hermitian matrices have real eigenvalues and are positive definite if and only if the eigenvalues are positive.

The *Schur product* of two n-dimensional vectors x and y is the vector obtained from element-by-element multiplication:

$$x \star y = [x_1 y_1, x_2 y_2, \ldots, x_n y_n]^T.$$

The Schur product is also called the *Hadamard product*.

The *vector p-norm* of $x = [x_1, x_2, \ldots, x_n]^T \in \mathbb{C}^n$ is denoted by $\|x\|_p$ and defined by

$$\|x\|_p = \begin{cases} (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}, & \text{if } 1 \leq p < \infty \\ \max\{|x_1|, |x_2|, \ldots, |x_n|\}, & \text{if } p = \infty \end{cases}.$$

The *matrix p-norm* of $B \in \mathbb{C}^{m \times n}$ is induced by the vector p-norm:

$$\|B\|_p = \max\{\|Bx\|_p : x \in \mathbb{C}^n \text{ with } \|x\|_p = 1\}.$$

Whenever the value of p is irrelevant or clear from context, the subscript will be omitted, and the norm will be denoted by $\|\cdot\|$.

For any particular p-norm, the *condition number* of a nonsingular matrix $B \in \mathbb{C}^{n \times n}$ is defined as

$$\text{cond}(B) = \|B\| \, \|B^{-1}\|.$$

By convention, if B is singular, $\text{cond}(B) = \infty$. The condition number of a matrix is always greater than or equal to 1. A matrix is said to be *well-conditioned* if its condition number is not large. Otherwise, it is *ill-conditioned*. It is a useful fact that for Hermitian matrices, the 2-norm condition number is the absolute value of the ratio of the extreme eigenvalues.

An algorithm for solving linear equations is *stable* (or *backward stable*) if for each matrix A and each right-hand side b, the computed solution $\tilde{x}$ to $Ax = b$ satisfies $\check{A}\tilde{x} = \check{b}$, where $\|\check{A} - A\|/\|A\|$ and $\|\check{b} - b\|/\|b\|$ are small.

An algorithm for solving linear equations is *weakly stable* if for each well-conditioned matrix A and each right-hand side b, the computed solution $\tilde{x}$ to $Ax = b$ is such that the *relative residual*, $\|A\tilde{x} - b\|/\|b\|$, is small. Equivalently, the *relative error*, $\|x - \tilde{x}\|/\|x\|$, is small. Stability implies weak stability but not vice versa.

Given a complex polynomial g of degree n, $\overline{g}$ denotes the polynomial obtained by conjugating the coefficients, $\overline{g}(z) = \overline{g(\overline{z})}$. The *reciprocal polynomial* $\hat{g}$ is the polynomial

$$\hat{g}(z) = z^n \overline{g}(1/z),$$

which can be obtained from g by conjugating and reversing the order of its power basis coefficients.

If $g(z) = \hat{g}(z)$, the polynomial is said to be *self reciprocal* or *conjugate symmetric*. The product of two conjugate-symmetric polynomials is conjugate symmetric.

In addition to symmetric polynomials, three types of highly symmetric vectors will be encountered in this work. The n-dimensional vector $x = [x_0, x_1, \ldots, x_{n-1}]^T \in \mathbb{R}^n$ is said to

have *real even (RE) symmetry* if $x_k = x_{n-k}$ and *real odd (RO) symmetry* if $x_k = -x_{n-k}$. An RO symmetric vector has the property that $x_0 = 0$ and, if n is even, $x_{n/2} = 0$. The vector x is said to have *real quarter-even (RQE) symmetry* if $x_k = x_{n-k-1}$. For example, the vectors

$$[1, 2, 3, 4, 5, 4, 3, 2]^T, \quad [0, 1, 2, 3, 0, -3, -2, -1]^T, \quad [1, 2, 3, 4, 4, 3, 2, 1]^T$$

exhibit RE, RO, and RQE symmetries, respectively.

A *flop* is a single floating-point arithmetic operation. Normally this will refer to one multiplication or addition of two floating-point numbers. If the number of floating-point operations required by an algorithm is bounded by a multiple of n^p, the algorithm is said to require $O(n^p)$ flops. The symbols $\alpha_{\mathbb{R}}$, $\mu_{\mathbb{R}}$, and $\delta_{\mathbb{R}}$ denote a single real addition, multiplication, and division operation, respectively, and $\alpha_{\mathbb{C}}$, $\mu_{\mathbb{C}}$ and $\delta_{\mathbb{C}}$ are defined similarly for complex operations. Operation counts involving complex arithmetic will assume that complex multiplication is carried out with four real multiplications and two real additions, i.e., $\mu_{\mathbb{C}} = 4\mu_{\mathbb{R}} + 2\alpha_{\mathbb{R}}$.

2.1.2 Circulant Matrices and FFTs

A *circulant matrix* of order n is an $n \times n$ Toeplitz matrix in which each column is a cyclic downshift of the previous column. In other words, a square circulant matrix C is a matrix of the form

$$C = \begin{bmatrix} c_1 & c_n & c_{n-1} & \cdots & c_2 \\ c_2 & c_1 & c_n & \cdots & c_3 \\ c_3 & c_2 & c_1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & c_n \\ c_n & c_{n-1} & \cdots & c_2 & c_1 \end{bmatrix}.$$

Circulant matrices have many special properties [43]. Chief among them is that circulant matrices are unitarily diagonalizable, and all circulant matrices are diagonalized by the same unitary matrix.

For a positive integer n, let ω be the primitive nth root of unity,

$$\omega = e^{-2\pi i/n}.$$

The *Fourier matrix* of order n is

$$\Omega = \Omega_n = [\omega^{jk}]_{j,k=0}^{n-1} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & w^{(n-1)(n-1)} \end{bmatrix}.$$

The Fourier matrix is nonsingular, and its inverse is given by

$$\Omega^{-1} = \frac{1}{n}\Omega^*.$$

It follows that $U = \Omega/\sqrt{n}$ is a unitary matrix.

Every square circulant matrix is diagonalized by U. More precisely, if C is any $n \times n$ circulant matrix and $U = \Omega_n/\sqrt{n}$, then

$$UCU^* = D,$$

where D is the diagonal matrix of eigenvalues of C.

The *discrete Fourier transform* (DFT) of an n-dimensional vector x is the vector $y = \Omega x$. For emphasis, this is sometimes called the *forward* DFT of x to distinguish it from the *backward* or *inverse discrete Fourier transform* (IDFT) of y,

$$x = \frac{1}{n}\Omega^* y.$$

With the vectors indexed from 0 to $n-1$, it follows that the elements of the DFT/IDFT pair satisfy

$$y_k = \sum_{j=0}^{n-1} x_j \, e^{-2\pi jki/n} \tag{2.1a}$$

and

$$x_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j \, e^{2\pi jki/n}, \tag{2.1b}$$

for $k = 0, 1, \ldots, n-1$.

While a matrix-vector product generally requires $O(n^2)$ flops, the computation of the DFT and IDFT can be performed in $O(n \log_2 n)$ flops by using the well-known computational procedures called *Fast Fourier Transforms* (FFTs). FFT algorithms are based on a divide-and-conquer strategy in which an n-dimensional DFT is split into smaller DFTs. For example, suppose n is an even number. Let $x_e = [x_{2j}]_{j=0}^{n/2-1}$ and $x_o = [x_{2j+1}]_{j=0}^{n/2-1}$ be the even-indexed and odd-indexed parts of the n-dimensional vector x. Then (2.1a) can be rewritten

$$y_k = \sum_{j=0}^{n/2-1} x_{2j} \, e^{-2\pi jki/(n/2)} + e^{-2\pi ki/n} \sum_{j=0}^{n/2-1} x_{2j+1} \, e^{-2\pi jki/(n/2)}. \tag{2.2}$$

For $k = 0, 1, \ldots, n/2 - 1$, the summations in (2.2) describe the elements of the DFTs of x_e and x_o, respectively. Since

$$e^{-2\pi j(k+n/2)i/(n/2)} = e^{-2\pi jki/(n/2)} \quad \text{and} \quad e^{-2\pi(k+n/2)i/n} = -e^{-2\pi ki/n},$$

it follows that the DFT of x can be obtained from the DFTs of x_e and x_o:

$$\begin{aligned} y_k &= \sum_{j=0}^{n/2-1} x_{2j}\, e^{-2\pi jki/(n/2)} + e^{-2\pi ki/n} \sum_{j=0}^{n/2-1} x_{2j+1}\, e^{-2\pi jki/(n/2)}, \\ y_{k+n/2} &= \sum_{j=0}^{n/2-1} x_{2j}\, e^{-2\pi jki/(n/2)} - e^{-2\pi ki/n} \sum_{j=0}^{n/2-1} x_{2j+1}\, e^{-2\pi jki/(n/2)}, \end{aligned} \tag{2.3}$$

for $k = 0, 1, \ldots, n/2 - 1$. In matrix notation, (2.3) takes the form

$$\Omega_n x = \begin{bmatrix} \Omega_{n/2} & \text{diag}([e^{-2\pi ki/n}]_{k=0}^{n/2-1})\Omega_{n/2} \\ \Omega_{n/2} & -\text{diag}([e^{-2\pi ki/n}]_{k=0}^{n/2-1})\Omega_{n/2} \end{bmatrix} \begin{bmatrix} x_e \\ x_o \end{bmatrix}, \tag{2.4}$$

from which it is clear that an n-dimensional DFT can be computed with two $n/2$-dimensional DFTs, $n/2$ complex multiplications, and n complex additions. The classic *Cooley-Tukey power-of-two FFT* [20] obtains its fast status by making repeated use of the splitting procedure described above. (See [31] for a tutorial review of FFT techniques and [37] for the history of the FFT.)

The exact number of flops required by an FFT depends on the nature of the integer n, the data type of the input vector, and the particular type of FFT algorithm. For example, if n is a power of 2 and the input vector is real, the *Duhamel-Hollmann split-radix FFT* [30]

requires $2n\log_2 n - 4n + 6$ flops. The more efficient *Johnson-Frigo modified split-radix FFT* [41] requires $\frac{17}{9}n\log_2 n - \frac{89}{27}n + O(\log_2 n)$ flops for the same type of vector.

Because circulant matrices are diagonalized by the Fourier matrix, matrix-vector operations involving circulant matrices can be carried out efficiently with FFT techniques. Indeed, since $UCU^* = D$ implies $\Omega C = D\Omega$, it follows that

$$\Omega C e_1 = D\Omega e_1 = D[1, 1, \ldots, 1]^T.$$

Therefore, the DFT of the first column of C gives the diagonal elements of D. By further exploiting this fact, the circulant matrix-vector product can be computed as follows:

$$Cx = U^*DUx = \frac{1}{n}\Omega^* D\Omega x = \frac{1}{n}\Omega^*[\mathrm{diag}(\Omega C e_1)\Omega x]$$

or

$$Cx = \frac{1}{n}\Omega^*\left[\Omega(Ce_1) \star \Omega x\right].$$

Thus, the product Cx follows from the IDFT of the Schur product of two DFTs. Using FFTs, this product can be computed in $O(n\log_2 n)$ flops—3 FFTs of length n and n additional complex multiplications associated with the Schur product. Just as circulant matrix-vector products reduce to Schur products in the transform domain, circulant systems can be solved with element-by-element division in the transform domain.

Toeplitz matrix-vector multiplication can also be accomplished efficiently with FFTs. The Toeplitz matrix M whose first column and row are

$$[m_0, m_1, \ldots, m_{n-1}]^T \qquad \text{and} \qquad [m_0, m_{-1}, \ldots, m_{-(n-1)}],$$

respectively, is embedded in the upper left corner of the circulant matrix of twice the size whose first column is

$$[m_0, m_1, \ldots, m_{n-1}, 0, m_{-(n-1)}, m_{-(n-2)}, \ldots, m_{-1}]^T.$$

In block form, this $2n \times 2n$ circulant matrix can be written

$$\begin{bmatrix} M & B \\ B & M \end{bmatrix},$$

and the Toeplitz matrix-vector product Mx forms the first n elements of the circulant matrix-vector product

$$\begin{bmatrix} M & B \\ B & M \end{bmatrix} \begin{bmatrix} x \\ 0_n \end{bmatrix},$$

where the 0_n represents the n-dimensional zero vector. In this way, a Toeplitz matrix-vector product can be computed in $O(n \log_2 n)$ flops.

Perhaps the most important application of FFTs in this work is the fast multiplication of polynomials. Suppose f and g are polynomials of degrees m and n, respectively:

$$f(z) = \sum_{j=0}^{m} f_j \, z^j \quad \text{and} \quad g(z) = \sum_{j=0}^{n} g_j \, z^j.$$

The polynomial product $f(z)g(z)$ has degree $n + m$ with $n + m + 1$ coefficients. Let C be the circulant matrix whose first column is

$$y = [f_0, f_1, \ldots, f_m, \underbrace{0, 0, \ldots, 0}_{n \text{ zeros}}]^T,$$

and let

$$x = [g_0, g_1, \ldots, g_n, \underbrace{0, 0, \ldots, 0}_{m \text{ zeros}}]^T.$$

The coefficients of the polynomial product are the elements of the circulant matrix-vector product Cx. This product is also called the *convolution* of the polynomials' zero-padded coefficient vectors. It will alternatively be denoted by

$$Cx = y * x.$$

As it is a circulant matrix-vector product, the polynomial product (or convolution) can be computed efficiently with FFTs of length $n+m+1$. Because the DFT is a linear transform,

addition and scalar multiplication of polynomials can be carried out in the transform domain. This indicates that a sum of products of polynomials, such as

$$f_1(z)g_1(z) + g_2(z)f_2(z),$$

can be computed with several FFTs (of zero-padded vectors), but only requires a single inverse FFT following the addition in the transform domain.

When a vector exhibits a certain type of symmetry, its DFT typically exhibits a related type of symmetry [55]. By exploiting these symmetries, the arithmetic complexity of the corresponding *symmetric FFTs* can be greatly reduced. For example, it is well known that the DFT of a real vector is conjugate symmetric, and its computation requires roughly half as many operations as a general complex FFT. Several types of symmetric vectors (often associated with symmetric polynomials) will be encountered in this dissertation. FFTs corresponding to these symmetries are discussed in [55]. The FFT algorithms used in this work, their references, flop counts, and other properties are summarized below.

1. **CFFTF**—Duhamel-Hollman Complex Split-Radix FFT [30]

    - Complex input and output vectors of size $n = 2^p$
    - Flop count denoted by $\psi_1(n)$, $n \geq 2$:

$$\psi_1(n) = (n\log_2 n - 3n + 4)\mu_{\mathbb{R}} + (3n\log_2 n - 3n + 4)\alpha_{\mathbb{R}} = 4n\log_2 n - 6n + 8$$

2. **RFFTF**—Duhamel-Hollman Real Split-Radix FFT [30]

   - Real input vector of size $n = 2^p$
   - Complex, conjugate-symmetric output vector
   - Flop count denoted by $\psi_2(n)$, $n \geq 2$:

$$\psi_2(n) = \left(\frac{n}{2}\log_2 n - \frac{3n}{2} + 2\right)\mu_{\mathbb{R}} + \left(\frac{3n}{2}\log_2 n - \frac{5n}{2} + 4\right)\alpha_{\mathbb{R}} = 2n\log_2 n - 4n + 6$$

3. **RFFTB**—Duhamel-Hollman Real Split-Radix Inverse FFT [30]

   - Complex, conjugate-symmetric input vector of size $n = 2^p$
   - Real output vector
   - Flop count denoted by $\psi_3(n)$, $n \geq 2$:

$$\psi_3(n) = \psi_2(n) + n\delta_{\mathbb{R}} = 2n\log_2 n - 3n + 6$$

4. **RDCTF**—RE-Symmetric FFT

   - Derived from RFFTF as described by Swarztrauber [55]
   - Real even input vector of size $n = 2^p \geq 4$
   - Real even output vector

- Flop count denoted by $\psi_4(n)$, $n \geq 4$:

$$\psi_4(n) = \psi_2(n/2) + \left(\frac{3n}{4} - 1\right)\mu_{\mathbb{R}} + (2n-3)\alpha_{\mathbb{R}} = n\log_2 n - \frac{n}{4} + 2$$

5. **RDCTB**—RE-Symmetric Inverse FFT

- Derived from RFFTF as described by Swarztrauber [55]
- Real even input vector of size $n = 2^p \geq 4$
- Real even output vector
- Flop count denoted by $\psi_5(n)$, $n \geq 4$:

$$\psi_5(n) = \psi_4(n) + \left(\frac{n}{2} + 1\right)\delta_{\mathbb{R}} = n\log_2 n + \frac{n}{4} + 3$$

6. **RDSTF**—RO-Symmetric FFT

- Derived from RFFTF as described by Swarztrauber [55]
- Real odd input vector of size $n = 2^p \geq 4$
- Purely imaginary, odd output vector
- Flop count denoted by $\psi_6(n)$, $n \geq 4$:

$$\psi_6(n) = \psi_2(n/2) + \left(\frac{n}{2} + 1\right)\mu_{\mathbb{R}} + \left(\frac{7n}{4} - 1\right)\alpha_{\mathbb{R}} = n\log_2 n - \frac{3n}{4} + 6$$

7. **RQETB**—RQE-Symmetric Inverse FFT

- Derived from RFFTF as described by Swarztrauber [55]
- Complex, conjugate-symmetric input vector of size $n = 2^p \geq 4$
- Input vector is the transform of an RQE-symmetric vector
- Real quarter-even output vector
- Flop count denoted by $\psi_7(n)$, $n \geq 4$:

$$\psi_7(n) = \psi_2(n/2) + (2n-5)\mu_{\mathbb{R}} + (2n-7)\alpha_{\mathbb{R}} + \frac{n}{2}\delta_{\mathbb{R}} = n\log_2 n + \frac{3n}{2} - 6$$

The operation counts given above do not include the computations of the powers of the roots of unity required by the FFT algorithms. For all of the FFTs used in this work, these values are precomputed, stored in sine/cosine tables, and retrieved as necessary.

2.2 Toeplitz Solvers

This section provides a summary of some results that are specific to the solution of Toeplitz systems. Several solution methods—those that have directly influenced the current research—are also reviewed.

An algorithm for the solution of the Toeplitz system $Mx = b$ is referred to as a *Toeplitz solver*. A Toeplitz solver is said to be *fast* if it requires $O(n^2)$ flops for the solution of an $n \times n$ system. Among the fast Toeplitz solvers are the Levinson-Durbin algorithm [32], Trench's inversion algorithm [58], the Bareiss algorithm for the Cholesky factorization [9],

and the split Levinson algorithm [23]. *Superfast* Toeplitz solvers require strictly less than $O(n^2)$ flops and are more efficient than fast solvers only for sufficiently large n. In this class of solvers are the generalized Schur algorithm [5] and those derived from algorithms for computing elements in the Padé table [13]. These algorithms require $O(n\,(\log_2 n)^2)$ flops in contrast to Stewart's superfast Schur algorithm with improved stability [54], which requires $O(n\,(\log_2 n)^3)$ flops.

Toeplitz solvers are generally divided into two main classes:

(A) those that compute M^{-1} or a decomposition of M^{-1} (*Levinson-type*), and

(B) those that compute a decomposition of M (*Schur-type*).

The Levinson-Durbin algorithm is in class (A), while the Bareiss algorithm is in class (B).

Most Toeplitz solvers are two-phase algorithms. In phase one, the algorithms compute the decomposition referred to above. In phase two, the decomposition is used to solve the system with a specific right-hand side. Because the second phase typically involves fewer operations than the first, emphasis is normally placed on phase one.

Although there are a wide variety of Toeplitz solvers, different solvers often have surprising connections. A number of authors have provided general frameworks for describing the relationships among these algorithms (for example, see [2, 15, 63]).

The Toeplitz solvers referenced above are known, or believed to be, at least weakly stable when applied to positive definite systems. On the other hand, when applied to indefinite or nonsymmetric systems, stability cannot be expected. In fact, simple examples can show that the algorithms can be highly unstable when applied to general Toeplitz matrices. Several

approaches for overcoming this unstable behavior have been developed (see [59] and the references therein). These "fixes" have the potential to greatly increase the computational complexity of an algorithm. Fortunately, many of the applications that give rise to Toeplitz systems also give rise to positive definite systems.

2.2.1 Yule-Walker Equations

From this point forward, unless otherwise indicated, M refers to the complex $(n+1) \times (n+1)$ Hermitian Toeplitz matrix

$$M = M_{n+1} = \begin{bmatrix} m_0 & m_{-1} & m_{-2} & \cdots & m_{-n} \\ m_1 & m_0 & m_{-1} & \cdots & m_{1-n} \\ m_2 & m_1 & m_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & m_{-1} \\ m_n & m_{n-1} & \cdots & m_1 & m_0 \end{bmatrix},$$

with $n \geq 1$. The *Yule-Walker equations* of order k are given by the system

$$M_k x = -[m_1, m_2, \ldots, m_k]^T; \quad k = 1, 2, \ldots, n, \tag{2.5}$$

where M_k is the $k \times k$ leading submatrix of M. If M is positive definite, then each leading principal submatrix is positive definite, and each system of Yule-Walker equations has a unique solution.

Yule-Walker equations appear in a number of applications in engineering, statistics, and mathematics. For example, in digital signal processing and time series analysis, they often arise in connection with least-squares parameter estimation. In Yule's own work [62], they arose in the context of modeling accidentally-disturbed periodic phenomena. (Walker [60] later extended Yule's ideas.)

The Yule-Walker equations are often written in the alternate form

$$\begin{bmatrix} m_0 & m_{-1} & \cdots & m_{-k} \\ m_1 & & & \\ \vdots & & M_k & \\ m_k & & & \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} \sigma_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{2.6}$$

where the rows indexed 1 through k form the original Yule-Walker system and σ_k is uniquely determined by the normalization $x_0 = 1$.

Because M is a Hermitian Toeplitz (persymmetric) matrix, the Yule-Walker equations (2.5) are equivalent to

$$JM_kJJx = -[m_k, m_{k-1}, \ldots, m_1]^T.$$

This matrix equation reduces to

$$M_k y = -[\overline{m}_k, \overline{m}_{k-1}, \ldots, \overline{m}_1]^T, \tag{2.7}$$

where $y = J\overline{x}$ is the reverse conjugate of the solution of (2.5). The alternate form associated with equation (2.7) is

$$\begin{bmatrix} & & & m_{-k} \\ & M_k & & m_{-(k-1)} \\ & & & \vdots \\ m_k & m_{k-1} & \cdots & m_0 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \sigma_k \end{bmatrix}, \tag{2.8}$$

where σ_k is determined by the normalization $y_k = 1$ and is equal to its counterpart in equation (2.6).

By renormalizing equation (2.6) so that $\sigma_k = 1$ (instead of $x_0 = 1$), the new solution gives the first column of M_{k+1}^{-1}. It follows that the solution of the Yule-Walker equations (2.5) implicitly gives the first column of M_{k+1}^{-1}. More precisely, if

$$x = [x_1, x_2, \ldots, x_k]^T$$

satisfies equation (2.5) and

$$\sigma_k = m_0 + m_{-1}x_1 + m_{-2}x_2 + \cdots + m_{-k}x_k,$$

then the first column of M_{k+1}^{-1} is given by

$$\frac{1}{\sigma_k}[1, x_1, x_2, \ldots, x_k]^T,$$

where σ_k is guaranteed to be a positive real number by the positive definite nature of M. In a similar way, the solution of equation (2.7) implicitly gives the last column of M_{k+1}^{-1}, which, as described above, is the reverse conjugate of the first column.

By solving the Yule-Walker equations of order n, the first (or last) column of M^{-1} can be computed. From this column, a complete, phase-one, decomposition of M^{-1} can be obtained from the Gohberg-Semencul formula or one of its variants [39].

2.2.2 Gohberg-Semencul Formula

In 1972, Gohberg and Semencul [33] presented a formula that expresses the inverse of Toeplitz matrix belonging to a certain class in terms of only the first (or last) column of its inverse. In particular, the *Gohberg-Semecul formula* is given as follows.

Theorem 2.1 (Gohberg-Semecul Formula)**.** *Let M be a nonsingular $(n+1)\times(n+1)$ Hermitian Toeplitz matrix with a nonsingular $n\times n$ leading submatrix. Let $x=[x_0,x_1,\ldots,x_n]^T$ be the first column of M^{-1}. Then $x_0\neq 0$ and*

$$M^{-1}=\frac{1}{x_0}\cdot\left(L(\begin{bmatrix}x_0\\x_1\\\vdots\\x_n\end{bmatrix})\cdot L(\begin{bmatrix}x_0\\x_1\\\vdots\\x_n\end{bmatrix})^*-L(\begin{bmatrix}0\\\overline{x}_n\\\vdots\\\overline{x}_1\end{bmatrix})\cdot L(\begin{bmatrix}0\\\overline{x}_n\\\vdots\\\overline{x}_1\end{bmatrix})^*\right),$$

where $L(y)$ denotes the lower-triangular Toeplitz matrix whose first column is y.

With the Gohberg-Semencul formula, the problem of solving $Mx=y$ reduces to the problem of solving the Yule-Walker equations of order n. Once the Yule-Walker solution is obtained, $M^{-1}y$ can be computed with FFTs, via the Gohberg-Semecul formula, in $O(n\log_2 n)$ flops.

There are a number Gohberg-Semecul-type formulas that offer a reduced arithmetic complexity. Among them is the circulant variant presented by Ammar and Gader [3]:

$$M^{-1}=\frac{1}{x_0}\cdot\left(L(\begin{bmatrix}x_0\\x_1\\\vdots\\x_n\end{bmatrix})\cdot C(\begin{bmatrix}x_0\\x_1\\\vdots\\x_n\end{bmatrix})^*-L(\begin{bmatrix}0\\\overline{x}_n\\\vdots\\\overline{x}_1\end{bmatrix})\cdot C(\begin{bmatrix}x_0\\x_1\\\vdots\\x_n\end{bmatrix})^*\right),$$

where x and L are as defined above and $C(y)$ denotes the circulant matrix whose first column is y. The use of this formula in the second phase of a two-phase Toeplitz solver provides a significant computational advantage over the Gohberg-Semecul formula [4].

2.2.3 Szegő Polynomials and the Levinson-Durbin Algorithm

The Hermitian positive definite (HPD) Toeplitz matrix $M = M_{n+1} = [m_{j-k}]_{j,k=0}^{n}$ has the unique factorization

$$M = LDL^*,$$

where L is a unit lower-triangular matrix and $D = \mathrm{diag}(\delta_0, \delta_1, \ldots, \delta_n)$ has positive (real) diagonal elements. Equivalently,

$$R^*MR = D, \tag{2.9}$$

where $R = [r_{j,k}]_{j,k=0}^{n} = (L^*)^{-1}$ is unit upper triangular. The matrix M induces an inner product on $\mathbb{C}^{n+1}$,

$$\langle x, y \rangle = y^*Mx,$$

and by equation (2.9), the columns of R are orthogonal with respect to this inner product. In particular, denoting the columns of R by r_k, it follows that

$$\langle r_k, r_\ell \rangle = r_\ell^* M r_k = \begin{cases} \delta_k, & k = \ell \\ 0, & k \neq \ell \end{cases}.$$

Under the identification

$$f(z) = \sum_{j=0}^{n} f_j \, z^j \quad \leftrightarrow \quad [f_0, f_1, \ldots, f_n]^T,$$

it is clear that the matrix also induces an inner product on $\mathbb{C}^n[z]$, the space of complex polynomials of degree at most n. The monic polynomials

$$\rho_k(z) = \sum_{j=0}^{n} r_{j,k} \, z^j; \; k = 0, 1, \ldots, n,$$

where $\deg(\rho_k) = k$, are therefore orthogonal with respect to the inner product. These polynomials are the *Szegő polynomials* associated with M (see [57]). Starting with $\rho_0(z) = 1$ and $\delta_0 = m_0$, they satisfy the Szegő recurrence relation

$$\rho_{k+1}(z) = z\rho_k(z) + \gamma_{k+1}\hat{\rho}_k(z), \tag{2.10a}$$

where $\hat{\rho}_k$ denotes the reciprocal polynomial. The numbers $\gamma_k, k = 1, 2, \ldots, n$, are given by

$$\gamma_{k+1} = -\left(\sum_{j=0}^{k} \overline{m}_{j+1} \, r_{j,k} \right) / \delta_k, \tag{2.10b}$$

where

$$\delta_{k+1} = \delta_k \left(1 - |\gamma_{k+1}|^2\right). \tag{2.10c}$$

Algorithm 2.1 Levinson-Durbin Algorithm

Input: $[m_0, m_1, \ldots, m_n]^T$ = first column of M
Initialize: $R = [r_{j,k} = 0]_{j,k=0}^{n}$; $\delta_0 = m_0$; $r_{0,0} = 1$
for $k = 0$ to $n-1$ **do**
$\gamma_{k+1} = -[\overline{m}_1, \overline{m}_2, \ldots, \overline{m}_{k+1}]^T [r_{0,k}, r_{1,k}, \ldots, r_{k,k}]/\delta_k$
$\delta_{k+1} = \delta_k(1 - |\gamma_{k+1}|^2)$
$[r_{0,k+1}, r_{1,k+1}, \ldots, r_{k+1,k+1}] = [0, r_{0,k}, r_{1,k}, \ldots, r_{k,k}] + \gamma_{k+1}[\overline{r}_{k,k}, \overline{r}_{k-1,k}, \ldots, \overline{r}_{0,k}, 0]$
end for
Output: R; $D = \mathrm{diag}(\delta_0, \ldots, \delta_n)$; $\gamma_1, \gamma_2, \ldots, \gamma_n$

The numbers $\delta_k, k = 0, 1, \ldots, n$, are the elements of the diagonal matrix D in equation (2.9). In applications, they arise as the norms of prediction error vectors and will henceforth be called *prediction errors*. Since M is a positive definite matrix, the prediction errors are necessarily real and positive, and they form a nonincreasing sequence. Through equation (2.10c), the positive definite nature of M is manifested in the inequalities

$$|\gamma_k| < 1; \quad k = 1, 2, \ldots, n. \tag{2.11}$$

Depending on the context in which they arise, the γ_k's are called *Schur parameters*, *reflection coefficients*, *partial correlation coefficients*, or even *Verblunsky coefficients* [53].

The Levinson-Durbin algorithm (Algorithm 2.1) for solving the Yule-Walker equations of order n is simply a matrix-vector formulation of the Szegő recurrence relation (2.10). In its kth step, the Levinson-Durbin algorithm solves the Yule-Walker equations (2.8), where $\sigma_k = \delta_k$. Upon completion, the algorithm yields the matrices R and D in the factorization (2.9), or alternatively, the last column of M^{-1}. Notice that this also provides factorizations, and last columns of inverses, for all smaller leading principal submatrices of M.

The Levinson-Durbin algorithm requires $2n^2 + O(n)$ flops when M is real or $8n^2 + O(n)$ flops when M is complex. An inductive proof of the validity of the algorithm is given in [6] and makes judicious use of the persymmetry and Hermitian symmetry of M.

Cybenko [21] was one of the first to investigate the numerical stability of the Levinson-Durbin algorithm. In floating-point arithmetic, with $\tilde{x}$ representing the computed solution of the nth-order Yule-Walker system (2.5), he found that the residual

$$y = M_n \tilde{x} + [m_1, m_2, \ldots, m_n]^T$$

satisfies

$$\|y\|_1 \leq \Delta \left(\frac{n^2}{2} + 11n \right) \left[\prod_{k=1}^{n} (1 + |\gamma_k|) - 1 \right] + O(\Delta^2), \tag{2.12}$$

where Δ is the machine precision. By comparing (2.12) with the analogous result for the well-behaved Cholesky algorithm, Cybenko concluded that the Levinson-Durbin and Cholesky algorithms have comparable residual bounds when all Schur parameters are positive. Bunch [16] later clarified Cybenko's results and concluded that the Levinson-Durbin algorithm is weakly stable when applied to the class of real symmetric positive definite Toeplitz matrices. Bunch also argued that it would be a "formidable task" to prove (strong) stability. Brent [12] further showed that

$$\|y\| = O\left(\mathrm{cond}(M_n)\, \Delta \left(\frac{32}{27} \right)^n \right),$$

which compares favorably to the residual bound for Gaussian elimination with partial pivoting, and he suggested that weak stability of the Levinson-Durbin algorithm is the most that could be expected.

2.2.4 Schur's Algorithm and the Cholesky Factorization

The Szegő recursions are used in the Levinson-Durbin algorithm to find the triangular factor R and the diagonal matrix D in the factorization $R^*MR = D$. They can also be used to find the factors L and D in the Cholesky factorization $M = LDL^*$. This is the essence of the fast Toeplitz solver presented by Bareiss [9] in 1969.

Sweet [56] and Bojanczyk et al. [11] showed that the Bareiss algorithm is stable when applied to symmetric positive definite Toeplitz matrices. In fact, the numerical properties of the algorithm are similar to those of Gaussian elimination without pivoting. Their results also suggest that the Levinson-Durbin algorithm can give much larger residuals than the Bareiss algorithm.

An algorithm for the fast Cholesky factorization of an HPD Toeplitz matrix is given in Algorithm 2.2. The use of the Szegő recursions can be seen in the innermost loop. Like the Levinson-Durbin algorithm, the fast Cholesky factorization requires $2n^2+O(n)$ or $8n^2+O(n)$ flops for the real or complex versions, respectively. A complete derivation of the algorithm is described in [6].

Algorithm 2.2 Fast Cholesky Factorization

Input: $[m_0, m_1, \ldots, m_n]^T$ = first column of M
Initialize: $T = [t_{j,k} = 0]_{j,k=0}^{n}$; $t_{0,0} = m_0$
for $j = 1$ to n **do**
    $t_{j,0} = m_j$
    $s = \overline{m}_j$ (s is a temporary variable)
    for $k = 1$ to $j - 1$ **do**
        $t_{j,k} = t_{j-1,k-1} + \gamma_k \cdot s$
        $s = s + \gamma_k \cdot \overline{t}_{j-1,k-1}$
    end for
    $\gamma_j = -s/t_{j-1,j-1}$
    $t_{j,j} = t_{j-1,j-1} \cdot (1 - |\gamma_j|^2)$
end for
Output: $T = LD$; $\gamma_1, \gamma_2, \ldots, \gamma_n$

It is known that the algorithm for the fast Cholesky factorization is a manifestation of Schur's algorithm for classifying certain analytic functions [51]. A *Schur function* is an analytic function that maps the open unit disk into its closure. Schur showed that each function in this class can be parameterized by a certain sequence of complex numbers, called *Schur parameters*. Given any Schur function ϕ_0, the corresponding sequence of Schur parameters $\{\gamma_k\}$ determines a continued fraction representation for ϕ_0. Schur's algorithm (Algorithm 2.3) provides a procedure for computing these parameters. Alternatively, Schur's algorithm can be viewed as a procedure for generating a sequence of Schur functions $\{\phi_k\}$ from the initial Schur function ϕ_0. It can be shown that the original and subsequent functions are Schur functions if and only if one of the following is true:

1. $|\gamma_k| < 1$ for $k = 1, 2, \ldots$ or

2. $|\gamma_k| < 1$ for $k = 1, 2, \ldots, n-1$, $|\gamma_n| = 1$, and ϕ_n is the constant function $\phi_n(z) = \gamma_n$.

Algorithm 2.3 Schur's Algorithm

Input: A complex function, $\phi_0(z)$
Initialize: $\gamma_1 = \phi_0(0)$; $k = 1$
while $|\gamma_k| < 1$ **do**
$\phi_k(z) = \dfrac{1}{z}\,\dfrac{\phi_{k-1}(z) - \gamma_k}{1 - \overline{\gamma}_k\,\phi_{k-1}(z)}$
$\gamma_{k+1} = \phi_k(0)$
$k = k + 1$
end while
Output: $\gamma_1, \gamma_2, \gamma_3 \ldots$

If the Schur function ϕ_k is represented as a ratio of formal power series

$$\phi_k(z) = \frac{\alpha_k(z)}{\beta_k(z)} = \frac{\alpha_{0,k} + \alpha_{1,k}z + \alpha_{2,k}z^2 + \cdots}{\beta_{0,k} + \beta_{1,k}z + \beta_{2,k}z^2 + \cdots}, \tag{2.13}$$

then Schur's algorithm can be formulated in terms of the numerator and denominator of ϕ_k. It is easy to establish that the functional iteration in Schur's algorithm takes the form

$$\alpha_k(z) = \frac{1}{z}(\alpha_{k-1}(z) - \gamma_k\beta_{k-1}(z)) \quad \text{and} \quad \beta_k(z) = \beta_{k-1}(z) - \overline{\gamma}_k\alpha_{k-1}(z). \tag{2.14}$$

With the requirement that $\beta_{0,0} > 0$, it follows that $\beta_{0,k} = (1 - |\gamma_k|^2)\,\beta_{0,k-1}$.

Stated in terms of α_k and β_k, the recurrence relation in Schur's algorithm clearly resembles the recurrence relations in the fast Cholesky factorization algorithm. In fact, the connection between the algorithms is made explicit by the following theorem (see [2, 6]).

Theorem 2.2. *Let $M = [m_{j-k}]_{j,k=0}^{n}$ be a Hermitian positive definite Toeplitz matrix. Define the polynomials*

$$\alpha_0^\star(z) = \sum_{j=0}^{n-1} -\overline{m}_{j+1}\, z^j$$

and

$$\beta_0^\star(z) = \sum_{j=0}^{n} \overline{m}_j\, z^j.$$

Then the terms of $\alpha_0^\star(z)$ and $\beta_0^\star(z)$ are the leading terms of power series $\alpha_0(z)$ and $\beta_0(z)$ (nonunique) whose ratio is a Schur function ϕ_0. Moreover, the elements of the lower triangular matrix $T = [t_{j,k}]_{j,k=0}^{n} = LD$ are given by $t_{j,k} = \overline{\beta}_{j-k,k}$ for $k \leq j$, where $\beta_k(z)$ is the denominator polynomial that results from k steps of Schur's algorithm applied to $\phi_0(z) = \alpha_0(z)/\beta_0(z)$. In addition, the sequences of Schur parameters $\{\gamma_k\}_{k=1}^{n}$ and prediction errors $\{\delta_k\}_{k=0}^{n}$ ($\delta_k = \beta_k(0) = \beta_{0,k}$) generated by Schur's algorithm, the fast Cholesky algorithm, and the Levinson-Durbin algorithm are identical.

Notice that operations on infinite series are required when applying Schur's algorithm to $\phi_0(z) = \alpha_0(z)/\beta_0(z)$. However, the computations can be arranged sequentially so that j terms of α_{k-1} and β_{k-1} are processed to obtain $j-1$ terms of α_k and β_k. This is the essence of the progressive Schur algorithm described in [6]. It follows that the polynomials $\alpha_0^\star(z)$ and $\beta_0^\star(z)$ are sufficient to generate the Cholesky factorization of M, as well as its Schur parameters and prediction errors.

2.2.5 The Generalized Schur Algorithm

In the previous section, Schur's algorithm was described as a procedure for generating a sequence of Schur functions. In particular, provided $|\gamma_k| < 1$, the Schur function ϕ_k is generated from the Schur function ϕ_{k-1} by the expression

$$\phi_k(z) = \frac{1}{z}\,\frac{\phi_{k-1}(z) - \gamma_k}{1 - \overline{\gamma}_k \phi_{k-1}(z)}.$$

Solving for ϕ_{k-1} gives

$$\phi_{k-1}(z) = \frac{\gamma_k + z\phi_k(z)}{1 + \overline{\gamma}_k z \phi_k(z)} = \tau_k(\phi_k(z)),$$

where τ_k is the Möbius transformation

$$\tau_k(s) = \frac{\gamma_k + zs}{1 + \overline{\gamma_k} zs}.$$

With this definition, an initial Schur function ϕ_0 satisfies

$$\phi_0 = \tau_1(\phi_1) = \tau_1(\tau_2(\phi_2)) = \cdots = \tau_1(\tau_2(\ldots(\tau_k(\phi_k)\ldots))).$$

This composition of k Möbius transformations gives k steps of *Schur's continued fraction* representation for ϕ_0.

Let T_k represent the composition

$$T_k(s) = (\tau_1 \circ \tau_2 \circ \cdots \circ \tau_k)(s)$$

so that $\phi_0 = T_k(\phi_k)$. It is shown in [6] that $T_k(s)$ can be written

$$T_k(s) = \frac{\xi_k(z) + \tilde{\eta}_k(z)s}{\eta_k(z) + \tilde{\xi}_k(z)s}, \tag{2.15}$$

where η_k, ξ_k, $\tilde{\eta}_k$, and $\tilde{\xi}_k$ are polynomials satisfying the recurrence relations

$$\begin{bmatrix} \tilde{\eta}_k(z) & \xi_k(z) \\ \tilde{\xi}_k(z) & \eta_k(z) \end{bmatrix} = \begin{bmatrix} \tilde{\eta}_{k-1}(z) & \xi_{k-1}(z) \\ \tilde{\xi}_{k-1}(z) & \eta_{k-1}(z) \end{bmatrix} \begin{bmatrix} z & \gamma_k \\ \overline{\gamma}_k z & 1 \end{bmatrix}, \quad \begin{bmatrix} \tilde{\eta}_0(z) & \xi_0(z) \\ \tilde{\xi}_0(z) & \eta_0(z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Furthermore, $\tilde{\eta}_k(z) = z^k \overline{\eta}_k(1/z)$ and $\tilde{\xi}_k(z) = z^k \overline{\xi}_k(1/z)$. With M, α_0, and β_0 defined as in Theorem 2.2, the kth Szegő polynomial associated with M is given by

$$\overline{\rho}_k(z) = \tilde{\eta}_k(z) + \frac{1}{z}\,\tilde{\xi}_k(z).$$

In this way, Schur's algorithm can be used to construct the Szegő polynomials and the first column of M^{-1}.

In the 1980's, Musicus [48] and de Hoog [22] independently presented a superfast Toeplitz solver that was later interpreted in terms of a doubling procedure applied to the composition T_n. This doubling generalization of Schur's algorithm was described and implemented by

Ammar and Gragg in a series of articles [5, 6, 7, 8]. Using the notation $T_{j,k}$ to represent the k-step composition starting with the Schur function ϕ_j (rather than ϕ_0), the key observations are the following:

- k steps of the continued fraction representation of ϕ_0 yield ϕ_k and $T_{0,k}$.
- ℓ steps of the continued fraction representation of ϕ_k yield $\phi_{k+\ell}$ and $T_{k,\ell}$.
- $T_{0,k+\ell} = T_{0,k} \circ T_{k,\ell}$.
- More generally, $T_{j,k+\ell} = T_{j,k} \circ T_{j+k,\ell}$ represents $k + \ell$ steps of the continued fraction representation of ϕ_j.

In order to make use of these observations, ϕ_k must be obtained from ϕ_0. Inverting equation (2.15) gives

$$T_{j,k}^{-1}(s) = \frac{\eta_{j,k}(z)s - \xi_{j,k}(z)}{\tilde{\eta}_{j,k}(z) - \tilde{\xi}_{j,k}(z)s},$$

from which it follows that $\phi_k = T_{0,k}^{-1}(\phi_0)$. In terms of the formal power series defined by equation (2.13), $T_{0,k}^{-1}(\phi_0)$ can be written

$$\phi_k = \frac{\alpha_k}{\beta_k} = T_{0,k}^{-1}(\frac{\alpha_0}{\beta_0}) = \frac{\alpha_0\eta_{0,k} - \beta_0\xi_{0,k}}{\beta_0\tilde{\eta}_{0,k} - \alpha_0\tilde{\xi}_{0,k}}.$$

It is shown in [6] that both the numerator and the denominator of this expression are divisible by z^k. Therefore it is natural to take

$$\begin{aligned} \alpha_k(z) &= \frac{\alpha_0(z)\eta_{0,k}(z) - \beta_0(z)\xi_{0,k}(z)}{z^k} \\ \beta_k(z) &= \frac{\beta_0(z)\tilde{\eta}_{0,k}(z) - \alpha_0(z)\tilde{\xi}_{0,k}(z)}{z^k}. \end{aligned} \tag{2.16}$$

At this point, the Schur function $\phi_k = \alpha_k/\beta_k$ can be used as an "initial" Schur function from which the ℓ-step composition $T_{k,\ell}$, and the polynomials $\eta_{k,\ell}$ and $\xi_{k,\ell}$, can be generated. The composition $T_{0,k+\ell}$ can then be constructed by composing $T_{0,k}$ and $T_{k,\ell}$. In particular, the composition gives

$$\begin{aligned} \xi_{0,k+\ell} &= \tilde{\eta}_{0,k}\xi_{k,\ell} + \xi_{0,k}\eta_{k,\ell} \\ \eta_{0,k+\ell} &= \tilde{\xi}_{0,k}\xi_{k,\ell} + \eta_{0,k}\eta_{k,\ell}. \end{aligned} \tag{2.17}$$

Notice that equations (2.16) and (2.17) apply more generally to any initial Schur function so that ϕ_{j+k} could be obtained from ϕ_j, $\eta_{j,k}$, $\xi_{j,k}$, $\tilde{\eta}_{j,k}$, and $\tilde{\xi}_{j,k}$. In this case, the compositions corresponding to equations (2.17) take the form

$$\begin{aligned} \xi_{j,k+\ell} &= \tilde{\eta}_{j,k}\xi_{j+k,\ell} + \xi_{j,k}\eta_{j+k,\ell} \\ \eta_{j,k+\ell} &= \tilde{\xi}_{j,k}\xi_{j+k,\ell} + \eta_{j,k}\eta_{j+k,\ell}. \end{aligned}$$

The generalized Schur algorithm of Ammar and Gragg is a doubling procedure based on the recursive use of equations (2.16) and (2.17). In practice, only k coefficients of α_0 and β_0 are required to compute $\eta_{0,k}$ and $\xi_{0,k}$. With this in mind, let $\alpha_k^{(p)}$ and $\beta_k^{(p)}$ denote the

Algorithm 2.4 Generalized Schur Algorithm

Input: $\alpha_0^{(n)}$ and $\beta_0^{(n)}$ obtained from M_{n+1}, where $n = 2^p$
Initialize: $\eta_{0,1} = 1$; $\xi_{0,1} = \gamma_1 = \alpha_0^{(1)}/\beta_0^{(1)}$
for $k = 1, 2, 4, \ldots, 2^{p-1}$ **do**
Use equations (2.16) to compute $\alpha_k^{(k)}$, $\beta_k^{(k)}$ from $\alpha_0^{(2k)}$, $\beta_0^{(2k)}$.
Treating α_k/β_k as an initial Schur function, compute $\eta_{k,k}$, $\xi_{k,k}$ from $\alpha_k^{(k)}$, $\beta_k^{(k)}$ just as $\eta_{0,k}$, $\xi_{0,k}$ were computed from $\alpha_0^{(k)}$, $\beta_0^{(k)}$.
Use equations (2.17) to compute $\eta_{0,2k}$, $\xi_{0,2k}$.
end for
Output: $\eta_{0,n}$; $\xi_{0,n}$; $\gamma_1, \gamma_2, \ldots, \gamma_n$

polynomials obtained from α_k and β_k by removing all terms of degree p or greater. Using this notation, the generalized Schur algorithm is described in Algorithm 2.4.

As can be seen from equations (2.16) and (2.17), the generalized Schur algorithm is rich in polynomial multiplication. By using FFT techniques to efficiently compute these products, the nth Szegő polynomial associated with M can be obtained in $O(n(\log_2 n)^2)$ flops.

An implementation of the algorithm and its application to real symmetric positive definite Toeplitz systems is described in [7] and [8]. When applied to the real Toeplitz matrix M_{n+1}, where $n = 2^p$, the algorithm requires fewer than $8n(\log_2 n)^2$ flops. Since the Levinson-Durbin algorithm requires more than $2n^2$ flops, the superfast generalized Schur algorithm has a smaller operation count when $n \geq 256$. Numerical results described in [8] indicate that the accuracy of the generalized Schur algorithm is comparable to that of the Levinson-Durbin algorithm. Therefore, it is expected that the algorithms have similar stability properties.

2.2.6 Conditioning of Toeplitz Matrices

While analyzing the numerical properties of the Levinson-Durbin algorithm, Cybenko [21] also established bounds on the value $\|M^{-1}\|_1$. His bounds give rather simple estimates for $\mathrm{cond}(M)$.

Suppose $M = M_{n+1}$ is a real symmetric positive definite Toeplitz matrix with the further restriction that $m_0 = 1$. Since M is positive definite and $m_0 = 1$, it follows that $|m_k| \leq 1$ for all k, and therefore

$$1 \leq \|M\|_1 \leq n + 1.$$

As a consequence, for moderately-sized Toeplitz matrices, the conditioning is essentially determined by $\|M^{-1}\|_1$. Given the Schur parameters and prediction errors associated with M, Cybenko showed that

$$\max\left\{\frac{1}{\delta_n}, \frac{1}{\prod_{k=1}^{n}(1-\gamma_k)}\right\} \leq \|M^{-1}\|_1 \leq \prod_{k=1}^{n}\frac{(1+|\gamma_k|)}{(1-|\gamma_k|)}, \tag{2.18}$$

from which it follows that

$$\max\left\{\frac{1}{\delta_n}, \frac{1}{\prod_{k=1}^{n}(1-\gamma_k)}\right\} \leq \mathrm{cond}(M) \leq (n+1)\prod_{k=1}^{n}\frac{(1+|\gamma_k|)}{(1-|\gamma_k|)}.$$

From his bounds on $\|M^{-1}\|_1$, Cybenko drew several striking conclusions. First, $\mathrm{cond}(M)$ is guaranteed large when δ_n is small. Unfortunately, in applications where δ_n measures a

prediction error, the smallness of δ_n is desirable. Such is the case in linear prediction. Next, if any γ_k is close to 1, cond(M) will be large. This type of situation occurs when there is a strong correlation between certain data values. In some applications this is also desirable, but it is indicative of nonstationarity, and the underlying assumptions that gave rise to the Toeplitz system are probably questionable. As long as no γ_k are close to 1 and n is not too large, the Toeplitz matrix M is well conditioned and the residual bound given by (2.12) is not large.

CHAPTER 3

SPLIT LEVINSON ALGORITHM

This chapter contains a complete derivation of the split Levinson algorithm for the computation of the nth Szegő polynomial associated with the HPD Toeplitz matrix $M = M_{n+1}$. Unlike the Szegő recursions, the split Levinson algorithm processes a family of conjugate-symmetric polynomials. The symmetry properties of these polynomials are exploited to yield an algorithm requiring significantly fewer multiplications than the Levinson-Durbin algorithm. The derivation given below is similar to the one presented by Krishna and Morgera [44]. Connections between the split Levinson symmetric polynomials and the Szegő polynomials are emphasized throughout. The parameters and formulas described in this chapter play important roles in subsequent chapters.

3.1 History

In 1986, the Levinson-Durbin algorithm for the solution of real symmetric positive definite Toeplitz systems was shown to contain certain redundancies. Delsarte and Genin [23] split the Levinson-Durbin algorithm into two more efficient algorithms, a symmetric version and an antisymmetric version, either of which could be used to obtain the nth Szegő polynomial. These "split" algorithms use three-term recurrence relations that generate conjugate-

symmetric polynomials. The symmetries in these polynomials give rise to redundancies identified by Delsarte and Genin. Their discovery made possible the splitting of several other important signal processing algorithms. In particular, Delsarte and Genin presented a split Schur algorithm in [24].

When first presented, a major drawback of the split Levinson algorithm was its restriction to real matrices. Shortly after its introduction, however, it was extended to the complex case [44]. It was also subjected to stability analysis [45], described under a broader framework [26], and even generalized [61].

3.2 Split Levinson Symmetric Polynomials

Suppose that the Szegő polynomials $\{\rho_k(z)\}_{k=0}^n$, Schur parameters $\{\gamma_k\}_{k=1}^n$, and prediction errors $\{\delta_k\}_{k=0}^n$ associated with the HPD Toeplitz matrix $M = M_{n+1} = [m_{j-k}]_{j,k=0}^n$ are given. Recall that the Szegő polynomials are monic polynomials with $\deg(\rho_k) = k$. They satisfy the recurrence relation

$$\rho_k(z) = z\rho_{k-1}(z) + \gamma_k\hat{\rho}_{k-1}(z) \tag{3.1}$$

and its reciprocal version

$$\hat{\rho}_k(z) = \hat{\rho}_{k-1}(z) + \overline{\gamma}_k z\rho_{k-1}(z), \tag{3.2}$$

where $\rho_0(z) = \hat{\rho}_0(z) = 1$. Since ρ_k is monic, $\hat{\rho}_k(0) = 1$ and it follows that $\rho_k(0) = \gamma_k$. The identities

$$\begin{aligned} \rho_k(z) - \gamma_k \hat{\rho}_k(z) &= z\rho_{k-1}(z)(1 - |\gamma_k|^2), \\ \hat{\rho}_k(z) - \overline{\gamma}_k \rho_k(z) &= \hat{\rho}_{k-1}(z)(1 - |\gamma_k|^2) \end{aligned}$$

are easy to verify from the recurrence relations (3.1) and (3.2). Furthermore, the following inequalities are guaranteed by the positive definite nature of M, via (2.11):

$$\rho_k(1) \neq 0, \quad \hat{\rho}_k(1) \neq 0; \quad k = 0, 1, \ldots, n.$$

In fact, it is well known that the zeros of the Szegő polynomials associated with an HPD Toeplitz matrix lie strictly inside the unit circle, while those of the reciprocal polynomials lie strictly outside [57].

A number of definitions and related formulas will now set the stage for the split Levinson algorithm.

Definition 3.1. *Let $\{\gamma_k\}_{k=1}^n$ be the sequence of Schur parameters associated with the HPD Toeplitz matrix $M = [m_{j-k}]_{j,k=0}^n$. With $\mu_0 = 1$, $\mu_1 = 1/m_0$, and $\lambda_1 = 2/m_0$, define the sequences $\{\mu_k\}_{k=0}^{n+1}$ and $\{\lambda_k\}_{k=1}^{n+1}$ recursively as follows. For $k = 1, 2, \ldots, n$,*

$$\mu_{k+1} = \frac{\overline{\mu}_k}{\lambda_k(\frac{\overline{\mu}_k}{\mu_k} - \overline{\gamma}_k)}, \tag{3.3a}$$

$$\lambda_{k+1} = 2\,\mathrm{Re}\left(\frac{\mu_{k+1}}{\mu_k}\right) - \frac{1}{\lambda_k}. \tag{3.3b}$$

Proposition 3.1. *The sequences described by Definition 3.1 are well defined. Moreover, each μ_k is a nonzero complex number and each λ_k is a positive real number.*

Proof. The proof is by induction. Obviously $\mu_0 \neq 0$. Since M is positive definite, m_0 is a positive real number. Therefore μ_1 and λ_1 are positive real numbers.

Now assume $\mu_j \neq 0$ and $\lambda_j > 0$ for some j, $1 \leq j \leq n$. Since $|\gamma_j| < 1$, it follows that $\frac{\overline{\mu}_j}{\mu_j} - \overline{\gamma}_j \neq 0$ and therefore,

$$\mu_{j+1} = \frac{\overline{\mu}_j}{\lambda_j(\frac{\overline{\mu}_j}{\mu_j} - \overline{\gamma}_j)}$$

is defined and nonzero. Furthermore, from Definition 3.1,

$$\lambda_{j+1} = \frac{\mu_{j+1}}{\mu_j} + \frac{\overline{\mu}_{j+1}}{\overline{\mu}_j} - \frac{1}{\lambda_j} = \frac{1}{\lambda_j}\left(\frac{\overline{\mu}_j/\mu_j}{\frac{\overline{\mu}_j}{\mu_j} - \overline{\gamma}_j} + \frac{\mu_j/\overline{\mu}_j}{\frac{\mu_j}{\overline{\mu}_j} - \gamma_j} - 1\right).$$

After obtaining the positive real common denominator

$$\left(\frac{\overline{\mu}_j}{\mu_j} - \overline{\gamma}_j\right)\left(\frac{\mu_j}{\overline{\mu}_j} - \gamma_j\right) = (1 + |\gamma_j|^2) - 2\,\mathrm{Re}(\frac{\mu_j}{\overline{\mu}_j}\overline{\gamma}_j),$$

the equation becomes

$$\lambda_{j+1} = \frac{1}{\lambda_j}\left(\frac{2 - 2\,\mathrm{Re}(\frac{\mu_j}{\overline{\mu}_j}\overline{\gamma}_j)}{(1 + |\gamma_j|^2) - 2\,\mathrm{Re}(\frac{\mu_j}{\overline{\mu}_j}\overline{\gamma}_j)} - 1\right).$$

Since $-1 < \mathrm{Re}(\mu_j\overline{\gamma}_j/\overline{\mu}_j) < 1$ and $(1 + |\gamma_j|^2) < 2$, it follows that λ_{j+1} is real and positive. □

There are several important identities that relate the quantities described in Definition 3.1. The following will be especially useful. All hold for $k = 1, 2, \ldots, n$.

$$\overline{\gamma}_k = \left(1 - \frac{\mu_k}{\lambda_k \mu_{k+1}}\right) \frac{\overline{\mu}_k}{\mu_k} \tag{3.4}$$

$$\overline{\mu}_k - \mu_k \overline{\gamma}_k = \frac{|\mu_k|^2}{\lambda_k \mu_{k+1}} \tag{3.5}$$

$$|\overline{\gamma}_k|^2 - 1 = \frac{|\mu_k|^2}{\lambda_k^2 |\mu_{k+1}|^2} - \frac{\mu_k}{\lambda_k \mu_{k+1}} - \frac{\overline{\mu}_k}{\lambda_k \overline{\mu}_{k+1}} \tag{3.6}$$

$$-\overline{\mu}_{k+1} = \frac{|\mu_k|^2}{\lambda_k(\overline{\mu}_k \gamma_k - \mu_k)} = \frac{\lambda_{k+1}(\mu_k \overline{\gamma}_k - \overline{\mu}_k)}{1 - |\gamma_k|^2} \tag{3.7}$$

$$\frac{\lambda_{k+1}}{\lambda_k} = \frac{|\mu_{k+1}|^2}{|\mu_k|^2}(1 - |\gamma_k|^2) \tag{3.8}$$

The first two follow immediately from (3.3a). Identity (3.6) can be obtained from (3.4) by direct computation. The left equality in (3.7) follows easily from (3.5), while the right can be verified by cross-multiplying and using (3.3b), (3.4), and (3.6). The final identity, (3.8), can be verified by using (3.5) to make a substitution into the right-hand side of (3.7).

The next proposition describes the relationship between the numbers λ_k and the prediction errors δ_k.

Proposition 3.2. *For $k = 1, 2, \ldots, n+1$, let μ_k and λ_k be defined as in Definition 3.1. Also let $\{\delta_j\}_{j=0}^n$ be the sequence of prediction errors associated with M. Then*

$$\lambda_k = 2\delta_{k-1}|\mu_k|^2. \tag{3.9}$$

Proof. A proof by induction is straightforward, requiring only (3.8) and (2.10c). □

The numbers λ_k are called *Jacobi parameters* by Delsarte and Genin because of their relationship to a certain tridiagonal matrix [26]. Notice that the positive definite nature of M is reflected in the positivity constraints

$$\lambda_k > 0; \quad k = 1, 2, \ldots, n+1.$$

The split Levinson polynomials and their properties are now described.

Definition 3.2. *Given the Szegő polynomials* $\{\rho_k(z)\}_{k=0}^{n}$ *associated with* M *and the parameters from Definition 3.1, define the* split Levinson polynomials $\{w_k(z)\}_{k=0}^{n+1}$ *as follows:* $w_0(z) = 1$ *and*

$$w_{k+1}(z) = \mu_{k+1}\hat{\rho}_k(z) + \overline{\mu}_{k+1} z \rho_k(z); \quad k = 0, 1, \ldots, n. \tag{3.10}$$

Proposition 3.3. *For* $k = 0, 1, \ldots, n+1$*, the split Levinson polynomial* w_k *has the following properties:*

i.) $w_k(0) = \mu_k$

ii.) $\deg(w_k) = k$

iii.) w_k *is conjugate symmetric. That is,* $w_k(z) = \hat{w}_k(z)$.

Proof. The properties obviously hold for $k = 0$. When $k \geq 1$, properties (i) and (ii) follow immediately from Definition 3.2, the properties of the Szegő polynomials, and the fact that μ_k is nonzero. For property (iii), notice that

$$\begin{aligned}
\hat{w}_k(z) &= z^k \overline{w}_k(1/z) \\
&= z^k \left(\overline{\mu}_k \tfrac{1}{z^{k-1}} \rho_{k-1}(z) \right) + z^k \left(\mu_k \tfrac{1}{z} \overline{\rho}_{k-1}(1/z) \right) \\
&= \overline{\mu}_k z \rho_{k-1}(z) + \mu_k z^{k-1} \overline{\rho}_{k-1}(1/z) \\
&= \overline{\mu}_k z \rho_{k-1}(z) + \mu_k \hat{\rho}_{k-1}(z) \\
&= w_k(z).
\end{aligned}$$

□

Proposition 3.4. *For $k = 0, 1, \ldots, n$, the split Levinson polynomial w_k satisfies*

$$\lambda_{k+1} w_k(z) = \mu_{k+1} \hat{\rho}_k(z) + \overline{\mu}_{k+1} \rho_k(z).$$

Proof. From Definition 3.1, $\mu_1 = \overline{\mu}_1 = 1/m_0$ and $\lambda_1 = 2/m_0$. Since $\rho_0(z) = \hat{\rho}_0(z) = 1$,

$$\lambda_1 w_0(z) = \frac{2}{m_0} = \frac{1}{m_0} + \frac{1}{m_0} = \mu_1 \hat{\rho}_0(z) + \overline{\mu}_1 \rho_0(z).$$

Therefore, the proposition holds for $k = 0$. Now assume $1 \leq k \leq n$. From the Szegő recursions (3.1) and (3.2), the right-hand side can be written

$$\mu_{k+1}(\hat{\rho}_{k-1}(z) + \overline{\gamma}_k z \rho_{k-1}(z)) + \overline{\mu}_{k+1}(z \rho_{k-1}(z) + \gamma_k \hat{\rho}_{k-1}(z)).$$

Using (3.4) to replace γ_k, gives

$$\mu_{k+1}\hat{\rho}_{k-1}(z)+\mu_{k+1}\left(1-\frac{\mu_k}{\lambda_k\mu_{k+1}}\right)\frac{\overline{\mu}_k}{\mu_k}z\rho_{k-1}(z)+\overline{\mu}_{k+1}z\rho_{k-1}(z)+\overline{\mu}_{k+1}\left(1-\frac{\overline{\mu}_k}{\lambda_k\overline{\mu}_{k+1}}\right)\frac{\mu_k}{\overline{\mu}_k}\hat{\rho}_{k-1}(z).$$

Expanding and factoring now gives

$$\mu_k\hat{\rho}_{k-1}(z)\left[\frac{\mu_{k+1}}{\mu_k}+\frac{\overline{\mu}_{k+1}}{\overline{\mu}_k}-\frac{1}{\lambda_k}\right]+\overline{\mu}_k z\rho_{k-1}(z)\left[\frac{\mu_{k+1}}{\mu_k}+\frac{\overline{\mu}_{k+1}}{\overline{\mu}_k}-\frac{1}{\lambda_k}\right].$$

Finally, using the definition of λ_{k+1}, (3.3b), in the expression above gives

$$\lambda_{k+1}(\mu_k\hat{\rho}_{k-1}(z)+\overline{\mu}_k z\rho_{k-1}(z)),$$

which reduces to $\lambda_{k+1}w_k(z)$ by definition of w_k. □

Proposition 3.5. *For $k=0,1,\ldots,n+1$, $w_k(1)$ is a nonzero real number. Furthermore, for $k\geq 1$,*

$$\lambda_k=\frac{w_k(1)}{w_{k-1}(1)}.$$

Proof. It follows from the conjugate symmetry of w_k that $w_k(1)$ is real. That $w_k(1)\neq 0$ and $\lambda_k=w_k(1)/w_{k-1}(1)$ follow by induction, using Proposition 3.4 and Definition 3.2. □

Proposition 3.6 (Split Levinson recurrence relation)**.** *For $k=1,2,\ldots n$, the split Levinson polynomials satisfy the three-term recurrence relation*

$$w_{k+1}(z)=(\zeta_k+\overline{\zeta}_k z)w_k(z)-zw_{k-1}(z), \tag{3.11}$$

where $\zeta_k = \mu_{k+1}/\mu_k$.

Proof. Assume $1 \leq k \leq n$. Definition 3.2 gives

$$w_{k+1}(z) = \mu_{k+1}\hat{\rho}_k(z) + \overline{\mu}_{k+1} z \rho_k(z).$$

With the Szegő recursions, the right-hand side becomes

$$\mu_{k+1}\left[\hat{\rho}_{k-1}(z) + \overline{\gamma}_k z \rho_{k-1}(z)\right] + \overline{\mu}_{k+1} z \left[z\rho_{k-1}(z) + \gamma_k \hat{\rho}_{k-1}(z)\right].$$

Expanding this expression and rearranging terms gives

$$\mu_{k+1}\hat{\rho}_{k-1}(z) + \overline{\mu}_{k+1} z^2 \rho_{k-1}(z) + \overline{\mu}_{k+1} z \gamma_k \hat{\rho}_{k-1}(z) + \mu_{k+1}\overline{\gamma}_k z \rho_{k-1}(z).$$

Using (3.4) to replace γ_k gives

$$\mu_{k+1}\hat{\rho}_{k-1}(z) + \overline{\mu}_{k+1} z^2 \rho_{k-1}(z) + z\hat{\rho}_{k-1}(z)\overline{\mu}_{k+1}\left(1 - \frac{\overline{\mu}_k}{\lambda_k \overline{\mu}_{k+1}}\right)\frac{\mu_k}{\overline{\mu}_k} + z\rho_{k-1}(z)\mu_{k+1}\left(1 - \frac{\mu_k}{\lambda_k \mu_{k+1}}\right)\frac{\overline{\mu}_k}{\mu_k}.$$

After expanding and rearranging terms, this becomes

$$\mu_{k+1}\hat{\rho}_{k-1}(z) + z\hat{\rho}_{k-1}(z)\mu_k\frac{\overline{\mu}_{k+1}}{\overline{\mu}_k} + z\rho_{k-1}(z)\overline{\mu}_k\frac{\mu_{k+1}}{\mu_k} + \overline{\mu}_{k+1} z^2 \rho_{k-1}(z) - z\hat{\rho}_{k-1}(z)\frac{\mu_k}{\lambda_k} - z\rho_{k-1}(z)\frac{\overline{\mu}_k}{\lambda_k}.$$

Finally, after factoring and using Definition 3.2 and Proposition 3.4, this expression reduces to

$$\left(\frac{\mu_{k+1}}{\mu_k} + \frac{\overline{\mu}_{k+1}}{\overline{\mu}_k} z\right) [\mu_k \hat{\rho}_{k-1}(z) + \overline{\mu}_k z \rho_{k-1}(z)] - z \left[\frac{\mu_k}{\lambda_k} \hat{\rho}_{k-1}(z) + \frac{\overline{\mu}_k}{\lambda_k} \rho_{k-1}(z)\right]$$

$$= (\zeta_k + \overline{\zeta}_k z) w_k(z) - z w_{k-1}(z).$$

□

At this point, an important observation is in order. The modified split Levinson recurrence relation

$$\varpi_{k+1}(z) = (\overline{\zeta}_k + \zeta_k z) \varpi_k(z) - z \varpi_{k-1}(z) \tag{3.12}$$

simply generates the split Levinson polynomials with conjugated coefficients. More precisely, if (3.12) is applied with $\varpi_k(z) = \overline{w}_k(z)$ and $\varpi_{k-1}(z) = \overline{w}_{k-1}(z)$, then $\varpi_{k+1}(z) = \overline{w}_{k+1}(z)$. Given the conjugate symmetry of the split Levinson polynomials, $\overline{w}_{k+1}$ is simply w_{k+1} with its coefficients reversed. Taking a slightly different point of view, if initialized with $\varpi_0 = \overline{w}_0$ and $\varpi_1 = \overline{w}_1$, the recurrence relation (3.12) generates the split Levinson polynomials associated with $\overline{M}$.

The next proposition shows how the Szegő polynomials can be obtained from the split Levinson polynomials.

Proposition 3.7. *With the split Levinson polynomials defined as in Definition 3.2, the Szegő polynomial ρ_k satisfies*

$$\mu_{k+1}(1 - z) \hat{\rho}_k(z) = w_{k+1}(z) - \lambda_{k+1} z w_k(z); \quad k = 0, 1, \ldots, n.$$

Proof. Assume $0 \leq k \leq n$. By Definition 3.2,

$$w_{k+1}(z) - \lambda_{k+1} z w_k(z) = \left[\mu_{k+1}\hat{\rho}_k(z) + \overline{\mu}_{k+1} z \rho_k(z)\right] - \lambda_{k+1} z w_k(z).$$

Now, by Proposition 3.4, the right-hand side becomes

$$\left[\mu_{k+1}\hat{\rho}_k(z) + \overline{\mu}_{k+1} z \rho_k(z)\right] - z\left[\mu_{k+1}\hat{\rho}_k(z) + \overline{\mu}_{k+1} \rho_k(z)\right].$$

After expanding, the expression above reduces to $\mu_{k+1}\hat{\rho}_k(z)(1-z)$. □

It follows from Proposition 3.7 that the Szegő polynomials and their reciprocals can be recovered from the split Levinson polynomials by the formulas:

$$\rho_k(z) = \frac{1}{\overline{\mu}_{k+1}(z-1)}\left[w_{k+1}(z) - \lambda_{k+1} w_k(z)\right]; \quad k = 0, 1, \ldots, n, \tag{3.13}$$

$$\hat{\rho}_k(z) = \frac{1}{\mu_{k+1}(1-z)}\left[w_{k+1}(z) - \lambda_{k+1} z w_k(z)\right]; \quad k = 0, 1, \ldots, n. \tag{3.14}$$

3.3 The Algorithm

The split Levinson algorithm processes the family of split Levinson polynomials via the recurrence relation (3.11). It is clear from the formulas presented above that the Szegő polynomials, Schur parameters, and predictions errors can be recovered from corresponding quantities associated with the split Levinson recursions. However, as it stands now, the split

Levinson parameters (μ_k, λ_k, and ζ_k) are defined in terms of the Szegő recursion's parameters. Before the split Levinson recursions can be useful alternatives to the Szegő recursions, the parameters must be made independent of the Schur parameters and prediction errors.

Recall that the coefficients of the kth Szegő polynomial and its reciprocal form vectors that satisfy the Yule-Walker equations (2.8) and (2.6), respectively. More precisely, if ρ_k and $\hat{\rho}_k$ are given by

$$\rho_k(z) = \sum_{j=0}^{k} r_{j,k}\, z^j \quad \text{and} \quad \hat{\rho}_k(z) = \sum_{j=0}^{k} \overline{r}_{k-j,k}\, z^j,$$

then

$$M_{k+1} \begin{bmatrix} r_{0,k} \\ \vdots \\ r_{k-1,k} \\ r_{k,k} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \delta_k \end{bmatrix} \quad \text{and} \quad M_{k+1} \begin{bmatrix} \overline{r}_{k,k} \\ \overline{r}_{k-1,k} \\ \vdots \\ \overline{r}_{0,k} \end{bmatrix} = \begin{bmatrix} \delta_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Now let $w_{j,k}$ denote the coefficient of z^j in the split Levinson polynomial w_k so that

$$w_k(z) = \sum_{j=0}^{k} w_{j,k}\, z^j; \quad k = 0, 1, \ldots, n+1.$$

It follows from Proposition 3.4 that

$$\lambda_{k+1} M_{k+1} \begin{bmatrix} w_{0,k} \\ w_{1,k} \\ \vdots \\ w_{k,k} \end{bmatrix} = \mu_{k+1} M_{k+1} \begin{bmatrix} \overline{r}_{k,k} \\ \vdots \\ \overline{r}_{1,k} \\ \overline{r}_{0,k} \end{bmatrix} + \overline{\mu}_{k+1} M_{k+1} \begin{bmatrix} r_{0,k} \\ r_{1,k} \\ \vdots \\ r_{k,k} \end{bmatrix} = \mu_{k+1} \begin{bmatrix} \delta_k \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \overline{\mu}_{k+1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \delta_k \end{bmatrix}.$$

Since λ_{k+1} and δ_k are positive real numbers, the vector equation above can be written

$$M_{k+1}\begin{bmatrix} w_{0,k} \\ w_{1,k} \\ \vdots \\ w_{k,k} \end{bmatrix} = \begin{bmatrix} \nu_k \\ 0 \\ \vdots \\ \overline{\nu}_k \end{bmatrix}; \quad k = 1, 2, \ldots, n, \tag{3.15}$$

where $\nu_k = \mu_{k+1}\delta_k/\lambda_{k+1}$. Equation (3.15) also gives the following expression for ν_k:

$$\nu_k = \sum_{j=0}^{k} \overline{m}_j \, w_{j,k}; \quad k = 1, 2, \ldots, n. \tag{3.16a}$$

While equations (3.15) and (3.16a) do not apply when $k = 0$, the definition

$$\nu_k = \frac{\mu_{k+1}\delta_k}{\lambda_{k+1}}$$

can be extended to include the case for $k = 0$ by defining, via Proposition 3.4,

$$\nu_0 = \frac{\mu_1\delta_0}{\lambda_1} = \frac{m_0}{2}. \tag{3.16b}$$

With the expressions for ν_k given above, it follows that $\nu_k \neq 0$ and

$$\frac{\nu_{k-1}}{\nu_k} = \frac{\mu_k\delta_{k-1}}{\lambda_k} \cdot \frac{\lambda_{k+1}}{\mu_{k+1}\delta_k}; \quad k = 1, 2, \ldots, n.$$

Algorithm 3.1 Split Levinson Algorithm

Input: $[m_0, m_1, \ldots, m_n]^T$ = first column of M
Initialize: $w_0(z) = 1$; $w_1(z) = (1+z)/m_0$; $\nu_0 = m_0/2$; $\lambda_1 = 2/m_0$
for $k = 1$ to n **do**

$$\nu_k = \sum_{j=0}^{k} \overline{m}_j \, w_{j,k}$$

$$\zeta_k = \overline{\nu}_{k-1}/\overline{\nu}_k$$

$$w_{k+1}(z) = (\zeta_k + \overline{\zeta}_k z) w_k(z) - z w_{k-1}(z)$$

$$\lambda_{k+1} = \zeta_k + \overline{\zeta}_k - 1/\lambda_k$$

$$\gamma_k = \left(1 - \frac{1}{\lambda_k \overline{\zeta}_k}\right) \frac{w_{0,k}}{\overline{w}_{0,k}}$$

end for

$$\rho_n(z) = \frac{1}{\overline{w}_{0,n+1}} \left(\frac{w_{n+1}(z) - \lambda_{n+1} w_n(z)}{z-1} \right)$$

$$\delta_n = \lambda_{n+1}/(2\,|w_{0,n+1}|^2)$$

Output: ρ_n; δ_n; $\gamma_1, \gamma_2, \ldots, \gamma_n$

Using (3.8) and (2.10c), this equation becomes

$$\frac{\nu_{k-1}}{\nu_k} = \frac{\overline{\mu}_{k+1}}{\overline{\mu}_k} = \overline{\zeta}_k; \quad k = 1, 2, \ldots, n, \tag{3.17}$$

where ζ_k was defined in Proposition 3.6.

Equations (3.16) and (3.17) now make the split Levinson recurrence relation (3.11) independent of the Szegő polynomials. With the intent of obtaining $\rho_n(z)$ and δ_n, the split Levinson algorithm is initialized with $w_0(z) = 1$, $w_1(z) = (1+z)/m_0$, $\nu_0 = m_0/2$ and $\lambda_1 = 2/m_0$ and recursively implements (3.16a), (3.17), (3.11), and (3.3b). The nth Szegő polynomial and prediction error follow from equations (3.13) and (3.9), respectively. If the Schur parameters are required, they can be obtained from (3.4), recalling that $w_k(0) = w_{0,k} = \mu_k$.

The split Levinson algorithm is given in Algorithm 3.1. Because of the symmetry in the split Levinson polynomials, some of the multiplications in (3.16a) are redundant. Furthermore, only one-half of the coefficients need be computed in the split Levinson recurrence relation (3.11). With these strategies implemented in Algorithm 3.1, the split Levinson algorithm requires $\frac{3}{2}n^2 + O(n)$ flops in the case that M is real and $6n^2 + O(n)$ flops in the case that M is complex. In each case, this amounts to a 25% reduction in operations over the Levinson-Durbin algorithm. A careful look shows that the savings are entirely in multiplications. Even though the split Levinson polynomials are symmetric, two consecutive polynomials are always required in algorithm. Therefore, there is no reduction is storage.

Krishna and Wang [45] investigated the stability of the split Levinson algorithm when applied to real symmetric positive definite Toeplitz matrices. They showed that the algorithm is weakly stable, and they established bounds on the residual similar to those found by Cybenko [21] in his analysis of the Levinson-Durbin algorithm. In particular, their bound on the residual, $y = M_n\tilde{x} + [m_1, m_2, \ldots, m_n]^T$, is given by

$$\|y\|_1 \leq \Delta \left(20n^3 + 18n^2 - 14n\right) \prod_{k=1}^{n}(1 + |\gamma_k|), \tag{3.18}$$

where Δ is the machine precision. Krishna and Wang also suggested that the weak stability of the algorithm is the most that could be expected. Their practical conclusion was that if n is large or M is ill-conditioned, the residual may be large.

It is interesting to view Cybenko's bounds on $\|M^{-1}\|_1$ (see section 2.2.6) in light of the parameters of the split Levinson algorithm. Assuming that M is a real symmetric positive definite matrix with $m_0 = 1$, it follows from (3.9), (3.4), and the definition of ζ_j that

$$\frac{1}{\delta_n} = \frac{2\mu_{n+1}^2}{\lambda_{n+1}} \quad \text{and} \quad \frac{1}{\prod_{j=1}^{n}(1-\gamma_j)} = \prod_{j=1}^{n} \lambda_j \zeta_j = \mu_{n+1} \prod_{j=1}^{n} \lambda_j.$$

Therefore, with $\kappa = \prod_{j=1}^{n} \lambda_j$, Cybenko's lower bound in (2.18) takes the form

$$\max\left\{ \frac{2\mu_{n+1}^2}{\lambda_{n+1}}, \mu_{n+1}\kappa \right\} \leq \|M^{-1}\|_1. \tag{3.19}$$

Now, with the given conditions on M and the help of (3.4) and (3.3b), it can be shown that

$$\lambda_{j+1}\lambda_j = \frac{1+\gamma_j}{1-\gamma_j}.$$

In the case that all Schur parameters are positive, it then follows that Cybenko's upper bound in (2.18) takes the form

$$\|M^{-1}\|_1 \leq \prod_{j=1}^{n} \lambda_{j+1}\lambda_j = \frac{\lambda_{n+1}\kappa^2}{2}. \tag{3.20}$$

3.4 Example

Consider the 4×4 Hermitian Toeplitz matrix

$$M = \begin{bmatrix} 8 & 4+i & 2 & 1-i \\ 4-i & 8 & 4+i & 2 \\ 2 & 4-i & 8 & 4+i \\ 1+i & 2 & 4-i & 8 \end{bmatrix}.$$

It is not difficult to verify that M is positive definite, as its leading principal minors are 8, 47, 268, and 1497. Applying Algorithm 3.1 in exact arithmetic yields the following results:

$$\begin{aligned} \rho_3(z) &= \tfrac{13+36i}{268} - \tfrac{10-15i}{134} z - \tfrac{131+60i}{268} z^2 + z^3 \\ \delta_3 &= \tfrac{1497}{268} \\ \{\gamma_k\}_{k=1}^3 &= \left\{ -\tfrac{4+i}{8}, -\tfrac{1-8i}{47}, \tfrac{13+36i}{268} \right\}. \end{aligned}$$

In addition,

$$\begin{aligned} \{\lambda_k\}_{k=1}^4 &\approx \{0.25, 1.30, 0.74, 1.37\} \\ \{\zeta_k\}_{k=1}^3 &\approx \{2.65 + 0.22i, 0.75 - 0.13i, 1.36 - 0.20i\} \\ w_4(z) &\approx (0.34 - 0.08i) - (0.14 - 0.17i)z - 0.07z^2 - (0.14 + 0.17i)z^3 + (0.34 + 0.08i)z^4 \\ w_3(z) &\approx (0.25 - 0.02i) - (0.13 - 0.11i)z - (0.13 + 0.11i)z^2 + (0.25 + 0.02i)z^3. \end{aligned}$$

Notice that, as expected, the positive definite nature of M is reflected in the inequalities $\delta_3 > 0$, $|\gamma_k| < 1$, and $\lambda_k > 0$.

CHAPTER 4

SPLIT SCHUR ALGORITHM

Schur's classical algorithm was presented in Section 2.2.4 as a means of parameterizing a Schur function and thereby obtaining a continued fraction representation for it. The algorithm was also described as a method for constructing a sequence of Schur functions from an initial Schur function. Written in terms of the numerators and denominators of Schur functions, Schur's algorithm led to a fast algorithm (Algorithm 2.2) for the Cholesky factorization of an HPD Toeplitz matrix. In certain applications, the Schur parameters $\{\gamma_k\}_{k=1}^n$ are the important quantities (see [36, 47]), and the matrix factorization itself is not required. Viewed in this light, Schur's algorithm provides an $O(n^2)$ method for computing the Schur parameters. In applications, the Schur parameters are usually called reflection coefficients, as in transmission line theory [17, 29] and seismology [50], or partial correlation coefficients, as in statistics.

After Delsarte and Genin split the Levinson-Durbin algorithm, they used similar approaches to split several classical algorithms in linear prediction theory [24, 25]. Among those algorithms was the Schur algorithm. Originally presented for the case of real data, the split Schur algorithm was later extended to complex data [26]. While the Schur algorithm is naturally associated with Schur functions, it turns out that the split Schur algorithm is similarly associated with Carathéodory functions.

In this chapter, a complete derivation of the split Schur algorithm is presented, and the connections to Carathéodory functions are described. The split Schur algorithm will be seen as a means of parameterizing functions in a certain class of Carathéodory-like functions. Just as Schur's algorithm provides a continued fraction representation for a Schur function, the split Schur algorithm provides a continued fraction representation for its kind of function.

4.1 Carathéodory Functions

A *Carathéodory function* is an analytic function $s = f(z)$ that maps the open unit disk $\{z : |z| < 1\}$ into the closed right half-plane $\{s : \mathrm{Re}(s) \geq 0\}$. It follows from the open mapping theorem that a nonconstant Carathéodory function must be nonzero on the open unit disk. It is easy to see that the conjugate and the reciprocal of a (nonzero) Carathéodory function are Carathédory functions, as is the sum of two Carathéodory functions.

Recall that a Schur function is an analytic function that maps the open unit disk into its closure. There is a one-to-one correspondence between the class of Carathéodory functions and the class of Schur functions. Indeed, $f(z)$ is a Carathéodory function if and only if

$$\phi(z) = \frac{\frac{1}{z}[f(0) - f(z)]}{\overline{f(0)} + f(z)} \tag{4.1}$$

is a Schur function (see [1]).

There is a connection between Carathéodory functions and Toeplitz matrices, which is made clear by the Carathéodory-Toeplitz theorem (see [1]).

Theorem 4.1 (Carathéodory-Toeplitz Theorem)**.** *Consider the power series*

$$f(z) = m + \sum_{j=1}^{\infty} m_j \, z^j$$

and the Hermitian Toeplitz matrix $M_{n+1} = [m_{j-k}]_{j,k=0}^{n}$*, where* $m_0 = 2\,\mathrm{Re}(m)$ *and* $m_{-j} = \overline{m}_j$*. The function* f *is a Carathéodory function if and only if the matrix* M_{n+1} *is nonnegative definite for each integer* $n \geq 0$*.*

4.2 Split Schur Functions

Suppose $M_\infty = [m_{j-k}]_{j,k=0}^{\infty}$ is an infinite Hermitian Toeplitz matrix with the property that every leading $k \times k$ submatrix is positive definite. Further assume that the elements of M_∞ are absolutely summable, i.e.,

$$\sum_{k=0}^{\infty} |m_k| < \infty.$$

Such a matrix can always be constructed (extended) in infinitely many ways from a finite HPD Toeplitz matrix via the Szegő recursions. Under the given conditions, it follows from the Weierstrass M-test that

$$F(z) = \frac{m_0}{2} + \overline{m}_1 z + \overline{m}_2 z^2 + \cdots + \overline{m}_n z^n + \cdots$$

is uniformly convergent on the open unit disk $\mathcal{D} = \{z : |z| < 1\}$. It also follows from the Carathéodory-Toeplitz theorem that $F(z)$ is a Carathéodory function.

Now define the powers series $\alpha_0(z)$ and $\beta_0(z)$ by

$$\begin{aligned} \alpha_0(z) &= \tfrac{1}{z}[F(0) - F(z)] &&= -\overline{m}_1 - \overline{m}_2 z - \overline{m}_3 z^2 - \cdots - \overline{m}_n z^{n-1} + \cdots \\ \beta_0(z) &= \overline{F(0)} + F(z) &&= m_0 + \overline{m}_1 z + \overline{m}_2 z^2 + \cdots + \overline{m}_n z^n + \cdots . \end{aligned}$$

These series converge uniformly on $\mathcal{D}$, and by the correspondence (4.1), the ratio $\phi_0 = \alpha_0/\beta_0$ is a Schur function. Starting with ϕ_0, Schur's algorithm (see Section 2.2.4) generates the sequence of Schur functions $\{\phi_k = \alpha_k/\beta_k\}_{k=0}^{\infty}$, where the numerators and denominators satisfy

$$\begin{aligned} \alpha_k(z) &= \tfrac{1}{z}(\alpha_{k-1}(z) - \gamma_k \beta_{k-1}(z)) \\ \beta_k(z) &= \beta_{k-1}(z) - \overline{\gamma}_k \alpha_{k-1}(z). \end{aligned} \tag{4.2}$$

It is easy to verify that the backward versions of the recurrence relations (4.2) are given by

$$\begin{aligned} \alpha_{k-1}(z) &= \frac{z\alpha_k(z) + \gamma_k \beta_k(z)}{1 - |\gamma_k|^2} \\ \beta_{k-1}(z) &= \frac{\overline{\gamma}_k z\alpha_k(z) + \beta_k(z)}{1 - |\gamma_k|^2}. \end{aligned} \tag{4.3}$$

As described in Theorem 2.2, the Schur functions give the Cholesky factorization, Schur parameters, and prediction errors associated with each leading principal submatrix of M_∞. Since those submatrices are positive definite, each Schur parameter in the infinite sequence of parameters has magnitude less than 1. It follows that the sequences of split Levinson

parameters defined in Definition 3.1 can be extended ad infinitum, while preserving the properties described in Proposition 3.1.

The splitting of Schur's algorithm begins with the splitting of the Schur functions that it generates. The *split Schur functions* are defined next, and a number of their important properties are laid out.

Definition 4.1. *With M_∞, $\{\alpha_k(z)\}_{k=0}^\infty$, and $\{\beta_k(z)\}_{k=0}^\infty$ as described above, let $\{\mu_k\}_{k=0}^\infty$ be the extended sequence of parameters from Definition 3.1. The* split Schur functions *associated with M_∞ (or $F(z)$) are defined as follows:*

$$\begin{aligned} h_{-1}(z) &= \tfrac{1}{2} - \tfrac{1}{2}z, \\ h_0(z) &= F(z) = \tfrac{m_0}{2} + \overline{m}_1 z + \overline{m}_2 z^2 + \cdots + \overline{m}_n z^n + \cdots, \\ h_{k+1}(z) &= \mu_{k+1}\beta_k(z) - \overline{\mu}_{k+1}\alpha_k(z); \quad k = 0, 1, 2, \ldots. \end{aligned}$$

Proposition 4.1. *For each k, the split Schur function $h_k(z)$ is uniformly convergent and nonzero on the open unit disk $\mathcal{D} = \{z : |z| < 1\}$.*

Proof. $h_{-1}(z)$ is clearly a nonzero polynomial on $\mathcal{D}$. By definition, h_0 is the uniformly convergent Carathéodory function $F(z)$. Since m_0 must be positive, h_0 is not identically zero. It follows from the open mapping theorem that $h_0(z)$ must be nonzero on $\mathcal{D}$.

Now since the coefficients of $F(z)$ are absolutely summable, so are the coefficients of $\alpha_0(z)$ and $\beta_0(z)$. It follows by induction that the coefficients of $\alpha_k(z)$ and $\beta_k(z)$ are also absolutely summable. Thus each series converges uniformly on $\mathcal{D}$ by the Weierstrass M-test. In turn, it follows that $h_k(z)$ is uniformly convergent for each k.

To show that $h_k(z)$ is nonzero, let $k \geq 1$ and suppose to the contrary that $h_k(z) = 0$ for some $z \in \mathcal{D}$. Then

$$h_k(z) = \mu_k \beta_{k-1}(z) - \overline{\mu}_k \alpha_{k-1}(z) = 0.$$

By Proposition 3.1, $\mu_k \neq 0$. Furthermore, since $\beta_{k-1}(z)$ is the denominator of a well-defined Schur function, it must be nonzero on $\mathcal{D}$. Therefore,

$$h_k(z) = 0 \quad \Longrightarrow \quad \frac{\overline{\mu}_k \alpha_{k-1}(z)}{\mu_k \beta_{k-1}(z)} = 1,$$

from which it follows that $|\alpha_{k-1}(z)/\beta_{k-1}(z)| = 1$. By the maximum principle, $\alpha_{k-1}(z)/\beta_{k-1}(z)$ must be a constant function with magnitude 1. However, this contradicts the fact that $|\alpha_{k-1}(0)/\beta_{k-1}(0)| = |\gamma_k| < 1$. □

At this point, a technical lemma is required. The purpose of the lemma is twofold. First, it will provide a new means for computing the ζ_k-parameters defined in Proposition 3.6. Next, it will aid in the derivation of a recurrence relation for the split Schur functions.

Lemma 4.1. *Given the extended sequences of parameters from Definition 3.1, the Schur function numerators* $\{\alpha_k\}_{k=0}^{\infty}$*, and the split Schur functions from Definition 4.1, it follows that*

$$\frac{\overline{\mu}_{k+1}}{\overline{\mu}_k} h_k(z) - h_{k-1}(z) = -\overline{\mu}_{k+1} z \alpha_k(z) + \frac{\overline{\mu}_k}{\lambda_k} z \alpha_{k-1}(z); \quad k = 1, 2, 3, \ldots.$$

Proof. For $k = 2, 3, 4, \ldots$, Definition 4.1 gives

$$\frac{\overline{\mu}_{k+1}}{\overline{\mu}_k} h_k(z) - h_{k-1}(z) = \frac{\overline{\mu}_{k+1}}{\overline{\mu}_k}(\mu_k \beta_{k-1}(z) - \overline{\mu}_k \alpha_{k-1}(z)) - \mu_{k-1}\beta_{k-2}(z) + \overline{\mu}_{k-1}\alpha_{k-2}(z).$$

After using (4.3) to replace α_{k-2} and β_{k-2}, the right-hand side can be written

$$\left[\frac{\overline{\mu}_{k+1}\mu_k}{\overline{\mu}_k} + \frac{\overline{\mu}_{k-1}\gamma_{k-1} - \mu_{k-1}}{1 - |\gamma_{k-1}|^2}\right] \beta_{k-1}(z) - \overline{\mu}_{k+1}\alpha_{k-1}(z) + z\left(\frac{\overline{\mu}_{k-1} - \mu_{k-1}\overline{\gamma}_{k-1}}{1 - |\gamma_{k-1}|^2}\right) \alpha_{k-1}(z).$$

With the use of (3.7), this expression reduces to

$$\left[\frac{\overline{\mu}_{k+1}\mu_k}{\overline{\mu}_k} - \frac{\mu_k}{\lambda_k}\right] \beta_{k-1}(z) - \overline{\mu}_{k+1}\alpha_{k-1}(z) + \frac{\overline{\mu}_k}{\lambda_k} z\alpha_{k-1}(z),$$

and with (3.4), it further reduces to

$$\overline{\mu}_{k+1}\gamma_k\beta_{k-1}(z) - \overline{\mu}_{k+1}\alpha_{k-1}(z) + \frac{\overline{\mu}_k}{\lambda_k} z\alpha_{k-1}(z).$$

Finally, with the recurrence relations (4.2), this becomes

$$-\overline{\mu}_{k+1} z\alpha_k(z) + \frac{\overline{\mu}_k}{\lambda_k} z\alpha_{k-1}(z),$$

as asserted.

While computationally tedious, the case for $k = 1$ can be verified directly by showing that each side is formally equal to

$$\frac{m_0\overline{m}_2 - 2\overline{m}_1\overline{m}_1 - m_0\overline{m}_1}{2(m_0 + \overline{m}_1)}z + \frac{m_0\overline{m}_3 - 2\overline{m}_1\overline{m}_2 - m_0\overline{m}_2}{2(m_0 + \overline{m}_1)}z^2 + \cdots$$

$$+ \frac{m_0\overline{m}_n - 2\overline{m}_1\overline{m}_{n-1} - m_0\overline{m}_{n-1}}{2(m_0 + \overline{m}_1)}z^{n-1} + \frac{m_0\overline{m}_{n+1} - 2\overline{m}_1\overline{m}_n - m_0\overline{m}_n}{2(m_0 + \overline{m}_1)}z^n + \cdots .$$

□

Proposition 4.2. *Let $\zeta_k = \mu_{k+1}/\mu_k$ as originally defined in Proposition 3.6. Then for $k = 0, 1, 2, \ldots,$*

$$\overline{\zeta}_k h_k(0) - h_{k-1}(0) = 0,$$

and therefore $\zeta_k = \overline{h_{k-1}(0)}/\overline{h_k(0)}$.

Proof. The result can be directly verified for $k = 0$. For $k \geq 1$, the result follows immediately from Lemma 4.1 and Proposition 4.1 □

Proposition 4.3 (Split Schur recurrence relation)**.** *With $\zeta_k = \overline{h_{k-1}(0)}/\overline{h_k(0)}$ as described in Proposition 4.2, the split Schur functions satisfy the following three-term recurrence relation:*

$$h_{k+1}(z) = \zeta_k h_k(z) + \frac{1}{z}\left[\overline{\zeta}_k h_k(z) - h_{k-1}(z)\right]; \quad k = 0, 1, 2, \ldots. \tag{4.4}$$

Proof. The case for $k = 0$ can be verified directly. For $k = 1, 2, 3, \ldots$, Lemma 4.1 and the definition of h_k make the right-hand side equal to

$$\frac{\mu_{k+1}}{\mu_k}(\mu_k\beta_{k-1}(z) - \overline{\mu}_k\alpha_{k-1}(z)) - \overline{\mu}_{k+1}\alpha_k(z) + \frac{\overline{\mu}_k}{\lambda_k}\alpha_{k-1}(z).$$

After expanding and rearranging terms, this can be written

$$\mu_{k+1}\left[\beta_{k-1}(z) - \left(\frac{\overline{\mu}_k}{\mu_k} - \frac{\overline{\mu}_k}{\lambda_k\mu_{k+1}}\right)\alpha_{k-1}(z)\right] - \overline{\mu}_{k+1}\alpha_k(z).$$

With (3.4), this reduces to

$$\mu_{k+1}\left[\beta_{k-1}(z) - \overline{\gamma}_k\alpha_{k-1}(z)\right] - \overline{\mu}_{k+1}\alpha_k(z).$$

After using the recurrence relations (4.2), this expression reduces to

$$\mu_{k+1}\beta_k(z) - \overline{\mu}_{k+1}\alpha_k(z),$$

which is precisely $h_{k+1}(z)$. □

The last result in this section highlights another connection between the split Schur functions and the split Levinson polynomials. This result will be used in the split Schur algorithm to simplify the computation of the Schur parameters.

Proposition 4.4. *Given the split Levinson parameters and the prediction errors associated with the leading principal submatrices of M_∞, let*

$$\nu_k = \frac{\mu_{k+1}\delta_k}{\lambda_{k+1}}; \quad k = 0, 1, 2, \ldots,$$

as described in Section 3.3. Then

$$\nu_k = h_k(0) \quad \textit{and} \quad \frac{\nu_k}{\overline{\nu}_k} = \frac{h_k(0)}{\overline{h_k(0)}} = \frac{\mu_{k+1}}{\overline{\mu}_{k+1}}.$$

Proof. The proof follows by induction using (3.16b), (3.17), and Proposition 4.2. □

4.3 The Algorithm

With Proposition 4.2, the split Schur recurrence relation (4.4) is independent of the parameters of the Levinson-Durbin and split Levinson algorithms. Furthermore, with the help of Proposition 4.4, the Schur parameters can be computed very easily from the formulas (3.3b) and (3.4):

$$\lambda_{k+1} = 2\,\mathrm{Re}(\zeta_k) - \frac{1}{\lambda_k}, \qquad \gamma_{k+1} = \left(1 - \frac{1}{\lambda_{k+1}\overline{\zeta}_{k+1}}\right)\frac{h_k(0)}{\overline{h_k(0)}}. \tag{4.5}$$

Initialized with $\zeta_0 = 1/m_0$, $\lambda_0 = \infty$, and h_{-1} and h_0 from Definition 4.1, the split Schur algorithm uses the split Schur recurrence relation and the formulas above to compute the

Algorithm 4.1 Progressive Split Schur Algorithm

Input: $F(z) = m_0/2 + \overline{m}_1 z + \overline{m}_2 z^2 + \cdots + \overline{m}_n z^n + \cdots$
Initialize: $h_{0,0} = m_0/2$; $h_{1,-1} = -1/2$; $\zeta_0 = 1/m_0$; $\lambda_0 = \infty$
for $k = 0, 1, 2, \ldots,$ **do**
  $h_{k+1,0} = \overline{m}_{k+1}$
  if $(k \neq 0)$ $h_{k+1,-1} = 0$
  $\lambda_{k+1} = 2\,\mathrm{Re}(\zeta_k) - \frac{1}{\lambda_k}$
  for $j = 0$ to k **do**
    $h_{k-j,j+1} = \zeta_j h_{k-j,j} + \overline{\zeta}_j h_{k-j+1,j} - h_{k-j+1,j-1}$
  end for
  $\zeta_{k+1} = \overline{h}_{0,k}/\overline{h}_{0,k+1}$
  $\gamma_{k+1} = \left(1 - \dfrac{1}{\lambda_{k+1}\overline{\zeta}_{k+1}}\right) \dfrac{h_{0,k}}{\overline{h}_{0,k}}$
end for
Output: $\gamma_1, \gamma_2, \gamma_3, \ldots$

split Schur functions and Schur parameters associated with M_∞. As in the case of applying Schur's algorithm to $\alpha_0(z)/\beta_0(z)$ (see Section 2.2.4), these computations involve operations on infinite power series. Nonetheless, the operations can be arranged sequentially so that each of the parameters λ_k, ζ_k, and γ_k can be obtained in a finite number of arithmetic operations. This is the essence of the progressive split Schur algorithm given in Algorithm 4.1, where coefficients of consecutive split Schur functions are computed one at a time. It is clear from the derivations of formulas (4.4) and (4.5) that the parameters λ_k and ζ_k are identical to the corresponding parameters of the split Levinson algorithm when that algorithm is applied to the finite leading principal submatrices of M_∞.

If the progressive split Schur algorithm is used to compute the Schur parameters of M_{n+1}, the leading $(n+1) \times (n+1)$ submatrix of M_∞, then the outer k-loop can be stopped at

$k = n - 1$. The resulting algorithm requires $\frac{3}{2}n^2 + O(n)$ flops in the real case or $6n^2 + O(n)$ flops in the complex case. These flop counts match those of the split Levinson algorithm.

Notice that the work in the progressive split Schur algorithm is back loaded in the sense that the number of operations required by the inner loop increases with k. In its application to the finite matrix M_{n+1}, the computations can be rearranged so that the work is front loaded.

Let $h_k^{(p)}$ denote the polynomial obtained from h_k by removing all terms of degree p or greater. In order to compute the final Schur parameter γ_n, only $h_n^{(1)}$ and $h_{n-1}^{(1)}$ are required. In order to compute the first term of each of h_n and h_{n-1}, two terms of h_{n-1} and h_{n-2} are required. In general, in order to compute $h_k^{(p)}$, only $h_{k-1}^{(p+1)}$ and $h_{k-2}^{(p+1)}$ are required. Thus, when used to compute the Schur parameters of M_{n+1}, the split Schur recurrence relation (4.4) can be replaced with its streamlined version

$$h_{k+1}^{(n-k)}(z) = \zeta_k h_k^{(n-k)}(z) + \frac{1}{z}\left[\overline{\zeta}_k h_k^{(n-k+1)}(z) - h_{k-1}^{(n-k+1)}(z)\right]; \quad k = 0, 1, 2, \ldots, n-1.$$

This results in the polynomial split Schur algorithm given in Algorithm 4.2, whose flop counts match those given above.

Algorithm 4.2 Polynomial Split Schur Algorithm

Input: $[m_0, m_1, \ldots, m_n]^T$ = first column of M_{n+1}
Initialize: $h_{-1}^{(n+1)}(z) = (1-z)/2$; $h_0^{(n+1)}(z) = m_0/2 + \overline{m}_1 z + \overline{m}_2 z^2 + \cdots + \overline{m}_n z^n$
$\zeta_0 = 1/m_0$; $\lambda_0 = \infty$
for $k = 0$ to $n-1$ **do**
$\lambda_{k+1} = 2\,\mathrm{Re}(\zeta_k) - \frac{1}{\lambda_k}$
$h_{k+1}^{(n-k)}(z) = \zeta_k h_k^{(n-k)}(z) + \frac{1}{z}[\overline{\zeta}_k h_k^{(n-k+1)}(z) - h_{k-1}^{(n-k+1)}(z)]$
$\zeta_{k+1} = \overline{h_k(0)}/\overline{h_{k+1}(0)}$
$\gamma_{k+1} = \left(1 - \dfrac{1}{\lambda_{k+1}\overline{\zeta}_{k+1}}\right) \dfrac{h_k(0)}{\overline{h_k(0)}}$
end for
Output: $\gamma_1, \gamma_2, \ldots, \gamma_n$

4.4 Connections to Carathéodory Functions

The split Schur algorithm is initialized with the Carathédory functions h_{-1} and h_0, but the subsequent role of Carathéodory functions is not immediately clear. The connection to Carathéodory functions can be illuminated by rewriting the recurrence relation (4.4).

Consider the ratios of split Schur functions,

$$\chi_k(z) = \frac{h_k(z)}{h_{k-1}(z)}; \quad k = 0, 1, 2, \ldots.$$

Provided that all leading principal submatrices of M_∞ are positive definite, Proposition 4.1 guarantees that χ_k is defined and nonzero on the unit disk $\mathcal{D} = \{z : |z| < 1\}$. Notice that

$\chi_k(0) = 1/\overline{\zeta}_k$ and that $(1-z)\chi_0(z) = 2h_0(z)$ is a Carathéodory function. Upon dividing by $h_k(z)$, the split Schur recurrence relation (4.4) takes the form

$$\chi_{k+1}(z) = \zeta_k + \frac{1}{z}\left[\overline{\zeta}_k - \frac{1}{\chi_k(z)}\right]; \quad k = 0, 1, 2, \ldots. \tag{4.6}$$

Proposition 4.5. *Let $\{\chi_k = h_k/h_{k-1}\}_{k=0}^{\infty}$ be the ratios of the split Schur functions from Definition 4.1. Also let $\{\lambda_k\}_{k=1}^{\infty}$ be the sequence of Jacobi parameters associated with M_∞. If $\frac{m_0}{2} + \sum_{j=1}^{\infty} \overline{m}_j$ exists and is nonzero, then*

$$\lim_{z\to 1} \chi_k(z) = \lambda_k; \quad k = 1, 2, 3, \ldots$$

Proof. Recall that the Jacobi parameters are nonzero since the leading principal submatrices of M_∞ are positive definite. The proof is by induction.

For $k = 1$, $\lambda_1 = 2/m_0$ and $\zeta_0 = 1/m_0$. By using

$$h_1(z) = \frac{1}{m_0}h_0(z) + \frac{1}{z}\left[\frac{1}{m_0}h_0(z) - h_{-1}(z)\right],$$

one can verify that

$$\lim_{z\to 1} \chi_1(z) = \lim_{z\to 1} \frac{h_1(z)}{h_0(z)} = \frac{2}{m_0}\left[\frac{m_0 + 2\overline{m}_1 + 2\overline{m}_2 + \cdots 2\overline{m}_n + \cdots}{m_0 + 2\overline{m}_1 + 2\overline{m}_2 + \cdots 2\overline{m}_n + \cdots}\right] = \frac{2}{m_0} = \lambda_1.$$

Now assume the result is true for $k = j$. Then by recurrence relation (4.6),

$$\lim_{z\to 1} \chi_{j+1}(z) = \lim_{z\to 1} \left(\zeta_j + \frac{1}{z}\left[\overline{\zeta}_j - \frac{1}{\chi_j(z)}\right]\right) = \zeta_j + \overline{\zeta}_j - \frac{1}{\lim_{z\to 1}\chi_j(z)} = 2\,\mathrm{Re}(\zeta_j) - \frac{1}{\lambda_j}.$$

The right-hand side is equal to λ_{j+1} by its definition. □

It turns out that the recurrence relation (4.6) is almost a recursion on Carathéodory functions. To be exact, it is a recursion on a new class of functions called *quasi-Carathéodory functions*.

Definition 4.2. *An analytic function $f(z)$ defined on the open unit disk $\mathcal{D} = \{z : |z| < 1\}$ is a* quasi-Carathéodory function (or qC-function) *if $(1-z)f(z)$ is a Carathéodory function.*

To show that (4.6) is a recursion on qC-functions, the following lemma of Delsarte and Genin [26] is required.

Lemma 4.2. *Let f_k be an arbitrary Carathéodory function and define f_{k+1} by means of*

$$f_k(z) = \frac{f_k(0) + \overline{f_k(0)}z}{1-z} - \frac{z}{(1-z)^2 f_{k+1}(z)}.$$

Then f_{k+1} is a Carathéodory function satisfying

$$0 \le \lim_{z\to 1} \frac{1}{(1-z)f_{k+1}(z)} \le 2\,\mathrm{Re}(f_k(0)),$$

with the possibility that $f_{k+1}(z) = \infty$.

Proof. A proof is given in [26]. □

Proposition 4.6. *Suppose χ_k is a qC-function and $\zeta_k = 1/\overline{\chi_k(0)}$. Then the function χ_{k+1} generated by the modified split Schur recurrence relation (4.6),*

$$\chi_{k+1}(z) = \zeta_k + \frac{1}{z}\left[\overline{\zeta}_k - \frac{1}{\chi_k(z)}\right],$$

is a qC-function satisfying $0 \leq \lim_{z\to 1} \chi_{k+1}(z) \leq 2\,\mathrm{Re}(\zeta_k)$.

Proof. Suppose χ_k is a qC-function. Then $(1-z)\chi_k$ is a Carathéodory function, as is its reciprocal $[(1-z)\chi_k(z)]^{-1}$. Let

$$f(z) = \frac{1}{(1-z)\chi_k(z)}.$$

It follows that $f(0) = \overline{\zeta}_k$. By Lemma 4.2, the function g, defined by

$$f(z) = \frac{\overline{\zeta}_k + \zeta_k z}{1-z} - \frac{z}{(1-z)^2 g(z)},$$

is a Carathéodory function. Rearranging gives

$$\frac{1}{(1-z)g(z)} = \zeta_k + \frac{1}{z}\left[\overline{\zeta}_k - \frac{1}{\chi_k(z)}\right] = \chi_{k+1}(z).$$

Since g is a Carathéodory function, so is $1/g$. Thus $(1-z)\chi_{k+1}(z)$ is a Carathéodory function. Furthermore, it follows from Lemma 4.2 that

$$\lim_{z\to 1} \chi_{k+1}(z) = \lim_{z\to 1} \frac{1}{(1-z)g(z)}$$

exists and lies between 0 and $2\,\mathrm{Re}(\zeta_k)$. □

Proposition 4.7. *Let $\chi_0(z) = h_0(z)/h_{-1}(z)$ be the ratio of the initial split Schur functions associated with M_∞. Also let $\{\chi_k\}_{k=1}^{\infty}$ be the sequence of functions generated by the modified split Schur recurrence relation (4.6), where $\zeta_k = 1/\overline{\chi_k(0)}$. Then each χ_k is a qC-function.*

Proof. Since h_0 is a Carathéodory function and $(1-z)\chi_0(z) = 2h_0(z)$, χ_0 is a qC-function. It follows from Proposition 4.6 that the recurrence relation (4.6) generates qC-function from qC-functions. □

By Propositions 4.6 and 4.7, the functions $\{\chi_k\}_{k=0}^{\infty}$ generated by the split Schur algorithm are qC-functions, and, under the condition of Proposition 4.5, their limits at $z = 1$ give the Jacobi parameters. In addition, the ζ_k's generated by the algorithm, henceforth called the *qC-parameters*, parameterize the original qC-function χ_0 in the following way. Solving the recurrence relation (4.6) for χ_k yields

$$\chi_k(z) = \frac{1}{\overline{\zeta}_k + \zeta_k z - z\chi_{k+1}(z)}. \tag{4.7}$$

Repeated use of this formula gives the continued fraction

$$\chi_0(z) = \cfrac{1}{\overline{\zeta}_0 + \zeta_0 z - \cfrac{z}{\overline{\zeta}_1 + \zeta_1 z - \cfrac{z}{\overline{\zeta}_2 + \zeta_2 z - \dots}}}. \tag{4.8}$$

Thus the polynomial split Schur algorithm generates n steps of a continued fraction representation for χ_0.

CHAPTER 5

DIVIDE-AND-CONQUER SPLIT SCHUR ALGORITHM

The split Schur algorithm was derived in Chapter 4 by splitting the Schur functions that are generated by Schur's algorithm. The recurrence relations used in Schur's algorithm to generate the numerators and denominators of Schur functions gave rise to a three-term recurrence relation for the split Schur functions. This three-term recurrence relation provides an efficient means for computing the Schur parameters associated with an HPD Toeplitz matrix.

From a different point of view, the split Schur algorithm is a recursion on quasi-Carathéodory (qC-) functions. It generates a sequence of qC-functions, as well as a sequence of complex numbers parameterizing an initial qC-function. The qC-parameters give rise to a continued fraction representation for a qC-function (see equation (4.8)) much like Schur's algorithm yields a continued fraction for a Schur function.

In Section 2.2.5, a doubling generalization of Schur's algorithm was described. The generalized Schur algorithm of Ammar and Gragg [5, 6, 7, 8] is a doubling procedure applied to a continued fraction representation of a certain Schur function. It is natural to wonder whether a similar doubling procedure could be applied to the quasi-Carathéodory continued fraction generated by the split Schur algorithm. This is indeed the case.

In this chapter, a new algorithm for solving HPD Toeplitz systems will be presented. The new algorithm is based on a doubling procedure applied to the continued fraction (4.8). The algorithm computes the nth Szegő polynomial, the nth prediction error, and the Schur parameters associated with the $(n+1) \times (n+1)$ HPD matrix M. Interestingly, if given the same input as the split Schur or split Levinson algorithms, the output of the new algorithm will be associated with $\overline{M}$ rather than M.

It will be seen that the new algorithm processes two families of conjugate-symmetric polynomials and is rich in polynomial multiplication. If implemented using FFTs for the polynomial multiplication, the new algorithm requires $O(n(\log_2 n)^2)$ flops. Thus the algorithm presented here is a superfast split Schur algorithm.

5.1 Main Ideas

The main ideas of the new algorithm are put forth in this section. In particular, a divide-and-conquer strategy is established by which the $(m + k)$-step continued fraction representation for the qC-function χ_0 can be obtained from an m-step and a k-step continued fraction.

Let $\{\zeta_k\}_{k=0}^{\infty}$ and $\{\chi_k(z)\}_{k=0}^{\infty}$ be the sequences of qC-parameters and qC-functions associated with the Toeplitz matrix M_∞, whose leading principal submatrices are Hermitian

positive definite and whose first column is absolutely summable. Define τ_{k+1} to represent the $(k+1)$st step of the continued fraction (4.8):

$$\tau_{k+1}(s) = \frac{1}{\overline{\zeta}_k + \zeta_k z - zs}.$$

With this definition, it follows from (4.7) that

$$\chi_k(z) = \tau_{k+1}(\chi_{k+1}(z))$$

and

$$\chi_0 = \tau_1(\chi_1) = \tau_1(\tau_2(\chi_2)) = \cdots = \tau_1(\tau_2(\ldots(\tau_k(\chi_k)\ldots))).$$

Now let T_k represent the composition

$$T_k(s) = (\tau_1 \circ \tau_2 \circ \cdots \circ \tau_k)(s).$$

Notice that T_k encapsulates k steps of χ_0's continued fraction representation (4.8). The finite continued fraction $T_k(s)$ can be simplified and written as a quotient of two multi-variable polynomials in s and z. With this in mind, define polynomials $p_{k+1}(z)$, $q_{k+1}(z)$, $u_{k+1}(z)$, and $v_{k+1}(z)$ such that

$$T_{k+1}(s) = \frac{u_{k+1} + v_{k+1}s}{p_{k+1} + q_{k+1}s}.$$

Since $T_{k+1}(s) = T_k(\tau_{k+1}(s))$, it follows that

$$\begin{aligned}\frac{u_{k+1} + v_{k+1}s}{p_{k+1} + q_{k+1}s} &= \frac{u_k + v_k\left(\frac{1}{\overline{\zeta}_k + \zeta_k z - zs}\right)}{p_k + q_k\left(\frac{1}{\overline{\zeta}_k + \zeta_k z - zs}\right)} \\ &= \frac{[(\overline{\zeta}_k + \zeta_k z)u_k + v_k] - zu_k s}{[(\overline{\zeta}_k + \zeta_k z)p_k + q_k] - zp_k s}.\end{aligned}$$

After equating terms, this gives

$$\begin{aligned}u_{k+1}(z) &= (\overline{\zeta}_k + \zeta_k z)u_k(z) + v_k(z) \\ v_{k+1}(z) &= -zu_k(z) \\ p_{k+1}(z) &= (\overline{\zeta}_k + \zeta_k z)p_k(z) + q_k(z) \\ q_{k+1}(z) &= -zp_k(z).\end{aligned} \tag{5.1a}$$

These recurrence relations are initialized by recognizing that

$$T_1(s) = \tau_1(s) = \frac{1}{\overline{\zeta}_0 + \zeta_0 z - zs},$$

from which it follows that

$$u_1(z) = 1, \quad v_1(z) = 0, \quad p_1(z) = \overline{\zeta}_0 + \zeta_0 z, \quad q_1(z) = -z. \tag{5.1b}$$

Taking $u_0(z) = 0$ and $p_0(z) = 1$, equations (5.1a) reduce to the pair of recurrence relations

$$\begin{aligned} u_{k+1}(z) &= (\overline{\zeta}_k + \zeta_k z)u_k(z) - zu_{k-1}(z) \\ p_{k+1}(z) &= (\overline{\zeta}_k + \zeta_k z)p_k(z) - zp_{k-1}(z). \end{aligned} \tag{5.2}$$

for $k = 1, 2, 3, \ldots$.

The recurrence relations above have the form of the modified split Levinson recurrence relation (3.12). Notice that the relation for the p_k's is initialized with $p_0 = 1$ and $p_1 = \overline{\zeta}_0 + \zeta_0 z$, which are precisely the conjugates of the split Levinson polynomials w_0 and w_1 (discussed in Chapter 3). Thus the p_k's generated in this way are the coefficient-conjugated split Levinson polynomials associated with the leading principal submatrices of M_∞, that is,

$$p_k(z) = \overline{w}_k(z); \quad k = 0, 1, 2, \ldots. \tag{5.3}$$

These polynomials have all the appropriately-modified properties that were described in Chapter 3. In particular,

$$\deg(p_k) = k, \quad p_k(0) = \overline{\mu}_k, \quad p_k(z) = \hat{p}_k(z); \quad k = 0, 1, 2, \ldots. \tag{5.4}$$

While the u_k's satisfy the same recurrence relation as the p_k's, they are initialized with $u_0(z) = 0$, $u_1(z) = 1$, and $u_2(z) = \overline{\zeta}_1 + \zeta_1 z$. The different initial conditions on the u_k's

and p_k's give rise to two distinct families of polynomials. Each u_k is a type of down-shifted, coefficient-conjugated split Levinson polynomial. It is easy to verify that

$$\deg(u_k) = k-1, \quad u_k(0) = \overline{\mu}_k/\overline{\mu}_1, \quad u_k(z) = \hat{u}_k(z); \quad k = 1, 2, 3, \ldots. \tag{5.5}$$

Viewed in light of the composition $T_k(s)$ and the recurrence relations (5.2), the split Schur algorithm processes two distinct families of conjugate-symmetric polynomials.

Notice that the recurrence relations (5.1) can be written in the matrix form

$$\begin{bmatrix} u_{k+1} & v_{k+1} \\ p_{k+1} & q_{k+1} \end{bmatrix} = \begin{bmatrix} u_k & v_k \\ p_k & q_k \end{bmatrix} \begin{bmatrix} (\overline{\zeta}_k + \zeta_k z) & -z \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} u_1 & v_1 \\ p_1 & q_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ (\overline{\zeta}_0 + \zeta_0 z) & -z \end{bmatrix}.$$

Computing determinants inductively leads to the following formula, which holds for $k = 0, 1, 2, \ldots$:

$$\det\left(\begin{bmatrix} p_{k+1} & q_{k+1} \\ u_{k+1} & v_{k+1} \end{bmatrix}\right) = v_{k+1}(z)p_{k+1}(z) - u_{k+1}(z)q_{k+1}(z) = z^{k+1}. \tag{5.6}$$

This determinant formula exposes a redundancy in the split Levinson polynomials—any one polynomial is completely determined by the other three.

Two important formulas, which have the semblance of the formula (5.6), can be deduced from the fact that $\chi_0(z) = T_{k+1}(\chi_{k+1}(z))$.

Proposition 5.1. *Given the split Schur functions associated with M_∞ and the polynomials described by the recurrence relations (5.1) (along with $u_0(z) = 0$), the following formulas are valid:*

$$\begin{aligned} p_{k+1}(z)h_0(z) - u_{k+1}(z)h_{-1}(z) &= z^{k+1}h_{k+1}(z); \quad k = -1, 0, 1, \ldots \\ v_{k+1}(z)h_{-1}(z) - q_{k+1}(z)h_0(z) &= z^{k+1}h_k(z); \qquad k = 0, 1, 2, \ldots . \end{aligned}$$

Proof. The first formula is obviously true for $k = -1$, and its truth for $k = 0$ follows easily from Proposition 4.3.

Now assume the first formula is true for $j \leq k$. By the recurrence relations (5.2),

$$p_{k+1}(z)h_0(z) - u_{k+1}(z)h_{-1}(z) = [(\overline{\zeta}_k + \zeta_k z)p_k(z) - zp_{k-1}(z)]h_0(z) - [(\overline{\zeta}_k + \zeta_k z)u_k(z) - zu_{k-1}(z)]h_{-1}(z).$$

After rearranging terms, the right-hand side becomes

$$(\overline{\zeta}_k + \zeta_k z)[p_k(z)h_0(z) - u_k(z)h_{-1}(z)] - z[p_{k-1}(z)h_0(z) - u_{k-1}(z)h_{-1}(z)].$$

With the induction hypothesis, this expression reduces to

$$(\overline{\zeta}_k + \zeta_k z)[z^k h_k(z)] - z[z^{k-1}h_{k-1}(z)].$$

This can be written

$$z^{k+1}\left[\frac{\overline{\zeta}_k}{z}h_k(z) + \zeta_k h_k(z) - \frac{1}{z}h_{k-1}(z)\right],$$

which is precisely $z^{k+1}h_{k+1}(z)$ by Proposition 4.3. Thus the first formula is proved.

The second formula follows immediately from the first by using $v_{k+1}(z) = -zu_k(z)$ and $q_{k+1}(z) = -zp_k(z)$. □

As is clear from the proof, the validity of Proposition 5.1 does not depend on how h_{-1} and h_0 were originally defined. The proposition remains valid for any initial split Schur functions provided that subsequent split Schur functions satisfy (4.4) and the polynomials p, q, u, v satisfy (5.1a) with the ζ's corresponding to the split Schur functions. In fact, Proposition 5.1 is simply a restatement of the identity

$$\chi_{k+1}(z) = T_{k+1}^{-1}(\chi_0(z)),$$

and the comments above simply describe the idea that

$$\chi_{k+1}(z) = (T_{k+1}^{-1} \circ T_j)(\chi_j(z)).$$

At this point, the task is to determine how the composition of ℓ steps, $T_\ell(s)$, can be broken into a composition of two "smaller" functions. This will determine how a doubling strategy can be implemented.

Let $\chi_0(z) = h_0(z)/h_{-1}(z)$ be the initial qC-function determined by the split Schur functions h_0 and h_{-1}. Beginning with this qC-function, m steps of the split Schur algorithm will yield the parameters $\zeta_0, \zeta_1, \ldots, \zeta_{m-1}$ and the corresponding polynomials p_m, q_m, u_m, and v_m. These give

$$\chi_0(z) = T_m(\chi_m(z)) = T_{0,m}(\chi_m(z)),$$

where

$$T_{0,m}(s) = \frac{u_{0,m} + v_{0,m}s}{p_{0,m} + q_{0,m}s}.$$

The additional subscript has been appended to indicate (and emphasize) that the process began with χ_0. It follows from Proposition 5.1 that $\chi_m(z) = h_m(z)/h_{m-1}(z)$, where

$$\begin{aligned} h_m(z) &= \frac{p_{0,m}(z)h_0(z) - u_{0,m}(z)h_{-1}(z)}{z^m} \\ h_{m-1}(z) &= \frac{v_{0,m}(z)h_{-1}(z) - q_{0,m}(z)h_0(z)}{z^m}. \end{aligned} \tag{5.7}$$

Now that χ_m has been computed, it can be considered an initial qC-function, and the split Schur algorithm can be applied to it. Taking k steps from χ_m, the split Schur algorithm will yield the parameters $\zeta_m, \zeta_{m+1}, \ldots, \zeta_{m+k-1}$ and the polynomials $p_{m,k}$, $q_{m,k}$, $u_{m,k}$, and $v_{m,k}$. These polynomials define the composition $T_{m,k}(s)$ so that

$$\chi_m(z) = T_{m,k}(\chi_{m+k}(z)),$$

where

$$T_{m,k}(s) = \frac{u_{m,k} + v_{m,k}s}{p_{m,k} + q_{m,k}s}.$$

Notice that the (m,k)-polynomials are associated with the Toeplitz matrix M_∞ through the intermediate qC-function χ_m. They are not explicitly associated with M_∞ in the same way that the $(0,m)$-polynomials are. Nonetheless, because they are generated by recurrence

relations of the form (5.1), the polynomials $p_{m,k}$ and $u_{m,k}$ have properties analogous to those of $p_{0,m}$ and $u_{0,m}$, including conjugate-symmetry. In particular,

$$\deg(p_{m,k}) = k, \qquad p_{m,k}(z) = \hat{p}_{m,k}(z), \qquad \deg(u_{m,k}) = k-1, \qquad u_{m,k}(z) = \hat{u}_{m,k}(z).$$

Once the composition $T_{m,k}$ has been obtained, it follows that $(T_{0,m} \circ T_{m,k})(s) = T_{0,m+k}(s)$. This gives

$$\frac{u_{0,m} + v_{0,m}\left(\dfrac{u_{m,k} + v_{m,k}s}{p_{m,k} + q_{m,k}s}\right)}{p_{0,m} + q_{0,m}\left(\dfrac{u_{m,k} + v_{m,k}s}{p_{m,k} + q_{m,k}s}\right)} = \frac{u_{0,m+k} + v_{0,m+k}s}{p_{0,m+k} + q_{0,m+k}s}.$$

After expanding and equating coefficients, formulas for the $(m+k)$-step polynomials are obtained:

$$\begin{aligned} u_{0,m+k}(z) &= u_{0,m}(z)p_{m,k}(z) + v_{0,m}(z)u_{m,k}(z) \\ v_{0,m+k}(z) &= u_{0,m}(z)q_{m,k}(z) + v_{0,m}(z)v_{m,k}(z) \\ p_{0,m+k}(z) &= p_{0,m}(z)p_{m,k}(z) + q_{0,m}(z)u_{m,k}(z) \\ q_{0,m+k}(z) &= p_{0,m}(z)q_{m,k}(z) + q_{0,m}(z)v_{m,k}(z). \end{aligned} \tag{5.8}$$

Recall that $q_{0,m+k}(z) = -zp_{0,m+k-1}(z)$ and that $p_{0,j}(z) = \overline{w}_j(z)$, where w_j is the jth split Levinson polynomial associated with M_∞. Thus $p_{0,m+k}$ and $q_{0,m+k}$ describe two consecutive coefficient-conjugated split Levinson polynomials associated with M_∞.

5.2 The Algorithm

In the last section, a divide-and-conquer approach to the split Schur algorithm was introduced. This approach naturally gives rise to the following doubling strategy:

A. Starting with the initial split Schur functions h_{-1} and h_0, take m steps of the split Schur algorithm to generate the qC-parameters $\zeta_0, \zeta_1, \ldots, \zeta_{m-1}$ and the corresponding polynomials $u_{0,m}$, $v_{0,m}$, $p_{0,m}$ and $q_{0,m}$.

B. Use equations (5.7) to compute the split Schur functions h_{m-1} and h_m.

C. With h_{m-1} and h_m as a new set of initial split Schur functions, take m steps of the split Schur algorithm to generate the qC-parameters $\zeta_m, \zeta_{m+1}, \ldots, \zeta_{2m-1}$ and the corresponding polynomials $u_{m,m}$, $v_{m,m}$, $p_{m,m}$ and $q_{m,m}$.

D. Use equations (5.8) to compose the $(0, m)$-polynomials and the (m, m)-polynomials to obtain the $(0, 2m)$-polynomials $u_{0,2m}$, $v_{0,2m}$, $p_{0,2m}$ and $q_{0,2m}$.

In theory, the operations involving the split Schur functions are operations on infinite power series. As in Chapter 4, when working with the finite HPD Toeplitz matrix $M = M_{n+1}$, the power series can be truncated to polynomials.

In the procedure described above, the split Schur algorithm processes the split Schur functions solely for the computation of the qC-parameters. As discussed at the end of Section 4.3, once a qC-parameter is computed, the computation of the next qC-parameter requires one fewer term of the corresponding split Schur functions. In the doubling strategy,

this amounts to the fact that h_{m-1} and h_m need only half the number of terms as h_{-1} and h_0 in order to compute the (m, m)-polynomials.

Let $h_k^{(p)}$ denote the polynomial obtained from h_k by removing all terms of degree p or greater. The split Schur functions in the doubling strategy can then be replaced by $h_{-1}^{(2m)}$, $h_0^{(2m)}$, $h_m^{(m)}$, and $h_{m-1}^{(m)}$. From equations (5.7), it follows that

$$h_m^{(m)}(z) = \text{first } m \text{ terms of } \frac{p_{0,m}(z)h_0^{(2m)}(z) - u_{0,m}(z)h_{-1}^{(2m)}(z)}{z^m} \tag{5.9a}$$

and

$$h_{m-1}^{(m)}(z) = \text{first } m \text{ terms of } \frac{v_{0,m}(z)h_{-1}^{(2m)}(z) - q_{0,m}(z)h_0^{(2m)}(z)}{z^m}. \tag{5.9b}$$

It is worthwhile to observe that these formulas imply that the polynomials $h_m^{(m)}$ and $h_{m-1}^{(m)}$ are determined by the middle one-third of the coefficients (i.e., coefficients of z^m through z^{2m-1}) of the expressions

$$p_{0,m}(z)h_0^{(2m)}(z) - u_{0,m}(z)h_{-1}^{(2m)}(z) \qquad \text{and} \qquad v_{0,m}(z)h_{-1}^{(2m)}(z) - q_{0,m}(z)h_0^{(2m)}(z),$$

respectively.

The doubling strategy described above can now be applied recursively. Suppose the HPD Toeplitz matrix M has the dimensions $(n+1) \times (n+1)$, where $n + 1 = 2^N$. The divide-and-conquer split Schur algorithm (DCSSA) is initialized with $p_{0,1}$, $q_{0,1}$, $u_{0,1}$, and $v_{0,1}$ as given in equation (5.1b) and $\zeta_0 = \overline{h_{-1}^{(1)}}/\overline{h_0^{(1)}}$. It then repeatedly applies itself, doubling the

Algorithm 5.1 Divide-and-Conquer Split Schur Algorithm (Version 1)

Input: $h_{-1}(z)$ and $h_0(z)$ associated with M_{n+1}, where $n+1 = 2^N$
Initialize: $\zeta_0 = \overline{h_{-1}^{(1)}/h_0^{(1)}}$; $p_{0,1}(z) = \overline{\zeta_0} + \zeta_0 z$; $q_{0,1}(z) = -z$; $u_{0,1}(z) = 1$; $v_{0,1}(z) = 0$
for $m = 1, 2, 4, \ldots, 2^{N-1}$ **do**

1. Use equations (5.9) to compute $h_{m-1}^{(m)}$ and $h_m^{(m)}$ from $h_{-1}^{(2m)}$ and $h_0^{(2m)}$.
2. Compute $p_{m,m}$, $q_{m,m}$, $u_{m,m}$, and $v_{m,m}$ from $h_{m-1}^{(m)}$ and $h_m^{(m)}$ just as $p_{0,m}$, $q_{0,m}$, $u_{0,m}$, and $v_{0,m}$ were computed from $h_{-1}^{(m)}$ and $h_0^{(m)}$.
3. Use equations (5.8) to compute $p_{0,2m}$, $q_{0,2m}$, $u_{0,2m}$, and $v_{0,2m}$.

end for
Output: $p_{0,n+1}$; $q_{0,n+1}$; $u_{0,n+1}$; $v_{0,n+1}$; $\zeta_0, \zeta_1, \ldots, \zeta_n$

polynomials until $p_{0,n+1}$, $q_{0,n+1}$, $u_{0,n+1}$, and $v_{0,n+1}$ are obtained. A preliminary version of the algorithm is described in Algorithm 5.1.

At its lowest level of recursion, the algorithm computes the $(0,1)$-polynomials, followed by the $(0,2)$-polynomials, followed by the $(0,4)$-polynomials, etc., until the final $(0,n+1)$-polynomials are computed. Of course, all along the algorithm is being used recursively to compute (and double) the intermediate polynomials whose compositions eventually give the $(0,2^k)$-polynomials.

The following list shows the order in which the polynomials are computed and composed in an 8×8 example:

$$\underbrace{(0,1),(1,1)}_{(0,2)},\overbrace{(0,2),\underbrace{(2,1),(3,1)}_{(2,2)},(2,2)}^{(0,4)},\overbrace{(0,4),\underbrace{(4,1),(5,1)}_{(4,2)},\overbrace{(4,2),\underbrace{(6,1),(7,1)}_{(6,2)},(6,2)}^{(4,4)},(4,4)}^{(0,8)},(0,8).$$

A one-step polynomial is computed each time the algorithm is recursed. At each of these steps, a qC-parameter is computed from a ratio of truncated split Schur functions. Thus

the entire set of qC-parameters is computed. For the 8×8 example, the pairs of split Schur functions are computed in the following order

$$(h_0^{(1)}, h_1^{(1)}),\ (h_1^{(2)}, h_2^{(2)}),\ (h_2^{(1)}, h_3^{(1)}),\ (h_3^{(4)}, h_4^{(4)}),\ (h_4^{(1)}, h_5^{(1)}),\ (h_5^{(2)}, h_6^{(2)}),\ (h_6^{(1)}, h_7^{(1)}),$$

and they are obtained via (5.9) from

$$(h_{-1}^{(2)}, h_0^{(2)}),\ (h_{-1}^{(4)}, h_0^{(4)}),\ (h_1^{(2)}, h_2^{(2)}),\ (h_{-1}^{(8)}, h_0^{(8)}),\ (h_3^{(2)}, h_4^{(2)}),\ (h_3^{(4)}, h_4^{(4)}),\ (h_5^{(2)}, h_6^{(2)}),$$

respectively.

Algorithm 5.2 gives a detailed function description of the DCSSA. Given the HPD Toeplitz matrix M_{n+1}, where $n+1 = 2^N$, the function is invoked with the input h_{-1}, h_0, and N. Its output is the set of polynomials $p_{0,n+1}$, $q_{0,n+1}$, $u_{0,n+1}$, and $v_{0,n+1}$. During the function execution, the step count is kept by the global variable k, which must be initialized to $k = 0$ before the function is initially invoked. The variable k indexes the sequences $\{\zeta_k\}$ and $\{\theta_k\}$, whose scopes must also be global. The ζ_k's are, of course, the qC-parameters. The numbers $\theta_k = h_k^{(1)}/\overline{h_k^{(1)}}$ are required for the computation of the Schur parameters (see equations (4.5)), but if preferred, they could be computed later from the qC-parameters. These sequences are not technically function output, but as globally defined sequences, their values are computed and stored throughout function execution.

After the DCSSA function has been applied, the output polynomials $p_{0,n+1}(z)$ and $q_{0,n+1}(z) = -zp_{0,n}(z)$ describe the coefficient-conjugated split Levinson polynomials asso-

Algorithm 5.2 Divide-and-Conquer Split Schur Algorithm (Version 2)

Remarks: *Given M_{n+1} with $n+1 = 2^N$, this function applies the divide-and-conquer split Schur algorithm. The variables k, ζ_k, and θ_k must be defined outside the scope of the function. $k = 0$ must be initialized before the function is applied.*

Begin Function

function $[P, Q, U, V] = \text{dcssa}(H_1, H_2, N)$

$\zeta_k = \overline{H_1^{(1)}}/\overline{H_2^{(1)}}$; $\theta_k = H_2^{(1)}/\overline{H_2^{(1)}}$

$P(z) = \overline{\zeta}_k + \zeta_k z$; $Q(z) = -z$; $U(z) = 1$; $V(z) = 0$

$k = k + 1$

for $j = 0$ to $N - 1$ **do**

$m = 2^j$

$\tilde{H}_1^{(m)}(z) = \text{first } m \text{ terms of } \dfrac{V(z)H_1^{(2m)}(z) - Q(z)H_2^{(2m)}(z)}{z^m}$

$\tilde{H}_2^{(m)}(z) = \text{first } m \text{ terms of } \dfrac{P(z)H_2^{(2m)}(z) - U(z)H_1^{(2m)}(z)}{z^m}$

$[P_1, Q_1, U_1, V_1] = \text{dcssa}(\tilde{H}_1^{(m)}, \tilde{H}_2^{(m)}, j)$

$P_2(z) = P(z)P_1(z) + Q(z)U_1(z); \quad Q_2(z) = P(z)Q_1(z) + Q(z)V_1(z)$

$U_2(z) = U(z)P_1(z) + V(z)U_1(z); \quad V_2(z) = U(z)Q_1(z) + V(z)V_1(z)$

$P(z) = P_2(z)$; $Q(z) = Q_2(z)$; $U(z) = U_2(z)$; $V(z) = V_2(z)$

end for

Output: $P(z), Q(z), U(z), V(z)$

End Function

ciated with M (which are the split Levinson polynomials associated with $\overline{M}$). Thus, by formula (3.13), the Szegő polynomial ρ_n is given by

$$\rho_n(z) = \frac{1}{p_{0,n+1}(0)(z-1)} \left[\overline{p}_{0,n+1}(z) + \frac{\lambda_{n+1}}{z} \overline{q}_{0,n+1}(z) \right].$$

This formula requires the value λ_{n+1} which can be obtained, along with the Schur parameters, from equations (4.5). The nth prediction error associated with M can be obtained from (3.9):

$$\delta_n = \frac{\lambda_{n+1}}{2|p_{0,n+1}(0)|^2}.$$

An entire procedure for using the DCSSA function to compute the nth Szegő polynomial, the nth prediction error, and the Schur parameters is described in Algorithm 5.3.

5.3 Example

Returning to the example at the end of Chapter 3, in which

$$M = \begin{bmatrix} 8 & 4+i & 2 & 1-i \\ 4-i & 8 & 4+i & 2 \\ 2 & 4-i & 8 & 4+i \\ 1+i & 2 & 4-i & 8 \end{bmatrix},$$

Algorithm 5.3 Divide-and-Conquer Split Schur Algorithm (Version 3)

Input: $[m_0, m_1, \ldots, m_n]^T$ = first column of M_{n+1}, where $n+1 = 2^N$
Declare: Global variables k, $\{\zeta_k\}$, and $\{\theta_k\}$
Initialize: $h_{-1}(z) = (1-z)/2$; $h_0(z) = m_0/2 + \overline{m}_1 z + \overline{m}_2 z^2 + \cdots + \overline{m}_n z^n$
$k = 0$ *(This k keeps the count in the DCSSA function.)*
$[p_{n+1}, q_{n+1}, u_{n+1}, v_{n+1}] = \mathrm{dcssa}(h_{-1}, h_0, N)$
$\lambda_1 = 2/m_0$
for $k = 1$ to n **do**
$\gamma_k = \left(1 - \dfrac{1}{\lambda_k \overline{\zeta}_k}\right) \theta_{k-1}$
$\lambda_{k+1} = 2\,\mathrm{Re}(\zeta_k) - \frac{1}{\lambda_k}$
end for
$\delta_n = \lambda_{n+1}/(2|p_{n+1}(0)|^2)$
$\rho_n(z) = \dfrac{1}{p_{n+1}(0)(z-1)} \left[\overline{p}_{n+1}(z) + \dfrac{\lambda_{n+1}}{z}\overline{q}_{n+1}(z)\right].$
Output: ρ_n; δ_n; $\gamma_1, \gamma_2, \ldots, \gamma_n$

Algorithm 5.3 produces identical output. However, as described above, the split-Levinson polynomials are coefficient-conjugated:

$$p_4(z) \approx (0.34 + 0.08i) - (0.14 + 0.17i)z - 0.07z^2 - (0.14 - 0.17i)z^3 + (0.34 - 0.08i)z^4$$

$$q_4(z) \approx -(0.25 + 0.02i)z + (0.13 + 0.11i)z^2 + (0.13 - 0.11i)z^3 - (0.25 - 0.02i)z^4.$$

Notice that $p_4(z) = \overline{w}_4(z)$ and $q_4(z) = -z\,\overline{w}_3(z)$, as expected.

CHAPTER 6

SUPERFAST IMPLEMENTATION OF THE DIVIDE-AND-CONQUER SPLIT SCHUR ALGORITHM

A divide-and-conquer approach to the split Schur algorithm was described in Chapter 5. The resulting new algorithm, abbreviated DCSSA, is rich in polynomial multiplication. If these multiplications are performed directly, the algorithm requires at least $O(n^2)$ flops when applied to a real or complex $(n+1) \times (n+1)$ Toeplitz matrix. In this chapter, a superfast, $O(n(\log_2 n)^2)$, implementation of the DCSSA is described for real symmetric positive definite Toeplitz matrices of size $n+1 = 2^N$. For real Toeplitz matrices, the split Levinson symmetric polynomials, the split Schur functions, and the qC-parameters are all real, as are the Szegő polynomials and Schur parameters.

The real-case implementation of the DCSSA described below follows Algorithm 5.1. In order to derive a superfast implementation, the polynomial multiplications in steps 1 and 3 must be written as convolutions and FFT techniques must be applied. It will be shown that the nth Szegő polynomial associated with $M = M_{n+1}$, the nth prediction error, and the Schur parameters can be obtained in $7n(\log_2 n)^2 + O(n \log_2 n)$ flops.

6.1 Symmetric Vectors

The DCSSA processes the polynomials u_m, v_m, p_m, and q_m described by the recurrence relations (5.1), as well as the truncated versions of the split Schur functions h_{m-1} and h_m described by equations (5.7). In Chapter 5, a second subscript was appended to the symbols u_m, v_m, p_m, and q_m to indicate (and emphasize) the qC-function from which the polynomials originated. For example, $u_{k,m}(z)$ is the polynomial obtained from m steps of the recurrence relation (5.1) by using $\chi_k(z) = h_k(z)/h_{k-1}(z)$ as the initial qC-function. Because the properties of the polynomials do not depend on their initial qC-functions, the first subscript will be omitted for the time being.

The first step in implementing a convolution-based, superfast version of Algorithm 5.1 is to associate the polynomials u_m, v_m, p_m, and q_m with corresponding vectors. There are several difficulties that arise if the polynomials are naively associated with their vectors of coefficients. In this section, the properties of the polynomials u_m, v_m, p_m, and q_m are reviewed, corresponding vectors are defined, and the properties of those vectors are described. Because Algorithm 5.1 is based on a doubling procedure, the subscript m will be restricted to a power of two.

As discussed in Chapter 5 (see properties (5.5) and (5.4)), the polynomials u_m and p_m have degrees $m-1$ and m, respectively. They are also conjugate symmetric in the sense that

$$u_m(z) = \hat{u}_m(z) = z^{m-1}\overline{u}_m(1/z) \quad \text{and} \quad p_m(z) = \hat{p}_m(z) = z^m\overline{p}_m(1/z).$$

By definition, the polynomials v_m and q_m are simply the shifted versions of $-u_{m-1}$ and $-p_{m-1}$:

$$v_m(z) = -zu_{m-1}(z), \qquad q_m(z) = -zp_{m-1}(z).$$

It follows that they have degrees $m-1$ and m, respectively. While not conjugate symmetric, they satisfy $v_m(0) = q_m(0) = 0$, and it is clear that $\frac{1}{z}v_m(z)$ and $\frac{1}{z}q_m(z)$ are conjugate symmetric.

In a real-valued implementation of the DCSSA, the conjugate-symmetric polynomials u_m, v_m/z, p_m, and q_m/z are, in fact, are real symmetric polynomials. As a consequence, their coefficient vectors read the same top-to-bottom as bottom-to-top. In Chapter 2, this type of vector symmetry was referred to as real quarter-even (RQE) symmetry. The following notation will be used from here on to associate the general polynomial f with its vector of coefficients $\mathfrak{f}$:

$$f(z) = \sum_{j=0}^{m} f_j z^j \quad \longleftrightarrow \quad \mathfrak{f} = [f_0, f_1, \ldots, f_m]^T = [f_j]_{j=0}^{m}.$$

All vectors associated with polynomials will be indexed from 0. Under this identification, the vectors $\mathfrak{u}_m$ and $\mathfrak{p}_m$ display RQE symmetry, while the vectors $\mathfrak{v}_m$ and $\mathfrak{q}_m$ do not. On the other hand, if $\mathfrak{v}_m$ and $\mathfrak{q}_m$ were upshifted, the new vectors would display RQE symmetry.

Notice that $\mathfrak{u}_m$ and $\mathfrak{v}_m$ are m-dimensional vectors, while $\mathfrak{p}_m$ and $\mathfrak{q}_m$ are $(m+1)$-dimensional vectors. Because m is a power of two in Algorithm 5.1, the vectors $\mathfrak{u}_m$ and $\mathfrak{v}_m$ are well suited for use with the efficient, split-radix, power-of-two FFTs described in Chapter 2. On the

other hand, $\mathfrak{p}_m$ and $\mathfrak{q}_m$ lack the same suitability. To accommodate this defect, define the modified polynomials $\dot{p}_m$ and $\dot{q}_m$, both of degree $m-1$, so that

$$p_m(z) = \dot{p}_m(z) + p_m(0)z^m$$

and

$$q_m(z) = z\dot{q}_m(z).$$

Since p_m is real symmetric, the modified polynomial $\dot{p}_m$ is obtained from p_m by simply removing its degree-m term. The modified polynomial $\dot{q}_m$ is simply the upshifted q_m, which is equal to $-p_{m-1}$. In order to keep notation consistent, the "modified" polynomials $\dot{u}_m$ and $\dot{v}_m$ are defined by

$$\dot{u}_m(z) = u_m(z) \quad \text{and} \quad \dot{v}_m(z) = v_m(z).$$

Each modified polynomial is associated with its corresponding m-dimensional vector of coefficients:

$$\dot{u}_m(z) \longleftrightarrow \dot{\mathfrak{u}}_m, \quad \dot{v}_m(z) \longleftrightarrow \dot{\mathfrak{v}}_m, \quad \dot{p}_m(z) \longleftrightarrow \dot{\mathfrak{p}}_m, \quad \dot{q}_m(z) \longleftrightarrow \dot{\mathfrak{q}}_m.$$

It is the coefficient vectors of the modified polynomials that will be processed in the superfast implementation of the DCSSA. These vectors inherit symmetry from their corresponding

polynomials. Specifically, the vectors $\dot{\mathfrak{u}}_m$ and $\dot{\mathfrak{q}}_m$ display RQE symmetry. The vector $\dot{\mathfrak{p}}_m$ has the form

$$[t_0, t_1, \ldots, t_{m/2-1}, t_{m/2}, t_{m/2-1}, \ldots, t_1]^T,$$

and therefore displays RE symmetry. Similarly, the vector $\dot{\mathfrak{v}}_m$ has the form

$$[0, s_1, \ldots, s_{m/2-1}, s_{m/2}, s_{m/2-1}, \ldots, s_1]^T,$$

and also displays RE symmetry.

The superfast implementation of the DCSSA will use FFT techniques to compute the polynomial products encountered in steps 1 and 3 of Algorithm 5.1. These products involve convolutions of the symmetric vectors defined above. The symmetry in these vectors will be exploited both in the computations of their DFTs and the computations of the convolutions. The following proposition highlights some of the important properties of the DFTs of the symmetric vectors.

Proposition 6.1. *Suppose that $x = [x_j]_{j=0}^{n-1}$ is an n-dimensional vector, where n is even, and $y = \Omega_n x$ is the DFT of x.*

i.) If x is a real vector, then y is complex, conjugate-symmetric, i.e., $y_j = \overline{y}_{n-j}$. Furthermore, y_0 and $y_{n/2}$ are real numbers.

ii.) If x is RE symmetric, then y is RE symmetric.

iii.) If x is RO symmetric, then y is purely imaginary and odd symmetric.

iv.) If x is RQE symmetric, then $[e^{-\pi ji/n}y_k]_{j=0}^{n-1}$ is RO symmetric with the exception that y_0 may be nonzero. Consequently, $y_{n/2} = 0$.

Proof. The first three results are well known. Discussions and proofs can be found in [14] or [55]. In part (iv), the fact that $[e^{-\pi ji/n}y_j]_{j=0}^{n-1}$ is a real vector is discussed in [55]. To establish its nearly odd symmetry, notice that

$$e^{-(n-j)\pi i/n}y_{n-j} = e^{-\pi i}e^{j\pi i/n}y_{n-j} = -e^{j\pi i/n}y_{n-j}.$$

Now since y is the DFT of a real vector, it is conjugate symmetric and thus, $y_{n-j} = \overline{y}_j$. It follows that

$$-e^{j\pi i/n}y_{n-j} = -e^{j\pi i/n}\overline{y}_j.$$

Finally, since these expressions describe real numbers, they are all equal to their conjugates. Therefore

$$e^{-(n-j)\pi i/n}y_{n-j} = -e^{-j\pi i/n}y_j.$$

This is the desired form of the vector. To establish that $y_{n/2} = 0$, simply substitute $j = n/2$. □

6.2 Zero-Padded Vectors

The symmetric vectors that will be processed in the superfast DCSSA were described above. However, before the algorithm can be implemented, another important detail must

be cleared up. The product of two polynomials of degree $m-1$ is a polynomial of degree $2m-2$, which has at most $2m-1$ nonzero coefficients. In order to compute such a product using power-of-two FFTs (see Section 2.1.2), the original coefficient vectors must be padded with zeros to length $2m$. The product of two $(m-1)$-degree polynomials is thus obtained as follows:

1. Pad each m-dimensional coefficient vector with m zeros to length $2m$.
2. Compute the DFT of each zero-padded vector.
3. Compute the Schur (element-by-element) product of the DFTs.
4. Compute the inverse DFT of the resulting vector to obtain the coefficient vector of the product polynomial.

In symbols, the coefficient vector of the product of the $(m-1)$-degree polynomials f and g is given by the vector convolution

$$\begin{bmatrix} \mathfrak{f} \\ 0_m \end{bmatrix} * \begin{bmatrix} \mathfrak{g} \\ 0_m \end{bmatrix} = \frac{1}{2m}\,\Omega_{2m}^{*}\left(\Omega_{2m}\begin{bmatrix} \mathfrak{f} \\ 0_m \end{bmatrix} \star\, \Omega_{2m}\begin{bmatrix} \mathfrak{g} \\ 0_m \end{bmatrix}\right),$$

where 0_m denotes the m-dimensional zero vector.

When computed with DFTs, the convolution requires the transforms of the zero-padded vectors. Any symmetry in the original vectors (other than real symmetry) is likely destroyed when the vectors are padded with zeros. Therefore, symmetric FFTs cannot be used directly to compute the DFTs once the vectors have been padded with zeros. The rest of this section

is devoted to developing and implementing a general method for computing the DFT of a zero-padded vector from the DFT of the original vector.

6.2.1 DFTs of Zero-Padded Vectors

Suppose n is a fixed even integer. Let $\omega = e^{-2\pi i/n}$ and, for any positive integer k, $W_k = \mathrm{diag}([\omega^j]_{j=0}^{k-1})$. As can be seen from equation (2.4), the Fourier matrix Ω_n satisfies

$$\Omega_n P_n = \begin{bmatrix} \Omega_m & W_m\Omega_m \\ \Omega_m & -W_m\Omega_m \end{bmatrix},$$

where P_n is the even/odd permutation matrix and $n = 2m$. Because the Fourier matrix (of any order) is symmetric, it follows that

$$P_n^T\Omega_n = \begin{bmatrix} \Omega_m & \Omega_m \\ \Omega_m W_m & -\Omega_m W_m \end{bmatrix}. \tag{6.1}$$

Therefore, the even/odd permutation of the DFT of the zero-padded m-dimensional vector x is given by

$$P_n^T\Omega_n \begin{bmatrix} x \\ 0_m \end{bmatrix} = \begin{bmatrix} \Omega_m x \\ \Omega_m W_m x \end{bmatrix}. \tag{6.2}$$

According to equation (6.2), the even-indexed part of the DFT of a zero-padded vector is the DFT of the original vector. The odd-indexed part of the DFT can be obtained from an

additional m-dimensional DFT. Equation (6.2) does not necessarily offer an improvement over simply computing the n-dimensional DFT of the zero-padded vector. On the other hand, if m also happens to be an even number, the matrix splitting can be repeated.

The root of unity associated with the Fourier matrix Ω_m is $e^{-2\pi i/m} = \omega^2$. Therefore, by using equation (6.1) to split Ω_m, one finds that

$$P_m^T \Omega_m W_m x = \begin{bmatrix} \Omega_{m/2} & \Omega_{m/2} \\ \Omega_{m/2} W_{m/2}^2 & -\Omega_{m/2} W_{m/2}^2 \end{bmatrix} \cdot W_m x. \tag{6.3}$$

Now partition x so that $x = \begin{bmatrix} a \\ b \end{bmatrix}$, where a and b are $m/2$-dimensional vectors. Since $\omega^{m/2} = -i$, it follows that

$$W_m x = \begin{bmatrix} W_{m/2} & 0 \\ 0 & -i\, W_{m/2} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} W_{m/2} a \\ -i\, W_{m/2} b \end{bmatrix},$$

where 0 refers to the zero matrix of order $m/2$. Substituting this into equation (6.3) and multiplying gives

$$P_m^T \Omega_m W_m x = \begin{bmatrix} \Omega_{m/2} W_{m/2} (a - bi) \\ \Omega_{m/2} W_{m/2}^3 (a + bi) \end{bmatrix}. \tag{6.4}$$

It turns out that the vector described by equation (6.4) is conjugate symmetric, as established by the following proposition.

Proposition 6.2. *Suppose m is divisible by two. Let $n = 2m$, $\omega = e^{-2\pi i/n}$, and $W_{m/2} = \mathrm{diag}([\omega^j]_{j=0}^{m/2-1})$. Then*

$$\Omega_{m/2} W_{m/2}^3 = J\overline{\Omega}_{m/2}\overline{W}_{m/2},$$

where J is the reversal matrix of order $m/2$.

Proof. Let $A = \Omega_{m/2} W_{m/2}^3$ and $B = \overline{\Omega}_{m/2}\overline{W}_{m/2}$. With rows and columns indexed from 0 to $m/2-1$, the (k,ℓ)-element of A is $\omega^{4k\ell}\omega^{3\ell} = \omega^{(4k+3)\ell}$. On the other hand, the (k,ℓ)-element of B is $\overline{\omega}^{4k\ell}\overline{w}^{\ell}$. Upon reversing the rows of B and using $\omega^{-2m} = 1$, the (k,ℓ)-element of JB is

$$\overline{\omega}^{4(m/2-1-k)\ell}\overline{\omega}^{\ell} = \left(\omega^{-2m}\right)^{\ell}\omega^{(4k+4)\ell}\omega^{-\ell} = \omega^{(4k+3)\ell}.$$

It follows that $A = JB$. □

By Proposition 6.2, the bottom half of the vector described in equation (6.4) is the reverse conjugate of the top half. Thus the DFT of the m-dimensional vector $W_m x$ can be computed from a related $m/2$-dimensional DFT. If equations (6.2) and (6.4) are combined, the $2m$-dimensional DFT of a zero-padded vector can be obtained from an m-dimensional DFT and an $m/2$-dimensional DFT. Depending on the nature of the vector x, this has the potential to significantly reduce the arithmetic complexity of its DFT. The cases in which x is RE symmetric or RQE symmetric are discussed next.

Proposition 6.3. *Let m be a multiple of four, $n = 2m$, $\omega = e^{-2\pi i/n}$, and $W_{m/2} = \mathrm{diag}([\omega^j]_{j=0}^{m/2-1})$. Suppose $x \in \mathbb{R}^m$ is partitioned into two vectors of length $m/2$:*

$$x = \begin{bmatrix} a \\ \\ b \end{bmatrix}.$$

Let y be the complex, $m/2$-dimensional vector $y = W_{m/2}(a - bi)$.

i.) If x is RE symmetric, then y has the form

$$[y_0, y_1, \ldots, y_{m/4-1}, y_{m/4}, -\overline{y}_{m/4-1}, \cdots - \overline{y}_1]^T.$$

Furthermore, $\mathrm{Re}(y_{m/4}) = 0$.

ii.) If x is RQE symmetric, then y has the form

$$[y_0, \ldots, y_{m/4-1}, -\overline{\omega y}_{m/4-1}, \ldots, -\overline{\omega y}_0]^T.$$

Proof. Assume that the vectors are indexed from 0. To prove the first part, suppose that x is RE symmetric, so that $b_{m/2-j} = a_j$ for $j = 1, 2, \ldots, m/2 - 1$. Now for $k = 1, 2, \ldots, m/2 - 1$, $y_k = \omega^k(a_k - ib_k) = \omega^k(a_k - ia_{m/2-k})$. On the other hand, since $\omega^{m/2} = -i$,

$$y_{m/2-k} = \omega^{m/2-k}(a_{m/2-k} - ia_k) = -i\overline{\omega}^k(-i)(a_k + ia_{m/2-k}) = -\overline{\omega}^k(a_k + ia_{m/2-k}).$$

It follows that $y_{m/2-k} = -\overline{y}_k$; $k = 1, 2, \ldots, m/2-1$. This gives the desired form. Furthermore, since $a_{m/4} = b_{m/4}$ and $\omega^{m/4} = \frac{\sqrt{2}}{2} - i\frac{\sqrt{2}}{2}$,

$$\mathrm{Re}(y_{m/4}) = \frac{\sqrt{2}}{2}a_{m/4} - \frac{\sqrt{2}}{2}a_{m/4} = 0.$$

To prove the second part, suppose that x is RQE symmetric, so that $b_{m/2-1-j} = a_j$ for $j = 0, 1, \ldots, m/2-1$. Then for $k = 0, 1, \ldots, m/2-1$, $y_k = \omega_k(a_k - ib_k) = \omega^k(a_k - ia_{m/2-1-k})$. On the other hand, since $\omega^{m/2} = -i$,

$$y_{m/2-1-k} = \omega^{m/2-1-k}(a_{m/2-1-k} - ia_k) = -i\overline{\omega\omega}^k(-i)(a_k + ia_{m/2-1-k}) = -\overline{\omega\omega}^k(a_k + ia_{m/2-1-k}).$$

It follows that $y_{m/2-1-k} = -\overline{\omega y}_k$; $k = 0, 1, \ldots, m/2-1$. This gives the desired form. □

The next two propositions describe how the results of Proposition 6.3 will be used in computing DFTs.

Proposition 6.4. *Referring to Proposition 6.3, suppose x is RE symmetric. Then*

$$y = t_0\, e_0 + O + Ei,$$

where $t_0 = \mathrm{Re}(y_0)$, $O \in \mathbb{R}^{m/2}$ is an RO-symmetric vector and $E \in \mathbb{R}^{m/2}$ is an RE-symmetric vector. Consequently, $Y = \Omega_{m/2}y$ can be computed from two symmetric $m/2$-dimensional FFTs. Furthermore, $\mathrm{Re}(Y_k) = t_0$ for all k.

Proof. The form of y follows directly from Proposition 6.3. To establish the other statements, notice that

$$Y = \Omega_{m/2} y = t_0 \Omega_{m/2} e_0 + \Omega_{m/2} O + i \Omega_{m/2} E.$$

By Proposition 6.1, $\Omega_{m/2} O$ is odd symmetric and purely imaginary. Similarly, $\Omega_{m/2} E$ is even symmetric and purely real. Thus $\Omega_{m/2}(O + Ei)$ is purely imaginary, and it follows that the real part of Y satisfies

$$\mathrm{Re}(Y) = t_0 \Omega_{m/2} e_0 = t_0 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}.$$

□

Proposition 6.5. *Referring to Proposition 6.3, suppose x is RQE symmetric. Let $Y = \Omega_{m/2} y$ and let $Z = \Omega_{m/4} y_e$, where $y_e = [y_{2j}]_{j=0}^{m/4-1}$ is the even-indexed part of y. Then for $k = 0, 1, \ldots, m/4 - 1$,*

$$\begin{aligned} Y_k &= Z_k - \overline{\omega} e^{4\pi i k/m} \overline{Z}_k \\ Y_{k+m/4} &= Z_k + \overline{\omega} e^{4\pi i k/m} \overline{Z}_k \end{aligned}.$$

Consequently, Y can be computed from one complex $m/4$-dimensional FFT.

Proof. According to the splitting equation (2.2), for $k = 0, 1, \ldots, m/2 - 1$,

$$Y_k = \sum_{j=0}^{m/4-1} y_{2j}\, e^{-2\pi j k i/(m/4)} + e^{-2\pi k i/(m/2)} \sum_{j=0}^{m/4-1} y_{2j+1}\, e^{-2\pi j k i/(m/4)}. \tag{6.5}$$

The symmetry in y described in Proposition 6.3ii is such that $y_{2j+1} = -\overline{\omega y}_{m/2-2j-2}$ for $k = 0, 1, \ldots, m/4 - 1$. Therefore, for $k = 0, 1, \ldots, m/4 - 1$, the splitting equation becomes

$$Y_k = \sum_{j=0}^{m/4-1} y_{2j}\, e^{-2\pi jki/(m/4)} - \overline{\omega} e^{-2\pi ki/(m/2)} \sum_{j=0}^{m/4-1} \overline{y}_{m/2-2j-2}\, e^{-2\pi jki/(m/4)}.$$

Reversing the order of summation in the second sum gives

$$Y_k = \sum_{j=0}^{m/4-1} y_{2j}\, e^{-2\pi jki/(m/4)} - \overline{\omega} e^{-2\pi ki/(m/2)} \sum_{j=0}^{m/4-1} \overline{y}_{2j}\, e^{-2\pi(m/4-1-j)ki/(m/4)}.$$

Simplifying this expression results in

$$Y_k = \sum_{j=0}^{m/4-1} y_{2j}\, e^{-2\pi jki/(m/4)} - \overline{\omega} e^{4\pi ki/m} \sum_{j=0}^{m/4-1} \overline{y}_{2j}\, e^{2\pi jki/(m/4)}.$$

After recognizing that

$$Z_k = \sum_{j=0}^{m/4-1} y_{2j}\, e^{-2\pi jki/(m/4)},$$

the equation above takes the form

$$Y_k = Z_k - \overline{\omega} e^{4\pi ki/m} \overline{Z}_k; \quad k = 0, 1, \ldots, m/4 - 1.$$

Since the summations in the original splitting equation (6.5) are periodic in k with period $m/4$ and since $e^{4\pi(k+m/4)i/m} = -e^{4\pi ki/m}$, it follows that

$$Y_{k+m/4} = Z_k + \overline{\omega} e^{4\pi ki/m} \overline{Z}_k; \quad k = 0, 1, \ldots, m/4 - 1.$$

□

In the next two sections, the general zero-padding strategies described above will be applied to the specific symmetric vectors processed by the DCSSA.

6.2.2 Zero Padding $\dot{\mathbf{u}}_m$ and $\dot{\mathbf{q}}_m$

In this section, a detailed procedure for zero padding and transforming the vectors $\dot{\mathbf{u}}_m$ and $\dot{\mathbf{q}}_m$ will be described. Because these vectors have similar properties, i.e., they are m-dimensional and RQE symmetric, they both require the same steps for zero padding and transforming. The procedure will be described for only $\dot{\mathbf{u}}_m$, but it applies identically to $\dot{\mathbf{q}}_m$.

Suppose m is a fixed positive integer divisible by four. Let $n = 2m$, $\omega = e^{-2\pi i/n}$, and, for any positive integer k, $W_k = \text{diag}([\omega^j]_{j=0}^{k-1})$. Assume that the m-dimensional, RQE-symmetric vector $\dot{\mathbf{u}}_m$ and its m-dimensional DFT $\Omega_m \dot{\mathbf{u}}_m$ are given. The following steps describe an efficient, FFT-based procedure for computing the DFT of the zero-padded $\dot{\mathbf{u}}_m$. In steps where arithmetic operations are required, the count is given in brackets. The particular FFT algorithms referred to below were summarized at the end of Section 2.1.2.

1. Let a and b be $m/2$-dimensional vectors such that

$$\dot{\mathbf{u}}_m = \begin{bmatrix} a \\ \\ b \end{bmatrix}.$$

2. With $y = W_{m/2}(a - bi)$, compute the even-indexed part of y: $y_e = [y_{2j}]_{j=0}^{m/4-1}$. Since $\omega^0 = 1$, no work is required to compute y_0. $[(\frac{m}{4} - 1)\,\mu_{\mathbb{C}}]$

3. Use CFFTF to compute $Z = \Omega_{m/4} y_e$. $[\psi_1(\frac{m}{4})]$

4. Use Proposition 6.5 to obtain $Y = \Omega_{m/2} y$. $[\frac{m}{4}\,\mu_{\mathbb{C}} + \frac{m}{2}\,\alpha_{\mathbb{C}}]$

5. Form the vector

$$\begin{bmatrix} Y \\ \\ J_{m/2}\overline{Y} \end{bmatrix}.$$

   It follows from equation (6.4) and Proposition 6.2 that this vector is the even/odd permutation of $\Omega_m W_m \dot{\mathbf{u}}_m$.

6. Permute the vector above to form $\Omega_m W_m \dot{\mathbf{u}}_m$.

7. Form the n-dimensional vector

$$\begin{bmatrix} \Omega_m \dot{\mathbf{u}}_m \\ \\ \Omega_m W_m \dot{\mathbf{u}}_m \end{bmatrix}.$$

   According to equation (6.2), this vector is the even/odd permutation of DFT of the zero-padded $\dot{\mathbf{u}}_m$.

8. Permute the vector above to form

$$\Omega_n \begin{bmatrix} \dot{\mathbf{u}}_m \\ 0_m \end{bmatrix}.$$

The procedure described above applies only when $m \geq 4$. In such a case, the total numbers of arithmetic operations are

$$\left(\frac{m}{2} - 1\right) \mu_{\mathbb{C}} + \frac{m}{2} \alpha_{\mathbb{C}} + \psi_1(m/4) = \left(\frac{m}{4} \log_2 m + \frac{3m}{4}\right) \mu_{\mathbb{R}} + \left(\frac{3m}{4} \log_2 m - \frac{m}{4} + 2\right) \alpha_{\mathbb{R}}.$$

For the cases in which $m = 1$ or $m = 2$, the DFT of the zero-padded vector can be obtained from the original vector and its DFT by direct computation involving no additional arithmetic operations.

6.2.3 Zero Padding $\dot{\mathbf{v}}_m$ and $\dot{\mathbf{p}}_m$

In this section, a detailed procedure for zero padding and transforming the vectors $\dot{\mathbf{v}}_m$ and $\dot{\mathbf{p}}_m$ will be described. In contrast to $\dot{\mathbf{u}}_m$ and $\dot{\mathbf{q}}_m$, the vectors $\dot{\mathbf{v}}_m$ and $\dot{\mathbf{p}}_m$ are RE symmetric. Several of the steps in the zero-padding procedure are significantly different than those described in the previous section. Nonetheless, the procedures for $\dot{\mathbf{v}}_m$ and $\dot{\mathbf{p}}_m$ are identical, so they will be described for only $\dot{\mathbf{v}}_m$.

Suppose m is divisible by four, and let $n = 2m$, $\omega = e^{-2\pi i/n}$, and $W_k = \mathrm{diag}([\omega^j]_{j=0}^{k-1})$. Assume that the m-dimensional, RE-symmetric vector $\dot{\mathfrak{v}}_m$ and its m-dimensional DFT $\Omega_m \dot{\mathfrak{v}}_m$ are given. The following steps describe an efficient, FFT-based procedure for computing the DFT of the zero-padded $\dot{\mathfrak{v}}_m$. In steps where arithmetic operations are required, the count is given in brackets. The particular FFT algorithms referred to below were summarized at the end of Section 2.1.2.

1. Let a and b be $m/2$-dimensional vectors such that

$$\dot{\mathfrak{v}}_m = \begin{bmatrix} a \\ \\ b \end{bmatrix}.$$

2. Compute $y = W_{m/2}(a - bi)$. By Proposition 6.3, y is completely determined by its first $m/4 + 1$ elements. Since $\omega^0 = 1$, no work is required to compute y_0. Furthermore, since $\dot{\mathfrak{v}}_m$ is RE symmetric, $a_{m/4} = b_{m/4}$. It follows that $y_{m/4} = -i\sqrt{2}a_{m/4}$ can be computed with one real multiplication. The remaining $m/4 - 1$ elements each require one complex multiplication. $[(\frac{m}{4} - 1)\,\mu_{\mathbb{C}} + 1\,\mu_{\mathbb{R}}]$

3. Let $y = a_0\, e_0 + O + Ei$ as in Proposition 6.4.

   (a) Use RDSTF to compute the purely imaginary, odd-symmetric vector $\Omega_{m/2}O$. $[\psi_6(\frac{m}{2})]$

   (b) Use RDCTF to compute the RE-symmetric vector $\Omega_{m/2}E$. $[\psi_4(\frac{m}{2})]$

(c) Add the purely imaginary vectors $\Omega_{m/2}O$ and $\Omega_{m/2}Ei$. Since $\Omega_{m/2}O$ is odd symmetric, its 0- and $m/4$-elements are zero. $[(\frac{m}{2}-2)\,\alpha_{\mathbb{R}}]$

(d) Form $Y = \Omega_{m/2}y$:

$$Y = \begin{bmatrix} a_o \\ \vdots \\ a_o \end{bmatrix} + \Omega_{m/2}(O + Ei).$$

If this procedure is being applied to $\dot{\mathfrak{v}}_m$, then $a_0 = v_m(0) = 0$. On the other hand, if it is being applied to $\dot{\mathfrak{p}}_m$. then $a_0 = p_m(0)$. [No additional operations]

4. Form the vector

$$\begin{bmatrix} Y \\ J_{m/2}\overline{Y} \end{bmatrix}.$$

It follows from equation (6.4) and Proposition 6.2 that this vector is the even/odd permutation of $\Omega_m W_m \dot{\mathfrak{v}}_m$.

5. Permute the vector above to form $\Omega_m W_m \dot{\mathfrak{v}}_m$.

6. Form the n-dimensional vector

$$\begin{bmatrix} \Omega_m \dot{\mathfrak{v}}_m \\ \Omega_m W_m \dot{\mathfrak{v}}_m \end{bmatrix}.$$

According to equation (6.2), this vector is the even/odd permutation of DFT of the zero-padded $\dot{\mathfrak{v}}_m$.

7. Permute the vector above to form

$$\Omega_n \begin{bmatrix} \dot{\mathfrak{v}}_m \\ 0_m \end{bmatrix}.$$

This procedure results in a transform whose even-indexed elements are purely real and whose odd-indexed elements are complex with constant real part. (That constant is either $v_m(0) = 0$ or $p_m(0)$ depending on which vector is being processed.) The procedure applies only when $m \geq 4$. The total numbers of arithmetic operations are

$$\left(\frac{m}{4} - 1\right) \mu_{\mathbb{C}} + 1\,\mu_{\mathbb{R}} + \left(\frac{m}{2} - 2\right) \alpha_{\mathbb{R}} + \psi_4(m/2) + \psi_6(m/2)$$

$$= \left(\frac{m}{4}\log_2 m + \frac{3m}{8} + 1\right) \mu_{\mathbb{R}} + \left(\frac{3m}{4}\log_2 m + \frac{m}{8}\right) \alpha_{\mathbb{R}}.$$

As in the previous section, the cases for $m = 1$ or $m = 2$ can be handled with no additional arithmetic operations.

6.2.4 Coefficient Vectors of p_m and q_m

The modified polynomials $\dot{u}_m(z)$, $\dot{v}_m(z)$, $\dot{p}_m(z)$ and $\dot{q}_m(z)$ were defined in such a way that their coefficient vectors could be processed with power-of-two FFTs. Unfortunately, steps 1 and 3 of Algorithm 5.1 require the original polynomials, not their modifications. Since $u_m = \dot{u}_m$ and $v_m = \dot{v}_m$, these polynomials present no problem. However, a method

must be developed by which the DFTs of the zero-padded coefficient vectors of $p_m(z) = \dot{p}_m(z) + p_m(0)z^m$ and $q_m(z) = z\dot{q}_m(z)$ are obtained from the DFTs of $\dot{\mathfrak{p}}_m$ and $\dot{\mathfrak{q}}_m$. The next two propositions pave the way for such a method.

Proposition 6.6. *Suppose x is a vector of length $2m$. Define X and Y by*

$$X = \Omega_{2m}x \quad \textit{and} \quad Y = \Omega_{2m}(x + te_m),$$

where $t \in \mathbb{R}$ and e_m (indexed from zero) is the $(m+1)$st column of I_{2m}. Then for $k = 0, 1, \ldots, 2m-1$,

$$Y_k = X_k + (-1)^k t.$$

Proof. The $(m+1)$st column of the $2m$-dimensional Fourier matrix, Ω_{2m}, is the $2m$-dimensional vector

$$[1, -1, 1, -1, \ldots, 1, -1]^T.$$

The result follows immediately from the linearity of the DFT. □

Proposition 6.7. *Suppose x is an RQE symmetric vector of length m. Define X and Y by*

$$X = \Omega_{2m}\begin{bmatrix} x \\ \\ 0_m \end{bmatrix} \quad \textit{and} \quad Y = \Omega_{2m}\begin{bmatrix} 0 \\ \\ x \\ \\ 0_{m-1} \end{bmatrix}.$$

Then $Y_e = \overline{X}_e$ and $Y_o = -\overline{X}_o$, where the e- and o-subscripts denote the even-indexed and odd-indexed parts.

Proof. Let $n = 2m$ and $\omega = e^{-2\pi i/n}$. By equation (2.1a),

$$X_k = \sum_{j=0}^{m/2-1} x_j \omega^{jk} + \sum_{j=0}^{m/2-1} x_{m/2+j} \omega^{(m/2+j)k}; \quad k = 0, 1, \ldots, n-1.$$

Since x is RQE symmetric, $x_{m/2+j} = x_{m/2-1-j}$. Using this symmetry and the fact that $\omega^{m/2} = -i$, the expression above is equivalent to

$$X_k = \sum_{j=0}^{m/2-1} x_j \omega^{jk} + (-i)^k \sum_{j=0}^{m/2-1} x_{m/2-1-j} \omega^{jk}.$$

After reversing the order of summation of the second sum, X_k is given by

$$X_k = \sum_{j=0}^{m/2-1} x_j [\omega^{jk} + (-i)^k \omega^{(m/2-1-j)k}],$$

and this reduces to

$$X_k = \sum_{j=0}^{m/2-1} x_j [\omega^{jk} + (-1)^k \overline{\omega}^{(j+1)k}].$$

Using the same reasoning, Y_k is given by

$$Y_k = \sum_{j=0}^{m/2-1} x_j \omega^k \omega^{jk} + \sum_{j=0}^{m/2-1} x_{m/2+j} \omega^k \omega^{(m/2+j)k}; \quad k = 0, 1, \ldots, n-1,$$

which reduces to

$$Y_k = \sum_{j=0}^{m/2-1} x_j[\omega^{(j+1)k} + (-1)^k \overline{\omega}^{jk}].$$

Now, if k is even, $(-1)^k = 1$ and $X_k = \overline{Y}_k$. On the other hand, if k is odd, $(-1)^k = -1$ and $X_k = -\overline{Y}_k$. □

Using Proposition 6.6, the DFT of the zero-padded coefficient vector of p_m can be easily obtained from that of $\dot{\mathfrak{p}}_m$:

$$\Omega_{2m} \begin{bmatrix} \mathfrak{p}_m \\ 0_{m-1} \end{bmatrix} = \Omega_{2m} \begin{bmatrix} \dot{\mathfrak{p}}_m \\ 0_m \end{bmatrix} + \begin{bmatrix} p_m(0) \\ -p_m(0) \\ \vdots \\ p_m(0) \\ -p_m(0) \end{bmatrix}. \tag{6.6}$$

As a consequence of the symmetry in the DFT of the zero-padded $\dot{\mathfrak{p}}_m$, which was described in Section 6.2.3, the vector expressed by equation (6.6) has a purely real even-indexed part and a purely imaginary odd-indexed part. Therefore, only an additional $m/2 + 1$ real additions are required in using (6.6) to obtain the DFT on the left from the DFT of the zero-padded $\dot{\mathfrak{p}}_m$. (In the case when $m = 1$, two real additions are required.)

Using Proposition 6.7, the DFT of the zero-padded coefficient vector of q_m,

$$\Omega_{2m}\begin{bmatrix} \mathfrak{q}_m \\ 0_{m-1} \end{bmatrix} = \Omega_{2m}\begin{bmatrix} 0 \\ \dot{\mathfrak{q}}_m \\ 0_{m-1} \end{bmatrix},$$

can be obtained from that of $\dot{q}_m$ by simply negating the imaginary parts of the even-indexed elements and negating the real parts of the odd-indexed elements. These steps require no additional arithmetic operations.

6.3 Split Schur Functions

The preceding sections focused on preparing the symmetric polynomials for processing in the superfast DCSSA. In this section, the focus is on the actual implementation of step 1 of Algorithm 5.1. This step applies equations (5.9),

$$h_m^{(m)}(z) = \text{first } m \text{ terms of } \frac{p_{0,m}(z)h_0^{(2m)}(z) - u_{0,m}(z)h_{-1}^{(2m)}(z)}{z^m}$$

and

$$h_{m-1}^{(m)}(z) = \text{first } m \text{ terms of } \frac{v_{0,m}(z)h_{-1}^{(2m)}(z) - q_{0,m}(z)h_0^{(2m)}(z)}{z^m},$$

in which m terms of the split Schur functions $h_{m-1}(z)$ and $h_m(z)$ are computed from $2m$ terms of $h_{-1}(z)$ and $h_0(z)$. Notice that the double subscript notation is once again being

used for the symmetric polynomials. As observed in Chapter 5, the formulas above indicate that the coefficients of $h_{m-1}^{(m)}$ and $h_m^{(m)}$ are, in fact, the coefficients of the terms of degree m through $2m-1$ of

$$v_{0,m}(z)h_{-1}^{(2m)}(z) - q_{0,m}(z)h_0^{(2m)}(z) \tag{6.7}$$

and

$$p_{0,m}(z)h_0^{(2m)}(z) - u_{0,m}(z)h_{-1}^{(2m)}(z), \tag{6.8}$$

respectively. Normally, the products in these expressions would be computed as convolutions of vectors padded with zeros to length $3m$. However, since only certain terms are required, it turns out that convolutions of $2m$-dimensional vectors will suffice.

Proposition 6.8. *Suppose* $f(z) = \sum_{j=0}^{m} f_j z^j$ *and* $g(z) = \sum_{j=0}^{2m-1} g_j z^j$ *are polynomials of degrees no greater than* m *and* $2m-1$*, respectively. Let* C *be the circulant matrix whose first column is*

$$[f_0, f_1, \ldots, f_m, \underbrace{0, 0, \ldots, 0}_{m-1 \ zeros}]^T,$$

and let

$$x = [g_0, g_1, \ldots, g_{2m-1}]^T.$$

Then the coefficients of the terms of $f(z)g(z)$ *of degrees* m *through* $2m-1$ *are the last* m *elements of the matrix-vector product* Cx*.*

Proof. Let C' be the circulant matrix whose first column is

$$[f_0, f_1, \ldots, f_m, \underbrace{0, 0, \ldots, 0}_{2m-1 \text{ zeros}}]^T,$$

and let

$$y = [g_0, g_1, \ldots, g_{2m-1}, \underbrace{0, 0, \ldots, 0}_{m \text{ zeros}}]^T.$$

The coefficients of $f(z)g(z)$ are given by $C'y$.

Now if the circulant matrix C is written in the block form

$$C = \begin{bmatrix} C_1 & C_2 \\ C_2 & C_1 \end{bmatrix},$$

then C' has the form

$$C' = \begin{bmatrix} C_1 & 0 & C_2 \\ C_2 & C_1 & 0 \\ 0 & C_2 & C_1 \end{bmatrix},$$

where the blocks are $m \times m$ matrices. Partition x into the two m-dimensional vectors a and b. Notice that

$$Cx = \begin{bmatrix} C_1 & C_2 \\ C_2 & C_1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} C_1 a + C_2 b \\ C_2 a + C_1 b \end{bmatrix},$$

while

$$C'y = \begin{bmatrix} C_1 & 0 & C_2 \\ C_2 & C_1 & 0 \\ 0 & C_2 & C_1 \end{bmatrix} \begin{bmatrix} a \\ b \\ 0_m \end{bmatrix} = \begin{bmatrix} C_1 a \\ C_2 a + C_1 b \\ C_2 a \end{bmatrix}.$$

Therefore, the middle one-third of $C'y$ is the last one-half of Cx. □

By using Proposition 6.8, $h_{m-1}^{(m)}$ and $h_m^{(m)}$ can be computed with $2m$-dimensional vectors—zero padding is required for the symmetric vectors, but not for $h_{-1}^{(2m)}$ and $h_0^{(2m)}$. The procedures for computing the coefficients of $h_{m-1}^{(m)}(z)$ and $h_m^{(m)}(z)$ are described next. As usual, the split Schur functions will be identified with their coefficient vectors using the notation

$$h_{m-1}^{(m)}(z) \longleftrightarrow \mathfrak{h}_{m-1}^{(m)} \quad \text{and} \quad h_m^{(m)}(z) \longleftrightarrow \mathfrak{h}_m^{(m)}.$$

6.3.1 Computing $\mathfrak{h}_{m-1}^{(m)}$

In terms of the modified symmetric polynomials, expression (6.7) takes the form

$$\dot{v}_{0,m}(z) h_{-1}^{(2m)}(z) - z\dot{q}_{0,m}(z) h_0^{(2m)}(z).$$

It follows that $\mathfrak{h}_{m-1}^{(m)}$ is given by the last m elements of

$$\begin{bmatrix} \dot{\mathfrak{v}}_{0,m} \\ 0_m \end{bmatrix} * \mathfrak{h}_{-1}^{(2m)} - \begin{bmatrix} 0 \\ \dot{\mathfrak{q}}_{0,m} \\ 0_{m-1} \end{bmatrix} * \mathfrak{h}_{0}^{(2m)},$$

which will be computed with DFTs:

$$\frac{1}{2m}\Omega_{2m}^{*}\left(\Omega_{2m}\begin{bmatrix} \dot{\mathfrak{v}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m}\mathfrak{h}_{-1}^{(2m)} - \Omega_{2m}\begin{bmatrix} \mathfrak{q}_{0,m} \\ 0_{m-1} \end{bmatrix} \star \Omega_{2m}\mathfrak{h}_{0}^{(2m)}\right). \tag{6.9}$$

Assuming that the DFTs of the zero-padded vectors are given, the following steps describe the computation of $\mathfrak{h}_{m-1}^{(m)}$. When arithmetic operations are required, the count is given in brackets at the end of the step.

1. Use RFFTF to compute the vector $\Omega_{2m}\mathfrak{h}_{-1}^{(2m)}$. $[\psi_2(2m)]$

2. Compute the Schur product of the DFTs:

$$\Omega_{2m}\begin{bmatrix} \dot{\mathfrak{v}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m}\mathfrak{h}_{-1}^{(2m)}.$$

   Since this involves the DFTs of real vectors, the result is conjugate symmetric. Therefore, only the first $m+1$ elements must be computed. Since $v(0) = 0$, the procedure for zero-padding and transforming $\dot{\mathfrak{v}}_{0,m}$, which was described in Section 6.2.3, results in a

vector whose even-indexed elements are purely real and whose odd-indexed elements are purely imaginary.

(a) The 0- and m-indexed elements of both DFTs are real. Each requires a single real multiplication. $[2\,\mu_{\mathbb{R}}]$

(b) The elements indexed $1, 3, \ldots, m-1$ each require an imaginary-by-complex multiplication. $[m\,\mu_{\mathbb{R}}]$

(c) The elements indexed $2, 4, \ldots, m-2$ each require a real-by-complex multiplication. $[(m-2)\,\mu_{\mathbb{R}}]$

3. Use RFFTF to compute the vector $\Omega_{2m}\mathfrak{h}_0^{(2m)}$. $[\psi_2(2m)]$

4. Compute the Schur product of the DFTs:

$$\Omega_{2m}\begin{bmatrix}\mathfrak{q}_{0,m}\\ 0_{m-1}\end{bmatrix} \star \Omega_{2m}\mathfrak{h}_0^{(2m)}.$$

As in step 2, the result is conjugate symmetric, and only the first $m+1$ elements must be computed.

(a) The 0-indexed elements of both DFTs are real. $[1\,\mu_{\mathbb{R}}]$

(b) The m-indexed element of the transform of the zero-padded $\mathfrak{q}_{0,m}$ is zero (when $m \geq 2$). No work is required for element m.

(c) The elements indexed 1 through $m-1$ each require a complex multiplication. $[(m-1)\,\mu_{\mathbb{C}}]$

5. Compute the difference of the Schur products. By symmetry, only the first $m+1$ elements are required.

   (a) The 0-indexed elements are real. $[1\,\alpha_{\mathbb{R}}]$

   (b) No work is required for the m-indexed element (see step 4(b)).

   (c) The elements indexed 1 through $m-1$ each require a complex addition. $[(m-1)\,\alpha_{\mathbb{C}}]$

6. Use RFFTB to compute the $2m$-dimensional inverse transform. $[\psi_3(2m)]$

7. $\mathfrak{h}_{m-1}^{(m)}$ is obtained from the last m elements of the inverse transform.

The total numbers of arithmetic operations required by the procedure described above are

$$2\psi_2(2m) + \psi_3(2m) + (2m+1)\,\mu_{\mathbb{R}} + (m-1)\,\mu_{\mathbb{C}} + (m-1)\,\alpha_{\mathbb{C}} + 1\,\alpha_{\mathbb{R}}$$

$$= (3m\log_2 m + 3)\,\mu_{\mathbb{R}} + (9m\log_2 m - 2m + 9)\,\alpha_{\mathbb{R}} + 2m\,\delta_{\mathbb{R}}.$$

This count is valid when $m \geq 2$. When $m = 1$, $\mathfrak{h}_{m-1}^{(m)}$ can be computed directly with 4 real additions and 2 divisions.

6.3.2 Computing $\mathfrak{h}_m^{(m)}$

In terms of the modified symmetric polynomials, expression (6.8) takes the form

$$[\dot{p}_{0,m}(z) + p_{0,m}(0)z^m]h_0^{(2m)}(z) - \dot{u}_{0,m}(z)h_{-1}^{(2m)}(z).$$

It follows that $\mathfrak{h}_m^{(m)}$ is given by the last m elements of

$$\begin{bmatrix} \mathfrak{p}_{0,m} \\ 0_{m-1} \end{bmatrix} * \mathfrak{h}_0^{(2m)} - \begin{bmatrix} \dot{\mathfrak{u}}_{0,m} \\ 0_m \end{bmatrix} * \mathfrak{h}_{-1}^{(2m)},$$

which will be computed with DFTs:

$$\frac{1}{2m}\Omega_{2m}^* \left(\Omega_{2m} \begin{bmatrix} \mathfrak{p}_{0,m} \\ 0_{m-1} \end{bmatrix} \star \Omega_{2m}\mathfrak{h}_0^{(2m)} - \Omega_{2m} \begin{bmatrix} \dot{\mathfrak{u}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m}\mathfrak{h}_{-1}^{(2m)} \right). \tag{6.10}$$

Assuming that all of the DFTs are given, the following steps describe the computation of $\mathfrak{h}_m^{(m)}$. When arithmetic operations are required, the count is given in brackets at the end of the step.

1. Compute the Schur product of the DFTs:

$$\Omega_{2m} \begin{bmatrix} \mathfrak{p}_{0,m} \\ 0_{m-1} \end{bmatrix} \star \Omega_{2m}\mathfrak{h}_0^{(2m)}.$$

Because this involves the DFTs of real vectors, the result is conjugate symmetric. Therefore, only the first $m+1$ elements must be computed. The procedure for computing the DFT of the zero-padded $\mathfrak{p}_{0,m}$, which was described in Sections 6.2.3 and 6.2.4, results in a vector whose even-indexed elements are purely real and whose odd-indexed elements are purely imaginary.

(a) The 0- and m-indexed elements of both DFTs are real. Each requires a single real multiplication. $[2\,\mu_{\mathbb{R}}]$

(b) The elements indexed $2, 4, \ldots, m-2$ each require a real-by-complex multiplication. $[(m-2)\,\mu_{\mathbb{R}}]$

(c) The elements indexed $1, 3, \ldots, m-1$ each require an imaginary-by-complex multiplication. $[m\,\mu_{\mathbb{R}}]$

2. Compute the Schur product of the DFTs:

$$\Omega_{2m} \begin{bmatrix} \dot{\mathfrak{u}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m} \mathfrak{h}_{-1}^{(2m)}.$$

As in step 1, the result is conjugate symmetric. Therefore, only the first $m+1$ elements must be computed.

(a) The 0-indexed elements of both DFTs are real. $[1\,\mu_{\mathbb{R}}]$

(b) The m-indexed element of the transform of the zero-padded $\dot{\mathfrak{u}}_m$ is zero (when $m \geq 2$). No work is required for element m.

(c) The elements indexed 1 through $m-1$ each require a complex multiplication. $[(m-1)\,\mu_{\mathbb{C}}]$

3. Compute the difference of the Schur products. By symmetry, only the first $m+1$ elements are required.

   (a) The 0-indexed elements are real. $[1\,\alpha_{\mathbb{R}}]$

   (b) No work is required for the m-indexed element (see step 2(b)).

   (c) The elements indexed 1 through $m-1$ each require a complex addition. $[(m-1)\,\alpha_{\mathbb{C}}]$

4. Use RFFTB to compute the $2m$-dimensional inverse transform. $[\psi_3(2m)]$

5. $\mathfrak{h}_m^{(m)}$ is obtained from the last m elements of the inverse transform.

The total numbers of arithmetic operations required by the procedure described above are

$$\psi_3(2m) + (2m+1)\,\mu_{\mathbb{R}} + (m-1)\,\mu_{\mathbb{C}} + (m-1)\,\alpha_{\mathbb{C}} + 1\,\alpha_{\mathbb{R}}$$

$$= (m\log_2 m + 4m - 1)\,\mu_{\mathbb{R}} + (3m\log_2 m + 2m + 1)\,\alpha_{\mathbb{R}} + 2m\,\delta_{\mathbb{R}}.$$

This count is valid when $m \geq 2$. When $m = 1$, $\mathfrak{h}_m^{(m)}$ can be computed, assuming the DFTs are given, with 1 real multiplication, 3 real additions, and 2 divisions.

6.4 Compositions of Symmetric Polynomials

In this section, procedures for implementing step 3 of Algorithm 5.1 will be described. Step 3 is the composition step in which the $(0, m)$-polynomials and the (m, m)-polynomials are composed to form the $(0, 2m)$-polynomials. These compositions follow from equations (5.8) and are given by

$$u_{0,2m}(z) = u_{0,m}(z)p_{m,m}(z) + v_{0,m}(z)u_{m,m}(z) \tag{6.11a}$$

$$v_{0,2m}(z) = u_{0,m}(z)q_{m,m}(z) + v_{0,m}(z)v_{m,m}(z) \tag{6.11b}$$

$$p_{0,2m}(z) = p_{0,m}(z)p_{m,m}(z) + q_{0,m}(z)u_{m,m}(z) \tag{6.11c}$$

$$q_{0,2m}(z) = p_{0,m}(z)q_{m,m}(z) + q_{0,m}(z)v_{m,m}(z). \tag{6.11d}$$

Because there are subtle differences in the implementations of the equations, each will be considered in its own subsection. It is assumed that the DFTs of the zero-padded coefficient vectors of the symmetric polynomials have been pre-computed according to the techniques described in Section 6.2. Since these DFTs and their Schur products are conjugate symmetric, the computations will only involve that first $m + 1$ elements of these vectors.

6.4.1 Computing the DFT of $\dot{\mathfrak{u}}_{0,2m}$

In terms of the modified symmetric polynomials, equation (6.11a) takes the form

$$\dot{u}_{0,2m}(z) = \dot{u}_{0,m}(z)[\dot{p}_{m,m}(z) + p_{m,m}(0)z^m] + \dot{v}_{0,m}(z)\dot{u}_{m,m}(z).$$

It follows that the DFT of $\dot{\mathfrak{u}}_{0,2m}$ is given by

$$\Omega_{2m}\dot{\mathfrak{u}}_{0,2m} = \Omega_{2m}\begin{bmatrix}\dot{\mathfrak{u}}_{0,m}\\ 0_m\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\mathfrak{p}_{m,m}\\ 0_{m-1}\end{bmatrix} + \Omega_{2m}\begin{bmatrix}\dot{\mathfrak{v}}_{0,m}\\ 0_m\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\dot{\mathfrak{u}}_{m,m}\\ 0_m\end{bmatrix}.$$

Because the m-indexed element of the DFT of $\dot{\mathfrak{u}}_{0,2m}$ is zero, only elements 0 through $m-1$ must be computed.

1. Compute the Schur product of the DFTs:

$$\Omega_{2m}\begin{bmatrix}\dot{\mathfrak{u}}_{0,m}\\ 0_m\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\mathfrak{p}_{m,m}\\ 0_{m-1}\end{bmatrix}.$$

   The DFT of the zero-padded $\mathfrak{p}_{m,m}$ is a vector whose even-indexed elements are purely real and whose odd-indexed elements are purely imaginary.

   (a) The 0-indexed elements of both DFTs are real. [$1\,\mu_{\mathbb{R}}$]

(b) The elements indexed $2, 4, \ldots, m-2$ each require a complex-by-real multiplication. $[(m-2)\,\mu_{\mathbb{R}}]$

(c) The elements indexed $1, 3, \ldots, m-1$ each require an complex-by-imaginary multiplication. $[m\,\mu_{\mathbb{R}}]$

2. Compute the Schur product of the DFTs:

$$\Omega_{2m} \begin{bmatrix} \dot{\mathfrak{v}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m} \begin{bmatrix} \dot{\mathfrak{u}}_{m,m} \\ 0_m \end{bmatrix}.$$

The DFT of the zero-padded $\dot{\mathfrak{v}}_{0,m}$ is a vector whose even-indexed elements are purely real and whose odd-indexed elements are purely imaginary.

(a) The 0-indexed elements of both DFTs are real. $[1\,\mu_{\mathbb{R}}]$

(b) The elements indexed $2, 4, \ldots, m-2$ each require a real-by-complex multiplication. $[(m-2)\,\mu_{\mathbb{R}}]$

(c) The elements indexed $1, 3, \ldots, m-1$ each require an imaginary-by-complex multiplication. $[m\,\mu_{\mathbb{R}}]$

3. Compute the sum of the Schur products.

(a) The 0-indexed elements are real. $[1\,\alpha_{\mathbb{R}}]$

(b) The elements indexed 1 through $m-1$ each require a complex addition. $[(m-1)\,\alpha_{\mathbb{C}}]$

The numbers of arithmetic operations required by the procedure described above are

$$(4m-2)\,\mu_{\mathbb{R}} + (2m-1)\,\alpha_{\mathbb{R}}.$$

These counts apply when $m \geq 2$. If $m = 1$, the computation of $\Omega_{2m}\dot{\mathfrak{u}}_{0,2m}$ requires no operations.

6.4.2 Computing the DFT of $\dot{\mathfrak{v}}_{0,2m}$

In terms of the modified symmetric polynomials, equation (6.11b) takes the form

$$\dot{v}_{0,2m}(z) = \dot{u}_{0,m}(z) z \dot{q}_{m,m}(z) + \dot{v}_{0,m}(z)\dot{v}_{m,m}(z).$$

It follows that the DFT of $\dot{\mathfrak{v}}_{0,2m}$ is given by

$$\Omega_{2m}\dot{\mathfrak{v}}_{0,2m} = \Omega_{2m}\begin{bmatrix} \dot{\mathfrak{u}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m}\begin{bmatrix} \mathfrak{q}_{m,m} \\ 0_{m-1} \end{bmatrix} + \Omega_{2m}\begin{bmatrix} \dot{\mathfrak{v}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m}\begin{bmatrix} \dot{\mathfrak{v}}_{m,m} \\ 0_m \end{bmatrix}.$$

Since $\dot{\mathfrak{v}}_{0,2m}$ is RE symmetric, its DFT is also RE symmetric. Therefore, only the real parts of the elements indexed 0 through m must be computed.

1. Compute the real parts of the Schur product of the DFTs:

$$\Omega_{2m} \begin{bmatrix} \dot{\mathfrak{u}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m} \begin{bmatrix} \mathfrak{q}_{m,m} \\ 0_{m-1} \end{bmatrix}.$$

   (a) The 0-indexed elements of both DFTs are real. $[1\,\mu_{\mathbb{R}}]$

   (b) The elements indexed 1 through $m-1$ each require the real part of a complex multiplication. $[(2m-2)\,\mu_{\mathbb{R}} + (m-1)\,\alpha_{\mathbb{R}}]$

   (c) The m-indexed element is zero. No work is required.

2. Compute the (purely real) Schur product of the DFTs:

$$\Omega_{2m} \begin{bmatrix} \dot{\mathfrak{v}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m} \begin{bmatrix} \dot{\mathfrak{v}}_{m,m} \\ 0_m \end{bmatrix}.$$

   These DFTs are vectors whose whose even-indexed elements are purely real and whose odd-indexed elements are purely imaginary. Each element of the Schur product requires 1 real multiplication. $[(m+1)\,\mu_{\mathbb{R}}]$

3. Compute the sum of the real parts of the Schur products.

   (a) The m-indexed element requires no work.

   (b) The elements indexed 0 through $m-1$ each require 1 real addition. $[m\,\alpha_{\mathbb{R}}]$

The numbers of arithmetic operations required by the procedure described above are

$$3m\,\mu_{\mathbb{R}} + (2m-1)\,\alpha_{\mathbb{R}}.$$

These counts apply when $m \geq 2$. If $m = 1$, the computation of $\Omega_{2m}\dot{\mathfrak{v}}_{0,2m}$ requires no operations.

6.4.3 Computing the DFT of $\dot{\mathfrak{p}}_{0,2m}$

In terms of the modified symmetric polynomials, equation (6.11c) takes the form

$$\dot{p}_{0,2m}(z) + p_{0,2m}(0)z^{2m} = [\dot{p}_{0,m}(z) + p_{0,m}(0)z^m][\dot{p}_{m,m}(z) + p_{m,m}(0)z^m] + z\dot{q}_{0,m}(z)\dot{u}_{m,m}(z),$$

which reduces to

$$\dot{p}_{0,2m}(z) = \dot{p}_{0,m}[\dot{p}_{m,m}(z) + p_{m,m}(0)z^m] + p_{0,m}(0)z^m\dot{p}_{m,m} + z\dot{q}_{0,m}(z)\dot{u}_{m,m}(z).$$

It follows that the DFT of $\dot{\mathfrak{p}}_{0,2m}$ is given by $\Omega_{2m}\dot{\mathfrak{p}}_{0,2m} =$

$$\Omega_{2m}\begin{bmatrix}\dot{\mathfrak{p}}_{0,m}\\ 0_m\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\mathfrak{p}_{m,m}\\ 0_{m-1}\end{bmatrix} + \begin{bmatrix}p_{0,m}(0)\\ -p_{0,m}(0)\\ \vdots\\ p_{0,m}(0)\\ -p_{0,m}(0)\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\dot{\mathfrak{p}}_{m,m}\\ 0_m\end{bmatrix} + \Omega_{2m}\begin{bmatrix}\mathfrak{q}_{0,m}\\ 0_{m-1}\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\dot{\mathfrak{u}}_{m,m}\\ 0_m\end{bmatrix}.$$

Since $\dot{\mathfrak{p}}_{0,2m}$ is RE symmetric, its DFT is also RE symmetric. Therefore, only the real parts of the elements indexed 0 through m must be computed.

1. Compute the real parts of the Schur product of the DFTs:

$$\Omega_{2m}\begin{bmatrix}\dot{\mathfrak{p}}_{0,m}\\ 0_m\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\mathfrak{p}_{m,m}\\ 0_{m-1}\end{bmatrix}.$$

   The first DFT has purely real even-indexed elements and complex odd-indexed elements with constant real part. The second DFT has purely real even-indexed elements and purely imaginary odd-indexed elements. The real parts of the Schur product require 1 real multiplication per element. $[(m+1)\,\mu_{\mathbb{R}}]$

2. Compute the real parts of the Schur product:

$$\begin{bmatrix} p_{0,m}(0) \\ -p_{0,m}(0) \\ \vdots \\ p_{0,m}(0) \\ -p_{0,m}(0) \end{bmatrix} \star \Omega_{2m} \begin{bmatrix} \dot{\mathfrak{p}}_{m,m} \\ 0_m \end{bmatrix}.$$

As above, the DFT on the right has purely real even-indexed elements and complex odd-indexed elements with constant real part.

   (a) The elements indexed $0, 2, \ldots, m$ are each require 1 real multiplication. $[(\frac{m}{2} + 1)\,\mu_{\mathbb{R}}]$

   (b) Because the real parts of the odd-indexed elements of the DFT of the zero-padded $\dot{\mathfrak{p}}_{m,m}$ are constant, the elements indexed $1, 3, \ldots, m-1$ are constant and only require a single real multiplication. $[1\,\mu_{\mathbb{R}}]$

3. Compute the real parts of the Schur product of the DFTs:

$$\Omega_{2m} \begin{bmatrix} \mathfrak{q}_{0,m} \\ 0_{m-1} \end{bmatrix} \star \Omega_{2m} \begin{bmatrix} \dot{\mathfrak{u}}_{m,m} \\ 0_m \end{bmatrix}.$$

   (a) The 0-indexed elements of both DFTs are real. $[1\,\mu_{\mathbb{R}}]$

(b) The elements indexed 1 through $m-1$ each require the real part of a complex multiplication. $[(2m-2)\,\mu_{\mathbb{R}} + (m-1)\,\alpha_{\mathbb{R}}]$

(c) The m-indexed element is zero. No work is required.

4. Add the real parts of the Schur products. This requires 1 real addition for the m-indexed element and 2 real additions for each of the elements indexed 0 through $m-1$. $[(2m+1)\,\alpha_{\mathbb{R}}]$

The numbers of arithmetic operations required by the procedure described above are

$$\left(\frac{7m}{2} + 2\right)\,\mu_{\mathbb{R}} + 3m\,\alpha_{\mathbb{R}}.$$

These counts apply when $m \geq 2$. If $m = 1$, the computation of $\Omega_{2m}\dot{\mathfrak{p}}_{0,2m}$ requires 2 multiplications and 3 additions.

6.4.4 Computing the DFT of $\dot{\mathfrak{q}}_{0,2m}$

In terms of the modified symmetric polynomials, equation (6.11d) takes the form

$$z\dot{q}_{0,2m}(z) = [\dot{p}_{0,m}(z) + p_{0,m}(0)z^m]z\dot{q}_{m,m}(z) + z\dot{q}_{0,m}(z)\dot{v}_{m,m}(z),$$

which reduces to

$$\dot{q}_{0,2m}(z) = [\dot{p}_{0,m}(z) + p_{0,m}(0)z^m]\dot{q}_{m,m}(z) + \dot{q}_{0,m}(z)\dot{v}_{m,m}(z).$$

It follows that the DFT of $\dot{\mathfrak{q}}_{0,2m}$ is given by

$$\Omega_{2m}\dot{\mathfrak{q}}_{0,2m} = \Omega_{2m}\begin{bmatrix} \mathfrak{p}_{0,m} \\ 0_{m-1} \end{bmatrix} \star \Omega_{2m}\begin{bmatrix} \dot{\mathfrak{q}}_{m,m} \\ 0_m \end{bmatrix} + \Omega_{2m}\begin{bmatrix} \dot{\mathfrak{q}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m}\begin{bmatrix} \dot{\mathfrak{v}}_{m,m} \\ 0_m \end{bmatrix}.$$

Since the m-indexed element of the DFT of $\dot{\mathfrak{q}}_{0,2m}$ is zero, only elements 0 through $m-1$ must be computed.

1. Compute the Schur product of the DFTs:

$$\Omega_{2m}\begin{bmatrix} \mathfrak{p}_{0,m} \\ 0_{m-1} \end{bmatrix} \star \Omega_{2m}\begin{bmatrix} \dot{\mathfrak{q}}_{m,m} \\ 0_m \end{bmatrix}.$$

   The DFT of the zero-padded $\mathfrak{p}_{0,m}$, is a vector whose even-indexed elements are purely real and whose odd-indexed elements are purely imaginary.

   (a) The 0-indexed elements of both DFTs are real. $[1\,\mu_{\mathbb{R}}]$

   (b) The elements indexed $2, 4, \ldots, m-2$ each require a real-by-complex multiplication. $[(m-2)\,\mu_{\mathbb{R}}]$

   (c) The elements indexed $1, 3, \ldots, m-1$ each require an imaginary-by-complex multiplication. $[m\,\mu_{\mathbb{R}}]$

2. Compute the Schur product of the DFTs:

$$\Omega_{2m} \begin{bmatrix} \dot{\mathfrak{q}}_{0,m} \\ 0_m \end{bmatrix} \star \Omega_{2m} \begin{bmatrix} \dot{\mathfrak{v}}_{m,m} \\ 0_m \end{bmatrix}.$$

The DFT of the zero-padded $\mathfrak{v}_{m,m}$, is a vector whose even-indexed elements are purely real and whose odd-indexed elements are purely imaginary.

   (a) The 0-indexed elements of both DFTs are real. $[1\,\mu_{\mathbb{R}}]$

   (b) The elements indexed $2, 4, \ldots, m-2$ each require a complex-by-real multiplication. $[(m-2)\,\mu_{\mathbb{R}}]$

   (c) The elements indexed $1, 3, \ldots, m-1$ each require an complex-by-imaginary multiplication. $[m\,\mu_{\mathbb{R}}]$

3. Compute the sum of the Schur products.

   (a) The 0-indexed elements are real. $[1\,\alpha_{\mathbb{R}}]$

   (b) The elements indexed 1 through $m-1$ each require a complex addition. $[(m-1)\,\alpha_{\mathbb{C}}]$

The numbers of arithmetic operations required by the procedure described above are

$$(4m-2)\,\mu_{\mathbb{R}} + (2m-1)\,\alpha_{\mathbb{R}}.$$

These counts apply when $m \geq 2$. If $m = 1$, the computation of $\Omega_{2m}\dot{\mathbf{q}}_{0,2m}$ requires no operations.

6.5 The Implementation

In the preceding sections, the details of the individual steps of an FFT-based implementation of the DCSSA were described. In this section, the overall implementation is described and its operation count is derived. The following technical lemma will be required in the derivation of the operation count.

Lemma 6.1. *Suppose n and n_0 are powers of two. If ψ is a function such that $\psi(n) = 2\psi(n/2) + 2an\log_2 n + bn + c$ for $n > n_0$, then*

$$\psi(n) = an(\log_2 n)^2 + (a+b)n\log_2 n + dn - c,$$

where d is determined by the initial value $\psi(n_0)$.

Proof. The lemma is easily verified by induction or derived from the first-order linear difference equation for $\psi_\nu = \psi(2^\nu)$. □

As presented in Chapter 5, Algorithm 5.1 is initialized with the one-term truncated split Schur functions $h_{-1}^{(1)}$ and $h_0^{(1)}$ and the $(0,1)$-polynomials $p_{0,1}$, $q_{0,1}$, $u_{0,1}$, and $v_{0,1}$. Because several of the operation counts for the individual steps described above are valid only for vectors of length 4 or greater, it makes sense to rewrite the initialization stage so that the

algorithm is initialized with four-term truncated split Schur functions and $(0,4)$-polynomials. This has the additional benefit of improving the algorithm's efficiency.

The following procedure describes a superfast implementation of Algorithm 5.1. At each step, maximum operation counts are given in brackets. $\Psi(m)$ represents the number of floating-point operations required to obtain the $(\cdot, m)$-symmetric polynomials. A formula for $\Psi(m)$ will be derived below.

Initialization

The four-step initialization stage proceeds follows. The first initialization coincides with $k = 0$.

Step 1: Starting with the four-term split Schur functions $h_{k-1}^{(4)}$ and $h_k^{(4)}$, use the split Schur algorithm to compute the qC-parameters ζ_k, ζ_{k+1}, ζ_{k+2}, and ζ_{k+3}. Specifically,

$$\zeta_k = h_{k-1}^{(1)}/h_k^{(1)}$$

and

$$\begin{aligned}
h_{k+1}^{(3)}(z) &= \zeta_k h_k^{(3)}(z) + \tfrac{1}{z}\left[\zeta_k h_k^{(4)}(z) - h_{k-1}^{(4)}(z)\right], & \zeta_{k+1} &= h_k^{(1)}/h_{k+1}^{(1)} \\
h_{k+2}^{(2)}(z) &= \zeta_{k+1} h_{k+1}^{(2)}(z) + \tfrac{1}{z}\left[\zeta_{k+1} h_{k+1}^{(3)}(z) - h_k^{(3)}(z)\right], & \zeta_{k+2} &= h_{k+1}^{(1)}/h_{k+2}^{(1)} \\
h_{k+3}^{(1)}(z) &= \zeta_{k+2} h_{k+2}^{(1)}(z) + \tfrac{1}{z}\left[\zeta_{k+2} h_{k+2}^{(2)}(z) - h_{k+1}^{(2)}(z)\right], & \zeta_{k+3} &= h_{k+2}^{(1)}/h_{k+3}^{(1)}.
\end{aligned}$$

$[6\,\mu_{\mathbb{R}} + 12\,\alpha_{\mathbb{R}} + 4\,\delta_{\mathbb{R}}]$

Step 2: Use recurrence relations (5.1) to obtain $p_{k,4}$, $q_{k,4}$, $u_{k,4}$ and $v_{k,4}$, and thereby obtain $\dot{\mathfrak{p}}_{k,4}$, $\dot{\mathfrak{q}}_{k,4}$, $\dot{\mathfrak{u}}_{k,4}$, and $\dot{\mathfrak{v}}_{k,4}$. It is tedious, but easy to verify that the recurrence relations give

$$\dot{\mathfrak{p}}_{k,4} = \begin{bmatrix} \zeta_k\,\zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} \\ 4\,\zeta_k\,\zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} - \zeta_{k+2}\,\zeta_{k+3} - \zeta_k\,\zeta_{k+3} - \zeta_k\,\zeta_{k+1} \\ 6\,\zeta_k\,\zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} - 2\,\zeta_{k+2}\,\zeta_{k+3} - 2\,\zeta_k\,\zeta_{k+3} - 2\,\zeta_k\,\zeta_{k+1} + 1 \\ 4\,\zeta_k\,\zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} - \zeta_{k+2}\,\zeta_{k+3} - \zeta_k\,\zeta_{k+3} - \zeta_k\,\zeta_{k+1} \end{bmatrix},$$

$$\dot{\mathfrak{q}}_{k,4} = \begin{bmatrix} -\zeta_k\,\zeta_{k+1}\,\zeta_{k+2} \\ -3\,\zeta_k\,\zeta_{k+1}\,\zeta_{k+2} + \zeta_{k+2} + \zeta_k \\ -3\,\zeta_k\,\zeta_{k+1}\,\zeta_{k+2} + \zeta_{k+2} + \zeta_k \\ -\zeta_k\,\zeta_{k+1}\,\zeta_{k+2} \end{bmatrix},$$

$$\dot{\mathfrak{u}}_{k,4} = \begin{bmatrix} \zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} \\ 3\,\zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} - \zeta_{k+3} - \zeta_{k+1} \\ 3\,\zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} - \zeta_{k+3} - \zeta_{k+1} \\ \zeta_{k+1}\,\zeta_{k+2}\,\zeta_{k+3} \end{bmatrix},$$

and

$$\dot{\mathfrak{v}}_{k,4} = \begin{bmatrix} 0 \\ -\zeta_{k+1}\,\zeta_{k+2} \\ -2\,\zeta_{k+1}\,\zeta_{k+2} + 1 \\ -\zeta_{k+1}\,\zeta_{k+2} \end{bmatrix}.$$

These symmetric vectors can be computed directly from the formulas above in far fewer operations than required by the recurrence relations, especially if certain products are temporarily stored rather than recomputed. $[13\,\mu_{\mathbb{R}} + 10\,\alpha_{\mathbb{R}}]$

Step 3: Use matrix multiplication to directly compute the DFT of each symmetric vector above. $[4\,\mu_{\mathbb{R}} + 10\,\alpha_{\mathbb{R}}]$

Step 4: Increment the counter, k, by four after each initialization: $k = k + 4$

The total numbers of arithmetic operations required by the initialization stage are

$$23\,\mu_{\mathbb{R}} + 32\,\alpha_{\mathbb{R}} + 4\,\delta_{\mathbb{R}}.$$

Therefore 59 flops are required for the four-step initialization, i.e., $\Psi(4) = 59$.

Main loop

After the four-step initialization stage, the main loop of the algorithm proceeds with $m = 4, 8, 16, \ldots, 2^{N-1}$.

Step 0: For $m \geq 4$, assume that the DFTs of $\dot{\mathfrak{u}}_{0,m}$, $\dot{\mathfrak{v}}_{0,m}$, $\dot{\mathfrak{p}}_{0,m}$, and $\dot{\mathfrak{q}}_{0,m}$ have been computed. $[\Psi(m)]$

Step 1: Compute the $\mathfrak{h}_{m-1}^{(m)}$ and $\mathfrak{h}_{m}^{(m)}$ from $\mathfrak{h}_{-1}^{(2m)}$ and $\mathfrak{h}_{0}^{(2m)}$.

a.) Obtain the symmetric vectors $\dot{\mathfrak{u}}_{0,m}$, $\dot{\mathfrak{v}}_{0,m}$, $\dot{\mathfrak{p}}_{0,m}$, and $\dot{\mathfrak{q}}_{0,m}$ by computing the inverse transforms of the DFTs in Step 0. This requires 2 RDCTBs and 2 RQETBs of length m. $[2\psi_5(m) + 2\psi_7(m)]$

b.) Zero pad each symmetric to length $2m$ and compute the DFTs. (See Sections 6.2.2 and 6.2.3.) $[(6m-14)\,\mu_{\mathbb{R}} + (6m-12)\,\alpha_{\mathbb{R}} + 2\psi_1(\frac{m}{4}) + 2\psi_4(\frac{m}{2}) + 2\psi_6(\frac{m}{2})]$

c.) Obtain the DFTs of the zero-padded $\mathfrak{p}_{0,m}$ and $\mathfrak{q}_{0,m}$ from those of $\dot{\mathfrak{p}}_{0,m}$ and $\dot{\mathfrak{q}}_{0,m}$. (See Section 6.2.4.) $[(\frac{m}{2}+1)\,\alpha_{\mathbb{R}}]$

d.) Use expressions (6.9) and (6.10) to obtain $\mathfrak{h}_{m-1}^{(m)}$ and $\mathfrak{h}_m^{(m)}$. (See Sections 6.3.1 and 6.3.2.) $[(12m-6)\,\mu_{\mathbb{R}} + (8m-6)\,\alpha_{\mathbb{R}} + 2\psi_2(2m) + 2\psi_3(2m)]$

Step 2: Using $h_{m-1}^{(m)}$ and $h_m^{(m)}$ as a new set of initial split Schur functions, compute the DFTs of $\dot{\mathfrak{u}}_{m,m}$, $\dot{\mathfrak{v}}_{m,m}$, $\dot{\mathfrak{p}}_{m,m}$, and $\dot{\mathfrak{q}}_{m,m}$ from $\mathfrak{h}_{m-1}^{(m)}$ and $\mathfrak{h}_m^{(m)}$ just as $\dot{\mathfrak{u}}_{0,m}$, $\dot{\mathfrak{v}}_{0,m}$, $\dot{\mathfrak{p}}_{0,m}$, and $\dot{\mathfrak{q}}_{0,m}$ were computed from $\mathfrak{h}_{-1}^{(m)}$ and $\mathfrak{h}_0^{(m)}$. $[\Psi(m)]$

Step 3: Compute the DFTs of $\dot{\mathfrak{u}}_{0,2m}$, $\dot{\mathfrak{v}}_{0,2m}$, $\dot{\mathfrak{p}}_{0,2m}$, and $\dot{\mathfrak{q}}_{0,2m}$.

a.) Obtain the symmetric vectors $\dot{\mathfrak{u}}_{m,m}$, $\dot{\mathfrak{v}}_{m,m}$, $\dot{\mathfrak{p}}_{m,m}$, and $\dot{\mathfrak{q}}_{m,m}$ by computing the inverse transforms of the DFTs in Step 2. This requires 2 RDCTBs and 2 RQETBs of length m. $[2\psi_5(m) + 2\psi_7(m)]$

b.) Zero pad each symmetric to length $2m$ and compute the DFTs. (See Sections 6.2.2 and 6.2.3.) $[(6m-14)\,\mu_{\mathbb{R}} + (6m-12)\,\alpha_{\mathbb{R}} + 2\psi_1(\frac{m}{4}) + 2\psi_4(\frac{m}{2}) + 2\psi_6(\frac{m}{2})]$

c.) Obtain the DFTs of the zero-padded $\mathfrak{p}_{m,m}$ and $\mathfrak{q}_{m,m}$ from those of $\dot{\mathfrak{p}}_{m,m}$ and $\dot{\mathfrak{q}}_{m,m}$. (See Section 6.2.4.) $[(\frac{m}{2}+1)\,\alpha_{\mathbb{R}}]$

d.) Use the techniques outlined in Section 6.4 to compute $\Omega_{2m}\dot{\mathfrak{u}}_{0,2m}$, $\Omega_{2m}\dot{\mathfrak{v}}_{0,2m}$, $\Omega_{2m}\dot{\mathfrak{p}}_{0,2m}$, and $\Omega_{2m}\dot{\mathfrak{q}}_{0,2m}$. $[(\frac{29m}{2}-2)\,\mu_{\mathbb{R}} + (9m-3)\,\alpha_{\mathbb{R}}]$

The total numbers of arithmetic operations required by one step of the main loop are

$$2\Psi(m) + 4\psi_1\left(\frac{m}{4}\right) + 2\psi_2(2m) + 2\psi_3(2m) + 4\psi_4\left(\frac{m}{2}\right) + 4\psi_5(m)$$

$$+4\psi_6\left(\frac{m}{2}\right) + 4\psi_7(m) + \left(\frac{77m}{2} - 36\right)\mu_{\mathbb{R}} + (30m - 31)\,\alpha_{\mathbb{R}}.$$

These counts reduce to

$$2\Psi(m) + (8m\,\log_2 m + 26m - 4)\,\mu_{\mathbb{R}} + \left(24m\,\log_2 m + \frac{19m}{2} + 9\right)\alpha_{\mathbb{R}} + (8m + 4)\,\delta_{\mathbb{R}}$$

$$= 2\Psi(m) + 32m\,\log_2 m + \frac{87m}{2} + 9.$$

After traversing the main loop, the $(0, 2m)$-symmetric polynomials have been computed from the $(0, m)$- and (m, m)-polynomials. Therefore

$$\Psi(2m) = 2\Psi(m) + 32m\,\log_2 m + \frac{87m}{2} + 9.$$

It follows from Lemma 6.1 and the initial condition $\Psi(4) = 59$ that

$$\Psi(m) = 8m\,(\log_2 m)^2 + \frac{55m}{4}\,\log_2 m - \frac{115m}{4} - 9.$$

The operation count given above is for the computations of $\Omega_m\dot{\mathfrak{u}}_{0,m}$, $\Omega_m\dot{\mathfrak{v}}_{0,m}$, $\Omega_m\dot{\mathfrak{p}}_{0,m}$, and $\Omega_m\dot{\mathfrak{q}}_{0,m}$ and the qC-parameters $\zeta_0, \zeta_1, \ldots, \zeta_{m-1}$. When the main loop terminates, these have been computed for $m = 2^N$.

At this point, $\dot{\mathfrak{p}}_{0,m}$ and $\dot{\mathfrak{q}}_{0,m}$ must be obtained from inverse transforms (1 RDCTB and 1 RQEBT) at a cost of

$$\psi_5(m) + \psi_7(m) = \left(\frac{m}{2}\log m + \frac{3m}{4} - 2\right)\mu_{\mathbb{R}} + \left(\frac{3m}{2} - 2\right)\alpha_{\mathbb{R}} + (m+1)\,\delta_{\mathbb{R}}$$

or $(2m\,\log_2 m - 7m/4 - 3)$ flops. Finally, several more operations are required to recover the sequence of Schur parameters, the final Szegő polynomial, and the final prediction error. The recovery of these quantities is described in Algorithm 5.3. In a real-valued implementation of Algorithm 5.3, ρ_{m-1}, δ_{m-1}, and $\gamma_1, \gamma_2, \ldots, \gamma_{m-1}$ can be obtained from $p_{0,m}$ $q_{0,m}$, and $\zeta_0, \zeta_1, \ldots, \zeta_{m-1}$ in

$$3m\,\mu_{\mathbb{R}} + (4m-3)\,\alpha_{\mathbb{R}} + 3m\,\delta_{\mathbb{R}} = 10m - 3\,\text{flops}.$$

(Here the polynomial division by $z-1$ is counted as $m-1$ additions via synthetic division.)

Notice that $u_{0,m}$ and $v_{0,m}$ are not required for the computation of the Szegő polynomial ρ_{m-1}. Thus, in the last iteration of the main loop of Algorithm 5.1, the computations of $\Omega_m \dot{\mathfrak{u}}_{0,m}$ and $\Omega_m \dot{\mathfrak{v}}_{0,m}$ can be skipped, saving a total of $11m/2 - 4$ operations (see Sections 6.4.1 and 6.4.2).

The observations and flop counts above lead to the following result.

Theorem 6.1. *Suppose n is a power of two with $n > 4$ and M_n is a symmetric positive definite Toeplitz matrix. Using the divide-and-conquer Split Schur algorithm, the computation*

of the corresponding Schur parameters, final Szegő polynomial, and final prediction error requires at most

$$8n\,(\log_2 n)^2 + 15.75n\,\log_2 n - 26n - 11$$

real arithmetic operations.

6.6 A Faster Implementation

The flop count of the superfast implementation described above can be significantly reduced by taking advantage of the determinant formula (5.6):

$$v_{k,m}(z)p_{k,m}(z) - u_{k,m}(z)q_{k,m}(z) = z^m.$$

This equation defines $p_{k,m}$ in terms of the other symmetric polynomials via the formula

$$p_{k,m}(z) = \frac{z^m + u_{k,m}(z)q_{k,m}(z)}{v_{k,m}(z)}.$$

Written in terms of the modified symmetric polynomials, the formula takes the form

$$\dot{p}_{k,m}(z) + p_{k,m}(0)z^m = \frac{z^m + \dot{u}_{k,m}(z)z\dot{q}_{k,m}(z)}{\dot{v}_{k,m}(z)}. \tag{6.12}$$

The polynomial division required by (6.12) is equivalent to the solution of a circulant system or a vector deconvolution (see Section 2.1.2). Thus, the computation can be carried out

in the transform domain using only element-by-element division. In practice, this provides the significant advantage of eliminating two length-m FFTs and four length-$m/2$ FFTs per step through the algorithm's main loop, while only adding the $O(m)$ flops associated with the use (6.12) in the transform domain. Of course, any one of the symmetric polynomials could be computed from the other three. In practice, however, there is a benefit to dividing by $\dot{v}_{k,m}$—as described in Section 6.2.3, the DFT of the zero-padded coefficient vector of $\dot{v}_{k,m}$ has purely real even-indexed elements and purely imaginary odd-indexed elements.

In order to derive a flop count for the determinant formula, first notice that the formula can be rearranged,transformed, and written in terms of convolutions to give the following:

$$\Omega_{2m}\begin{bmatrix}\dot{\mathfrak{v}}_{k,m}\\ 0_m\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\mathfrak{p}_{k,m}\\ 0_{m-1}\end{bmatrix} = \begin{bmatrix}1\\ -1\\ \vdots\\ 1\\ -1\end{bmatrix} + \Omega_{2m}\begin{bmatrix}\dot{\mathfrak{u}}_{k,m}\\ 0_m\end{bmatrix} \star \Omega_{2m}\begin{bmatrix}\mathfrak{q}_{k,m}\\ 0_{m-1}\end{bmatrix}.$$

The DFT of the zero-padded $\mathfrak{p}_{k,m}$ has purely real even-indexed elements and complex odd-indexed elements whose real parts are constant. It can be obtained by first computing the right-hand side of the equation above and then performing deconvolution with element-wise division. Because the DFT is conjugate symmetric, only elements 0 through m must be computed.

Assuming that the DFTs of the zero-padded symmetric vectors have been computed, the following steps describe the computation of the the transform of the zero-padded $\mathfrak{p}_{k,m}$.

1. Compute the 0-indexed element. Because the 0-indexed elements of the DFTs are real, this requires one real multiplication, one real addition by 1, and one real division. $[1\,\mu_{\mathbb{R}} + 1\,\alpha_{\mathbb{R}} + 1\,\delta_{\mathbb{R}}]$

2. Compute the m-indexed element. For $m > 2$, this is simply the reciprocal of the m-indexed element of the DFT of the zero-padded $\dot{\mathfrak{v}}_{k,m}$. $[1\,\delta_{\mathbb{R}}]$

3. Compute the elements indexed $2, 4, \ldots, m-2$. These are purely real, and the computation of each requires the real part of a complex multiplication, a real addition by 1, and a real division. $[(m-2)\,\mu_{\mathbb{R}} + (m-2)\,\alpha_{\mathbb{R}} + \frac{1}{2}(m-2)\,\delta_{\mathbb{R}}]$

4. Compute the real parts of the elements indexed $1, 3, \ldots, m-1$. Because the real parts of the odd-indexed elements have the same (constant) value, these can be computed, once and for all, from the imaginary part of a single complex multiplication, followed by a purely imaginary division. $[2\,\mu_{\mathbb{R}} + 1\,\alpha_{\mathbb{R}} + 1\,\delta_{\mathbb{R}}]$

5. Compute the imaginary parts of the elements indexed $1, 3, \ldots, m-1$. The computation of each requires the real part of a complex multiplication, a real addition by 1, and a purely imaginary division. $[m\,\mu_{\mathbb{R}} + m\,\alpha_{\mathbb{R}} + \frac{m}{2}\,\delta_{\mathbb{R}}]$

The numbers of arithmetic operations required by this procedure are

$$(2m+1)\,\mu_{\mathbb{R}} + 2m\,\alpha_{\mathbb{R}} + (m+2)\,\delta_{\mathbb{R}}.$$

Thus, the transform of the zero-padded $\mathfrak{p}_{k,m}$ can be obtained from the transforms of the other vectors in $5m+3$ operations. By reversing the procedure described in Section 6.2.4, an additional $m/2+1$ real subtractions are required to obtain the transform of the zero-padded $\dot{\mathfrak{p}}_{k,m}$.

At this point, the faster DCSSA implementation can be described and its operation count derived.

Initialization

The four-step initialization stage proceeds exactly as described in Section 6.5 with the exception that $\dot{\mathfrak{p}}_{k,4}$ and its transform need not be computed. The total numbers of arithmetic operations required by the new, streamlined initialization stage are

$$15\,\mu_{\mathbb{R}} + 23\,\alpha_{\mathbb{R}} + 4\,\delta_{\mathbb{R}}.$$

Therefore 42 flops are required for the initialization, i.e., $\Psi(4) = 42$.

Main loop

After the four-step initialization stage, the main loop of the algorithm proceeds with $m = 4, 8, 16, \ldots, 2^{N-1}$.

Step 0: For $m \geq 4$, assume that the DFTs of $\dot{\mathfrak{u}}_{0,m}$, $\dot{\mathfrak{v}}_{0,m}$, and $\dot{\mathfrak{q}}_{0,m}$ have been computed.

$[\Psi(m)]$

Step 1: Compute the $\mathfrak{h}_{m-1}^{(m)}$ and $\mathfrak{h}_{m}^{(m)}$ from $\mathfrak{h}_{-1}^{(2m)}$ and $\mathfrak{h}_{0}^{(2m)}$.

a.) Obtain the symmetric vectors $\dot{\mathbf{u}}_{0,m}$, $\dot{\mathbf{v}}_{0,m}$, and $\dot{\mathbf{q}}_{0,m}$ by computing the inverse transforms of the DFTs in Step 0. This requires 1 RDCTB and 2 RQETBs of length m. $[\psi_5(m) + 2\psi_7(m)]$

b.) Zero pad each symmetric vector to length $2m$ and compute the DFTs. (See Sections 6.2.2 and 6.2.3.) $[(5m-11)\,\mu_{\mathbb{R}} + (5m-8)\,\alpha_{\mathbb{R}} + 2\psi_1(\frac{m}{4}) + \psi_4(\frac{m}{2}) + \psi_6(\frac{m}{2})]$

c.) Obtain the DFT of the zero-padded $\mathbf{q}_{0,m}$ from that of $\dot{\mathbf{q}}_{0,m}$. (See Section 6.2.4.) No work is required.

d.) Use the determinant formula in the transform domain to compute the DFT of the zero-padded $\mathbf{p}_{0,m}$. $[(2m+1)\,\mu_{\mathbb{R}} + 2m\,\alpha_{\mathbb{R}} + (m+2)\,\delta_{\mathbb{R}}]$

e.) Obtain the DFT of the zero-padded $\dot{\mathbf{p}}_{0,m}$ from that of $\mathbf{p}_{0,m}$. (See Section 6.2.4.) $[(\frac{m}{2}+1)\,\alpha_{\mathbb{R}}]$

f.) Use expressions (6.9) and (6.10) to obtain $\mathfrak{h}_{m-1}^{(m)}$ and $\mathfrak{h}_{m}^{(m)}$. (See Sections 6.3.1 and 6.3.2.) $[(12m-6)\,\mu_{\mathbb{R}} + (8m-6)\,\alpha_{\mathbb{R}} + 2\psi_2(2m) + 2\psi_3(2m)]$

Step 2: Using $h_{m-1}^{(m)}$ and $h_{m}^{(m)}$ as a new set of initial split Schur functions, compute the DFTs of $\dot{\mathbf{u}}_{m,m}$, $\dot{\mathbf{v}}_{m,m}$, and $\dot{\mathbf{q}}_{m,m}$ from $\mathfrak{h}_{m-1}^{(m)}$ and $\mathfrak{h}_{m}^{(m)}$ just as $\dot{\mathbf{u}}_{0,m}$, $\dot{\mathbf{v}}_{0,m}$, and $\dot{\mathbf{q}}_{0,m}$ were computed from $\mathfrak{h}_{-1}^{(m)}$ and $\mathfrak{h}_{0}^{(m)}$. $[\Psi(m)]$

Step 3: Compute the DFTs of $\dot{\mathbf{u}}_{0,2m}$, $\dot{\mathbf{v}}_{0,2m}$, and $\dot{\mathbf{q}}_{0,2m}$.

a.) Obtain the symmetric vectors $\dot{\mathbf{u}}_{m,m}$, $\dot{\mathbf{v}}_{m,m}$, and $\dot{\mathbf{q}}_{m,m}$ by computing the inverse transforms of the DFTs in Step 2. This requires 1 RDCTB and 2 RQETBs of length m. $[\psi_5(m) + 2\psi_7(m)]$

b.) Zero pad each symmetric vector to length $2m$ and compute the DFTs. (See Sections 6.2.2 and 6.2.3.) $[(5m-11)\,\mu_{\mathbb{R}}+(5m-8)\,\alpha_{\mathbb{R}}+2\psi_1(\frac{m}{4})+\psi_4(\frac{m}{2})+\psi_6(\frac{m}{2})]$

c.) Obtain the DFT of the zero-padded $\mathfrak{q}_{m,m}$ from that of $\dot{\mathfrak{q}}_{m,m}$. (See Section 6.2.4.) No work is required.

d.) Use the determinant formula in the transform domain to compute the DFT of the zero-padded $\mathfrak{p}_{m,m}$. $[(2m+1)\,\mu_{\mathbb{R}}+2m\,\alpha_{\mathbb{R}}+(m+2)\,\delta_{\mathbb{R}}]$

e.) Obtain the DFT of the zero-padded $\dot{\mathfrak{p}}_{m,m}$ from that of $\mathfrak{p}_{m,m}$. (See Section 6.2.4.) $[(\frac{m}{2}+1)\,\alpha_{\mathbb{R}}]$

f.) Use the techniques outlined in Section 6.4 to compute $\Omega_{2m}\dot{\mathfrak{u}}_{0,2m}$, $\Omega_{2m}\dot{\mathfrak{v}}_{0,2m}$, and $\Omega_{2m}\dot{\mathfrak{q}}_{0,2m}$. $[(11m-4)\,\mu_{\mathbb{R}}+(6m-3)\,\alpha_{\mathbb{R}}]$

The total numbers of arithmetic operations required by one step of the main loop are

$$2\Psi(m)+4\psi_1\left(\frac{m}{4}\right)+2\psi_2(2m)+2\psi_3(2m)+2\psi_4\left(\frac{m}{2}\right)+2\psi_5(m)$$

$$+2\psi_6\left(\frac{m}{2}\right)+4\psi_7(m)+(37m-30)\,\mu_{\mathbb{R}}+(29m-23)\,\alpha_{\mathbb{R}}+(2m+4)\,\delta_{\mathbb{R}}.$$

These counts reduce to

$$2\Psi(m)+(7m\,\log_2 m+\frac{105m}{4}-8)\,\mu_{\mathbb{R}}+\left(21m\,\log_2 m+\frac{41m}{4}+7\right)\,\alpha_{\mathbb{R}}+(9m+6)\,\delta_{\mathbb{R}}$$

$$=2\Psi(m)+28m\,\log_2 m+\frac{91m}{2}+5.$$

After one step through the main loop, the $(0, 2m)$-symmetric polynomials have been computed. Therefore

$$\Psi(2m) = 2\Psi(m) + 28m \log_2 m + \frac{91m}{2} + 5.$$

It follows from Lemma 6.1 and the initial condition $\Psi(4) = 42$ that

$$\Psi(m) = 7m \left(\log_2 m\right)^2 + \frac{63}{4} m \log_2 m - \frac{201m}{4} + 5.$$

The operation count given above is for the computations of $\Omega_m \dot{\mathbf{u}}_{0,m}$, $\Omega_m \dot{\mathbf{v}}_{0,m}$, and $\Omega_m \dot{\mathbf{q}}_{0,m}$, and the qC-parameters $\zeta_0, \zeta_1, \ldots, \zeta_{m-1}$. When the main loop terminates, these have been computed for $m = 2^N$. However, the remaining vector, $\Omega_m \dot{\mathbf{p}}_{0,m}$ (with $m = 2^N$), has not yet been computed. It can be obtained by using the techniques described in Section 6.4.3 with an operation count of

$$\left(\frac{7m}{4} + 2\right) \mu_{\mathbb{R}} + \frac{3m}{2} \alpha_{\mathbb{R}}.$$

From this point, the implementation mirrors that given in Section 6.5. Inverse transforms, Schur parameters, and the final Szegő polynomial must be computed at a total cost of $2m \log_2 m + 33m/4 - 6$ flops, while the computations of $\Omega_m \dot{\mathbf{u}}_{0,m}$ and $\Omega_m \dot{\mathbf{v}}_{0,m}$ can be eliminated at a savings of $11m/2 - 4$ flops. These observations and flop counts lead to the following result.

Theorem 6.2. *Suppose n is a power of two with $n > 4$ and M_n is a symmetric positive definite Toeplitz matrix. Using the divide-and-conquer Split Schur algorithm with the determinant formula, the computation of the corresponding Schur parameters, final Szegő polynomial, and final prediction error requires at most*

$$7n\,(\log_2 n)^2 + 17.75n\,\log_2 n - 44.25n + 5$$

real arithmetic operations.

A final caveat is in order. Although the determinant formula uniquely defines one polynomial in terms of the others, the element-by-element division associated with its use in the transform domain can be problematic. At any stage of the computation, if any one of the elements of the DFT of the zero-padded $\dot{\mathfrak{v}}_{k,m}$ is zero or numerically small, the division could be catastrophic. A practical implementation of the algorithm should guard against this failure. As $\dot{\mathfrak{v}}_{k,m}$ is computed, zero-padded, and transformed, the magnitude of its elements should be compared against a lower threshold. If an unacceptably small element is found, the determinant formula should not be used at that particular step of the process. In this case, a work-around should be developed to directly compute $\dot{\mathfrak{p}}_{k,m}$. The resulting hybrid algorithm would have a flop count somewhere between those given by Theorems 6.1 and 6.2.

CHAPTER 7

NUMERICAL TESTS

In the previous chapter, a superfast implementation of the divide-and-conquer split Schur algorithm (DCSSA) was presented. The original implementation (version A) requires approximately $8n\,(\log_2 n)^2$ flops, while the modified version (version B) makes use of the determinant formula (5.6) to reduce the flop count to approximately $7n\,(\log_2 n)^2$. These implementations have been coded in the C++ programming language[1]. The coded versions use the double precision, `double`, data type, in which floating-point numbers are stored in 8 bytes and the machine epsilon is approximately 2.22×10^{-16}. The results of several numerical experiments are presented in this chapter.

Numerical experiments involving four different test matrices are described below. Each test matrix is a real symmetric positive definite Toeplitz matrix whose order is a power of two. In each test, the accuracy of the algorithm is measured in a number of different ways. First, the Yule-Walker residual is computed. If $\tilde{\rho}_n(z) = \sum_{j=0}^{n} \tilde{r}_{j,n} z^j$ denotes the computed

[1] A slower, non-FFT, implementation was also coded and tested in the computer algebra system *Maxima*.

nth Szegő polynomial associated with the $(n+1) \times (n+1)$ Toeplitz matrix M_{n+1}, then this residual vector is given by

$$\text{Resid} = M_n \begin{bmatrix} \tilde{r}_{0,n} \\ \tilde{r}_{1,n} \\ \vdots \\ \tilde{r}_{n-1,n} \end{bmatrix} + \begin{bmatrix} m_n \\ m_{n-1} \\ \vdots \\ m_1 \end{bmatrix}.$$

Also computed in each test are the errors in the computed Szegő polynomials, Schur parameters, split Levinson polynomials, and qC-parameters. In some cases, the computed values are compared to known exact values. In other cases, "exact" values are obtained from other algorithms using the extended precision, `long double`, data type (floating point numbers are stored in 16 bytes and the machine epsilon is approximately 1.08×10^{-19}, compliant with IEEE 754). In these cases, "exact" Szegő polynomials and Schur parameters are obtained from an extended-precision implementation of the Levinson-Durbin algorithm (Algorithm 2.1). The "exact" split Levinson polynomials are obtained via formulas (3.10) and (5.3) from the Szegő polynomials. Finally, the "exact" qC-parameters are obtained from an extended precision implementation of the split Schur algorithm (SSA) (Algorithm 4.2). All "computed" values in all tests are obtained from the double-precision implementations, version A or version B, of the DCSSA (Algorithm 5.3).

The accuracy tests use the vector 1-norm to measure the magnitude of the residual and error vectors. Recall that for $x = [x_1, x_2, \ldots, x_n]^T$, the 1-norm is given by

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|.$$

In the tables that follow, the norms of the Yule-Walker residual, the errors in the coefficients of the final Szegő and split Levinson polynomials, and the errors in the Schur and qC-parameters are displayed. The tilde is used to denote a computed value. All numerical tests were conducted on an ASUS CM6850 Series PC with an Intel Core i5–2320 CPU. C++ code was compiled with the 64-bit TDM-GCC compiler (version 4.8.1).

7.1 Test Matrix 1

In the first set of tests, the test matrix is a random, strictly diagonally dominant, symmetric Toeplitz matrix of order $K = 2^n$ with 1's along the main diagonal. The elements of the first column are given by

$$m_j = \begin{cases} 1, & j = 0 \\ \text{rand}(-\frac{0.999}{K-1}, \frac{0.999}{K-1}), & j = 1, 2, \ldots, K-1, \end{cases}$$

where $\text{rand}(a, b)$ denotes a random number from the uniform distribution on (a, b).

It is well known that strictly diagonally dominant, symmetric matrices with positive diagonal elements are positive definite [38]. It is also likely that these matrices are well conditioned—the off-diagonal elements are such that the deleted row sums should be approximately 1/2 and subsequently, Geršgorin discs would put the eigenvalues between 1/2 and 3/2, resulting in a 2-norm condition number of less than 3.

The first test used only version A (without the determinant formula) of the DCSSA. The results are summarized in Table 7.1. For the sake of comparison, the split Levinson algorithm (SLA) (Algorithm 3.1) was also tested. Some corresponding errors in the double-precision SLA are summarized in Table 7.2.

Table 7.1: Errors in DCSSA (version A) associated with test matrix 1

Order, K	$\|\text{Resid}\|$	$\|\tilde{\gamma} - \gamma\|$	$\|\tilde{\rho}_K - \rho_K\|$	$\|\tilde{p}_{K+1} - p_{K+1}\|$	$\|\tilde{\zeta} - \zeta\|$
8	5.76e-15	1.34e-15	7.18e-15	1.42e-15	1.67e-15
16	4.43e-14	6.80e-15	4.50e-14	4.11e-14	5.00e-15
32	9.99e-14	1.80e-14	1.07e-13	3.71e-14	1.32e-14
64	3.96e-12	6.12e-14	4.17e-12	9.68e-13	1.45e-13
128	1.22e-11	2.67e-13	7.38e-11	1.22e-11	2.55e-13
256	5.43e-11	1.42e-12	2.12e-10	5.43e-11	7.92e-13
512	1.63e-10	4.60e-12	9.12e-10	1.63e-10	1.49e-11
1024	4.76e-10	1.86e-11	6.22e-09	4.76e-10	4.90e-11
2048	1.82e-08	1.04e-10	4.77e-08	1.82e-08	1.89e-10
4096	3.90e-07	1.55e-09	3.36e-07	3.90e-07	6.24e-10
8192	8.81e-07	5.47e-09	3.78e-06	8.81e-07	1.63e-08

Table 7.2: Errors in SLA associated with test matrix 1

Order, K	$\|\tilde{p}_{K+1} - p_{K+1}\|$	$\|\tilde{\zeta} - \zeta\|$
8	1.85e-15	1.44e-15
16	2.42e-15	5.66e-15
32	7.84e-15	1.47e-14
64	4.93e-14	8.43e-14
128	8.33e-14	5.77e-13
256	6.17e-13	2.96e-12
512	1.26e-12	9.66e-12
1024	1.53e-12	2.40e-11
2048	4.81e-12	2.77e-10
4096	2.89e-11	1.12e-09
8192	3.74e-11	7.39e-09

7.2 Test Matrix 2

The matrices for test 2 are obtained from the Fourier coefficients of the positive, even function $f(x) = x^2 + 1$ on $[-\pi, \pi]$. Specifically, the elements of the first column of the test matrix M_K are given by

$$m_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\, dx = \frac{\pi^2 + 3}{3},$$

$$m_j = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) \cos(jx)\, dx = \frac{2(-1)^j}{j^2}, \quad j = 1, 2, \ldots, K-1.$$

It is known that the eigenvalues of such matrices are bounded by the minimum and maximum values of f [35]. Thus, for any K, M_K is well conditioned, with its 2-norm condition number not exceeding $\pi^2 + 1$.

The results from DCSSA version A are summarized in Table 7.3. Those from DCSSA version B (with determinant formula) are summarized in Table 7.4. Recall that the use of

the determinant formula in version B requires division by the elements of the DFT of the zero-padded $\dot{\mathfrak{v}}_{k,m}$. The minimum divisor encountered in this division is listed in Table 7.4.

Table 7.3: Errors in DCSSA (version A) associated with test matrix 2

Order, K	$\|\text{Resid}\|$	$\|\tilde{\gamma} - \gamma\|$	$\|\tilde{\rho}_K - \rho_K\|$	$\|\tilde{p}_{K+1} - p_{K+1}\|$	$\|\tilde{\zeta} - \zeta\|$
8	6.61e-15	8.21e-16	3.80e-15	1.72e-15	2.75e-15
16	9.56e-14	2.72e-15	7.99e-14	1.20e-14	9.05e-15
32	1.70e-12	7.89e-15	1.56e-12	1.35e-13	4.03e-14
64	1.51e-11	7.09e-14	1.45e-11	4.57e-12	1.86e-13
128	2.00e-11	1.93e-13	1.98e-11	2.65e-11	7.26e-13
256	3.72e-10	1.12e-12	3.68e-10	8.90e-11	2.10e-12
512	7.47e-10	8.41e-12	7.44e-10	4.29e-10	1.85e-11
1024	1.11e-08	1.39e-11	1.11e-08	2.24e-09	1.69e-10
2048	5.19e-08	1.54e-10	5.18e-08	1.08e-08	5.06e-10
4096	1.62e-06	8.74e-10	1.62e-06	1.09e-07	2.94e-09
8192	1.23e-05	2.69e-10	1.22e-06	6.03e-06	1.93e-08

Table 7.4: Errors in DCSSA (version B) associated with test matrix 2

Order, K	$\|\text{Resid}\|$	$\|\tilde{\gamma} - \gamma\|$	$\|\tilde{\rho}_K - \rho_K\|$	$\|\tilde{p}_{K+1} - p_{K+1}\|$	$\|\tilde{\zeta} - \zeta\|$	Min divisor
8	1.53e-14	8.01e-16	1.09e-14	1.85e-15	3.64e-15	0.422
16	6.07e-14	6.07e-15	5.29e-14	4.28e-14	1.92e-14	0.163
32	3.63e-13	2.13e-14	3.25e-13	2.68e-13	3.21e-13	0.101
64	5.62e-12	3.51e-13	5.62e-12	5.25e-13	2.51e-11	0.0501
128	2.07e-10	1.06e-11	2.09e-11	1.80e-11	1.75e-09	0.0254
256	3.38e-08	1.87e-09	3.16e-09	2.20e-09	6.48e-07	0.0127
512	5.96e-06	3.29e-07	5.53e-07	3.76e-07	2.29e-04	0.00635

7.3 Test Matrix 3

The elements of the first column of matrix 3 are given by

$$m_j = \theta^{j^2}, \quad j = 0, 1, \ldots, K-1,$$

where $\theta \in (-1, 1)$. These test matrices are numerically banded, with the band width determined by the choice of θ and the machine precision. The exact forms of the Schur parameters and the Szegő polynomials are known [34]. In fact, the Schur parameters satisfy

$$\gamma_j = (-\theta)^j, \quad j = 1, 2, \ldots, K-1.$$

It follows that the matrices are positive definite, and by Cybenko's inequality (2.18), they are fairly well conditioned for small θ. Because the exact values of the Schur parameters are known, the exact values of the qC-parameters can be computed recursively from formulas (4.5). Beginning with $\zeta_0 = 1$ and $\lambda_0 = \infty$,

$$\lambda_{j+1} = 2\zeta_j - \frac{1}{\lambda_j}, \quad \zeta_{j+1} = \frac{1}{\lambda_{j+1}(1 - (-\theta)^{j+1})}; \quad j = 0, 1, \ldots, K-2.$$

Tests were performed with $\theta = -0.5$. Estimates computed with Scilab (version 5.3.3) indicate that the tested matrices have condition numbers less than 18. The results from DCSSA version A are summarized in Table 7.5. At the most basic level, the split algorithms

can be thought of as methods for computing the qC-parameters. After all, it is this set of parameters that gives rise to the split Levinson polynomials and the Schur parameters. A direct comparison of the DCSSA, SLA, and SSA is given in Table 7.6. This table gives the errors in the qC-parameters, compared to their exact values, when the different split algorithms are applied in double precision. As can be seen, the accuracy of DCSSA version B deteriorates rapidly.

Table 7.5: Errors in DCSSA (version A) associated with test matrix 3 with $\theta = -0.5$

Order, K	$\|\text{Resid}\|$	$\|\tilde{\gamma} - \gamma\|$	$\|\tilde{\rho}_K - \rho_K\|$	$\|\tilde{p}_{K+1} - p_{K+1}\|$	$\|\tilde{\zeta} - \zeta\|$
8	2.04e-15	1.67e-15	6.45e-15	7.24e-15	9.99e-16
16	7.22e-15	1.42e-14	1.73e-14	5.15e-14	2.11e-15
32	3.86e-14	2.89e-14	2.49e-13	1.31e-13	1.21e-14
64	2.47e-13	1.09e-13	1.85e-12	7.49e-13	3.39e-14
128	8.77e-13	6.31e-13	6.88e-12	3.95e-12	1.20e-13
256	7.97e-12	1.71e-12	6.44e-11	1.69e-11	2.65e-12
512	2.27e-10	7.21e-12	1.85e-09	2.17e-10	1.08e-11
1024	2.46e-09	6.26e-11	2.02e-08	8.15e-09	3.39e-11
2048	3.98e-08	1.86e-10	3.27e-07	2.94e-08	9.92e-11
4096	2.07e-07	6.05e-10	1.71e-06	7.42e-07	3.64e-10
8192	1.46e-07	2.49e-09	1.21e-06	1.94e-06	1.54e-09

Table 7.6: Errors in qC-parameters, $\|\tilde{\zeta} - \zeta\|$, from DCSSA (versions A and B), SLA, and SSA (Test matrix 3 with $\theta = -0.5$)

Order, K	DCSSA(A)	DCSSA(B)	SLA	SSA
8	1.20e-15	4.34e-16	1.14e-15	2.92e-16
16	2.28e-15	8.05e-15	4.21e-15	7.01e-16
32	1.21e-14	4.72e-13	9.53e-15	4.58e-15
64	3.38e-14	1.65e-11	6.10e-14	1.69e-14
128	1.19e-13	2.52e-09	2.08e-13	4.64e-14
256	2.65e-12	5.18e-07	5.01e-13	1.05e-13
512	1.08e-11	2.16e-04	1.09e-12	2.23e-13
1024	3.39e-11	1.82e-01	2.26e-12	4.59e-13
2048	9.93e-11	****	4.61e-12	9.32e-13
4096	3.64e-10	****	9.31e-12	1.88e-12
8192	1.54e-09	****	1.87e-11	3.76e-12

7.4 Test Matrix 4

The final test matrices have 2's along the main diagonal and 1's elsewhere. Their first columns have the form

$$[2, 1, 1, \ldots, 1]^T.$$

These matrices have some very nice properties as summarized in the following proposition.

Proposition 7.1. *Let M_n be the symmetric Toeplitz matrix whose 1st column is the n-dimensional vector $[2, 1, 1, \ldots, 1]^T$.*

i.) The eigenvalues of M_n are $n + 1$, with multiplicity 1, and 1, with multiplicity $n - 1$. Consequently, the 2-norm condition number is $n + 1$.

ii.) The split Levinson polynomials and qC-parameters are given by

$$p_k(x) = \frac{(-1)^{k-1}+3}{4(k+1)}\left[kx^k - \sum_{j=1}^{k-1} 2x^j + k\right], \quad k = 1, 2, \ldots, n,$$

$$\zeta_k = \frac{2^{(-1)^k}(k+1)^2}{k(k+2)}, \quad k = 1, 2, \ldots, n-1.$$

iii.) The Szegő polynomials, prediction errors, and Schur parameters are given by

$$\rho_k(x) = x^k - \sum_{j=0}^{k-1} \frac{x^j}{k+1}, \quad k = 0, 1, \ldots, n-1,$$

$$\delta_k = \frac{k+2}{k+1}, \quad k = 0, 1, \ldots, n-1,$$

$$\gamma_k = -\frac{1}{k+1}, \quad k = 1, 2, \ldots, n-1.$$

Proof. For part (i), notice that M_n is a circulant matrix. Therefore its eigenvalues can be obtained from the DFT of its first column. Direct computation using (2.1a) gives

$$\Omega_n \begin{bmatrix} 2 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} n+1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

The condition number is the ratio of greatest to least eigenvalues.

Part (ii) is straightforward to establish by induction using Algorithm 3.1. It helps to establish along the way that

$$\nu_k = \frac{(-1)^{k-1}+3}{4}\left(\frac{k+2}{k+1}\right).$$

Part (iii) also follows readily by induction using formulas (2.10). □

As is clear from the proposition, these matrices are poorly conditioned for large n. Nonetheless, the knowledge of their exact polynomials and parameters makes them convenient for testing purposes.

The results from DCSSA version A are summarized in Table 7.7. Results in this table are based on comparison with exact values, rather than extended precision computed values. Table 7.8 gives the errors in the qC-parameters, compared to their exact values, when the different split algorithms are applied in double precision. Once again, the accuracy of DCSSA version B deteriorates rapidly.

Table 7.7: Errors in DCSSA (version A) associated with test matrix 4

Order, K	$\|\text{Resid}\|$	$\|\tilde{\gamma} - \gamma\|$	$\|\tilde{\rho}_K - \rho_K\|$	$\|\tilde{p}_{K+1} - p_{K+1}\|$	$\|\tilde{\zeta} - \zeta\|$
8	1.04e-14	7.91e-15	8.65e-15	8.33e-16	1.89e-15
16	1.23e-13	5.69e-14	8.07e-14	2.37e-15	6.22e-15
32	9.64e-13	4.47e-13	6.28e-13	4.86e-14	8.06e-14
64	4.44e-12	3.02e-12	4.15e-12	1.91e-13	7.11e-13
128	8.62e-11	1.78e-11	2.82e-11	2.03e-12	5.36e-12
256	3.71e-10	1.88e-10	2.33e-10	1.59e-11	1.37e-11
512	6.59e-09	1.01e-09	1.31e-09	1.90e-10	3.18e-10
1024	4.67e-08	2.45e-09	3.27e-09	3.16e-09	1.14e-08
2048	4.72e-07	1.94e-08	2.82e-08	1.69e-08	1.52e-07
4096	5.68e-06	5.69e-07	7.95e-07	2.37e-07	1.65e-06
8192	6.31e-06	5.01e-06	6.31e-06	1.16e-06	5.38e-05

Table 7.8: Errors in qC-parameters, $\|\tilde{\zeta} - \zeta\|$, from DCSSA (versions A and B), SLA, and SSA (Test matrix 4)

Order, K	DCSSA(A)	DCSSA(B)	SLA	SSA
8	1.89e-15	3.44e-15	3.11e-15	2.11e-15
16	6.22e-15	1.27e-14	1.69e-14	5.00e-15
32	8.06e-14	2.11e-13	5.84e-14	5.88e-15
64	7.11e-13	1.79e-12	1.50e-13	1.82e-14
128	5.36e-12	6.98e-11	3.25e-13	5.47e-14
256	1.37e-11	4.06e-09	3.05e-12	9.38e-14
512	3.18e-10	1.73e-06	6.54e-12	5.78e-13
1024	1.14e-08	****	3.42e-11	2.11e-12
2048	1.52e-07	****	1.21e-10	4.59e-12
4096	1.65e-06	****	9.34e-10	6.17e-12
8192	5.38e-05	****	3.32e-09	1.58e-11

The DCCSA version A was also tested with matrix 4 using the computer algebra system *Maxima* (version 5.25.1). This tested implementation is a slower implementation with computational complexity $O(n^2)$, which uses direct polynomial multiplication rather than FFT techniques. This test was carried out for two reasons: to assess the impact of the FFT techniques on accuracy and to experiment with different levels of precision. The results corresponding to matrix 4 with order 1024 are shown in Table 7.9.

Table 7.9: Errors in *Maxima* DCSSA (version A) associated with test matrix 4 of order 1024

d-digit arithmetic	$\|\text{Resid}\|$	$\|\tilde{p}_{K+1} - p_{K+1}\|$	$\|\tilde{\zeta} - \zeta\|$
$d = 10$	2.08e-02	3.93e-05	1.35e-06
$d = 15$	3.07e-07	3.57e-09	3.54e-11
$d = 20$	2.59e-12	3.57e-14	1.12e-16
$d = 25$	2.00e-17	2.43e-19	1.32e-21

7.5 Discussion of Test Results

It is clear from Tables 7.4, 7.6, and 7.8 that the DCSSA version B (the implementation making use of the determinant formula) performs very poorly as the matrix order increases. Of course, division by small quantities is an ever-present concern in the version B implementation. While the minimum divisor was monitored in one of the tests (Table 7.4), no efforts were made to restrict or work around the division. A hybrid algorithm, such as described at

the end of Section 6.6, was not implemented. Based on the numerical tests, it appears that the direct implementation of the determinant formula in version B is problematic.

Focusing on the DCSSA version A, all of the test results show a consistent steady decrease in accuracy as the order of the matrix increases. For the most part, it appears that these decreases in accuracy cannot be explained by the matrix conditioning. Matrices 1 and 2, for instance, have small, bounded condition numbers, while the condition number of matrix 4 increases linearly with order. Nonetheless, the accuracy results for these matrices are very similar.

Nearly all tests of version A show a loss of about three digits of accuracy per ten-fold increase in the order of the matrix. This is not inconsistent with the split Levinson algorithm residual bound, inequality (3.18), given by Krishna and Wang [45]. However, as shown in Tables 7.2, 7.6, and 7.8, the split Levinson and split Schur algorithms tend to outperform the DCSSA.

The decrease in accuracy does not appear to be related to the FFT-based polynomial multiplication. This conclusion is drawn from Table 7.9, where the results associated with slow, standard polynomial multiplication are given. With the test matrix of order 1024, the results in this table, corresponding to $d = 15$, are similar to those in the appropriate row of Table 7.7.

Table 7.9 shows that the DCSSA performs with increased accuracy when the algorithm is implemented within increased precision. This, of course, is expected of any numerically reliable algorithm. It does not indicate the stability of the DCSSA, but it does indicate

that the algorithm can produce accurate results. In general, the numerical stability of the algorithm remains questionable.

CHAPTER 8

SUMMARY AND CONCLUSIONS

This dissertation has been concerned with the direct numerical solution of Hermitian positive definite Toeplitz systems. The contributions of this work are summarized in this chapter. In addition, a number of related research problems are proposed for future investigation.

8.1 Contributions

The most significant contributions of the research described in this dissertation are the development and implementation of a new superfast Toeplitz solver. The divide-and-conquer split Schur algorithm (DCSSA) is a doubling procedure applied to a continued fraction representation of a quasi-Carathéodory function. When implemented with FFT techniques, the new algorithm requires approximately $8n\,(\log_2 n)^2$ arithmetic operations to compute the final Szegő polynomial and all Schur parameters associated with an $n \times n$ real symmetric positive definite Toeplitz matrix. By taking advantage of a redundancy present in some of the intermediate quantities, the operation count can be further reduced to $7n\,(\log_2 n)^2$. This appears to be the smallest operation count of all known direct Toeplitz solvers.

The implementation of the DCSSA requires a number of new or relatively unknown results concerning discrete Fourier transforms (DFTs) of symmetric vectors. In particular, the descriptions of the symmetries in the transforms of zero-padded real quarter-even and real even vectors do not appear to be well known. Consequently, the techniques described for exploiting the symmetry of the original vector in the computation of the DFT after padding with zeros are new. These results are contained in Propositions 6.3–6.7.

While the split Schur algorithm has been thoroughly described in the literature, the derivation given in Chapter 4 is unique. This derivation is based on splitting Schur functions into their numerators and denominators, forming linear combinations, called split Schur functions, and then processing the split Schur functions. Ratios of the split Schur functions are called quasi-Carathéodory (qC-) functions. Viewed in light of these functions, the split Schur algorithm is a recursive procedure for generating qC-functions, just as Schur's classical algorithm generates Schur functions. In this context, the qC-parameters defined in Chapter 4 are the analogs of the Schur parameters.

8.2 Future Research

While the current research has closed some doors, it has opened many others. The following problems are left for the future.

1. Competitiveness and stability of the algorithm

The numerical stability of the algorithm remains in question. While the weak stability of the DCSSA version A is not ruled out by numerical tests, the rate at which its accuracy decreases is disturbing. The accuracy comparisons with the split Levinson and split Schur algorithms, described in Chapter 7, indicate that the DCSSA falls rather short. Similarly, the numerical tests of Ammar and Gragg's superfast generalized Schur algorithm [8] suggest that in its current form, the DCSSA is not competitive with other algorithms. Future work is required to assess the stability of the algorithm and optimize its performance.

2. Effective use of the determinant formula

It is clear from the numerical tests of the DCSSA version B that serious problems arise with the naive use of the determinant formula (5.6) to accelerate the algorithm. Questions abound as to how to apply the formula effectively. For instance,

a.) Which symmetric polynomial should be solved for?

b.) Could division by zero be encountered? If so, can effective work-arounds be implemented?

c.) What is the best way to implement the polynomial division required by the use of the determinant formula?

d.) Are there other (better) ways to take advantage of the redundancies implied by the determinant formula?

In general, a deeper understanding of the relationships and redundancies among the split Levinson polynomials is required. (Some work along these lines has been accomplished by Delsarte and Genin [26, 27, 28].)

3. Properties of qC-functions and qC-parameters

Schur's classical algorithm [51], described in Section 2.2.4, provides a constructive proof of the Carathéodory-Toeplitz theorem [1]. The split Schur algorithm provides a similar proof. However, the properties and applications of the qC-functions and parameters that arise in the split Schur algorithm are not well studied. The qC-parameters and Jacobi parameters characterize the positive definite nature of a Toeplitz matrix. They can also be used to compute Cybenko's bounds on the condition number of a Toeplitz matrix. Are there other useful properties and applications of the qC-functions and parameters?

4. Further properties of the DFTs of zero-padded symmetric vectors

Many applications call for the zero padding of vectors before their discrete Fourier transforms are computed [14]. There are well-known techniques for exploiting symmetry in the computation of DFTs. However, after vectors are padded with zeros, their original symmetry is destroyed. There appears to be little in the literature concerning the exploitation of symmetry in zero-padding. A thorough investigation of this issue, and the subsequent development of state-of-the-art routines, could be very useful.

REFERENCES

[1] N. I. Akhiezer, *The Classical Moment Problem and Some Related Questions in Analysis*, Oliver and Boyd Ltd., 1965.

[2] G. S. Ammar, *Classical foundations of algorithms for solving positive definite toeplitz equations*, Calcolo, 33 (1996), pp. 99–113.

[3] G. S. Ammar and P. Gader, *New decompositions of the inverse of a toeplitz matrix*, in Signal Processing, Scattering, and Operator Theory, and Numerical Methods, M. A. Kaashoek, J. H. van Schuppen, and A. C. M. Ran, eds., Proceedings of the International Symposium MTNS-89, Vol. III, Birkhaüser, 1990, pp. 421–428.

[4] ——, *A variant of the Gohberg-Semencul formula involving circulant matrices*, SIAM J. Matrix Anal. Appl., 12 (1991), pp. 534–540.

[5] G. S. Ammar and W. B. Gragg, *Implementation and use of the generalized Schur algorithm*, in Computational and Combinatorial Methods in Systems Theory, C. I. Byrnes and A. Lindquist, eds., Amsterdam, 1986, North-Holland, pp. 265–279.

[6] ——, *The generalized Schur algorithm for the superfast solution of Toeplitz systems*, in Rational Approximation and its Applications in Mathematics and Physics, J. Gilewicz, M. Pindor, and W. Siemaszko, eds., no. 1237 in Lecture Notes in Mathematics, Berlin, 1987, Springer-Verlag, pp. 315–330.

[7] ——, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 61–76.

[8] ——, *Numerical experience with a superfast real Toeplitz solver*, Lin. Alg. Appl., 121 (1989), pp. 185–206.

[9] E. H. Bareiss, *Numerical solution of linear equations with Toeplitz and vector Toeplitz matrices*, Numer. Math., 13 (1969), pp. 404–424.

[10] R. R. Bitmead and B. D. O. Anderson, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Lin. Alg. Appl., 34 (1980), pp. 103–116.

[11] A. W. Bojanczyk, R. P. Brent, F. R. de Hoog, and D. R. Sweet, *On the stability of the bareiss and related toeplitz factorization algorithms*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 40–57.

[12] R. P. Brent, *Parallel algorithms and numerical stability for Toeplitz solvers*. Presented at the SIAM Conference on Linear Algebra in Signals, Systems, and Control, Seattle (August 1993).

[13] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun, *Fast solution of Toeplitz systems of equations and computation of Padé approximants*, J. Algorithms, 1 (1980), pp. 259–295.

[14] W. L. Briggs and V. E. Henson, *The DFT: An Owner's Manual for the Discrete Fourier Transform*, SIAM, 1995.

[15] A. Bultheel and P. Dewilde, *On the relation between Pade approximation algorithms and Levinson/Schur recursive methods*, in Signal Processing: Theories and Applications, M. Kunt and F. de Coulon, eds., North-Holland, 1980, pp. 517–523.

[16] J. R. Bunch, *The weak and strong stability of algorithms in numerical linear algebra*, Lin. Alg. Appl., 88/89 (1987), pp. 49–66.

[17] R. E. Caflisch, *An inverse problem for Toeplitz matrices and the synthesis of discrete transmission lines*, Lin. Alg. Appl., 38 (1981), pp. 207–225.

[18] R. H. Chan and M. K. Ng, *Conjugate gradient methods for Toeplitz systems*, tech. report, Department of Mathematics, The Chinese University of Hong Kong, October 1994.

[19] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu, *A superfast algorithm for Toeplitz systems of linear equations*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1247–1266.

[20] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.

[21] G. Cybenko, *The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations*, SIAM J. Sci. Stat. Comput., 1 (1980), pp. 303–319.

[22] F. de Hoog, *A new algorithm for solving Toeplitz systems of equations*, Lin. Alg. Appl., 88/89 (1987), pp. 123–138.

[23] P. Delsarte and Y. Genin, *The split Levinson algorithm*, IEEE Trans. Acoust. Speech Signal Process., ASSP-34 (1986), pp. 470–478.

[24] ——, *On splitting the classical algorithms in linear prediction theory*, IEEE Trans. Acoust. Speech Signal Process., ASSP-35 (1987), pp. 645–653.

[25] ——, *A survey of the split approach based techniques in digital signal processing applications*, Philips J. Res., 43 (1988), pp. 346–374.

[26] ——, *The tridiagonal approach to Szegő's orthogonal polynomials, Toeplitz linear systems, and related interpolation problems*, SIAM J. Math. Anal., 19 (1988), pp. 718–735.

[27] ——, *Tridiagonal approach to the algebraic environment of Toeplitz matrices, Part I: Basic results*, SIAM J. Matrix Anal. Appl., 12 (1991), pp. 220–238.

[28] ——, *Tridiagonal approach to the algebraic environment of Toeplitz matrices, Part II: Zero and eigenvalue problems*, SIAM J. Matrix Anal. Appl., 12 (1991), pp. 432–448.

[29] B. W. Dickinson, *An inverse problem for Toeplitz matrices*, Lin. Alg. Appl., 59 (1984), pp. 79–83.

[30] P. Duhamel, *Implementation of "split-radix" FFT algorithms for complex, real, and real-symmetric data*, IEEE Trans. Acoust. Speech Signal Process., ASSP-34 (1986), pp. 285–295.

[31] P. Duhamel and M. Vetterli, *Fast Fourier transforms: A tutorial review and a state of the art*, Signal Processing, 19 (1990), pp. 259–299.

[32] J. Durbin, *The fitting of time-series models*, Rev. Int. Stat. Inst., 28 (1960), pp. 233–244.

[33] I. C. Gohberg and A. A. Semencul, *On the inversion of finite toeplitz matrices and their continuous analogs* (in Russian), Mat. Issled., 7 (1972), pp. 201–223.

[34] 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.

[35] U. Grenander and G. Szegő, *Toeplitz Forms and Their Applications*, Chelsea Publishing Co., 2nd ed., 1984.

[36] R. A. Haddad and T. W. Parsons, *Digital Signal Processing: Theory, Applications, and Hardware*, Computer Science Press, 1991.

[37] M. T. Heideman, D. H. Johnson, and C. S. Burrus, *Gauss and the history of the fast Fourier transform*, IEEE ASSP Magazine, 1 (1984), pp. 14–21.

[38] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[39] T. Huckle, *Computations with Gohberg-Semencul-type formulas for Toeplitz matrices*, Lin. Alg. Appl., 273 (1998), pp. 169–198.

[40] J. R. Jain, *An efficient algorithm for a large Toeplitz set of linear equations*, IEEE Trans. Acoust. Speech Signal Process., 27 (1979), pp. 612–615.

[41] S. G. Johnson and M. Frigo, *A modified split-radix FFT with reduced arithmetic complexity*, IEEE Trans. Signal Process., 55 (2007), pp. 111–119.

[42] S. M. Kay and S. L. Marple, Jr., *Spectrum analysis—A modern perspective*, Proceedings of the IEEE, 69 (1981), pp. 1380–1419.

[43] I. Kra and S. R. Simanca, *On circulant matrices*, Notices Amer. Math. Soc., 59 (2012), pp. 368–377.

[44] H. Krishna and S. D. Morgera, *The Levinson recurrence and fast algorithms for solving Toeplitz systems of linear equations*, IEEE Trans. Acoust. Speech Signal Process., ASSP-36 (1987), pp. 839–848.

[45] H. Krishna and Y. Wang, *The split Levinson algorithm is weakly stable*, SIAM J. Numer. Anal., 30 (1993), pp. 1498–1508.

[46] N. Levinson, *The Wiener RMS (root mean square) error criterion in filter design and prediction*, J. Math. Phys., 25 (1947), pp. 261–278.

[47] J. Makhoul, *Linear prediction: A tutorial review*, Proceedings of the IEEE, 63 (1975), pp. 561–580.

[48] B. R. Musicus, *Levinson and fast Cholesky algorithms for Toeplitz and almost Toeplitz matrices*, tech. report, Research Lab. of Electronics, M. I. T., 1984.

[49] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, Inc., 1978.

[50] F. Scherbaum, *Seismic imaging of the site response using microearthquake recordings. Part I. Method*, Bull. Seism.Soc. Am., 77 (1987), pp. 1905–1923.

[51] I. Schur, *Uber Potenzreihen, die in Innern des Einheitskrises Beschränkt Sind*, J. Reine Angnew. Math., 147 (1917), pp. 205–232.

[52] M. Shanir P.P., Y. U. Khan, and E. Khan, *Classification of EEG signal for left and right wrist movements using AR modelling*, in Proceedings of Conference on Modern Trends in Electronics and Communication Systems, MTECS 2008, Aligarh Muslim University, 2008, pp. 65–69.

[53] B. Simon, *OPUC on one foot*, Bull. Amer. Math. Soc. (N.S.), 42 (2005), pp. 431–460.

[54] M. Stewart, *A superfast Toeplitz solver with improved numerical stability*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 669–693.

[55] P. N. Swarztrauber, *Symmetric FFTs*, Math. Comp., 47 (1986), pp. 323–346.

[56] D. R. Sweet, *Numerical Methods for Toeplitz Matrices*, PhD thesis, University of Adelaide, 1982.

[57] G. Szegő, *Orthogonal Polynomials*, American Mathematical Society, 4th ed., 1975.

[58] W. F. Trench, *An algorithm for the inversion of finite Toeplitz matrices*, SIAM J. Appl. Math., 12 (1964), pp. 515–522.

[59] M. Van Barel, G. Heinig, and P. Kravanja, *A stabilized superfast solver for nonsymmetric Toeplitz systems*, Tech. Report TW 293, Katholieke Universiteit Leuven, October 1999.

[60] G. Walker, *On periodicity in series of related terms*, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci., 131 (1931), pp. 518–532.

[61] A. E. Yagle, *New analogues of split algorithms for Toeplitz-plus-Hankel matrices*, in IEEE International Conference on Acoustics, Speech, and Signal Processing, Toronto, 1991, pp. 2253–2256.

[62] G. U. Yule, *On a method of investigating periodicities in disturbed series, with special reference to Wolfer's sunspot numbers*, Philos. Trans. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci., 226 (1927), pp. 267–298.

[63] H.-M. Zhang and P. Duhamel, *On the methods for solving Yule-Walker equations*, IEEE Trans. Signal Process., 40 (1992), pp. 2987–3000.

[64] S. Zohar, *Toeplitz matrix inversion: The algorithm of W. F. Trench*, J. ACM, 16 (1969), pp. 592–601.