Rapid Generation of Jacobi Matrices for Measures Modified by Rational Factors

Amber Sumner
RAPID GENERATION OF JACOBI MATRICES FOR MEASURES MODIFIED BY RATIONAL FACTORS

by

Amber C. Sumner

A Dissertation
Submitted to the Graduate School, the College of Science and Technology, and the Department of Mathematics of The University of Southern Mississippi in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

March 2018
RAPID GENERATION OF JACOBI MATRICES FOR MEASURES MODIFIED BY RATIONAL FACTORS

by Amber C. Sumner

March 2018

Approved by:

Dr. James V. Lambers, Committee Chair
Associate Professor, Mathematics

Dr. Jiu Ding, Committee Member
Professor, Mathematics

Dr. Haiyan Tian, Committee Member
Professor, Mathematics

Dr. John Harris, Committee Member
Associate Professor, Mathematics

Dr. Bernd Schroeder
Chair, Department of Mathematics

Dr. Karen S. Coats
Dean of the Graduate School
ABSTRACT

RAPID GENERATION OF JACOBI MATRICES FOR MEASURES MODIFIED BY RATIONAL FACTORS

by Amber C. Sumner

March 2018

Orthogonal polynomials are important throughout the fields of numerical analysis and numerical linear algebra. The Jacobi matrix $J_n$ for a family of $n$ orthogonal polynomials is an $n \times n$ tridiagonal symmetric matrix constructed from the recursion coefficients for the three-term recurrence satisfied by the family. Every family of polynomials orthogonal with respect to a measure $d\lambda(t)$ on a real interval $[a, b]$ satisfies such a recurrence. Given a modified measure $d\tilde{\lambda}(t) = r(t)d\lambda(t)$, where $r(t)$ is a rational function, an important problem is to compute the Jacobi matrix $\tilde{J}_n$ corresponding to $d\tilde{\lambda}(t)$ from knowledge of $J_n$. There already exist efficient methods to accomplish this when $r(t)$ is a polynomial, so we focus on the case where $r(t)$ is the reciprocal of a polynomial. Working over the field of real numbers, this means considering the case where $r(t)$ is the reciprocal of a linear or irreducible quadratic factor, or a product of such factors. Existing methods for this type of modification are computationally expensive. Our goal is to develop a faster method based on inversion of existing procedures for the case where $r(t)$ is a polynomial. The principal challenge in this project is that this inversion requires working around missing information. This can be accomplished by treating this information as unknown parameters and making guesses that can be corrected iteratively.
ACKNOWLEDGMENTS

I would like to first thank my son Race for giving me motivation to accomplish this goal. I would like thank my advisor and committee chair, Dr. James V. Lambers, for all of his guidance, wisdom, reassurance, and late nights of editing my work. Thank you to my committee members Dr. Jiu Ding, Dr. Haiyan Tian, and Dr. John Harris for your time and efforts in agreeing to read and review my dissertation. I would like to thank all of my colleagues at William Carey University, especially my department chair, Dr. Benjamin F. Dribus, for being patient and understanding with me as I complete my dissertation. Thank you to my peers at the University of Southern Mississippi Haley Dozier and Brianna Bingham for always being available to lend an ear (or an office) when I need it. Last but not least, I would like to thank my family for being patient and supportive throughout the years of my pursuing this degree. If it were not for the support of everyone around me, I could not have accomplished this goal. I do not take lightly the support and encouragement that has been shown to me. Thank you.
# TABLE OF CONTENTS

ABSTRACT ................................................................. ii

ACKNOWLEDGMENTS .................................................. iii

LIST OF ILLUSTRATIONS ............................................. vi

LIST OF TABLES ........................................................ vii

LIST OF ABBREVIATIONS .............................................. viii

NOTATION AND GLOSSARY ............................................ viii

1 Introduction ........................................................... 1
  1.1 Problem Statement .............................................. 1
  1.2 Applications ................................................... 1
  1.3 Previous Work .................................................. 1
  1.4 Outline .......................................................... 2

2 Background ............................................................ 3
  2.1 Gaussian Quadrature ............................................ 3
  2.2 Properties of orthogonal polynomials ......................... 7
  2.3 Minimal Solutions .............................................. 11
  2.4 Inverse Cholesky ............................................... 12
  2.5 Jagels’ and Reichel’s work on even powers ................. 14

3 Building Blocks for Algorithms .................................. 17
  3.1 Reverse Cholesky ............................................... 17
  3.2 Reverse QR ..................................................... 19
  3.3 Solving nonlinear equations .................................. 19
  3.4 Safe-guarded Methods ......................................... 21

4 Modification by Linear Factors .................................. 25
  4.1 Multiplication Procedure ...................................... 25
  4.2 Reversal Procedure ............................................ 27
  4.3 Derivation of the objective function for guessing ....... 28
  4.4 Interval for guessing and quartic equation ............... 29
  4.5 Algorithm description ........................................ 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Modification by Irreducible Quadratic Factors</td>
<td>31</td>
</tr>
<tr>
<td>5.1</td>
<td>Explicit Double Shift</td>
<td>31</td>
</tr>
<tr>
<td>5.2</td>
<td>Implicit Double Shift</td>
<td>34</td>
</tr>
<tr>
<td>6</td>
<td>NUMERICAL EXPERIMENTS</td>
<td>42</td>
</tr>
<tr>
<td>6.1</td>
<td>Linear Factors</td>
<td>43</td>
</tr>
<tr>
<td>6.2</td>
<td>Irreducible Quadratic Factors</td>
<td>58</td>
</tr>
<tr>
<td>6.3</td>
<td>Linear and Quadratic Factors with no Iteration</td>
<td>68</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>BIBLIOGRAPHY</td>
<td>73</td>
</tr>
</tbody>
</table>
# LIST OF ILLUSTRATIONS

**Figure**

6.1 Polynomials corresponding to $J_n$. .................................................. 44  
6.2 Polynomials corresponding to $\tilde{J}_n$. ............................................. 45  
6.3 Polynomials corresponding to $J_n$ on $[(c - 1), b]$. .......................... 45  
6.4 Polynomials corresponding to $\tilde{J}_n$ on $[(c - 1), b]$. ....................... 46  
6.5 Objective function $f$. ................................................................. 47  
6.6 Objective function $f$. ................................................................. 48  
6.7 Objective function $f$. ................................................................. 49  
6.8 Polynomials corresponding to $J_n$. .................................................. 50  
6.9 Polynomials corresponding to $\tilde{J}_n$. ............................................. 50  
6.10 Polynomials corresponding to $J_n$ on $[(c - 1), b]$. .......................... 51  
6.11 Polynomials corresponding to $\tilde{J}_n$ on $[(c - 1), b]$. ....................... 51  
6.12 Polynomials corresponding to $J_n$. .................................................. 54  
6.13 Polynomials corresponding to $\tilde{J}_n$. ............................................. 54  
6.14 Polynomials corresponding to $J_n$ on $[(c - 1), b]$. .......................... 55  
6.15 Polynomials corresponding to $\tilde{J}_n$. ............................................. 55  
6.16 Objective function $f$. ................................................................. 56  
6.17 Objective function $f$. ................................................................. 56  
6.18 Objective function $f$. ................................................................. 57  
6.19 Polynomials corresponding to $J_n$. .................................................. 58  
6.20 Polynomials corresponding to $\tilde{J}_n$. ............................................. 60  
6.21 Objective function $f$ for $c = -2i$. ................................................... 61  
6.22 Objective function $f$ for $c = -1.5i$. .............................................. 62  
6.23 Objective function $f$ for $c = -1.25i$ ............................................. 62  
6.24 Objective function $f$ for $c = -1.1i$ .............................................. 63  
6.25 Polynomials corresponding to $J_n$. .................................................. 64  
6.26 Polynomials corresponding to $\tilde{J}_n$. ............................................. 64  
6.27 Objective function $f$ for $c = -2i$. ................................................... 65  
6.28 Objective function $f$ for $c = -3i$. ................................................... 65  
6.29 Objective function $f$ for $c = -4i$ ................................................... 66  
6.30 Objective function $f$ for $c = -5i$ ................................................... 66  
6.31 Objective function $f$ for $c = -5.5i$. .............................................. 67  
6.32 Objective function $f$ for $c = -5.75i$. .............................................. 67
LIST OF TABLES

Table

6.1 Example 2 Errors ....................................................... 52
6.2 Example 2 Graphs .......................................................... 53
6.3 Example 4 Errors .......................................................... 58
6.4 Example 4 Graphs .......................................................... 59
6.5 Example 1 Errors .......................................................... 60
6.6 Example 2 Errors .......................................................... 63
6.7 Example 1 Errors .......................................................... 69
6.8 Example 2 ................................................................. 70
6.9 Example 1 Errors .......................................................... 70
6.10 Example 2 Errors .......................................................... 71
6.11 Example 3 ................................................................. 71
NOTATION AND GLOSSARY

General Usage and Terminology

The notation used in this text represents fairly standard mathematical and computational usage. In many cases these fields tend to use different preferred notation to indicate the same concept, and these have been reconciled to the extent possible, given the interdisciplinary nature of the material. The capital letters, $A, B, \ldots$ are used to denote matrices. Functions which are denoted in boldface type typically represent vector valued functions. Lower case letters such as $i, j, k, l, m, n$ and sometimes $p$ and $d$ are used to denote indices. Vectors and matrices are typeset in square brackets, e.g., $[\cdot]$. In general the norms are typeset using double pairs of lines, e.g., $||\cdot||$, and the absolute value of numbers is denoted using a single pairs of lines, e.g., $|\cdot|$. 
Chapter 1

Introduction

1.1 Problem Statement

The focus of this dissertation is on methods for modification of a Jacobi matrix when the underlying measure is modified by a rational weight function. We design and implement an algorithm that produces the Jacobi matrix $\tilde{J}_n$ corresponding to the measure $d\tilde{\lambda}(t) = r(t) \, d\lambda(t)$, where $r(t)$ is a rational function, given the Jacobi matrix $J_n$ corresponding to $d\lambda(t)$.

1.2 Applications

One reason we wish to obtain such a matrix is to obtain Gaussian quadrature rules for approximating integrals using rational approximations of integrands rather than customary polynomials. Integrands of particular interest are rapidly decaying exponentials, which arise in time-stepping methods for systems of ordinary differential equations in which larger time steps are used. Another application would be in the use of Generalized Jacobi Polynomials (GJPs) [9, 18]. Recently developed polynomials $\{\phi_j\}$ orthogonal with respect to $\omega \equiv 1$ on the interval $(-1, 1)$ satisfying the boundary conditions $p(1) = 0$ and $p(-1) = p(1) = 0$ can be modified to obtain the three-term recurrence for the GJPs [15]. The modified Jacobi matrix $\tilde{J}_n$ is the Jacobi matrix for the GJPs that satisfy the same boundary conditions, and can be found by modification by dividing by one or more linear factors.

1.3 Previous Work

The existing methods for this type of modification are computationally expensive. For example, the Inverse Cholesky (IC) algorithm, due to Elhay and Kautsky [2], solves for the modified Jacobi matrix $\tilde{J}_n$ exactly, but this comes at the cost of finding the factorization $(J_n - c I)^{-1} = LL^T$ for a linear factor, which requires $O(n^3)$ floating point operations. Another potential problem with the IC method is that it produces a less accurate modified Jacobi matrix when a pole of the rational function is far from the interval $[a, b]$ based on numerical experiments performed by the authors.
Another existing method is *Minimal Solution* Method, due to Gautschi [4]. This method is also computationally expensive because it requires working with a larger Jacobi matrix than what is required, and the expense increases dramatically when a pole is close to \([a, b]\). There has been work done for this problem specifically if the linear divisor is raised to an even power

\[
d\tilde{\lambda}(t) = (t - c)^{-2l}d\lambda(t),
\]

by Jagels and Reichel in [14] however this limits us to only a small subset of the problems we wish to solve.

### 1.4 Outline

The necessary background information is presented in Chapter 2, including discussion of notation, properties of orthogonal polynomials, as well as other methods for modification of a Jacobi matrix. These methods include Minimal Solutions from [4], Inverse Cholesky from [2], and Jagels’ and Reichel’s work on even powers [14]. Chapter 2 also describes known methods for modification of a Jacobi matrix where the modification is a linear divisor, and irreducible quadratic divisor. From these cases, one can perform modifications by a rational weight function as well. Chapter 3 presents the tools necessary to build the algorithms that we develop such as reverse Cholesky (not to be confused with Inverse Cholesky discussed in Chapter 2), reverse QR, and nonlinear equation solvers. Chapter 4 begins with an introduction to the method that will be reversed later in this dissertation, i.e. modification by multiplying by linear factors [4]. The later part of this chapter discusses one of the new methods of this dissertation, the reversal procedure for the linear case. Chapter 5 presents the reversal procedure for the irreducible quadratic case. Finally, numerical results and conclusions are given in Chapters 6 and 7, respectively.
Chapter 2

Background

2.1 Gaussian Quadrature

The bilinear form
\[ \langle f, g \rangle_\lambda = \int_a^b f(t)g(t) \, d\lambda(t) \]  
(2.1)
defines an inner product on the vector space \( P \) of polynomials \( p(t) \) on the interval \([a,b]\) with respect to some measure \( d\lambda(t) \). The integral can be approximated using Gaussian quadrature rules
\[ \int_a^b f(t) \, d\lambda(t) \approx \sum_{i=1}^n w_i f(t_i), \]  
(2.2)
where \( w_i \) are the weights and \( t_i \) are the nodes. These rules are able to evaluate the integral of a polynomial exactly up to degree \( 2n - 1 \) using only \( n \) nodes [11]. To maximize efficiency and accuracy, the number of nodes must be minimized. Therefore, having to only calculate \( n \) nodes to get accuracy up to degree \( 2n - 1 \) is preferable to other quadrature rules. Gaussian quadrature rules can be constructed by moment matching, also known as direct construction, where the goal for a given \( n \) is to select nodes and weights so that the first \( 2n \) moments are computed exactly by
\[ \sum_{i=1}^n t_i^k w_i = \int_a^b t^k \, d\lambda(t) = \mu_k \quad \text{for} \quad k = 0, 1, \ldots, 2n - 1. \]  
(2.3)
The drawback to using direct construction is that (2.3) requires one to solve a system of nonlinear equations by a root finding method such as the secant method or Newton’s method, which can be quite difficult.

This gives motivation to calculate the nodes and weights for the quadrature rule in another way. A subset of monic polynomials of \( P \)
\[ P = \{ p_0, p_1, p_2, \ldots, \} \]  
(2.4)
is orthogonal under the measure \( d\lambda(t) \) if
\[ \langle p_k, p_i \rangle_\lambda = 0 \quad \text{for} \ i \neq k \]
and orthonormal if
\[
\langle p_k, p_i \rangle = \begin{cases} 0 & k \neq i \\ 1 & k = i. \end{cases}
\]

The basis elements of \( P \) (2.4) satisfy a three-term recurrence
\[
p_k(t) = (t - \alpha_k)p_{k-1}(t) - \beta_k^2p_{k-2}(t) \quad \text{for } k > 1,
\]
which we will discuss in more detail in the next subsection. This uniquely determines the coefficients
\[
\alpha_k = \frac{\langle tp_{k-1}, p_{k-1} \rangle}{\langle p_{k-1}, p_{k-1} \rangle} \quad \text{for } k \geq 1,
\]
and
\[
\beta_k^2 = \frac{\langle p_k, p_k \rangle}{\langle p_{k-1}, p_{k-1} \rangle} \quad \text{for } k \geq 1,
\]
where we define
\[
p_{-1} \equiv 0, \quad p_0 \equiv 1.
\]

If we define
\[
\beta_0 = \langle p_0, p_0 \rangle,
\]
and define polynomials \( q_k \) to be
\[
q_k(t) = \frac{p_k(t)}{\beta_0 \beta_1 \cdots \beta_k} \quad \text{for } k = 0, 1, \ldots, n,
\]
then the polynomials \( q_0, q_1, \ldots, q_n \) are orthonormal. From the vector space viewpoint, \( t : q(t) \to tq(t) \) is a linear operator on \( P \). The Jacobi matrix \( J_n \) is the matrix representing the operator \( t \) with respect to the basis \( Q = \{ q_0, q_1, \ldots, q_{n-1} \} \). That is,
\[
tq_\ell = \sum_{j=0}^{n-1} (J_n)_{j+1, \ell+1}q_j + \beta_n q_n \quad \text{for } \ell = 0, 1, \ldots, n - 1
\]
for each \( \ell \). Carrying out this matrix vector multiplication yields,
\[
\begin{align*}
tq_0(t) &= \beta_1 q_1(t) + \alpha_1 q_0(t) \\
tq_1(t) &= \beta_2 q_2(t) + \alpha_1 q_1(t) + \beta_1 q_0(t) \\
tq_2(t) &= \beta_3 q_3(t) + \alpha_2 q_2(t) + \beta_2 q_1(t) \\
&\vdots \\
tq_{k-1}(t) &= \beta_k q_k(t) + \alpha_{k-1} q_{k-1}(t) + \beta_{k-1} q_{k-2}(t)
\end{align*}
\]
which, in vector form, becomes
\[ t\mathbf{q}(t) = \begin{bmatrix}
\alpha_0 q_0(t) + \beta_1 q_1(t) \\
\beta_1 q_0(t) + \alpha_1 q_1(t) + \beta_2 q_2(t) \\
\beta_2 q_1(t) + \alpha_2 q_2(t) + \beta_3 q_3(t) \\
\vdots \\
\beta_{n-1} q_{n-2}(t) + \alpha_{n-1} q_{n-1}(t) + \beta_n q_n(t)
\end{bmatrix}. \]

From the above, we can conclude that
\[ t\mathbf{q}(t) = \begin{bmatrix}
\alpha_1 & \beta_1 \\
\beta_1 & \alpha_1 & \beta_2 \\
& \beta_2 & \ddots & \ddots \\
& & \ddots & \ddots & \beta_{n-1} \\
& & & \beta_{n-1} & \alpha_n
\end{bmatrix} \begin{bmatrix}
q_0(t) \\
q_1(t) \\
q_2(t) \\
\vdots \\
q_{n-1}(t)
\end{bmatrix} = J_n \mathbf{q}(t), \]

where \( J_n \), called a Jacobi matrix, is a symmetric tridiagonal matrix of the form
\[ J_n = \begin{bmatrix}
\alpha_1 & \beta_1 \\
\beta_1 & \alpha_1 & \beta_2 \\
& \beta_2 & \ddots & \ddots \\
& & \ddots & \ddots & \beta_{n-1} \\
& & & \beta_{n-1} & \alpha_n
\end{bmatrix}. \quad (2.6) \]

The roots of \( q_n \) give the nodes for the quadrature rule (2.3) and the eigenvectors of \( J_n \) give the weights [8]. Define a polynomial \( g(t) \) of at most degree \( 2n - 1 \), therefore \( g(t) \in \mathcal{P}_{2n-1} \). Let \( p_{n-1}(t) \) be the interpolant polynomial of degree \( n - 1 \) of \( g(t) \), where the interpolation points are the roots for \( q_n(t) \). We can now represent the interpolation error \( e(t) \) as
\[ e(t) = g(t) - p_{n-1}(t). \]

It can be seen that \( e(t) \) has roots at each of the roots of \( q_n(t) \), where \( q_n(t) \) is the \( n^{th} \) polynomial in the family of orthogonal polynomials \( \{q_i(t)\}_{i=0}^n \). Therefore, \( e \) can be factored into the form
\[ e(t) = q_n(t)r(t), \]
where \( r(t) \in \mathcal{P}_{n-1} \). Using the fact that \( q_n(t) \) is orthogonal to any polynomial in the space \( \mathcal{P}_{n-1} \),
\[ \int_a^b g(t) d\lambda(t) = \int_a^b p_{n-1}(t) d\lambda(t) + \int_a^b q_n(t)r(t) d\lambda(t) = \int_a^b p_{n-1}(t) d\lambda(t). \]
Therefore, we can use the roots of $q_n(t)$ as the nodes for the quadrature rule. The authors of [8] explain why we need orthogonal polynomials for Gaussian quadrature. For the family of orthonormal polynomials $q_n(t)$, the recurrence relation is as follows:

$$t q_n(t) = J_n q_n(t) + \beta_n q_n(t) e_n.$$ 

If $t$ is the root we are looking for, the last term is eliminated because $q_n(t) = 0$ and we get

$$t q_n(t) = J_n q_n(t),$$

where we now have an eigenvalue problem and $t$, an eigenvalue of $J_n$, gives the nodes we need.

Using the Jacobi matrix to obtain the nodes and weights for the quadrature rule is much more efficient than direct construction, requiring only $O(n)$ operations because the matrix is symmetric and tridiagonal. The method used to find the eigenvalues of a matrix with this structure is the symmetric QR algorithm. In [7] the authors discuss the symmetric QR algorithm in which a general symmetric matrix $A$ is reduced to tridiagonal form. However, in our case, we have the advantage of working with the Jacobi matrix that already has tridiagonal structure.

Using this method to compute the nodes and weights is preferable for a number of different reasons. One of the most important things considered in numerical analysis is computational cost. If $T \in \mathbb{R}^{n \times n}$ is tridiagonal, then its QR factorization can be computed by applying a sequence of $n - 1$ Givens rotations. This requires $O(n)$ floating point operations. Another nice property is that since we are dealing with a symmetric matrix, the left eigenvector $\lambda y^H = y^H A$ is the same as the right eigenvector; therefore, the eigenvalues are perfectly conditioned.

Prescribing $p_0(t) = 1$ allows us to generate the first orthonormal polynomial

$$q_0 = \frac{1}{\beta_0},$$

where

$$\beta_0 = \sqrt{\mu_0},$$

$$\mu_0 = \langle p_0, p_0 \rangle.$$

The first diagonal entry of $J_n$ can be computed

$$\alpha_1 = \langle q_0, tq_0 \rangle = \frac{1}{\mu_0} \langle t, 1 \rangle = \frac{\mu_1}{\mu_0}.$$
by using the moments. Likewise, the off-diagonal entry is given by the following:

$$\beta_1^2 = \frac{\langle (t - \alpha_1), (t - \alpha_1) \rangle}{\beta_0^2}$$

$$= \frac{\int t^2 - 2\alpha_1 t + \alpha_1^2}{\beta_0^2}$$

$$= \frac{\mu_2 - 2\alpha_1 \mu_1 + \alpha_1^2 \mu_0}{\mu_0}$$

$$= \frac{\mu_2 - 2\frac{\mu_1^2}{\mu_0} \mu_1 + \frac{\mu_1^2}{\mu_0} \mu_0}{\mu_0}$$

$$= \frac{\mu_2 - \frac{\mu_1^2}{\mu_0}}{\mu_0}$$

$$= \frac{\mu_0 \mu_2 - \mu_1^2}{\mu_0^2}.$$

Constructing \( \alpha_j \) requires the moments \( \mu_0, \mu_1, \ldots, \mu_{2j-1} \) and \( \beta_j \) requires \( \mu_0, \mu_1, \ldots, \mu_{2j} \). We can also define the moments corresponding to the measure \( d\lambda(t) = (t-c)^{-1}d\tilde{\lambda}(t) \) beginning with

$$\tilde{\mu}_0 = \int_a^b \frac{\tilde{p}_0(t)}{t-c} d\tilde{\lambda}(t) = \langle \tilde{p}_0, 1 \rangle.$$

The \( k^{th} \) moment of modified measure can be computed by

$$\tilde{\mu}_k = \int_a^b t^k d\tilde{\lambda}(t). \quad (2.7)$$

2.2 Properties of orthogonal polynomials

Here we will discuss some properties of orthogonal polynomials. A sequence of polynomials \( p_0(t), p_1(t), \ldots, p_n(t) \) on an interval \([a,b] \) is called orthogonal with respect to an inner product \( \langle p_n, p_m \rangle = \int_a^b p_n(t)p_m(t) \omega(t) dt \) if \( \langle p_n, p_m \rangle = 0 \) for \( m \neq n \). Here, \( \omega(t) \) is called the weight function for the inner product. Such a sequence is often defined to be orthonormal; that is, it is assumed to satisfy the additional property that \( \langle p_m, p_n \rangle = 1 \). The polynomials \( q_k(t) \) defined in the previous section are orthonormal. This can be seen by taking the inner
product of $q_k$ with itself as follows,

$$\langle q_k, q_k \rangle = \frac{\langle p_k, p_k \rangle}{\beta_0 \beta_1 \cdots \beta_k} = \langle p_k, p_k \rangle \beta_0^{-1} \beta_1^{-1} \cdots \beta_k^{-1} = \langle p_k, p_k \rangle \frac{1}{\langle p_{k-1}, p_{k-1} \rangle \langle p_{k-2}, p_{k-2} \rangle \cdots \langle p_0, p_0 \rangle} \langle p_0, p_0 \rangle = 1.$$  

A sequence of orthogonal polynomials may be used to form a basis for the vector space of functions defined on the interval $[a, b]$.

One of the most important properties of orthogonal polynomials for the purposes of this dissertation is that they satisfy a three-term recurrence relation

$$\beta_j q_{j+1}(t) = (t - \alpha_j)q_j(t) - \beta_{j-1}q_{j-1}(t), \quad (2.8)$$

where the recursion coefficients $\beta_j, \alpha_j, \beta_{j-1}$ are defined in terms of inner products of pairs of orthogonal polynomials. These coefficients supply the entries of the $n \times n$ Jacobi matrix

$$J_n = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \cdots & \\ & \cdots & \cdots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{bmatrix} \quad (2.9)$$

of which we wish to obtain a modification. The coefficients can be obtained by forcing orthogonality to eliminate terms in the three-term recurrence relation. First, multiplying (2.8) by $p_j(t)$ and integrating over $[a, b]$ with respect to $d\lambda(t)$ yields

$$q_j(t)\beta_j q_{j+1}(t) = q_j(t)(t - \alpha_j)q_j(t) - q_j(t)\beta_{j-1}q_{j-1}(t)$$

$$q_j(t)\beta_j q_{j+1}(t) = q_j(t)\beta_j(t) - q_j(t)\alpha_j q_j(t) - q_j(t)\beta_{j-1}q_{j-1}(t)$$

$$\langle q_j, \beta_j q_{j+1} \rangle = \langle q_j, tq_j \rangle - \langle q_j, \alpha_j q_j \rangle - \langle q_j, \beta_{j-1}q_{j-1} \rangle$$

$$\beta_j \langle q_j, q_{j+1} \rangle = \langle q_j, tq_j \rangle - \alpha_j \langle q_j, q_j \rangle - \beta_j \langle q_j, q_{j-1} \rangle$$

$$0 = \langle q_j, tq_j \rangle - \alpha_j \langle q_j, q_j \rangle$$

$$\alpha_j \langle q_j, q_j \rangle = \langle q_j, tq_j \rangle$$

$$\alpha_j = \frac{\langle q_j, tq_j \rangle}{\langle q_j, q_j \rangle},$$
then multiplying by \( q_{j-1}(t) \) and integrating over \([a,b]\) with respect to \(d\lambda(t)\) leads to

\[
q_{j-1}(t)\beta_j q_{j+1}(t) = q_{j-1}(t)(t - \alpha_j)q_j(t) - q_{j-1}(t)\beta_j - 1(t)q_{j-1}(t)
\]

\[
q_{j-1}(t)\beta_j q_{j+1}(t) = q_{j-1}(t)tq_j(t) - q_{j-1}(t)\alpha_j q_j(t) - q_{j-1}(t)\beta_j - 1q_{j-1}(t)
\]

\[
\langle q_{j-1}, \beta_j q_{j+1} \rangle = \langle q_{j-1}, tq_j \rangle - \langle q_{j-1}, \alpha_j q_j \rangle - \langle q_{j-1}, \beta_j - 1q_{j-1} \rangle
\]

\[
\beta_j \langle q_{j-1}, q_{j+1} \rangle = \langle q_{j-1}, tq_j \rangle - \alpha_j \langle q_{j-1}, q_j \rangle - \beta_j - 1\langle q_{j-1}, q_{j-1} \rangle
\]

\[
0 = \langle q_{j-1}, tq_j \rangle - \beta_j - 1\langle q_{j-1}, q_{j-1} \rangle
\]

\[
\beta_j - 1\langle q_{j-1}, q_{j-1} \rangle = \langle q_{j-1}, tq_j \rangle
\]

\[
\beta_{j-1} = \frac{\langle q_{j-1}, tq_j \rangle}{\langle q_{j-1}, q_{j-1} \rangle} = \frac{\langle tq_{j-1}, q_j \rangle}{\langle q_{j-1}, q_{j-1} \rangle} = \frac{\langle q_{j}, q_j \rangle}{\langle q_{j-1}, q_{j-1} \rangle} \text{ for } j \geq 1.
\]

The existence of such a three-term recurrence relation is a nontrivial consequence of the fact that the inner product we are using arises from a measure on a real interval. The sequence \( p_0(t), p_1(t), \ldots, p_n(t) \) of orthogonal polynomials is constructed by applying the Gram-Schmidt process to the standard basis \( 1, t, \ldots, t^n \) of the space of polynomials of degree at most \( n \). The recursion coefficients depend in a nontrivial way on the choice of the standard basis as the input data for the Gram-Schmidt process.

**Theorem 2.2.1. Three-term Recurrence Relation**

For monic, orthogonal polynomials, there exist sequences of coefficients \( \alpha_j \) for \( j = 1, 2, \ldots, n \) and \( \beta_j \) for \( j = 1, 2, \ldots, n \) such that

\[
p_{j+1}(t) = (t - \alpha_j)p_j(t) - \beta_j^{2}p_{j-1}(t) \text{ for } j = 1, 2, \ldots, n
\]  

(2.10)

where

\[
p_{-1} \equiv 0 \quad p_0 \equiv 1
\]  

(2.11)

and

\[
\alpha_j = \frac{\langle tq_j, p_j \rangle}{\langle p_j, p_j \rangle} \text{ for } j = 0, 1, \ldots, n
\]

\[
\beta_j = \frac{\langle p_j, p_j \rangle}{\langle p_{j-1}, p_{j-1} \rangle} \text{ for } j = 1, 2, \ldots, n.
\]

**Proof:**

We begin by computing the first two monic, orthogonal polynomials

\[
p_0(t) \equiv 1
\]

and since \( p_1(t) \) is a monic degree one polynomial, then

\[
p_1(t) = t - \alpha_1
\]
for some constant \( \alpha_1 \). By orthogonality,

\[
0 = \langle p_1, p_0 \rangle_{\lambda} = \langle t - \alpha_1, 1 \rangle = \int_a^b d\lambda(t)t - \alpha_1 \int_a^b d\lambda(t),
\]

and since the weight function is positive in the interval of integration, we can solve for \( \alpha_1 \)

\[
\alpha_1 = \frac{\int_a^b d\lambda(t)t}{\int_a^b d\lambda(t)}.
\]

In general, we seek to construct a polynomial of degree \( n + 1 \) that has the form

\[
p_j(t) = tp_{j-1} - \sum_{i=0}^{j-1} c_ip_i(t) \quad \text{for } k = 0, \ldots, j - 1.
\]

Taking the inner product of (2.12) with \( p_k(t) \),

\[
\langle p_j(t), p_k(t) \rangle = \langle tp_{j-1}(t), p_k(t) \rangle - \sum_{i=0}^{j-1} c_i \langle p_i(t), p_k(t) \rangle
= \langle tp_{j-1}(t), p_k(t) \rangle - c_k \langle p_k(t), p_k(t) \rangle
\]

Solving the above for \( c_k \) gives

\[
p_j(t) = tp_{j-1}(t) - \sum_{i=0}^{j-1} \frac{\langle p_i(t), tp_{j-1}(t) \rangle}{\langle p_i(t), p_i(t) \rangle} p_i(t).
\]

Since we have the property that we can swap the \( t \) in the inner product, \( \langle p_i(t), tp_{j-1}(t) \rangle = \langle tp_i(t), p_{j-1}(t) \rangle \) we can conclude that \( \langle p_i(t), tp_{j-1}(t) \rangle = 0 \) when \( i < j - 2 \). This shows that sequences of orthogonal polynomials satisfy a three term recurrence relation

\[
p_j(t) = (t - \alpha_j)p_{j-1}(t) - \beta_j \beta_{j-1} p_{j-2}(t), \quad j > 1
\]

where the recursion coefficients are

\[
\alpha_j = \frac{\langle p_{j-1}, tp_{j-1} \rangle}{\langle p_{j-1}, p_{j-1} \rangle} \quad \text{(2.15)}
\]

and

\[
\beta_j^2 = \frac{\langle p_{j-1}, tp_j \rangle}{\langle p_{j-1}, p_{j-1} \rangle} = \frac{\langle tp_{j-1}, p_j \rangle}{\langle p_{j-1}, p_{j-1} \rangle} = \frac{\langle p_j, p_j \rangle}{\langle p_{j-1}, p_{j-1} \rangle}, \quad j \geq 1.
\]

\[\blacksquare\]
2.3 Minimal Solutions

This method, given by Gautschi in [4] and [5], shows how to use the minimal solution to the three term recurrence for the polynomials orthogonal with respect to the measure \( d\tilde{\lambda}(t) \), where

\[
d\tilde{\lambda}(t) = (t - c)^{-1} d\lambda(t) \quad \text{where} \quad c \notin [a,b] \tag{2.17}
\]

and

\[
d\tilde{\lambda}(t) = [(t + ci)(t + \bar{c}i)]^{-1} d\lambda(t) \tag{2.18}
\]

to compute the modified moments of the measures \( \tilde{\mu}_k = \langle 1, q_k \rangle_{\tilde{\lambda}} \). The following algorithm gives the modified moments: Once the modified moments are computed, then the modified

Algorithm 1 Modified Moments: This algorithm takes as input the Jacobi matrix \( J_n \) of size \( n \times n \), the first moments \( \mu_0 \), a shift \( c \) outside the interval, and an integer \( m = m(n,v) < n \). The result is the vector \( \hat{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})^T \) where the modified moments are given by \( \tilde{\mu}_i = \phi_i \) if \( c \) is real, and \( \tilde{\mu}_i = \frac{\text{imag}(\phi_i)}{\text{imag}(c)} \) if \( c \) is complex.

1: \( p_n = 0 \)
2: \textbf{for} \( i = n : -1 : 2 \) \textbf{do}
3: \( p_{i-1} = \beta_{i-1} / (c - \alpha_i - \beta_i p_i) \)
4: \textbf{end for}
5: \( \phi_0 = \sqrt{\mu_0} / (\beta_1 p_1 - (c - \alpha_1)) \)
6: \textbf{for} \( i = 1 : n - 1 \) \textbf{do}
7: \( \phi_i = \phi_{i-1} p_i \)
8: \textbf{end for}

Jacobi matrix is computed by the lower triangular lanczos (LTL) method that is given by Golub and Kautsky in [10]. The following LTL algorithm takes as input a \( k \times k \) matrix \( J \), vectors \( u = L e_1 \) that stores the modified moments, where \( L^{-1} \) satisfies \( p = L^{-1} \tilde{p} \), and \( c \) of dimension \( k \), where \( c \) relates the old polynomials of degree \( n \) to all of the new polynomials. In the above relationship \( p = L\tilde{p} \), \( p \) refers to the new measure, and \( \tilde{p} \) the old measure. However, since we are trimming the matrix of the last row and column, \( c \) leaves the results unaffected. This results in the tridiagonal modified Jacobi matrix

\[
\tilde{J}_n = \begin{bmatrix}
\tilde{\alpha}_1 & \tilde{\beta}_1 & & \\
\tilde{\beta}_1 & \tilde{\alpha}_2 & & \\
& \ddots & \ddots & \\
& & \tilde{\beta}_{n-1} & \tilde{\alpha}_n
\end{bmatrix}.
\tag{2.19}
\]

Therefore, the Jacobi matrix corresponding to a rational weight function of either a linear factor or irreducible quadratic factor can be computed using this method. However, the
Algorithm 2 Lower Triangular Lanczos

1: \( \mathbf{v}_0 = 0, \mathbf{v}_1 = \mathbf{u}, \gamma_1 = 1, \tilde{\beta}_0 = 1, \lambda_1 = (\mathbf{e}_1^T \mathbf{v}_1)^{-1}, \tilde{\alpha}_1 = \lambda_1 \mathbf{e}_1^T \mathbf{v}_1 \)
2: \textbf{for} \( j = 2 : k \) \textbf{do}
3: \( \mathbf{v}_j = ((\mathbf{J} - \tilde{\alpha}_{j-1} \mathbf{I}) \mathbf{v}_{j-1} - \gamma_{j-1} \mathbf{v}_{j-2}) / \tilde{\beta}_{j-2} + \mathbf{e}_j^T \mathbf{e}_k \)
4: \( \gamma_j = \lambda_1 j - 1 \mathbf{e}_{j-1}^T \mathbf{J} \mathbf{v}_j \)
5: \( \lambda_j = (\mathbf{e}_j^T \mathbf{v}_j)^{-1} \)
6: \( \tilde{\beta}_{j-1} = (\gamma_j \tilde{\beta}_{j-2})^{1/2} \)
7: \( \tilde{\alpha}_j = \lambda_j \mathbf{e}_j^T (\mathbf{J} \mathbf{v}_j - \gamma_j \mathbf{v}_{j-1} + \tilde{\beta}_{j-1} \mathbf{e}_k) \)
8: \textbf{end for}

The major drawback of this method is that, in practice, it requires very large initial matrices to produce a result of satisfactory dimensions. This approach is also computationally expensive, requiring \( O(n^2) \) floating point operations.

2.4 Inverse Cholesky

Suppose the polynomials \( \mathbf{q}(t) = (q_0(t), q_1(t), \ldots, q_{n-1}(t))^T \) and \( q_n(t) \) are orthonormal on the interval \([a, b]\) with respect to some measure \( d\lambda(t) \). The coefficients of the Jacobi matrix, \( J_n \), can be obtained from the three term recurrence relation for orthogonal polynomials,

\[
\beta_{j+1} q_{j+1}(t) = (t - \alpha_j) q_j(t) - \beta_j q_{j-1}(t). \tag{2.20}
\]

Thus we have the symmetric, tridiagonal matrix (2.9). From this recurrence relation, we can derive an identity that will be important throughout this paper. Now, it is easy to see that the first recurrence relation is

\[
\beta_1 q_1(t) = (t - \alpha_0) q_0(t) - \beta_0 q_{-1}(t). \tag{2.21}
\]

Rearranging the above, we get

\[
t q_0(t) = \alpha_0 q_0(t) + \beta_1 q_1(t), \tag{2.22}
\]

since \( q_{-1} \equiv 0 \). If we take a look at the subsequent recurrence relations, it is easy to see a pattern emerging. For a brief moment, let us turn our attention to the following: One can easily see that (2.22) is precisely what we found from \( J\mathbf{q}(t) \). We can conclude that

\[
t \mathbf{q}(t) = J_n \mathbf{q}(t) + \beta_n q_n(t) \mathbf{e}_n. \tag{2.23}
\]

Taking the outer product with \( \mathbf{q}(t)^T \) we get,

\[
t \mathbf{q}(t) \mathbf{q}(t)^T = J_n \mathbf{q}(t) + \beta_n q_n(t) \mathbf{q}(t)^T.
\]
Due to orthogonality, the last term vanishes once integrated. This, moreover, implies that

\[ J_n = \int_a^b r\mathbf{q}(t)\mathbf{q}(t)^T d\lambda(t). \]  

(2.24)

Now suppose we have a second set of polynomials \( \tilde{\mathbf{q}}(t) = (\tilde{\mathbf{q}}_0(t), \tilde{\mathbf{q}}_1(t), \ldots, \tilde{\mathbf{q}}_{n-1}(t)) \) orthogonal on the interval \([a, b]\) with respect to the measure \( d\tilde{\lambda}(t) \), where \( d\tilde{\lambda}(t) = (t - c)^{-1}d\lambda(t) \). The previous properties apply to these polynomials as well. In the paper [2] by Elhay and Kautsky, the Inverse Cholesky (IC) method is used to obtain the Jacobi matrix \( \tilde{\mathbf{J}} \) modified by the measure \( d\tilde{\lambda}(t) \). If \( \tilde{\mathbf{J}} \) is the matrix satisfying (2.23) with \( \tilde{\mathbf{q}}(t) \), then it can be shown from [3] that

\[ \tilde{\mathbf{J}}_n = L_n^{-1}JL_n + \mathbf{e}_n\beta_n\theta_n\mathbf{c}^T, \]  

(2.25)

where \( \theta_n \) is some constant, and \( L_n \) is found as a Cholesky factor of

\[ J_n^{-1}(I - \mathbf{e}_n\mathbf{d}^T) = L_nL_n^T, \]  

(2.26)

which implies \( I - J_n\mathbf{e}_n\mathbf{d}^T = J_nL_nL_n^T \), where \( \mathbf{d} = \beta_n\theta_n\mathbf{c} \), and \( L_n \) is a lower triangular matrix with elements

\[
\begin{bmatrix}
  l_{11} & 0 & 0 & \ldots & 0 \\
l_{21} & l_{22} & 0 & \ldots & 0 \\
l_{31} & l_{32} & l_{33} & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
l_{n1} & l_{n2} & l_{n3} & \ldots & l_{nn}
\end{bmatrix}.
\]  

(2.27)

From the above equation (2.26), the authors [2] developed the Inverse Cholesky algorithm based on the following element by element relation:

\[
\delta_{ij} - \delta_{ni}\mathbf{d}_j = \beta_{i-1}\sum_{k=1}^{n} l_{i-1,k}l_{jk} + \alpha_i\sum_{k=1}^{n} l_{ik}l_{jk} + \beta_i\sum_{k=1}^{n} l_{i+1,k}l_{jk} \\
- \beta_{i-1}\sum_{k=1}^{\min(i-1,j)} l_{i-1,k}l_{jk} + \alpha_i\sum_{k=1}^{j} l_{ik}l_{jk} + \beta_i\sum_{k=1}^{j-1} l_{i+1,k}l_{jk} + \beta_i\beta_{i-1}l_{i+1,j}l_{jj}.
\]  

(2.28)

(2.29)

The algorithm (3) takes the symmetric tridiagonal matrix \( J \) as input and requires that \( l_{11} \neq 0 \). This algorithm determines the \( L \) and \( \mathbf{d} \) from (2.26). A similar algorithm is described in the paper [2] for modification by dividing by a quadratic factor. For this case, the Cholesky factor \( L_n \) satisfies the relation

\[ I = J_n^2L_nL_n^T + \mathbf{e}_n\mathbf{d}^T + \mathbf{e}_{n-1}\mathbf{f}^T. \]  

(2.30)

For the purposes of the following algorithm, it is assumed that the upper left \( 2 \times 2 \) block of \( L \) from (2.30) is prescribed. The algorithm (4) takes the symmetric pentadiagonal matrix \( J_n^2 \)
Algorithm 3 Linear Factor Inverse Cholesky

1: $\bar{\beta}_0 = 0$
2: $l_{21} = (1/l_{11} - \alpha_1 l_{11})/\beta_1$
3: for $i = 2 : n - 1$
  4: $l_i + 1,1 = -(\beta_{i-1} l_{i-1,1} + \alpha_i l_{i1})/\beta_i$
5: end for
6: $d_1 = -(\beta_{n-1} l_{n-1,1} + \alpha_n l_{n1})$
7: for $j = 2 : n$
  8: $s = \beta_{j-2} L_{j-2,1:j-2} L_{j-1,1:j-2}^T + \alpha_{j-1} L_{j-1,1:j-1} L_{j,1:j-1}^T + \beta_{j-1} L_{j,1:j-1} L_{j,1:j-1}^T$
  9: $l_{jj} = \sqrt{-s/\beta_{j1}}$
10: for $i = j : n - 1$
  11: $s = \beta_{i-1} L_{i-1,\min(i-1,j)} + \alpha_i L_{i,1:j} L_{j,1:j}^T + \beta_i L_{i+1,1:j-1} L_{j,1:j-1}^T$
  12: $l_{i+1,j} = (\delta_{ij} - s)/\beta_{i1}$
13: end for
14: $d_j = -(\beta_{n-1} L_{n-1,1:min(n-1,j)} L_{j,1:min(n-1,j)}^T + \alpha_n L_{n,1:j} L_{j,1:j}^T)$
15: end for

and this $2 \times 2$ block of $L_n$ as input. The steps to compute $d$ and $f$ are omitted. Based on the relation

\[
(\delta_{ij} - \delta_{n,i-1} f_j - \delta_n d_j) = \gamma_{i-2} \sum_{k=1}^{\min(i-1,j)} l_{i-2,k} l_{jk} + \beta_{i-1} \sum_{k=1}^{\min(i-1,j)} l_{i-1,k} l_{jk} + \alpha_i \sum_{k=1}^j l_{ik} l_{jk} + \beta_i \sum_{k=1}^j l_{i+1,k} l_{jk} + \gamma_i \sum_{k=1}^{j-1} l_{i+2,k} l_{jk} + \gamma_i l_{i+2,j} l_{jj},
\]

(2.31)

we get the following Algorithm 4.

2.5 Jagels’ and Reichel’s work on even powers

In [14], Jagels and Reichel describe an approach to determine recursion coefficients for Gauss quadrature rules associated with measures of the form

\[
\tilde{\lambda}(t) = (t - c)^{2l} \lambda(t).
\]

(2.32)

The proposed numerical method is based on this relation (2.32) between standard and rational Gauss quadrature rules. The authors discuss a method to compute Gauss quadrature rules associated with a measure of the form

\[
d\tilde{\lambda}(t) = \tilde{c} \frac{d\lambda(t)}{t^{2m-2}},
\]

(2.33)
which is used for working on the measure (2.32) with a change of variable. Let $p(t)$ and $q(t)$ be polynomials, then taking the inner product

$$\langle p, q \rangle_{\lambda} = \left\langle p(t), q(t) \right\rangle_{\lambda}$$

(2.34)

shows that standard Gauss rules may be considered rational Gauss rules with respect to the inner product (2.1). To compute the rational Gauss rule

$$R_{\tau}(f) = \sum_{k=1}^{\tau} w_k^2 f(x_k),$$

(2.35)
the entries of the pentadiagonal matrix \( H_\tau \) must first be determined, where the entries of \( H_{\tau-j} \) \((j = 0)\) are the recursion coefficients of \( \phi_j(t) \), where

\[
\Phi_{\tau-j}(t) = [\phi_0(t), \ldots, \phi_i(t), \phi_{-1}(t), \ldots, \phi_{-m+1}, \ldots, \phi_{m-j}]
\] (2.36)

are vectors of orthonormal Laurent polynomials. The Gauss rule (2.35) can be considered a \( \tau \) point Gauss rule associated with the measure (2.33). To compute the necessary Gauss rule, the authors then evaluate the eigenvalues of \( H_\tau \) give the nodes, and the squares of the first components of the normalized eigenvectors of \( H_\tau \) give the weights.
Chapter 3
Building Blocks for Algorithms

In this chapter, we discuss necessary tools for developing the methods of this dissertation. Since the goal is to reverse the procedure for modification of a Jacobi matrix by multiplication, and the multiplication procedure requires certain factorizations, then we also must reverse the algorithms for those factorizations as well. Later in this chapter, we discuss non-linear equation solvers such as the secant method and some convergence accelerators. Root finding methods play an important role in the linear case in particular. That is, they allow for the unknown parameter of the method to be found.

3.1 Reverse Cholesky

Reverse Cholesky is just like the "regular" Cholesky factorization
\[ A = GG^T, \]
but in reverse order
\[ A = G^T G, \]
where \( G \) is a lower triangular matrix. Reverse Cholesky starts at the \((n,n)\) entry, whereas the Cholesky factorization (3.1) begins with the \((1,1)\) entry. Let us examine how to compute the Cholesky factors \( G \) from the following equations:

\[
\begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & \ddots & \cdots & \vdots \\
  \vdots & \ddots & \ddots & \vdots \\
  a_{n1} & \cdots & \cdots & a_{nn}
\end{bmatrix} =
\begin{bmatrix}
  g_{11} & g_{21} & \cdots & g_{n1} \\
  \vdots & \ddots & \cdots & \vdots \\
  \vdots & \ddots & \ddots & \vdots \\
  g_{n1} & \cdots & \cdots & g_{nn}
\end{bmatrix}
\]

If we take a look at a 5 \( \times \) 5 example, then for the fifth row, we have
\[
\begin{align*}
  a_{55} &= g_{55}^2 \\
  a_{54} &= g_{55}g_{54} \\
  a_{53} &= g_{55}g_{53} \\
  a_{52} &= g_{55}g_{52} \\
  a_{51} &= g_{55}g_{51}. 
\end{align*}
\]
For the fourth row, we have

\[ a_{45} = g_4 g_{55} \]
\[ a_{44} = g_{44}^2 + g_{45}^2 \]
\[ a_{43} = g_{44} g_{43} + g_{45} g_{53} \]
\[ a_{42} = g_{44} g_{42} + g_{45} g_{52} \]
\[ a_{41} = g_{44} g_{41} + g_{45} g_{51}. \]

For the third row, we have

\[ a_{35} = g_3 g_{55} \]
\[ a_{34} = g_{34} g_{44} + g_{35} g_{54} \]
\[ a_{33} = g_{33}^2 + g_{34}^2 + g_{35}^2 \]
\[ a_{32} = g_{33} g_{32} + g_{34} g_{42} + g_{35} g_{52} \]
\[ a_{31} = g_{33} g_{31} + g_{34} g_{41} + g_{35} g_{51} \]

From these entries, we can see a pattern emerging in the calculation of the entries of the reverse Cholesky factors. This leads to the following algorithm that takes a symmetric positive definite matrix \( A \) as input, and gives the reverse Cholesky factor \( G \) as output. All of

\begin{algorithm}
\begin{algorithmic}[1]
\State \textbf{for} \( i = n : 1 \) \textbf{do}
\State \hspace{1em} \textbf{for} \( j = i + 1 : n \) \textbf{do}
\State \hspace{2em} \( a_{ij} = a_{ij} - l_{ji}^2 \)
\State \hspace{1em} \textbf{end for}
\State \hspace{1em} \( l_{ii} = \sqrt{a_{ii}} \)
\State \hspace{1em} \textbf{for} \( j = 1 : i - 1 \) \textbf{do}
\State \hspace{2em} \textbf{for} \( k = i + 1 : n \) \textbf{do}
\State \hspace{3em} \( a_{ij} = a_{ij} - l_{ki} \ast l_{kj} \)
\State \hspace{2em} \textbf{end for}
\State \hspace{2em} \( l_{ij} = a_{ij} / l_{ii} \)
\State \hspace{1em} \textbf{end for}
\State \textbf{end for}
\end{algorithmic}
\end{algorithm}

the entries of \( G \) can be found this way. \( A = G^T G \) is the reverse Cholesky factorization of \( A \) if and only if \( P^T A P = LL^T \) is the Cholesky factorization of \( P^T A P \), where \( L = P^T G^T P \) and \( P \) is the permutation matrix that reverses the order of rows or columns.
3.2 Reverse QR

Similar to the previous section on reverse Cholesky, we can also reverse the algorithm for the QR factorization. In this case, we wish to find the matrix factorization $A = RQ$, where $Q$ is orthogonal and $R$ is upper triangular. First, we right multiply by $Q^T$, since $Q$ is an orthogonal matrix, to get $AQ^T = R$. Let

$$Q^T = G_1 \ldots G_k \quad \text{with } k = n(n-1)/2$$

where $G_1, \ldots, G_k$ are Givens rotations. Givens rotations are used to zero specific off diagonal entries of a matrix, and are defined by the matrix

$$G = \begin{bmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{bmatrix}.$$

This operation is called a rotation because it is orthogonal, and therefore length-preserving, and also because there is an angle $\theta$ such that $\sin \theta = \sigma$ and $\cos \theta = \gamma$. Its effect is to rotate a vector clockwise through the angle $\theta$. In particular,

$$\begin{bmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{bmatrix}^T \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \rho \\ 0 \end{bmatrix}$$

where $\rho = \sqrt{\alpha^2 + \beta^2}$, $\alpha = \rho \cos \theta$ and $\beta = \rho \sin \theta$. Starting with the $n-2$ column and last row, we use Givens rotations to rotate the $n-2$ and $n-1$ columns, performing $AQ^T = R$, and zeroing the $n, n-2$ entry:

$$\begin{bmatrix} \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & \times & 0 \\ \times & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ \end{bmatrix} \rightarrow \begin{bmatrix} \times & \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ \end{bmatrix}.$$

Then move to the right rotating the $n-1$ and $n$ columns. Once that row is complete, then we move up to the row above and perform the same operations until we achieve an upper triangular structure. Each Givens rotations only requires $O(1)$ floating point operations if $A$ is banded; therefore, the entire process requires $O(n)$ floating point operations.

3.3 Solving nonlinear equations

To obtain the modified Jacobi matrix, it is necessary to solve a nonlinear equation and sometimes a system of nonlinear equations. In this section we will discuss various root finding
methods, along with some safeguarded methods and methods for accelerated convergence. The methods discussed in this section are standard numerical analysis methods and can be found in the texts [12, 13, 16, 19].

### 3.3.1 Bisection

Bisection is among the simplest of the root finding methods used to solve \( f(x) = 0 \), where \( f \) is a continuous function defined on a closed interval \([a, b]\), and \( f(a) \) and \( f(b) \) are opposite in sign. Then the Intermediate Value Theorem can be used to conclude that a solution is bracketed in the interval where the function evaluations have opposite signs. The idea is that given an interval and an initial guess, the interval is bisected in two by computing the midpoint \( c = \frac{a + b}{2} \), then it is necessary to again use the Intermediate Value Theorem to determine where the solutions lies: either between \( f(a) \) and \( f(c) \) or between \( f(c) \) and \( f(b) \). The iteration continues until satisfactory convergence is achieved. The most desirable trait of bisection is that we are guaranteed a solution exists in the interval if the criteria is satisfied. However, convergence for this method is very slow.

**Algorithm 6 Bisection**

```
1: for \( i = 1, 2, \ldots \) do
2: \( m = (a + b)/2 \)
3: if \( f(m) = 0 \) then
4: \( x^* = m \)
5: return \( x^* \)
6: end if
7: if \( f(a)f(m) < 0 \) then
8: \( b = m \)
9: else
10: \( a = m \)
11: end if
12: end for
```

### 3.3.2 Secant method

The secant method is a well known root finding algorithm that uses a succession of roots of secant lines to approximate the root of nonlinear equations. The secant method takes two initial guesses \( x_0 \) and \( x_1 \) as input, and uses the \( y \) values of the function at those initial guesses to form a secant line between the two initial guesses. This computes the next iterate,

\[
x_n = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}.
\]
This iteration stops when a given error tolerance is achieved. The order of convergence for the secant method is superlinear.

**Algorithm 7** Secant Method

1: for $i = 1, 2, \ldots$ do
2:   if $f(x^i)$ is sufficiently small then
3:     $x^* = x^i$
4:   return $x^*$
5: end if
6: $x^{(k+1)} = x^i - \frac{f(x^i)}{f(x^i) - f(x^{(k-1)})} x^i - x^{(k-1)}$
7: if $|x^{(k+1)} - x^i|$ is sufficiently small then
8:     $x^* = x^{(k+1)}$
9: end if
10: end if
11: end for

### 3.4 Safe-guarded Methods

The *Regula Falsi Method*, or method of false position, is a root finding method for solving an equation in one unknown. Like bisection, regula falsi is guaranteed to converge, but sometimes experiences lower rates of convergence than Newton’s method or the secant method. It is a modification of the secant method in which the two initial iterates $x^{(0)}$ and $x^{(1)}$ are chosen so that the product of the iterates evaluated at the function is negative, $f(x^{(0)}) \cdot f(x^{(1)}) < 0$, therefore guaranteeing that a solution lies between $x^{(0)}$ and $x^{(1)}$. This condition also guarantees that the next iterate $x^{(2)}$ will lie between the previous two iterates $x^{(0)}$ and $x^{(1)}$, as can be seen by applying the Intermediate Value Theorem to the secant line passing through $(x^{(0)}, f(x^{(0)}))$ and $(x^{(1)}, f(x^{(1)}))$.

It follows that if $f(x^{(2)}) \neq 0$, then a solution must exist between $x^{(0)}$ and $x^{(2)}$, or between $x^{(1)}$ and $x^{(2)}$. In the first case, we use the secant line passing through the points $(x^{(0)}, f(x^{(0)}))$ and $(x^{(2)}, f(x^{(2)}))$ to compute the next iterate $x^{(3)}$. In the latter case, we use the secant line passing through the points $(x^{(1)}, f(x^{(1)}))$ and $(x^{(2)}, f(x^{(2)}))$. Continuing this iteration, a sequence of increasingly smaller intervals can be obtained that are guaranteed to contain a solution, as in the Bisection Method, but the interval is updated using a superlinearly convergent method, the Secant Method, rather than simply being bisected. The algorithm for regula falsi takes the interval $(a, b)$ as inputs. In some certain examples, such as solving $f(x) = 0$ where $f(x) = x^2 - 2$, the right endpoint of each interval does not change, while the left endpoint converges towards a solution. The unfortunate result is that the search
Algorithm 8 Regula Falsi

1: for $i = 1, 2, \ldots$ do
2: $c_i = \frac{af(b) - bf(a)}{f(b) - f(a)}$
3: if $f(c) = 0$ or $b - a$ is sufficiently small then
4: $x^* = c$
5: return $x^*$
6: end if
7: if $f(a)f(c) < 0$ then
8: $b = c$
9: else
10: $a = c$
11: end if
12: end for

interval will not converge to zero as in bisection; it will converge to the interval $(x^*, x^{(1)})$. However, with a modification of algorithm (8), this issue can be resolved resulting in the Illinois Algorithm [1]. The idea behind this method is that the endpoint of the interval that is not changing, denoted $x^{(K)}$, must be identified, and then the next iterate is forced to be on the same side of the root as $x^{(K)}$. This is done by halving the $y$ value of the previous data point if it turns out that the new $y$ value has the same sign as the previous one. A modification of the regula falsi algorithm can be made to correct this issue. Instead of maintaining endpoints of the search interval $(a, b)$ where $a < b$, we need only to prescribe that $a = x^{(k-1)}$ and $b = x^{(k)}$, that is, the previous two iterates.

Algorithm 9 Illinois Algorithm

1: for $i = 1, 2, \ldots$ do
2: $c_i = \frac{af(b) - bf(a)}{f(b) - f(a)}$
3: if $f(c) = 0$ or $b - c$ is sufficiently small then
4: $x^* = c$
5: return $x^*$
6: end if
7: if $f(b)f(c) < 0$ then
8: $a = b$  $f(a) = f(b)$
9: else
10: $f(a) = f(a)/2$
11: end if
12: $b = c$  $f(b) = f(c)$
13: end for
We are also interested in accelerating the convergence of such methods previously discussed. One way of doing this is called Aitken’s $\Delta^2$ Method. This is done by constructing 

$$\hat{x}^k = x^k - \frac{(\Delta x^k)^2}{\Delta^2 x^k} \quad k = 0, 1, 2, \ldots,$$

where

$$\Delta x^k = x^{k+1} - x^k$$

and

$$\Delta^2 x^k = \Delta(x^{k+1} - x^k)$$

$$= (x^{k+2} - x^{k+1}) - (x^{k+1} - x^k)$$

$$= x^{k+2} - 2x^{k+1} + x^k$$

If an iterate falls outside of the interval, we can first bisect, creating a smaller interval and therefore making the iterates fall inside the interval.

Methods for solving a system of nonlinear equations include methods such as Newton’s method and Broyden’s method. Newton’s method for a single nonlinear equation can be generalized to a system of nonlinear equations using a fixed-point iteration.

**Algorithm 10** Newton’s Method for a system of equations

1: while not converged do  
2: $y_k = F(x^{(k)})$  
3: solve $[J_F(x^{(k)})]s_k = -y_k$ for $s_k$  
4: $x^{(k+1)} = x^{(k)} + s_k$  
5: $k = k + 1$  
6: end while  
7: $x^* = x^{(k)}$

In the above algorithm $J_F$ represents the Jacobian of $F$. This presents the main drawback of this method: the need to compute $J_F$ and then solve a system of linear equations every iteration. Another disadvantage of this method is the inability to use previously computed information. Broyden’s method is derived from Newton’s method to overcome the challenges of having to compute a partial derivative and having to solve a system of equations each iteration. The idea behind Broyden’s method is that we will no longer compute the Jacobian every iteration. Instead, a rank-one update of the form

$$B = A + uv^T$$

is used to update the Jacobian each iteration.
Algorithm 11: Broyden’s Method

1: Choose $x^{(0)}$
2: $A_0 = J_F(x^{(0)})$
3: $s_1 = -A_0^{-1}F(x^{(0)})$
4: $x^{(1)} = x^{(0)} + s_1$
5: $k = 1$
6: while not converged do
7: $y_k = F(x^{(k)}) - F(x^{(k-1)})$
8: $w_k = A_{k-1}^{-1}y_k$
9: $c = 1 / s_k^T w_k$
10: $A_k^{-1} = A_{k-1}^{-1} + c(s_k - w_k)s_k^TA_k^{-1}$
11: $s_{k+1} = -A_k^{-1}F(x^{(k)})$
12: $x^{(k+1)} = x^{(k)} + s_{k+1}$
13: $k = k + 1$
14: end while
In this chapter, we will discuss the multiplication procedure that modifies a measure by multiplying the original measure by a linear factor \( (t - c) \) in section (4.11). In Section 4.2, the reversal procedure for the linear case is described where the original measure is modified by dividing by a linear factor \( (t - c)^{-1} \).

\section{Multiplication Procedure}

Given a symmetric tridiagonal matrix \( J_n \) (2.6) corresponding to the set of polynomials \( p(t) = (p_0(t), p_1(t), p_2(t), \ldots, p_{n-1}(t))^T \) that are orthonormal with respect to the measure \( d\lambda(t) \), one can obtain, by the multiplication procedure due to [4], the modified matrix \( \tilde{J}_n \) corresponding to the set of polynomials \( \tilde{p}(t) = (\tilde{p}_0(t), \tilde{p}_1(t), \tilde{p}_2(t), \ldots, \tilde{p}_{n-1}(t))^T \) that are orthonormal with respect to the measure \( d\tilde{\lambda}(t) \), where \( \tilde{\lambda}(t) = \omega(t - c)\lambda(t) \), where \( c \) is outside or on the boundary of the support interval of \( \lambda(t) \), and \( \omega \) is chosen to be \( \pm 1 \) so that \( d\tilde{\lambda}(t) \) is positive. We choose these values to ensure that the matrix

\[ \omega(J_n - cI) \]

is symmetric and positive definite, so that it admits a Cholesky decomposition,

\[ \omega(J_n - cI) = L_nL_n^T \]

where \( L_n \) is lower bidiagonal. From (4.1) we can investigate the relationship between the old polynomials \( p(t) \) and the new polynomials \( \tilde{p}(t) \). Starting with the fact that

\[ \omega(J_n - cI) = \omega J_n - \omega cI \] (4.3)

substituting from (2.24) and using the fact that the polynomials \( p(t) \) are orthogonal with respect to the measure \( d\lambda(t) \), we see that (4.3) is equivalent to

\[ L_nL_n^T = \omega \int_a^b tp(t)p(t)^T d\lambda(t) - \omega c \int_a^b p(t)p(t)^T d\lambda(t) \]

\[ = \omega \int_a^b (t - c)p(t)p(t)^T d\lambda(t) \]

\[ = \int_a^b p(t)p(t)^T d\tilde{\lambda}(t) \] (4.6)
Multiplying on the left by $L_n^{-1}$ and on the right by $L_n^{-T}$ yields

$$
I = L_n^{-1} \int_a^b p(t)p(t)^T d\tilde{\lambda}(t)L_n^{-T} \quad (4.7)
$$

$$
= L_n \int_a^b L_n^{-1} p(t)p(t)^T L_n^{-T} d\tilde{\lambda}(t)L_n^T. \quad (4.8)
$$

Using the relation,

$$
p(t) = L_n \tilde{p}(t)
$$

we see that

$$
\int \tilde{p}(t)\tilde{p}(t)^T d\tilde{\lambda}(t) = I. \quad (4.10)
$$

This shows that the new polynomials $\tilde{p}(t)$ are orthonormal with respect to the measure $d\tilde{\lambda}(t)$. Now, from [4] we can compute $\tilde{J}_n$ by using the fact that the shifted matrix $(\tilde{J}_n - cI)$ satisfies

$$
(\tilde{J}_n - cI) = \int_a^b (t-c)\tilde{p}(t)\tilde{p}(t)^T d\tilde{\lambda}(t)
$$

$$
= (t-c)(L_n^{-1} p(t))(L_n^{-1} p(t))^T \omega(t-c)\tilde{\lambda}(t)
$$

$$
= (t-c)L_n^{-1} p(t)p(t)^T L_n^{-T} \omega(t-c)\tilde{\lambda}(t)
$$

$$
= \omega L_n^{-1} \int_a^b (t-c)^2 p(t)p(t)^T d\tilde{\lambda}(t)L_n^{-T}.
$$

Using the Lanczos relation (2.23),

$$
(t-c)p(t) = (J_n - cI)p(t) + \beta_n p_n(t)e_n,
$$

and multiplying it by its transpose on both sides of the equation we have

$$
(t-c)p(t)(t-c)p(t)^T = (J_n - cI)p(t)(t-c)p(t)^T + \beta_n p_n(t)e_n
$$

$$
= (J_n - cI)p(t)p(t)^T (J_n - cI) + \beta_n p_n(t)e_n e_n^T
$$

$$
= (J_n - cI)p(t)p(t)^T (J_n - cI) + \beta_n p_n(t)e_n e_n^T + \beta_n p_n(t)e_n e_n^T.
$$

Substituting the above into (4.11), and using properties of orthonormality we obtain

$$
\tilde{J}_n - cI = \omega L_n^{-1} \int_a^b (t-c)^2 p(t)p(t)^T \lambda(t)L_n^{-T}
$$

$$
= \int_a^b ((J_n - cI)p(t)p(t)^T (J - cI) + \beta_n p_n^2 e_n e_n^T)\lambda(t)L_n^{-T}
$$

$$
= \omega L_n^{-1} (J_n - cI) \int_a^b p(t)p(t)^T \lambda(t)((J_n - cI) + \beta_n e_n e_n^T)L_n^{-T}
$$

$$
= \omega L_n^{-1} (J_n - cI) I(J_n - cI) + \beta_n e_n e_n^T)L_n^{-T}
$$

$$
= \omega L_n^{-1} (J_n - cI)^2 + \beta_n e_n e_n^T)L_n^{-T}.
and substituting (4.2) into the above we have

\[
\tilde{J}_n - cI = \omega L_n^{-1} \left( \frac{1}{\omega^2} L_n L_n^T L_n L_n^T + \beta_n e_n e_n^T \right) L_n^{-T}
\]

\[
\tilde{J}_n = cI + \frac{1}{\omega} L_n^T L_n + \omega L_n^{-1} \beta_n e_n e_n^T L_n^T
\]

\[
= cI + \frac{1}{\omega} L_n^T L_n + \omega \beta_n (L_n^{-1} e_n)(L_n^{-1} e_n)^T
\]

\[
= cI + \frac{1}{\omega} L_n^T L_n + \omega \beta_n e_n e_n^T.
\]

We now have a formula to obtain \(\tilde{J}_n\) and if \(c < a\) for interval \([a, b]\), and if we let \(\omega = 1\) we obtain,

\[
\tilde{J}_n = L_n^T L_n + cI + \frac{\beta_n}{l_{nn}^2} e_n e_n^T.
\]  

(4.11)

4.2 Reversal Procedure

In the previous section, we discussed the procedure for obtaining the matrix \(\tilde{J}_n\) from the matrix \(J_n\) that is modified by the measure \(d\tilde{\lambda}(t) = (t - c)d\lambda(t)\), where \(d\lambda(t)\) is the measure that the original polynomials \(p(t)\) are orthogonal with respect to. We now consider that the modified matrix \(\tilde{J}_n\) is given, and wish to obtain the original matrix \(J_n\) when it is modified by the weight function \(d\lambda(t) = \frac{1}{(t-c)}d\tilde{\lambda}(t)\). We do this by reversing the procedure for multiplication by a linear factor. In the multiplication procedure, we perform the Cholesky factorization,

\[
J_n - cI = L_n L_n^T,
\]

multiply the Cholesky factors in reverse order, shift back, and add on a rank one update,

\[
\tilde{J}_n = L_n^T L_n + cI + \frac{\beta_n}{l_{nn}^2} e_n e_n^T.
\]  

(4.12)

To reverse this procedure, we must first find the Cholesky factors of (4.11). Since the factors \(L_n\) are in reverse order, we have to use the reverse Cholesky procedure described in Section 3.1. However, complications arise in the reversal that were of no concern in the multiplication procedure. The reverse Cholesky procedure is applied to the matrix

\[
\tilde{J}_n - cI - \left( \frac{\beta_n}{l_{nn}^2} \right) e_n e_n^T,
\]

where \(\beta_n\) is unknown because the super and subdiagonals are where the entries \(\beta_1, \beta_2, \ldots, \beta_{n-1}\) lie, therefore the Jacobi matrix \(J_n\) of size \(n \times n\) only contains \(\beta_1, \ldots, \beta_{n-1}\). Therefore, we have to use a secant like iteration to converge to this value. If the guess is off, the error will propagate through the entire matrix, since the reverse Cholesky algorithm begins with the \((n,n)\) entry of the matrix.
4.3 Derivation of the objective function for guessing

Since we are using an initial guess and iteration to find the value of $\beta_n$, we need a way to check to see if it is correct, and the matrix that results is also correct. The recurrence relations are not reliable for checking this. We know that the resulting polynomials should be orthonormal,

$$\langle p_j, p_k \rangle_{\lambda} = \delta_{j,k},$$

where

$$\delta_{j,k} = \begin{cases} 
0 & \text{if } i \neq j \\
1 & \text{if } i = j 
\end{cases}$$

Therefore, we use that to check, using information that is known. From the relationship $p(t) = L_n \tilde{p}(t)$, for example in the case where $n = 5$,

$$\begin{bmatrix} p_0 \\
p_1 \\
p_2 \\
p_3 \\
p_4 \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\
l_{21} & l_{22} & 0 & 0 & 0 \\
0 & l_{32} & l_{33} & 0 & 0 \\
0 & 0 & l_{43} & l_{44} & 0 \\
0 & 0 & 0 & l_{54} & l_{55} \end{bmatrix} \begin{bmatrix} \tilde{p}_0 \\
\tilde{p}_1 \\
\tilde{p}_2 \\
\tilde{p}_3 \\
\tilde{p}_4 \end{bmatrix}. \quad (4.14)$$

Let us take the inner product $\langle p_1, p_3 \rangle_{\lambda}$, and from (4.14), we can make the substitutions:

$$p_3 = l_{32} \tilde{p}_2 + l_{33} \tilde{p}_3. \quad (4.15)$$

Now, rearranging and regrouping we get,

$$\langle p_1, p_3 \rangle_{\lambda} = l_{32} \langle p_1, \tilde{p}_2 \rangle_{\lambda} + l_{33} \langle p_1, \tilde{p}_3 \rangle_{\lambda}$$

From the above, we get, in general,

$$\langle p_j, p_k \rangle_{\lambda} = l_{k,k-1} \langle p_j, \tilde{p}_{k-1} \rangle_{\lambda} + l_{k,k} \langle p_j, \tilde{p}_k \rangle_{\lambda}, \quad \text{for } k > 1, \quad (4.16)$$

where we have the base case: $\langle p_j, p_1 \rangle_{\lambda} = l_{11} \langle p_j, \tilde{p}_1 \rangle_{\lambda}$. Rewriting terms from above using shifted power form, we get

$$\langle p_j, \tilde{p}_k \rangle_{\lambda} = \langle p_j(c) + \sum_{i=1}^{j} a_{ij} (t-c)^i, \tilde{p}_k \rangle_{\lambda}$$

$$= p_j(c) \langle 1, \tilde{p}_k \rangle_{\lambda} + \sum_{i=1}^{j} a_{ij} \langle (t-c)^{i-1}, \tilde{p}_k \rangle_{\lambda}$$

$$= p_j(c) \langle 1, \tilde{p}_k \rangle_{\lambda},$$

where the last term vanishes due to orthogonality only if $i - 1 < k$. Likewise, we compute

$$\langle p_j, \tilde{p}_{k-1} \rangle_{\lambda} = p_j(c) \langle 1, \tilde{p}_{k-1} \rangle_{\lambda}.$$
from (4.16). The necessary initial value is \( \langle 1, \tilde{p}_0 \rangle_\lambda = \tilde{p}_0 \mu_0 \). We now need to compute the inner product \( \langle 1, \tilde{p}_k \rangle_\lambda \) from the recurrence relation for \( \tilde{p}_k \). For values of \( k \geq 1 \)

\[
\langle 1, \tilde{p}_k \rangle_\lambda = \frac{1}{\beta_k} \left[ \langle 1, \tilde{p}_{k-1} \rangle_\lambda + (c - \alpha_k) \langle 1, \tilde{p}_{k-1} \rangle_\lambda - \tilde{p}_{k-1} \langle 1, \tilde{p}_{k-2} \rangle \right],
\]

where \( \langle 1, \tilde{p}_{k-1} \rangle_\lambda = 0 \) when \( k > 1 \) due to orthogonality, and \( \langle 1, \tilde{p}_{k-1} \rangle_\lambda = \mu_0^{1/2} \) when \( k = 1 \). All of the values in the above expression are known, and therefore the inner product (4.16) can be computed.

### 4.4 Interval for guessing and quartic equation

As stated previously, the value of \( \beta_n \) is unknown, and this value is critical in computing the modified Jacobi matrix that we want to find. Therefore, we have to iterate using a secant method to converge to this solution along with regula falsi to accelerate convergence. The interval we use is some interval in the interior of \([0, z]\) where we find the value of \( z \) from looking at the \((n \times n)\) entry of (4.12)

\[
\tilde{j}_{nn} = l_{nn}^2 + c + \beta_n / l_{nn}^2
\]

(4.17)

and multiplying the entire equation through by \( l_{nn}^2 \) and rearranging, we get a quadratic equation in \( l_{nn}^2 \)

\[
l_{nn}^4 - (\tilde{j}_{nn} - c)l_{nn}^2 + \beta_n = 0.
\]

Solving this quadratic equation leads to the interval \( z = (c - \alpha_n) / 2 \), where \( \alpha_n \) is the \( n \times n \) entry of \( J_n \).

### 4.5 Algorithm description

We give algorithms for computing the modified Jacobi matrix. The main function Algorithm 12 generates the Jacobi matrix and the necessary moments for the input to the iteration. Algorithm 13 is the objective function \( f \) that we wish to find a solution for.
Algorithm 12 Main: This function is the main function called. The parameters \([a, b], w, n,\) and \(c\) are given as input. This function calls the test case Algorithm 6, given in Chapter 6, to produce the necessary Jacobi matrices, moments, and orthonormal polynomials. It then calls Secant method Algorithm 7 that takes the linear reversal Algorithm 13 as the function handle \(f\).

1: \([J, \tilde{J}, \mu, \tilde{\mu}, q(t), \tilde{q}(t)] = \text{testcase}(a, b, r, w, n)\)
2: \(z = |c - \alpha_{n-1}| / 2\)
3: \([x] = \text{secant}(h * z, k * z, f)\)
4: \(\tilde{L} = \text{reversecholesky}(J - cI - \delta_1e_ne_n^T)\)
5: \(\tilde{J} = \tilde{L}\tilde{L}^T + cI\)

Algorithm 13 Linear Factor Reversal: This algorithm takes as input the shift \(c\), the moments \(\mu_0\) and \(\tilde{\mu}_0\), a scalar value \(x\), and the Jacobi matrix \(J_n\) and returns the modified Jacobi matrix \(\tilde{J}_n\). This algorithm is the objective function that we wish to find the root of by secant method, then carries out the reversal procedure outlined in Section (4.2).

1: \(\beta_n = x\)
2: \(p = r^4 + (c - J_{n-1,n-1})t^2 + \beta_n^2\)
3: \(r = \text{roots}(p)\)
4: \(r_1 = \max|r|\)
5: \(\delta_1 = (\beta_n / r_1)^2\)
6: \(p_0 = L_{1,1}\)
7: \(\tilde{p}_0 = 1 / \sqrt{\mu_0}\)
8: \(\text{for } i = 1 : n \text{ do}\)
9: \(\tilde{p}_i = ((c - J_{i-1,i-1}) * \tilde{p}_{i-1} - J_{i-1,i-2} * \tilde{p}_{i-2}) / J_{i,i-1}\)
10: \(\text{end for}\)
11: \(p = L_{1,1}\tilde{p}\)
12: \(\mu_0 = 1 / L_{1,1}^2\)
13: \(\langle 1, \tilde{p}_1 \rangle = \mu_0 / \sqrt{\mu_0}\)
14: \(\langle 1, \tilde{p}_2 \rangle = (\sqrt{\mu_0} + c - J_{1,1}\langle 1, \tilde{p}_1 \rangle) / J_{2,1}\)
15: \(\text{for } i = 2 : n \text{ do}\)
16: \(\langle 1, \tilde{p}_i \rangle = ((c - J_{i-1,i-1})\langle 1, \tilde{p}_{i-1} \rangle - J_{i-2,i-2}\langle 1, \tilde{p}_{i-2} \rangle) / J_{i,i-1}\)
17: \(\text{end for}\)
18: \(\langle p_j, p_k \rangle_\lambda = L_{k,k-1}\langle p_j, \tilde{p}_{k-1} \rangle_\lambda + L_{k,k}\langle p_j, \tilde{p}_k \rangle_\lambda\)
Chapter 5

Modification by Irreducible Quadratic Factors

Here we will discuss the case that the Jacobi matrix is modified by the weight function \( \tilde{\lambda}(t) = (t - c)^{-1}(t - \bar{c})^{-1} \lambda(t) \), that is an irreducible quadratic factor. Section 4.1 outlines the multiplication procedure that would start with \( J \) and achieve the result of \( J^2 \), this is modification by multiplying by an irreducible quadratic factor. The goal of this chapter is to reverse the multiplication procedure, starting with \( J^2 \). This division by an irreducible quadratic factor will result in what we call the "original" matrix \( J \). There are two main ideas in this chapter:

1. Two-stage algorithm: The most straightforward approach is to apply the previous reversal procedure from Section 4.2 twice. The two stage approach has an intermediate step to achieve the final result. Starting with \( J_2 \to J_1 \to J \), each step involves dividing by one of the irreducible quadratic factors. There are some drawbacks to this method, in particular the fact that we must deal with complex arithmetic among other things such as solving a quartic equation which is more difficult because of the complex shifts, which means that there is no way to bracket the solution.

2. Implicit double shift algorithm: Here, we consider the second degree polynomial of \( J_2 \),

\[
J^2 - 2\text{Re}(c)J + |c|^2 I.
\]

This produces a pentadiagonal matrix to which we apply the reversal procedure to obtain the modified matrix. The advantage of this approach is that complex arithmetic is not necessary.

5.1 Explicit Double Shift

5.1.1 Orthogonal polynomials in the complex plane

Just as we have a weighted inner product of two real functions, we also have a weighted inner product of two functions \( f \) and \( g \) in the complex plane

\[
\langle f, g \rangle_\omega = \int_{\gamma} f(\lambda) \overline{g(\lambda)} \omega(\lambda) |d\lambda|,
\] (5.1)
where \( \omega(\lambda) \) is a positive valued weight function, and \( \gamma \) is an arc in the complex plane. Similar to the real case, we say these functions are orthonormal if they satisfy

\[
\langle f, g \rangle_\omega = \begin{cases} 
1 & : i = j \\
0 & : i \neq j 
\end{cases}
\]

However, a sequence of orthogonal polynomials in the complex plane does not necessarily satisfy a three-term recurrence relation [17]. This is because, in general, for an arc in the complex plane

\[
\langle \lambda f, g \rangle_\omega \neq \langle f, \lambda g \rangle_\omega,
\]

whereas in the real case, the independent variable is transferable from one side to the other in the inner product. The authors of [17] resort to \textit{formally orthogonal polynomials} to overcome the complications we are faced with. A different bilinear form is used to allow the operator to switch from one side to the other, and we do not conjugate.

5.1.2 Two-stage algorithm

The two stage multiplication algorithm can be executed by the following computations:

\[
\begin{align*}
J - c_1 &= L_1 L_1^T \\
J_1 &= L_1^T L_1 + c_1 I + \delta_1 e_n e_n^T \\
J_1 - c_2 I &= L_2 L_2^T \\
J_2 &= L_2^T L_2 + c_2 I + \delta_2 e_n e_n^T,
\end{align*}
\]

where \( \delta_1 = \frac{\bar{\beta}_n}{(L_1)^{2}_{nn}} \) and \( \delta_2 = \frac{\bar{\beta}_n}{(L_2)^{2}_{nn}} \). In this section, we discuss our method for computing the matrix \( J \) from the matrix \( J_2 \) that is modified by the weight function \( (t - c_1)^{-1}(t - c_2)^{-1} \), where \( c_1 \) and \( c_2 = \bar{c}_1 \) are complex conjugates of each other. We achieve this by applying the reversal procedure for dividing by a linear factor twice, with an intermediate step \( J_1 \) that is complex. Let us define

\[
J_2 = \begin{bmatrix}
\bar{\alpha}_1 & \bar{\beta}_1 & 0 & \ldots & 0 \\
\bar{\beta}_1 & \bar{\alpha}_2 & \bar{\beta}_2 & \ldots & \vdots \\
0 & \bar{\beta}_2 & \bar{\alpha}_3 & \ldots & \vdots \\
0 & 0 & \ddots & \ddots & \ddots \\
0 & \ldots & \ldots & \bar{\beta}_{n-1} & \bar{\alpha}_n
\end{bmatrix}
\]

(5.2)
and

\[
J_1 = \begin{bmatrix}
\alpha_1 & \bar{\beta}_1 & 0 & \ldots & 0 \\
\bar{\beta}_1 & \alpha_2 & \bar{\beta}_2 & \ddots & \vdots \\
0 & \bar{\beta}_2 & \alpha_3 & \ddots & \vdots \\
0 & 0 & \ddots & \ddots & \bar{\beta}_{n-1} \\
0 & \ldots & 0 & \bar{\beta}_{n-1} & \alpha_n
\end{bmatrix}.
\] (5.3)

Now, similarly as in the linear case, we must use reverse Cholesky to find the factors \( L_1 \) and \( L_2 \) using the following algorithm:

\[
J_2 - c_2 I + \delta_2 e_n e_n^T = L_2^T L_2
\]

\[
J_1 = L_2 L_2^T + c_2 I
\]

\[
J_1 - c_1 I + \delta_1 e_n e_n^T = L_1^T L_1
\]

\[
J = L_1 L_1^T + c_1 I.
\]

Let \( J_2 \) be the given matrix, then we use the reverse Cholesky algorithm to obtain \( L_2 \) from (5.4). In this case, we will iterate to solve for the unknown entry \( \delta_2 \) as in the linear case. Next, \( J_1 \) can be found by multiplying the Cholesky factors \( L_2 \) and \( L_2^T \) in reverse order. Then the process is repeated, and we have the Cholesky factor \( L_1 \). Once we have \( L_1 \), we can compute the matrix \( J \). The objective function for the two stage algorithm is equivalent to two steps of the objective function described for the linear case in subsection (4.3).

### 5.1.3 Quartic equation

The drawbacks of this method are dealing with complex arithmetic, which is more expensive. Also, this method requires us to solve a quartic equation, and because of complex shifts there is no way to bracket the solution.

**Algorithm 14** Main: This function is the main function called. The parameters \([a, b], w, n, x\) and \( c \) are given as input. This function calls the test case Algorithm 6 to produce the necessary Jacobi matrices, moments, and orthonormal polynomials. It then calls Broyden’s method Algorithm 11 that takes the two-step irreducible quadratic reversal Algorithm 13 as the function handle \( f \).

1: \([J, \tilde{J}, \mu, \tilde{\mu}, q(t), \tilde{q}(t)] = \text{testcase}(a, b, r, w, n)\)
2: \([\delta_2] = \text{broyden}(x, f)\)
3: \(L_2 = \text{reversecholesky}(J - c_2 I_n - \delta_2 e_n e_n^T)\)
4: \(J_1 - c_1 I_n = L_2 L_2^T\)
5: \([\delta_1] = \text{broyden}(x, f)\)
6: \(L_1 = \text{reversecholesky}(J_1 - c_1 I_n - \delta_1 e_n e_n^T)\)
7: \(J - c I_n = L_1 L_1^T\)
Algorithm 15 Irreducible Quadratic Reversal- Two Stage Algorithm: This algorithm applies the algorithm for the linear case twice. This algorithm takes as input the shift \( c \), the moments \( \mu_0 \) and \( \tilde{\mu}_0 \), a scalar value \( x \), and the Jacobi matrix \( J_n \) and returns the modified Jacobi matrix \( \tilde{J}_n \). This algorithm is the objective function that we wish to find the root of by secant method, then carries out the reversal procedure outlined in Section (4.2).

1: \( \tilde{\beta}_n = x \)
2: \( p = t^4 + (c - J_{n-1,n-1})t^2 + \beta_n^2 \)
3: \( r = \text{roots}(p) \)
4: \( r_1 = \max|r| \)
5: \( \hat{\delta}_1 = (\beta_n/r_1)^2 \)
6: \( p_0 = L_{11} \)
7: \( \hat{\rho}_0 = 1/\sqrt{\mu_0} \)
8: \( \textbf{for} \ i = 1 \ : \ n \ \textbf{do} \)
9: \( \tilde{p}_i = \frac{(c - J_{i-1,i-1})\tilde{p}_{i-1} - J_{i-1,i-2}\tilde{p}_{i-2})/J_{i,i-1}}{J_{i,i-1}} \)
10: \( \textbf{end for} \)
11: \( p = L_{1,1}\tilde{p} \)
12: \( \mu_0 = 1/L_{2,1}^2 \)
13: \( \langle 1, \tilde{p}_1 \rangle = \mu_0/\sqrt{\mu_0} \)
14: \( \langle 1, \tilde{p}_2 \rangle = (\sqrt{\mu_0} + c - J_{1,1}\langle 1, \tilde{p}_1 \rangle)/J_{21} \)
15: \( \textbf{for} \ i = 2 \ : \ n \ \textbf{do} \)
16: \( \langle 1, \tilde{p}_i \rangle = ((c - J_{i-1,i-1})\langle 1, \tilde{p}_{i-1} \rangle - J_{i-2,i-2}\langle 1, \tilde{p}_{i-2} \rangle)/J_{i,i-1} \)
17: \( \textbf{end for} \)
18: \( \langle p_j, p_k \rangle_\lambda = L_{k,k-1}\langle p_j, \tilde{p}_{k-1} \rangle_\lambda + L_{k,k}\langle p_j, \tilde{p}_k \rangle_\lambda \)

5.2 Implicit Double Shift

The goal of this section is to compute the matrix \( J \) corresponding to the measure \( d\hat{\lambda}(t) = r(t)d\lambda(t) \), where \( r(t) \) is an irreducible quadratic rational weight function, if we are given \( J_2 \). We can carry out this process in one step if we consider the second degree polynomial of \( J \). To derive the second degree polynomial of \( J_2 \) we start with

\[
L^T L = L_{2,1}^T L_{11} L_2 \]

\[
= L_{2,1}^T (J_1 - c_1 I - \delta_1 e_n e_n^T) L_2
\]

\[
= L_{2,1}^T J_1 L_2 - c_1 L_{2,1}^T L_2 - \delta_1 L_{2,1}^T e_n e_n^T L_2
\]

\[
= L_{2,1}^T (L_2 L_2^T + c_2 I)L_2 - c_1 (J_2 - c_2 I - \delta_2 e_n e_n^T) - \delta_1 L_{2,1}^T e_n e_n^T L_2
\]

\[
= (J_2 - c_2 I - \delta_2 e_n e_n^T)^2 + (c_2 - c_1)(J_2 - c_2 I - \delta_2 e_n e_n^T) - \delta_1 L_{2,1}^T e_n e_n^T L_2
\]

\[
= J_2^2 - 2c_2 J_2 + c_2^2 I - \delta_2 J_2 e_n e_n^T - \delta_2 e_n e_n^T J_2 + 2c_2 \delta_2 e_n e_n^T + \delta_2^2 e_n e_n^T + (c_2 - c_1)(J_2 - c_2 I - \delta_2 e_n e_n^T) - \delta_1 L_{2,1}^T e_n e_n^T L_2
\]

\[
= J_2^2 - (c_1 + c_2)J_2 + c_1 c_2 I - \delta_2 J_2 e_n e_n^T - \delta_2 e_n e_n^T J_2 + (c_1 + c_2) \delta_2 e_n e_n^T - \delta_1 L_{2,1}^T e_n e_n^T L_2.
\]
Therefore the second degree polynomial of \(J_2\) is equal to the following.

\[
J_2^2 - 2\text{Re}(c)J_2 + |c|^2I - \delta_2 J_2 e_n e_n^T - \delta_2 e_n e_n^T J_2 + (c_1 + c_2)\delta_2 e_n e_n^T - \delta_1 L_2^T e_n e_n^T L_2 = (L_1 L_2)^T (L_1 L_2),
\]

where the terms \(\delta_1 = \frac{\bar{b}_n}{(L_1)_{2,n}}\) and \(\delta_2 = \frac{\bar{b}_n}{(L_2)_{2,n}}\). These perturbations only affect the lower \(2 \times 2\) block of the matrix. To obtain the desired matrix, we need to first find the Cholesky factor \(L\) by performing the reverse Cholesky algorithm on (5.4). In this case, the matrix that comes from (5.4), \(J_2^2\), is a pentadiagonal matrix, as opposed to the tridiagonal matrices, \(J_n\), that we have previously seen.

### 5.2.1 Interval for guessing

An objective function for guessing the unknown entries is derived here, similar to what was done in the linear case that

\[
\langle p_j, p_k \rangle_\lambda = 0 \quad \text{for} \quad j \neq k. \tag{5.5}
\]

That is, the inner product of two constructed polynomials of differing degrees is actually zero with respect to the measure \(d\lambda(t)\). The necessary initial values are

\[
\begin{align*}
\langle 1, \tilde{p}_0 \rangle_\lambda &= \tilde{p}_0 \mu_0 \\
\langle 1, \tilde{p}_1 \rangle_\lambda &= m \mu_1 + b \mu_0,
\end{align*}
\]

where the relationship between \(m\) and \(b\) is given by

\[
\tilde{p}_1(t) = m t + b
\]

\[
0 = \langle \tilde{p}_1, \tilde{p}_0 \rangle_\lambda = m \tilde{p}_0 \langle t, 1 \rangle_\lambda + b \tilde{p}_0 \langle 1, 1 \rangle_\lambda = \tilde{p}_0 (m \mu_1 + b \mu_0)
\]

where \(m \mu_1 + b \mu_0 = 0\) gives \(b = -\frac{m \mu_1}{\mu_0}\). Taking the inner product \(\langle \tilde{p}_1, \tilde{p}_1 \rangle_\lambda = \langle m t + b, m t + b \rangle\) implies that \(m^2 \mu_2 + 2m b \mu_1 + b^2 \mu_0 = 1\). Substituting the previously found formula for \(b\) gives \(1 = m^2 (\mu_2 - \frac{\mu_1^2}{\mu_0})\). From the above, we get, in general,

\[
\langle p_j, p_k \rangle_\lambda = l_{k,k-1} \langle p_j, \tilde{p}_{k-1} \rangle_\lambda + l_{k,k-2} \langle p_j, \tilde{p}_{k-2} \rangle_\lambda + l_{k,k} \langle p_j, \tilde{p}_k \rangle_\lambda \tag{5.6}
\]

where

\[
\begin{align*}
\langle p_j, \tilde{p}_{k-1} \rangle_\lambda &= \langle p_j(c) + p_j[c, \bar{c}](t - c) + (t^2 + c^2) q_{j-2}, \tilde{p}_{k-1} \rangle_\lambda \\
&= \langle p_j(c) + p_j[c, \bar{c}](t - c), \tilde{p}_{k-1} \rangle_\lambda + \langle (t^2 + c^2) q_{j-2}, \tilde{p}_{k-1} \rangle_\lambda \\
&= \langle p_j(c) + p_j[c, \bar{c}](t - c), \tilde{p}_{k-1} \rangle_\lambda + \langle q_{j-2}, \tilde{p}_{k-1} \rangle_\lambda,
\end{align*}
\]
where \( p_j[c, \vec{c}] = \frac{p_j(c) - p_j(\vec{c})}{c - \vec{c}} \) is a divided difference, with \( q_{j-2} \) being a polynomial of degree \( j - 2 \), and when \( j - 2 < k - 1 \) the last term of the above is zero. Then we get the following:

\[
\langle p_j, \tilde{p}_{k-1} \rangle_\lambda = \langle p_j(c) + p_j[c, \vec{c}](t - c), \tilde{p}_{k-1} \rangle_\lambda \\
= \langle p_j[c, \vec{c}]t, \tilde{p}_{k-1} \rangle_\lambda + \langle p_j(c) - p_j[c, \vec{c}]c, \tilde{p}_{k-1} \rangle_\lambda \\
= p_j[c, \vec{c}] \langle t, \tilde{p}_{k-1} \rangle_\lambda + (p_j(c) - p_j[c, \vec{c}]c) \langle 1, \tilde{p}_{k-1} \rangle_\lambda.
\]

(5.7)

Let us now derive a formula for \( \langle t, \tilde{p}_{k-1} \rangle_\lambda \) where \( k > 1 \)

\[
\langle t, \tilde{p}_{k-1} \rangle_\lambda = \left\langle t, \frac{1}{\tilde{p}_{k-1}} \left[ (t - \alpha_{k-1}) \tilde{p}_{k-2} - \beta_{k-2} \tilde{p}_{k-3} \right] \right\rangle_\lambda \\
= \frac{1}{\tilde{p}_{k-1}} \langle t, t \tilde{p}_{k-2} \rangle_\lambda - \frac{\alpha_{k-1}}{\beta_{k-1}} \langle t, \tilde{p}_{k-2} \rangle_\lambda - \frac{\beta_{k-2}}{\beta_{k-1}} \langle t, \tilde{p}_{k-3} \rangle_\lambda \\
= \frac{1}{\tilde{p}_{k-1}} \langle 1, \tilde{p}_{k-2} \rangle_\lambda - \frac{c^2}{\tilde{p}_{k-1}} \langle 1, \tilde{p}_{k-2} \rangle_\lambda - \frac{\alpha_{k-1}}{\beta_{k-1}} \langle t, \tilde{p}_{k-2} \rangle_\lambda - \frac{\beta_{k-2}}{\beta_{k-1}} \langle t, \tilde{p}_{k-3} \rangle_\lambda,
\]

where \( \langle t, \tilde{p}_0 \rangle_\lambda = \frac{\mu_0}{\sqrt{\mu_0}} \). Now that we are able to compute \( \langle t, \tilde{p}_{k-1} \rangle \) from (2.1) with known information, we give a way to compute \( \langle 1, \tilde{p}_{k-1} \rangle_\lambda \) similarly as follows:

\[
\langle 1, \tilde{p}_{k-1} \rangle_\lambda = \left\langle 1, \left[ \frac{1}{\tilde{p}_{k-1}} (t - \tilde{\alpha}_{k-1}) \tilde{p}_{k-2} - \tilde{\beta}_{k-2} \tilde{p}_{k-3} \right] \right\rangle_\lambda \\
= \frac{1}{\tilde{p}_{k-1}} \langle t, \tilde{p}_{k-2} \rangle_\lambda - \frac{\tilde{\alpha}_{k-1}}{\tilde{p}_{k-1}} \langle 1, \tilde{p}_{k-2} \rangle_\lambda - \frac{\tilde{\beta}_{k-2}}{\tilde{p}_{k-1}} \langle 1, \tilde{p}_{k-3} \rangle_\lambda,
\]

where \( \langle 1, \tilde{p}_0 \rangle = \frac{\mu_0}{\sqrt{\mu_0}} \).

### 5.2.2 Similarity transformation involving \( L \)

The matrix we wish to obtain, \( J \), is related to the given matrix \( J_2 \) by \( L \), where \( L = L_1 L_2 \). To find the relationship, first we note that

\[
L_1^T = (J_1 - c_1 I - \delta_1 e_n e_n^T)L_1^{-1}.
\]

Now, we substitute this into

\[
J - c_1 I = L_1 L_1^T,
\]
where we have the following:

\[ J = L_1 (J_1 - c_1 I - \delta_1 e_n e_n^T) L_1^{-1} + c_1 I \]

\[ J = L_1 (L_2 L_2^T + c_2 I - c_1 I - \delta_1 e_n e_n^T) L_1^{-1} + c_1 I \]

\[ L_1^{-1} J L_1 = J_1 - c_1 I - \delta_1 e_n e_n^T + c_1 I \]

\[ L_2^{-1} L_1^{-1} J L_1 L_2 = L_2^{-1} (L_2 L_2^T + c_2 I) L_2 \]

\[ L_2^{-1} L_1^{-1} J L_1 L_2 = L_2^{-1} L_2 + c_2 I + \delta_2 e_n e_n^T - \delta_1 L_2^{-1} e_n e_n^T L_2 \]

\[ L_2^{-1} L_1^{-1} J L_1 L_2 \approx J_2, \quad (5.8) \]

where the error, \( \delta_2 e_n e_n^T - \delta_1 L_2^{-1} e_n e_n^T L_2 \), is zero except for the last row. From the above, now we can multiply on the left by \( L \), and multiply on the right by \( L^{-1} \) to get \( J = LJ_2 L^{-1} \). We verify that this is true, starting with

\[ J \approx LJ_2 L^{-1} \]

\[ = (L_1 L_2)(L_2 L_2^T + c_2 I + \delta_2 e_n e_n^T)(L_1 L_2)^{-1} \]

\[ = (L_1 L_2)(L_2 L_2^T + c_2 I + \delta_2 e_n e_n^T)(L_2^{-1} L_1^{-1}) \]

\[ = L_1 L_2 L_2^T L_2 L_1^{-1} + c_2 L_1 L_2 L_2^{-1} L_1^{-1} + \delta_2 L_1 L_2 e_n e_n^T L_2^{-1} L_2^{-1} \]

\[ = L_1 (J_1 - c_2 I) L_1^{-1} + c_2 I + \delta_2 L_1 L_2 e_n e_n^T L_2^{-1} L_2^{-1} \]

\[ = L_1 (L_1^T L_1 + c_1 I + \delta_1 e_n e_n^T - c_2 I) L_1^{-1} + c_2 I + \delta_2 L_1 L_2 e_n e_n^T L_2^{-1} L_1^{-1} \]

\[ = L_1 L_1^T L_1 L_1^{-1} + c_1 I + \delta_1 L_1 e_n e_n^T L_1^{-1} - c_2 I + c_2 I + \delta_2 L_1 L_2 e_n e_n^T L_2^{-1} L_1^{-1} \]

\[ = L_1 L_1^T + c_1 I + \delta_1 L_1 e_n e_n^T L_1^{-1} + \delta_2 L_1 L_2 e_n e_n^T L_2^{-1} L_1^{-1} \]

\[ = J + \delta_1 L_1 e_n e_n^T L_1^{-1} + \delta_2 L_1 L_2 e_n e_n^T L_2^{-1} L_1^{-1} \]

From the last step we can see that the terms \( \delta_1 L_1 e_n e_n^T L_1^{-1} \) and \( \delta_2 L_1 L_2 e_n e_n^T L_2^{-1} L_1^{-1} \) perturb the matrix \( J \), but only in the lower \( 2 \times 2 \) block, so we settle for a \( J \) that has size \( (n - 1) \times (n - 1) \). In this computation, it is necessary to invert the matrix \( L \) to carry out the similarity transformation. Usually inversion is an expensive numerical procedure; however, if done strategically, the expensive of the computation can be reduced. For a full matrix, inversion costs \( O(\frac{2}{3} n^3) \) floating point operations by Gaussian elimination. \( L \) has the structure of being lower three-diagonal. If we factor \( L = LD \) where \( L \) is a unit lower matrix and \( D \) is the diagonal elements, we can rewrite the similarity transformation as

\[ J = \tilde{L}_1 \ldots \tilde{L}_{n-1} D J_2 D^{-1} L_{n-1}^{-1} \ldots L_1^{-1}, \quad (5.9) \]
where

\[
\hat{L}_1 = \begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
m_{2,1} & 1 & 0 & \ldots & 0 \\
m_{3,1} & 0 & \ddots & \vdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & \ldots & 0 & \ddots & 0 \\
0 & \ldots & \ldots & 0 & 1
\end{bmatrix},
\]

and \( m_{i,j} \) plays the role of a multiplier. In general, the matrix \( \hat{L}_j \) has the structure

\[
\hat{L}_j = \begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
0 & 1 & 0 & \ldots & 0 \\
\vdots & m_{j+1,j} & \ddots & \vdots & 0 \\
0 & m_{j+2,j} & 0 & \ddots & 0 \\
0 & 0 & \ldots & 0 & 1
\end{bmatrix}
\]

where the multipliers lie in the \( j^{th} \) column. From equation (5.9), it can be seen that we also need the inverses of \( L_{n-1}, \ldots, L_1 \). A matrix with this structure is trivial to invert, and the inverse has the structure

\[
\hat{L}_j^{-1} = \begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
0 & 1 & 0 & \ldots & 0 \\
\vdots & -m_{j+1,j} & \ddots & \vdots & 0 \\
0 & -m_{j+2,j} & 0 & \ddots & 0 \\
0 & 0 & \ldots & 0 & 1
\end{bmatrix}
\]

The following figures show the structure of \( J_2 \) as it is constructed by (5.9). It can be seen from the figures that as each multiplication on the left and right occurs, only two new nonzeros are introduced each time. Then the nonzeros are eliminated, so there is no major fill in issues with the construction of \( J_2 \).
Carrying out the product \((J - cI)(J - \bar{c}I)\),

\[
(J - cI)(J - \bar{c}I) = (L_1L_1^T)(L_1L_1^T + (c + \bar{c})I)
\]

\[
= L_1(L_1^T L_1) + L_1L_1I + (c + \bar{c})I
\]

\[
= L_1(J_1 - cJ)L_1^T + L_1L_1^T(c + \bar{c})
\]

\[
= L_1J_1L_1^T + \bar{c}L_1L_1^T
\]

\[
= L_1(J_2L_2L_2^T)L_1^T + \bar{c}L_1L_1^T
\]

\[
= L_1L_2L_2^T L_1^T + \bar{c}L_1L_1^T - \bar{c}L_1L_1^T
\]

\[
= L_1L_2L_2^T L_1^T
\]

\[
= LL^T.
\]

Similarly, we would have

\[
(J_2 - cI)(J_2 - \bar{c}I) = L^T L.
\]

Therefore, in the case where \(J\) and \(J_2\) are large enough so that the perturbation terms involving \(\delta_1\) and \(\delta_2\) make no meaningful contribution, \(J_2\) is obtained by two iterations of the \(LR\) algorithm applied to \(J\). In Chapter 6, we discuss the case that no iteration is used. In that case, the reversal procedure is undoing two iterations of the \(LR\) algorithm. Also, since the \(LR\) algorithm preserves bandwidth with each row or column operation, the reversal procedure does so as well. Therefore, the similarity transformation (5.8) can be carried out in \(O(n)\) floating point operations. The \(QR\) factorization is analogous to the previously discussed \(LR\) factorization. In both methods, the reversal procedure reverses the order of the matrix multiplication. The similarity transformation involving \(Q = [(J - cI)L_1^{-T}L_2^{-T}]^H\) [5], not to be confused with the \(Q\) from the \(RQ\) algorithm that is real, using Givens rotations is given below:

\[
J = QJ_2Q^H.
\]

Below, we show that the matrix \(Q\) is orthogonal:

\[
Q^HQ = [(J - cI)L_1^{-T}L_2^{-T}]^H[(J - cI)L_1^{-T}L_2^{-T}]
\]

\[
= [JL_1^{-T}L_2^{-T} - cL_1^{-T}L_2^{-T}] [JL_1^{-T}L_2^{-T} - cL_1^{-T}L_2^{-T}]
\]

\[
= [L_2^{-1}L_1^{-1}J - \bar{c}L_2^{-1}L_1^{-1}L_1^{-1}][JL_1^{-T}L_2^{-T} - cL_1^{-T}L_2^{-T}]
\]

\[
= L_2^{-1}L_1^{-1}J^2L_1^{-T}L_2^{-T} - cL_2^{-1}L_1^{-1}JL_1^{-T}L_2^{-T} - \bar{c}L_2^{-1}L_1^{-1}JL_1^{-T}L_2^{-T} + |c|^2L_2^{-1}L_1^{-1}L_1^{-T}L_2^{-T}
\]

\[
= L^{-1}(J^2 - (c + \bar{c})J + I)L^{-T}
\]

\[
= L^{-1}LL^TL^{-T}
\]

\[
= I.
\]

The drawback to using this \(Q\) is that it is complex.
Algorithm 16 Main: This function is the main function called. The parameters $[a, b], w, n, x$ and $c$ are given as input. This function calls the test case algorithm (6) to produce the necessary Jacobi matrices, moments, and orthonormal polynomials. It then calls Broyden’s method algorithm (11) that takes the irreducible quadratic reversal algorithm (17) as the function handle $f$.

1: $[J, \tilde{J}, \mu, \tilde{\mu}, q(t), \tilde{q}(t)] = \text{testcase}(a, b, r, w, n)$
2: $L_1 = \text{cholesky}(\tilde{J} - cI_n)$
3: $L_1 = (L_1)_{1:n-1,1:n-1}$
4: $J_1 = L_1^T L + cI_{n-1}$
5: $L_2 = \text{cholesky}(J_1 - \tilde{c} * I_{n-1})$
6: $L_2 = (L_2)_{1:n-2,1:n-2}$
7: $L = r(L_1 * L_2)$
8: $J_2^2 = J^2 + |c|^2 I_{n-2}$
9: $E = L_1^T L - J_2^2$
10: $[x] = \text{broyden}(x, f)$
11: $L = \text{reversecholesky}(J_2^2)$
12: $J = LJ_2L^{-1}$

Algorithm 17 Irreducible Quadratic Reversal-Implicit Double Shift: This algorithm takes as input the matrix $J$, the moments $\mu$, and $\tilde{\mu}$, and gives as output the inner products that we want to be zero.

1: for $i = 1 : n$ do
2: \hspace{1em} $\tilde{p}_i = ((c - J_{i-1,i-1}) \ast \tilde{p}_{i-1} - J_{i-1,i-2} \ast \tilde{p}_{i-2}) / J_{i,i-1}$
3: end for
4: \hspace{1em} $\langle p_j, \tilde{p}_0 \rangle_\lambda = \tilde{p}_0(p_j[c, \tilde{c}]\mu_1 + (p_j[c] - p_j[c - \tilde{c}]\mu_0))$
5: for $i = 1 : n$ do
6: \hspace{2em} $\langle 1, \tilde{p}_{k-1} \rangle_\lambda = \frac{1}{\beta_{k-1}} \left( \langle 1, t\tilde{p}_{k-2} \rangle_\lambda + \tilde{\alpha}_{k-1} \langle 1, \tilde{p}_{k-2} \rangle_\lambda - \tilde{\beta}_{k-2} \langle 1, \tilde{p}_{k-3} \rangle_\lambda \right)$
7: end for
8: for $k = 1 : n$ do
9: \hspace{2em} $\langle t, \tilde{p}_{k-1} \rangle_\lambda = \frac{1}{\beta_{k-1}} \left( \langle 1, \tilde{p}_{k-2} \rangle_\lambda - c^2 \langle 1, \tilde{p}_{k-2} \rangle_\lambda - \tilde{\alpha}_{k-1} \langle t, \tilde{p}_{k-2} \rangle_\lambda - \tilde{\beta}_{k-2} \langle t, \tilde{p}_{k-3} \rangle_\lambda \right)$
10: end for
11: for $j = 1 : n$ do
12: \hspace{2em} $\langle p_j, \tilde{p}_{k-1} \rangle_\lambda = p_j[c, \tilde{c}] \langle t, \tilde{p}_{k-1} \rangle_\lambda + p_j[c] - (p_j[c, \tilde{c}]c) \langle 1, \tilde{p}_k \rangle_\lambda$
13: end for
14: $\langle p_j, p_k \rangle_\lambda = L_{k,k-1} \langle p_j, \tilde{p}_{k-1} \rangle_\lambda + L_{k,k-2} \langle p_j, \tilde{p}_{k-2} \rangle_\lambda$
Chapter 6

NUMERICAL EXPERIMENTS

In this chapter, we will discuss how the test cases are produced, and the results from numerical experiments using the methods of this dissertation. The test case Algorithm 6 makes a call to the function `lgwt.m` obtained from MATLAB Central, written by Greg von Winckle, that produces the Gauss quadrature nodes and weights to generate the test cases. We use Gauss quadrature for this instead of the recurrence relation with polynomial coefficients because the latter method of generating a test case suffers from catastrophic cancellation.
Algorithm 18 Test Case: This algorithm produces the matrices $J$ and $\tilde{J}$, the moments $\mu_0$ and $\tilde{\mu}_0$ for the linear case and the moments $\mu_0, \mu_1, \tilde{\mu}_0, \tilde{\mu}_1$ for the quadratic case, and the orthonormal polynomials $q_1(t), \ldots, q_n(t)$. The function takes in as input the interval $[a, b]$, the dimension $n$, the weight function $d\lambda(t)$, and the modification measure $r(t)$.

1: $md = 2(n - 1) + 1 + \text{length}(w - 1)$
2: $ng = \lceil md + 1 \rceil / 2$
3: $[xg, wg] = lgwt(ng, a, b)$
4: $wtg = w(xg)$
5: $rtg = r(xg)$
6: if $r$ is a linear modification then
   7: $\mu_0 = wg^T * wtg$
   8: $\tilde{\mu}_0 = wg^T(wtg / rtg)$
   9: else
  10: $\mu_0 = wg^T wtg$
  11: $\mu_1 = wg^T(xg * wtg)$
  12: $\tilde{\mu}_0 = wg^T * (wtg / rtg)$
  13: $\tilde{\mu}_1 = wg^T * (xg * wtg / rtg)$
end if
14: $\beta_0 = \sqrt{\mu_1}$
15: for $j = 1 : n$ do
  16: $\alpha_j = wg^T(wtg * q_j^2 * xg)$
  17: $J_{j, j} = \alpha_j$
  18: $p_j = (xg - \alpha_j) * q_{j - 1} - \beta_0 * q_{j - 1}$
  19: $\beta_j = \sqrt{wg^T(wtg * p_j^2)}$
  20: $J_{j, j+1} = \beta_j$
  21: $J_{j+1, j} = \beta_j$
  22: $q_{j - 1} = p_{j - 1} / \beta_j$
end for
14: $\beta_0 = \sqrt{\tilde{\mu}_1}$
15: for $j = 1 : n$ do
  16: $\alpha_j = wg^T((wtg * q_j^2 * xg) / rtg)$
  17: $\tilde{J}_{j, j} = \alpha_j$
  18: $p_{j - 1} = (xg - \alpha_j) * q_{j - 1} - \beta_0 * q_{j - 2}$
  19: $\beta_j = \sqrt{wg((wtg * p_j^2) / rtg)}$
  20: $\tilde{J}_{j, j+1} = \beta_j$
  21: $\tilde{J}_{j+1, j} = \beta_j$
  22: $\tilde{q}_{j - 1} = p_{j - 1} / \beta_j$
end for

6.1 Linear Factors

In the following examples the Jacobi matrix $J_n$ that is being modified is produced by the previous Algorithm 6. In each example, we vary the parameters $n$ (dimension of the Jacobi
matrix), the shift $c$, and the interval $[a, b]$.

1. **Example 1:** For this example we modify the original measure $d\lambda(t) = dt$ by a weight function $r(t) = 1/(t - c)$ and we choose the following parameters:

   - $c = -2$
   - interval $[-1, 1]$
   - $n = 10$.

The following Figures 6.1 and 6.2 show the orthonormal polynomials $q(t) = [q_0(t), q_1(t), \ldots, q_n(t)]$ and the modified polynomials $\tilde{q}(t) = [\tilde{q}_0(t), \tilde{q}_1(t), \ldots, \tilde{q}_n(t)]$ that correspond to $J_n$ and $\tilde{J}_n$ respectively.

![Figure 6.1: Polynomials corresponding to $J_n$.](image)

We also show the same polynomials from Figures 6.1 and 6.2 on larger intervals that contain the shift $c$ in Figures 6.3 and 6.4. It can be seen from the latter two figures that the polynomials are reasonably behaved on the interval $[a, b]$, however they tend to grow rapidly outside of $[a, b]$. Therefore, a shift that is moderately far outside of the interval leads to dealing with polynomials that have rapid growth and are not well behaved.
Figure 6.2: Polynomials corresponding to $\tilde{J}_n$.

Figure 6.3: Polynomials corresponding to $J_n$ on $[(c-1), b]$. 
Figure 6.4: Polynomials corresponding to $\tilde{J}_n$ on $[(c - 1), b]$.

We produce the following error matrix by

$$\text{err} = J_t - J_{\text{mod}}$$

where

$$J_t = \text{real}(L*L' + cI)$$

$$L = \text{reversechol}(J - cI - d1*(e_n*e_n')),$$

$$d1 = \frac{\beta_n}{r},$$

where $r$ is the chosen root of the quartic equation $t^4 + (c - \alpha_{n-1})t^2 + \beta_n^2$, and $e_n$ is the $n^{th}$ column vector from the identity matrix.
The matrix 2-norm is defined by 

$$
\|A\|_2 = \sigma_{\text{max}}(A),
$$

where $\sigma_{\text{max}}(A)$ is the largest singular value of $A$. The error matrix, for this example, is the matrix 2-norm of the error 

$$
\text{errnorm} = 7.8408 \times 10^{-8}.
$$

The following Figure 6.5 is a plot of the objective function $f$ where the y-axis represents the inner products $\langle p_j, p_k \rangle_\lambda$.

Figure 6.5: Objective function $f$. 
(a) This example is a variation of the previous example where all of the parameters are the same except we choose shift \( c = -1.1 \). The matrix 2-norm of the error matrix for this example is

\[
\text{errnorm} = 3.6099 \times 10^{-8}.
\]

The following Figure 6.6 is a plot of the objective function \( f \) where the \( y \)-axis represents the inner products \( \langle p_j, p_k \rangle_{\lambda} \).

(b) This example is a variation of the previous example where all of the parameters are the same except we choose shift \( c = -3 \).

The matrix 2-norm of the error matrix for this example is

\[
\text{errnorm} = 2.4939 \times 10^{-4}.
\]

The following Figure 6.7 is a plot of the objective function \( f \) where the \( y \)-axis represents the inner products \( \langle p_j, p_k \rangle_{\lambda} \). For this shift, we see that the result is less accurate than previous results. From the Figure 6.7, it can be seen that the \( y \) values are much smaller than in Figure 6.6. This function is poorly conditioned compared to previous examples, and therefore, the result is less accurate.
2. **Example 2:** For this example we modify the original measure $d\lambda(t) = dt$ by a weight function $r(t) = 1/(t - c)$ and we choose the following parameters:

- interval $[0, 1]$
- $n = 10$.

The shift $c$ will be varied during this example, to show how the error behaves as the shift moves further outside of the interval. The following Figures 6.8 and 6.9 show the orthonormal polynomials $q$ and the modified polynomials $\tilde{q}(t)$ that correspond to $J_n$ and $\tilde{J}_n$ respectively. As in the previous example, we also show the same polynomials from Figures 6.8 and 6.9 on larger intervals that contain the shift $c$ in Figures 6.10 and 6.11.

The Table 6.1 shows different shifts close to the interval, starting at -1 and slowly progressing away from the interval, and the behavior of the error as $c$ decreases. We see that as the shift $c$ approaches $-2$, the error increases.

The following Table of figures 6.2 is a plot of the objective function $f$ where the y-axis represents the inner products $\langle p_j, p_k \rangle_\lambda$. Here we see that the behavior of each graph
Figure 6.8: Polynomials corresponding to $J_n$.

Figure 6.9: Polynomials corresponding to $\tilde{J}_n$. 
Figure 6.10: Polynomials corresponding to $J_n$ on $[(c-1), b]$. 

Figure 6.11: Polynomials corresponding to $\tilde{J}_n$ on $[(c-1), b]$. 
is smooth, until we reach the last graph where $c = -2$.

3. **Example 3:** For this example we modify the original measure $d\lambda(t) = dt$ by a weight function $r(t) = 1/(t - c)$ and we choose the following parameters:

- $c = -2$
- interval $[-1, 0]$
- $n = 10$.

The following Figures 6.12 and 6.13 show the orthonormal polynomials $q$ and the modified polynomials $\tilde{q}(t)$ that correspond to $J_n$ and $\tilde{J}_n$ respectively. Again, we also show the same polynomials from Figures 6.12 and 6.13 on larger intervals that contain the shift $c$ in Figures 6.14 and 6.15.

The matrix 2-norm of the error matrix for this example is $\text{errnorm} = 2.5416 \times 10^{-5}$.

The following Figure 6.16 is a plot of the objective function $f$ where the $y$-axis represents the inner products $\langle p_j, p_k \rangle_{\lambda}$.

(a) This example is a variation of Example 3 where all of the parameters are the same except we choose shift $c = -1.1$.

The matrix 2-norm of the error matrix for this example is $\text{errnorm} = 3.2134 \times 10^{-7}$.
The following Figure 6.17 is a plot of the objective function $f$ where the y-axis represents the inner products $\langle p_j, p_k \rangle_\lambda$.

(b) This example is a variation of the previous example where all of the parameters are the same except we choose shift $c = -3$. 
Figure 6.12: Polynomials corresponding to $J_n$.

Figure 6.13: Polynomials corresponding to $\tilde{J}_n$. 
Figure 6.14: Polynomials corresponding to $J_n$ on $[(c - 1), b]$.

Figure 6.15: Polynomials corresponding to $\tilde{J}_n$. 
Figure 6.16: Objective function $f$.

Figure 6.17: Objective function $f$. 
The matrix 2-norm of the error matrix for this example is
\[ \text{errnorm} = 0.9396. \]

The following Figure 6.18 is a plot of the objective function \( f \) where the y-axis represents the inner products \( \langle p_j, p_k \rangle_\lambda \).

4. **Example 4:** In this example, we will modify the original measure \( d\lambda(t) = dt \) by the same weight function \( r(t) = (t - c)^{-1} \) as the previous examples. However, this time we will vary the size of the matrix \( n \), and choose a value of \( c = -1.1 \) on the interval \([-1, 1]\). We choose a shift close to the interval because the method performs well under these criteria, so that we can analyze the effects of changing the dimension \( n \). The polynomials for this example can be seen in test case 1, in Figures 6.1, 6.2, 6.3, and 6.4. The following Table 6.3 gives the errors for each value of \( n \). In the case that \( n = 40 \) we see the error is much worse than for smaller values of \( n \). This is due to excessive roundoff error from computing the inner product \( \langle 1, \tilde{p}_k \rangle_\lambda \).

The following Table of figures 6.4 is a plot of the objective function \( f \) where the y-axis represents the inner products \( \langle p_j, p_k \rangle_\lambda \).
<table>
<thead>
<tr>
<th>dimension $n$</th>
<th>error $e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 20$</td>
<td>$e = 3.1708 \times 10^{-8}$</td>
</tr>
<tr>
<td>$n = 30$</td>
<td>$e = 4.3347 \times 10^{-7}$</td>
</tr>
<tr>
<td>$n = 40$</td>
<td>$e = 0.0342$</td>
</tr>
</tbody>
</table>

*Table 6.3: Example 4 Errors*

### 6.2 Irreducible Quadratic Factors

1. **Example 1:** For this example we will vary the values of shift $c$ to analyze the behavior and we modify the original measure $d\lambda(t) = dt$ by a weight function $r(t) = 1/(t^2 + c^2)$ for the following parameters:
   
   - interval $[-1, 1]$
   - $n = 10$.

The following Figures 6.19 and 6.20 show the orthonormal polynomials $q$ and the modified polynomials $\tilde{q}(t)$ that correspond to $J_n$ and $\tilde{J}_n$ respectively.

*Figure 6.19: Polynomials corresponding to $J_n$.***
Table 6.4: Example 4 Graphs
Figure 6.20: Polynomials corresponding to $\tilde{J}_n$.

<table>
<thead>
<tr>
<th>shift $c$</th>
<th>error $e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c = -2i$</td>
<td>$e = 1.4483 \times 10^{-4}$</td>
</tr>
<tr>
<td>$c = -1.5i$</td>
<td>$e = 3.7043 \times 10^{-4}$</td>
</tr>
<tr>
<td>$c = -1.25i$</td>
<td>$e = 6.3396 \times 10^{-4}$</td>
</tr>
<tr>
<td>$c = -1.1i$</td>
<td>$e = 8.9442e \times 10^{-4}$</td>
</tr>
</tbody>
</table>

Table 6.5: Example 1 Errors

The Table 6.5 gives the errors for different shifts.
The error matrix below shows how the error for the shift $c = -2i$ increases as $n$ approaches the last entry of the matrix.

$$
\text{err1} = \\
1.0e-03 * \\
\begin{bmatrix}
0.0000 & 0.0000 & 0 & 0 & 0 & 0 & 0 \\
-0.0000 & -0.0000 & -0.0000 & 0 & 0 & 0 & 0 \\
-0.0000 & -0.0000 & 0.0000 & 0.0000 & 0 & 0 & 0 \\
0.0000 & 0.0000 & -0.0000 & -0.0000 & -0.0000 & 0 & 0 \\
0.0000 & 0.0000 & -0.0000 & -0.0005 & 0.0000 & 0.0042 & 0 \\
-0.0000 & -0.0000 & 0.0004 & 0.0000 & -0.0037 & -0.0000 & -0.0041 \\
-0.0000 & -0.0005 & 0.0000 & 0.0079 & -0.0000 & -0.1446 & 0.0000
\end{bmatrix}
$$

The Figures 6.21, 6.22, 6.23, and 6.24 show the objective function $f$ for the different values of $c$ where each graph shows how each component of the vector-valued objective function varies with each component of the input and the exact value is used as a starting point.

2. Example 2: For this example we will vary the values of shift $c$ to analyze the behavior and we modify the original measure $d\lambda(t) = dt$ by a weight function $r(t) = 1/(t^2 + c^2)$.
Figure 6.22: Objective function $f$ for $c = -1.5i$.

Figure 6.23: Objective function $f$ for $c = -1.25i$
Figure 6.24: Objective function $f$ for $c = -1.1i$

<table>
<thead>
<tr>
<th>shift $c$</th>
<th>error $e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c = -2i$</td>
<td>$e = 0.0014$</td>
</tr>
<tr>
<td>$c = -3i$</td>
<td>$e = 5.9286 \times 10^{-4}$</td>
</tr>
<tr>
<td>$c = -4i$</td>
<td>$e = 2.5543 \times 10^{-4}$</td>
</tr>
<tr>
<td>$c = -5i$</td>
<td>$e = 1.2175 \times 10^{-4}$</td>
</tr>
<tr>
<td>$c = -5.5i$</td>
<td>$e = 8.7211 \times 10^{-5}$</td>
</tr>
<tr>
<td>$c = -5.75i$</td>
<td>$e = 7.4449 \times 10^{-5}$</td>
</tr>
</tbody>
</table>

Table 6.6: Example 2 Errors

for the following parameters:

- interval $[-2, 2]$
- $n = 10$.

The following Figures 6.25 and 6.26 show the orthonormal polynomials $q$ and the modified polynomials $\tilde{q}(t)$ that correspond to $J_n$ and $\tilde{J}_n$ respectively.

The Table 6.6 gives the errors for different shifts. The Figures 6.27, 6.28, 6.29, 6.30, 6.31, and 6.32 show the objective function $f$ for the different values of $c$. 
Figure 6.25: Polynomials corresponding to $J_n$.

Figure 6.26: Polynomials corresponding to $\tilde{J}_n$. 
Figure 6.27: Objective function $f$ for $c = -2i$.

Figure 6.28: Objective function $f$ for $c = -3i$. 
Figure 6.29: Objective function $f$ for $c = -4i$

Figure 6.30: Objective function $f$ for $c = -5i$
Figure 6.31: Objective function $f$ for $c = -5.5i$.

Figure 6.32: Objective function $f$ for $c = -5.75i$. 
6.3 Linear and Quadratic Factors with no Iteration

In this section, we perform the reversal method to obtain a modified Jacobi matrix by ignoring the unknown terms that we previously found by solving either a single nonlinear equation in the case of dividing by a linear factor or a system of nonlinear equations in the case of dividing by an irreducible quadratic factor. As the dimension $n$ increases the modified moments have been observed to tend to zero for a high enough degree if the objective function is ill-conditioned. Therefore, the objective function will return zero up to round off error if the degree is large enough, no matter the guess of the unknown parameter. This is equivalent to undoing one step of the LR algorithm [5] for the linear case, and two steps for the quadratic case. If we find that the modified moments do not tend to zero, then the objective function is well conditioned and the iteration can be used. To determine how large the original matrix must be, we can compute the inner products $\langle 1, \tilde{p}_k \rangle$ and $\langle t, \tilde{p}_k \rangle$ and see how many iterations it takes for them to get sufficiently small, which is similar to the approach taken in minimal solutions [5]. From the observed error in the previous numerical experiments, we see that the accuracy of the unknown entry or entries affects a portion of the lower right rows and columns of the modified matrix. Therefore, there is an upper left portion of the modified matrix that remains uncontaminated by the unknown values. We use this knowledge to obtain the modified matrix by starting with a larger dimension, and then trimming to achieve adequate accuracy. To perform this operation, we first compute $L$ by taking the reverse Cholesky factorization of the unperturbed matrix $J - cI$ in the linear case, and the second degree polynomial of $J, J^2 - \text{Re}(c)J + |c|^2I$ in the irreducible quadratic case. This $L$ of size $n$ is contaminated from not performing the necessary rank two update, so at this point, we remove some of the last rows and columns. Once $L$ is trimmed down to become more accurate, we can compute the modified Jacobi matrix by $\tilde{J} = LL^T + cI$ for the linear case and $\tilde{J} = LJJL^{-1}$ for the irreducible quadratic case. The steps to performing the reversal algorithm are as follows:

- **Linear Case:**
  1. $J - cI = L_m^T L_m$
  2. $L_n = L_m(1:n, 1:n)$
  3. $\tilde{J} = L_n L_n^T + cI = L_n JJL_n^{-1}$

- **Quadratic Case:**
  1. $J_2^2 - 2\text{Re}(c)J_2 + |c|^2I = L_m^T L_m$
The following are examples of modification by dividing by a linear factor \((t - c)\).

1. **Example 1:** In this example, we use an interval of \([-1, 1]\), starting with a matrix of dimension \(n = 50\), then trim down to dimension \(n = 40\). We vary the value of \(c\), and see that the smaller the imaginary part gets, the larger the error gets. The errors are given in Table 6.7.

2. **Example 2:** In this example, we use an interval of \([-1, 1]\), starting with a matrix of dimension \(n = 200\), then trim down to dimension different sizes for different shifts. It can be seen that the closer the shift is to the interval, the more we must trim down to achieve accuracy. If the original matrix has to be trimmed down significantly, then we must start with a larger matrix. For this example, we start with a matrix of size \(n = 200\) each time, and trim down the matrix until enough accuracy is achieved. The errors are given in Table 6.8.

The following are examples of modification by dividing by an irreducible quadratic factor \((t^2 + c^2)\).

1. **Example 1:** In this example, we use an interval of \([-1, 1]\), starting with a matrix of dimension \(n = 50\), then trim down to dimension \(n = 40\). We vary the value of \(c\), and see that the smaller the imaginary part gets, the larger the error gets. The errors are given in Table (6.9).

2. **Example 2:** In this example, we use an interval of \([0, 1]\), starting with a matrix of dimension \(n = 50\), then trim down to dimension \(n = 40\). We vary the value of \(c\), and see that the smaller the imaginary part gets, the larger the error gets. The errors are given in Table (6.10).

<table>
<thead>
<tr>
<th>shift (c)</th>
<th>error (e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c = -2)</td>
<td>(e = 8.2313 \times 10^{-16})</td>
</tr>
<tr>
<td>(c = -1.3)</td>
<td>(e = 1.1149 \times 10^{-14})</td>
</tr>
<tr>
<td>(c = -1.1)</td>
<td>(e = 2.5204 \times 10^{-9})</td>
</tr>
<tr>
<td>(c = -1.01)</td>
<td>(e = 1.3037 \times 10^{-4})</td>
</tr>
<tr>
<td>(c = -1.001)</td>
<td>(e = 0.0025)</td>
</tr>
</tbody>
</table>

*Table 6.7: Example 1 Errors*
3. **Example 3:** In this example, we use an interval of $[-1, 1]$, starting with a matrix of dimension $n = 200$, then trim down to dimension different sizes for different shifts. It can be seen that the smaller the imaginary part of the shift is, the more the matrix must be trimmed down to achieve accuracy. If the original matrix has to be trimmed down significantly, then we must start with a larger matrix. For this example, we start with a matrix of size $n = 200$ each time, and trim down the matrix until enough accuracy is achieved. The errors are given in Table 6.11.
<table>
<thead>
<tr>
<th>shift $c$</th>
<th>error $e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c = -2i$</td>
<td>$e = 1.4711 \times 10^{-15}$</td>
</tr>
<tr>
<td>$c = -i$</td>
<td>$e = 7.9155 \times 10^{-16}$</td>
</tr>
<tr>
<td>$c = -5i$</td>
<td>$e = 4.8751 \times 10^{-11}$</td>
</tr>
<tr>
<td>$c = -.3i$</td>
<td>$e = -3.6068 \times 10^{-9}$</td>
</tr>
<tr>
<td>$c = -.2i$</td>
<td>$e = 1.5933 \times 10^{-7}$</td>
</tr>
<tr>
<td>$c = -.1i$</td>
<td>$e = -0.1 \times 10^{-6}$</td>
</tr>
<tr>
<td>$c = -.001i$</td>
<td>$e = -0.001 \times 0.0035$</td>
</tr>
</tbody>
</table>

Table 6.10: Example 2 Errors

<table>
<thead>
<tr>
<th>shift $c$</th>
<th>trimmed $n$</th>
<th>error $e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c = -i$</td>
<td>$n = 190$</td>
<td>$e = 1.0939 \times 10^{-9}$</td>
</tr>
<tr>
<td>$c = -i$</td>
<td>$n = 180$</td>
<td>$e = 4.8514 \times 10^{-16}$</td>
</tr>
<tr>
<td>$c = -5i$</td>
<td>$n = 190$</td>
<td>$e = 7.986 \times 10^{-6}$</td>
</tr>
<tr>
<td>$c = -.5i$</td>
<td>$n = 180$</td>
<td>$e = 5.2795 \times 10^{-10}$</td>
</tr>
<tr>
<td>$c = -.5i$</td>
<td>$n = 170$</td>
<td>$e = 3.4931 \times 10^{-14}$</td>
</tr>
<tr>
<td>$c = -.5i$</td>
<td>$n = 160$</td>
<td>$e = 3.913 \times 10^{-16}$</td>
</tr>
<tr>
<td>$c = -.25i$</td>
<td>$n = 190$</td>
<td>$e = 0.0013$</td>
</tr>
<tr>
<td>$c = -.25i$</td>
<td>$n = 180$</td>
<td>$e = 9.2082 \times 10^{-6}$</td>
</tr>
<tr>
<td>$c = -.25i$</td>
<td>$n = 170$</td>
<td>$e = 6.5271 \times 10^{-8}$</td>
</tr>
<tr>
<td>$c = -.25i$</td>
<td>$n = 160$</td>
<td>$e = 4.6266 \times 10^{-10}$</td>
</tr>
<tr>
<td>$c = -.25i$</td>
<td>$n = 150$</td>
<td>$e = 3.279 \times 10^{-12}$</td>
</tr>
<tr>
<td>$c = -.25i$</td>
<td>$n = 140$</td>
<td>$e = 2.3385 \times 10^{-14}$</td>
</tr>
<tr>
<td>$c = -.25i$</td>
<td>$n = 130$</td>
<td>$e = 6.3916 \times 10^{-16}$</td>
</tr>
<tr>
<td>$c = -.1i$</td>
<td>$n = 190$</td>
<td>$e = 0.0233$</td>
</tr>
<tr>
<td>$c = -.1i$</td>
<td>$n = 170$</td>
<td>$e = 4.2699 \times 10^{-4}$</td>
</tr>
<tr>
<td>$c = -.1i$</td>
<td>$n = 150$</td>
<td>$e = 7.8728 \times 10^{-6}$</td>
</tr>
<tr>
<td>$c = -.1i$</td>
<td>$n = 130$</td>
<td>$e = 1.4516 \times 10^{-7}$</td>
</tr>
<tr>
<td>$c = -.1i$</td>
<td>$n = 100$</td>
<td>$e = 3.6344 \times 10^{-10}$</td>
</tr>
<tr>
<td>$c = -.1i$</td>
<td>$n = 10$</td>
<td>$e = 3.3141 \times 10^{-16}$</td>
</tr>
</tbody>
</table>

Table 6.11: Example 3
Chapter 7

Conclusion

This dissertation provides an alternate method for producing a modified Jacobi matrix, \( \tilde{J}_n \) when the underlying measure, \( d\lambda(t) \) corresponding to \( J_n \), is modified by a rational weight function. For the linear case, the reversal procedure presented in Chapter 4 requires only \( O(n) \) floating point operations. While Inverse Cholesky is the most expensive, requiring \( O(n^3) \) floating point operations, the Minimal Solutions method depends on the parameter \( m \), the number of modified moments. The resulting algorithm requires \( O(n^2) + O(m) \) operations, along with the computational expense of the LTL algorithm which is \( O(n^2) \). This can be quite expensive if \( m \) is large, and the closer the pole is to the interval, the larger \( m \) is. The use of an iterative method to converge to an unknown is not something utilized by the other known methods, and this idea works quite nicely in the linear case, since we have a bracketed solution.

For the irreducible quadratic case, it is feasible to start with a larger index of the recurrence relation, perform the reverse Cholesky to produce \( L_m \), then trim to \( L_n \) to perform the similarity transformation. Since \( L_m \) has to be produced as a matrix with dimensions \( m \times m \), this method requires \( O(m) \) floating point operations.

While we do have a less expensive method than known methods, there are still a few challenges to overcome. The iteration for the irreducible quadratic case is ill-conditioned. This is still worth looking into because it behaves linearly near the solution. Stabilization of this nonlinear system would lead to rapid convergence to a solution. Since this is a system of three unknowns, it is not expensive to solve. For the quadratic case, the initial guess is not a concern, since the objective function behaves linearly, a solution will be achieved rapidly.
BIBLIOGRAPHY


