Spring 5-1-2021

On the Treatment of Bilinear Forms Involving Matrix Functions as Perturbations of Quadratic Forms

Keelia Altheimer

ON THE TREATMENT OF BILINEAR FORMS INVOLVING MATRIX FUNCTIONS AS PERTURBATIONS OF QUADRATIC FORMS

by

Keelia Shom Altheimer-Bienemy

A Thesis
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Mathematics and Natural Sciences
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Approved by:

Dr. James V. Lambers, Committee Chair
Dr. Jiu Ding
Dr. Haiyan Tian
Dr. John Harris

May 2021

ABSTRACT

Optimizing the approximation of bilinear forms using a one-sided perturbation, where the computation of the approximate solution is generated faster while using less storage or reusing information is possible. Standard methods for approximation like Lanczos and others exist; however, separating the work a little bit and having the bilinear form relate to the quadratic form can yield a more efficient algorithm. When we have an application where the needs are different, it is helpful to understand how the processes like symmetric Lanczos and unsymmetric Lanczos relate to one another so that we can find a way to break things down to accommodate specific cases. The objective of this research is to build an efficient algorithm to approximate bilinear forms with matrix functions utilizing the quadrature rule for approximating quadratic forms without applying the standard methods of approximation.

ACKNOWLEDGMENTS

Foremost, I am thankful to God for allowing me to begin and complete such an endeavor, and for His love and grace throughout my journey.

I would like to express my sincere gratitude to my academic advisor, Dr. James V. Lambers, for rescuing me and guiding me through the research and discovery process; I will be forever grateful. You have shown me what it means to be a scholar and a genuinely caring human being. You answered my emails at all times of the day and night, supported me through the good and bad moments, shared stories of mistakes and triumphs, and laughed with me. You are remarkable, and I am thankful for you.

I would like to thank my distinguished committee members, Dr. Jiu Ding, Dr. Haiyan Tian, and Dr. John Harris for their support and valuable feedback. A special thank you to Dr. Jiu Ding for being an exemplary professor. You are the kind the professor I aspire to be. Thank you to my professors, the School of Mathematics and Natural Sciences, and the University of Southern Mississippi for the opportunity to pursue my degree and for providing me with quality instruction. A heartfelt thank you to Dr. Wallace Pye, Dr. Joseph Kolibal and Dr. Sung Lee for the roadmap and the stepping stones to accomplish my goal.

In addition, my deepest appreciation to my life advisors, Dr. Carol Spain, Dr. Kim LeDuff, Dr. Louise Perkins and Dr. Kimberley Davis for holding my hand, providing guidance every step of the way, and for teaching me the rules of the road in this pursuit as a woman, mother, daughter and wife. I thank you.

I would like to express the warmest thank you to my supportive and loving mom (Jennice T. Chandler), my dad (Vincent S. Chandler) who taught me to never quit, my sweet father (Douglas R. Altheimer) who is no longer with us, and my beautiful and wise grandmother (Charlie Mae Gilmore) who told me that I was born to be a teacher. Also, a loving thank you to my aunts, uncles, cousins and friends who cheered me on until the end no matter how long it took.

To my sweet, smart and encouraging daughter, Charlie LaRue Bienemy, you willed me to completion. I hope that my achievement teaches you to never give up on your goals and dreams; more importantly, never give up on yourself. Last but surely not least, the greatest thank you to my loving, patient and supportive husband, Reneé C. Bienemy. You are my hero and I am blessed to call you husband and friend. We did it!

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

CF - Closed-form

CR - Coefficient relations

KSS - Krylov subspace spectral methods

PDE - Partial differential equation

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 blackboard fonts are used to denote standard sets of numbers: $\mathbb{R}$ for the field of real numbers and $\mathbb{C}$ for the complex field. The capital letters, $A, B, \cdots$ are used to denote matrices, including capital greek letters, e.g., Λ for a diagnonal matrix. Functions which are denoted in boldface type typically represent vector valued functions, and real valued functions usually are set in lower case roman or greek letters. Lower case letters such as i, j, k, l, m, n and sometimes p and d are used to denote indices, while lower case bold letters are used to denote vectors. Vectors are typset in square brackets, e.g., $[\cdot]$, and matrices are typeset in parenthesese, e.g., $(\cdot)$. In general the norms are typeset using double pairs of lines, e.g., $||\cdot||$, and the abolute value of numbers is denoted using a single pairs of lines, e.g., $|\cdot|$.

Chapter 1

Introduction

1.1 Introduction

The aim of this dissertation is to optimize the approximation of bilinear forms involving infinitely differentiable matrix functions. Numerical algorithms are used and sometimes developed to solve or derive an approximate solution to problems that emerge in scientific applications. The main goal is to develop a more efficient algorithm for approximating bilinear quantities of the form

$$\mathbf{u}^T f(A)\mathbf{v} \tag{1.1}$$

where $\mathbf{u}$ and $\mathbf{v}$ are given N-vectors, A is an $N \times N$ symmetric matrix, and f is a smooth function on a given interval on the real line to produce good approximations of bilinear forms involving matrix functions using fewer steps and less storage.

Many applications require approximations of the quantity (1.1), such as approximation in signal processing (the scattering amplitude), error estimation in the conjugate method and least-squares problems, estimation of the trace of the inverse and the determinant, and spectral methods for PDEs, to name a few. Motivations for seeking approximations of bilinear quantities may include, computing a solution directly is not an option, elements of a very large matrix function are too costly to compute, approximate solutions within specific error bounds are sufficient, or simply approximating bilinear quantities can be done efficiently. Methods used to approximate bilinear forms include, but is not limited to, Conjugate Gradient Method (CG), Biconjugate Gradient Method (BiCG), Galerkin Method [7], and Aitken's Method [8]. Instead, we will consider an approximation approach based on the Lanczos algorithm to optimize bilinear forms.

In building an efficient algorithm, we seek to express every step in the algorithm of approximating bilinear forms in terms of the corresponding steps of approximating quadratic forms utilizing one quadrature rule. In [10], techniques pioneered by Golub and Meurant of approximating bilinear forms with matrix functions allows bilinear quantities to be written as matrix Riemann-Stieltjes integrals

$$\mathbf{u}^T f(A)\mathbf{v} = \int_a^b f(\lambda)\, d\alpha(\lambda) = I[f], \tag{1.2}$$

and approximated using a Gaussian quadrature rule over the spectrum of A, where nodes and weights can be computed implementing the symmetric Lanczos algorithm if $\mathbf{u} = \mathbf{v}$ and the unsymmetric Lanczos algorithm if $\mathbf{u} \neq \mathbf{v}$. When numerically approximating an integral, we are approximating our integrand $f(\lambda)$ by a polynomial that interpolates f at the nodes, and the quadrature rule integrates the interpolating polynomial exactly. Chapter 2 is devoted to background and theory of the relationship between the Gaussian quadrature rule and the Lanczos algorithms (symmetric and unsymmetric). We will also discuss the role orthogonal polynomials play in computing the coefficients of the three-term recurrence relations, since approximation via Gauss quadrature requires orthogonal polynomials with respect to our integral measure $\alpha(\lambda)$ in (1.2). The Lanczos algorithm with the correct inner-product can always find the right set of orthogonal polynomials; due to this fact, we choose Gauss quadrature compared to some other integration rules.

Introduced in [16], Lambers applied these techniques in developing a numerical method called a Krylov Subspace Spectral (KSS) method, which utilizes Gaussian quadrature to compute components of the solution of variable coefficient diffusion problems. Each Fourier coefficient of the linear parabolic partial differential equation (PDE), once spatial discretized, results in a bilinear form involving a matrix function $\mathbf{u}^T f(A)\mathbf{v}$. This method proved more accurate than the traditional spectral method. However, accurately approximating (1.1) for $\mathbf{u} \neq \mathbf{v}$ using Gaussian quadrature and implementing the unsymmetric Lanczos algorithm has its limitations. For $\mathbf{u}$ and $\mathbf{v}$ not sufficiently close, may lead to a bad approximation, and can lead to negative weights which numerically destabilizes the quadrature rule [1]. Thus, the work of Lambers in [14, 16] gives rise to our motivations for developing a more accurate approximation approach of (1.1). If we can handle the quadratic form

$$\mathbf{u}^T f(A)\mathbf{u} \tag{1.3}$$

analytically, which is what KSS methods can do, then it's not a good idea to just run unsymmetric Lanczos in the ordinary way with the following perturbation

$$\mathbf{u}^T f(A)(\mathbf{u} + d\mathbf{v}), \tag{1.4}$$

especially if $\mathbf{u}$ depends on a parameter as in KSS methods [14]. It is preferable to be able to compute bilinear form quantities directly from quadratic form quantities (1.3), instead of performing Lanczos all over again because it is wasteful.

As a result, we utilize an alternative approach to approximate bilinear forms like (1.1) for $\mathbf{u} \neq \mathbf{v}$ using the same quadrature rule discussed previously to overcome these limitations. Additionally, we keep in mind that when the initial vectors $\mathbf{u}$ and $\mathbf{v}$ are equal, positive weights are guaranteed. By continuity, if the initial vectors $\mathbf{u}$ and $\mathbf{v}$ are *close enough*,

positive weights are guaranteed. We derive the perturbation (1.4) which arises out of expressing (1.1) as a difference quotient involving the following perturbation

$$\mathbf{u}^T f(A)\mathbf{v} = \frac{1}{d}[\mathbf{u}^T f(A)(\mathbf{u}+d\mathbf{v}) - \mathbf{u}^T f(A)\mathbf{u}], \tag{1.5}$$

where A is still an $N \times N$ symmetric positive definite matrix, $\mathbf{u}$ is the left-side initial N-vector, $\mathbf{u}+d\mathbf{v}$ is the right-side initial N-vector as a pertubation of the left, and d is a small positive constant [1, 14]. Thus, we approximate the perturbed bilinear form (1.4) by carrying out the *Perturbation Approach* algorithm to compute the perturbed quantities of the modified Jacobi matrix, which we will thoroughly explain in Chapter 3.

Chapter 4 provides the numerical results to show the efficiency of our algorithm discussed previously, when approximating bilinear quantities in the form of (1.1). Additionally, the outcomes from investigating the measurement of sensitivity of Gaussian Quadrature nodes and weights, and recursion coefficients associated with the Gaussian Quadrature rules are presented in this chapter. The sensitivity measures are utilized to help choose d so that the $\hat{\beta}_j$ are all positive real numbers, because real positive $\hat{\beta}_j$ guarantees positive Gaussian quadrature weights. Thus, the sensitivity study of all quantities for $d = 0$ will ultimately aid in determining if Lanczos is ill-conditioned or not. Conclusions will follow in Chapter 5.

Chapter 2

BACKGROUND

In this chapter, we present the Lanczos algorithms, symmetric and unsymmetric (commonly referred to as nonsymmetric). We also provide the framework for the motivations to pursue this work. For our information, what we refer to today as the Lanczos algorithm was born as a numerical method developed by Cornelius Lanczos in 1950 to solve the classic *Eigenvalue Problem*, which Lanczos called the Method of Minimized Iterations[19]. Let us first define the types of problems we are working with, quadratic forms and bilinear forms for real symmetric matrices.

2.1 Quadratic and Bilinear Forms

Quadratic forms, for an N-vector $\mathbf{u}$ and an $N \times N$ real symmetric matrix A is a scalar function Q defined by

$$Q(\mathbf{u}) = \mathbf{u}^T A\mathbf{u} = \sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}u_i u_j. \tag{2.1}$$

A quadratic form is said to be postive definite whenever A is a positive definite matrix. In other words, $Q(\mathbf{u})$ is a positive definite form if and only if $\mathbf{u}^T A\mathbf{u} > 0$ for all $\mathbf{u} \neq 0$ [21].

For N-vectors $\mathbf{u}$ and $\mathbf{v}$, and an $N \times N$ symmetric matrix A, a function B of the form

$$B(\mathbf{u},\mathbf{v}) = \mathbf{u}^T A\mathbf{v} = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}u_i v_j \tag{2.2}$$

is called a **bilinear form**. A bilinear form is symmetric if the scalar product $B(\mathbf{u},\mathbf{v}) = B(\mathbf{v},\mathbf{u})$. $B(\mathbf{u},\mathbf{v})$ is positive definite if the scalar product $B(\mathbf{u},\mathbf{u}) > 0$ for all $\mathbf{u} \neq 0$ [23].

Since A is symmetric, it has real eigenvalues

$$a = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N = b, \tag{2.3}$$

and orthonormal eigenvectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_N$ such that

$$A\mathbf{q}_j = \lambda_j \mathbf{q}_j, \quad j = 1,2,...,N. \tag{2.4}$$

Expressing A in terms of its eigenvalues λ_j and eigenvectors $\mathbf{q}_j$ allows the bilinear quantity involving matrix function f to be written as follows,

$$\mathbf{u}^T f(A)\mathbf{v} = \mathbf{u}^T f\left(\sum_{j=1}^{N} \lambda_j \mathbf{q}_j \mathbf{q}_j^T\right)\mathbf{v} = \sum_{j=1}^{N} f(\lambda_j)\mathbf{u}^T \mathbf{q}_j \mathbf{q}_j^T \mathbf{v} \tag{2.5}$$

As presented in [10], it follows that the bilinear form (2.2) can also be written as a matrix Riemann-Stieltjes integral

$$\mathbf{u}^T f(A)\mathbf{v} = \sum_{j=1}^{N} f(\lambda_j)\mathbf{u}^T \mathbf{q}_j \mathbf{q}_j^T \mathbf{v} = \int_a^b f(\lambda)\, d\alpha(\lambda) = I[f], \tag{2.6}$$

where a and b are the smallest and largest eigenvalues, and the measure $\alpha(\lambda)$ is defined as

$$\alpha(\lambda) = \begin{cases} 0 & \lambda < a, \\ \sum_{j=1}^{k} \mathbf{u}^T \mathbf{q}_j \mathbf{q}_j^T \mathbf{v} & \lambda_k \leq \lambda < \lambda_{k+1}, \\ \sum_{j=1}^{N} \mathbf{u}^T \mathbf{q}_j \mathbf{q}_j^T \mathbf{v} & \lambda \geq b. \end{cases}$$

The Riemann-Stieltjes integral with matrix function f,

$$I[f] = \int_a^b f(\lambda)\, d\alpha(\lambda) \tag{2.7}$$

can be approximated using a Gaussian quadrature rule over the spectrum of A using techniques developed by Golub and Meurant, where nodes and weights can be computed implementing the symmetric Lanczos algorithm if $\mathbf{u} = \mathbf{v}$, and the unsymmetric Lanczos algorithm if $\mathbf{u} \neq \mathbf{v}$. When numerically approximating an integral (2.7), we are approximating our integrand $f(\lambda)$ by a polynomial that interpolates the function f at the nodes, and the quadrature rule integrates the interpolating polynomial exactly.

2.2 Approximation via Gaussian quadrature

This section provides an overview of the approximation of an expression of the form (2.2) using Gaussian quadrature and techniques covered in [10].

To use Gauss quadrature, we need polynomials that are orthogonal with respect to our integral measure $\alpha(\lambda)$. We choose Gauss quadrature compared to some other integration rules because we can always find the right set of orthogonal polynomials. The Lanczos algorithm with the correct inner-product will give us the orthogonal polynomials needed. The benefit of Gauss quadrature is that we start with an orthogonal polynomial of degree 0, which is a constant, and then the recurrence relation generates polynomials of degree $1, 2, \ldots, N$. Once we have the polynomial of degree N, we stop. This is equivalent to having

our $N \times N$ Jacobi matrix, and roots of the $N^t h$ degree polynomial are the nodes. In general, polynomial roots are difficult to obtain; our work is the best way to get the roots, because, if the roots we want are the eigenvalues of a symmetric matrix (the Jacobi matrix is symmetric), then we can compute the roots efficiently and they are better conditioned. Eigenvalues of a symmetric matrix are well conditioned; this is a guarantee. We are actually not using this aspect of Gauss quadrature; however, it is such an important benefit. Benefits of Gauss quadrature we are making use of include:

- The degree of accuracy
- We are guaranteed to have positive weights as long as our measure is positive and increasing. This is why it's important to choose u and v close.
- Because Gauss quadrature has positive weights, we can be sure that as the number of nodes builds to infinity that approximation really is converging.

Positive weights are good for robustness, and degree of accuracy is good for efficiency.

2.3 Symmetric Lanczos Algorithm

To accurately approximate $I[f]$ (2.7) for $\mathbf{u} = \mathbf{v}$ using Gaussian quadrature, we can apply the symmetric Lanczos algorithm to A with initial vector $\mathbf{u}$ as follows [17]:

Let $X_1 = \mathbf{x}_1$ be an $N \times 1$ given matrix, such that $X_1^T X_1 = I_2$. Let $\mathbf{x}_0 = 0$ and $\mathbf{r}_0 = \mathbf{u}$. Then, for $j = 1, 2, \ldots$, we compute

$\mathbf{r}_0 = \mathbf{u}$
$\mathbf{x}_0 = \mathbf{0}$
for $j = 1, \ldots, K$
$\quad \beta_{j-1} = \|\mathbf{r}_{j-1}\|_2$
$\quad \mathbf{x}_j = \mathbf{r}_{j-1}/\beta_{j-1}$
$\quad \mathbf{v}_j = A\mathbf{x}_j$
$\quad \alpha_j = \mathbf{x}_j^T \mathbf{v}_j$
$\quad \mathbf{r}_j = \mathbf{v}_j - \alpha_j \mathbf{x}_j - \beta_{j-1}\mathbf{x}_{j-1}$
end

where $\mathbf{r}_j$ is computed using the $3-$term recurrence relation that defines the orthogonal polynomials needed to use Gauss quadrature, and whose output is stored after each iteration.

The recursion coefficients $\alpha_1, \ldots, \alpha_K$ and $\beta_1, \ldots, \beta_K$ computed by the symmetric Lanczos iteration yields the tridiagonal matrix T_K

$$T_K = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{K-2} & \alpha_{K-1} & \beta_{K-1} \\ & & & \beta_{K-1} & \alpha_K \end{bmatrix} \tag{2.8}$$

which is the *Jacobi matrix*. Finally, we obtain the approximation

$$\begin{aligned} \mathbf{u}^T f(A)\mathbf{u} &= \int_a^b f(\lambda)\, d\alpha(\lambda) \\ &= \sum_{j=1}^{K} f(t_j) w_j + \text{error} \end{aligned}$$

where the nodes t_j are the eigenvalues of T_K produced by the symmetric Lanczos algorithm, and the weights w_j are equal to the squares of the first components of the normalized eigenvectors of T_K.

2.4 Unsymmetric Lanczos

For the case $\mathbf{u} \neq \mathbf{v}$, we can apply the unsymmetric Lanczos algorithm to A with initial vectors $\mathbf{u}$ and $\mathbf{v}$. Here we will assume that $\mathbf{u}$ and $\mathbf{v}$ are sufficiently close in effort to ensure a numerically stable quadrature rule. To accurately approximate $I[f]$ (2.7) for $\mathbf{u} \neq \mathbf{v}$ using Gaussian quadrature, we can apply the unsymmetric Lanczos algorithm to A with initial vectors $\mathbf{u}$ and $\mathbf{v}$ as follows: After K iterations, we have

$$AX_K = X_K T_K + \beta_K \mathbf{x}_{K+1} \mathbf{e}_K^T, \tag{2.9}$$

where $X_K^H X_K = I_K$, and

$$\hat{T}_K = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & & \\ \hat{\beta}_1 & \hat{\alpha}_2 & \hat{\beta}_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \hat{\beta}_{K-2} & \hat{\alpha}_{K-1} & \hat{\beta}_{K-1} \\ & & & \hat{\beta}_{K-1} & \hat{\alpha}_K \end{bmatrix} \tag{2.10}$$

is the symmetric and tridiagonal Jacobi matrix. The recursion coefficients $\hat{\alpha}_1, \ldots, \hat{\alpha}_K$ and $\hat{\beta}_1, \ldots, \hat{\beta}_K$ are produced by the Unsymmetric Lanczos iteration as follows: Let $\hat{X}_1 = \hat{\mathbf{x}}_1$ be an $N \times 1$ given matrix, such that $\hat{X}_1^T \hat{X}_1 = \hat{I}_1$. Let $\hat{\mathbf{x}}_0, \hat{\mathbf{y}}_0 = 0$, $\hat{\mathbf{p}}_0 = \mathbf{u}$ and $\hat{\mathbf{r}}_0 = \mathbf{v}$. Then, for $j = 1, 2, \ldots$, we compute

$\hat{\mathbf{p}}_0 = \mathbf{u}$
$\hat{\mathbf{r}}_0 = \mathbf{v}$
$\hat{\mathbf{x}}_0 = \mathbf{0}$
$\hat{\mathbf{y}}_0 = \mathbf{0}$
for $j = 1, \ldots, K$
$\quad \hat{\beta}_{j-1}^2 = \hat{\mathbf{p}}_{j-1}^T \hat{\mathbf{r}}_{j-1}$
$\quad \hat{\mathbf{x}}_j = \hat{\mathbf{r}}_{j-1} / \hat{\beta}_{j-1}$
$\quad \hat{\mathbf{y}}_j = \hat{\mathbf{p}}_{j-1} / \hat{\beta}_{j-1}$
$\quad \hat{\mathbf{w}}_j = A\hat{\mathbf{x}}_j$
$\quad \hat{\mathbf{t}}_j = A\hat{\mathbf{y}}_j$
$\quad \hat{\alpha}_j = \hat{\mathbf{y}}_j^T \hat{\mathbf{w}}_j$
$\quad \hat{\mathbf{r}}_j = \hat{\mathbf{w}}_j - \hat{\alpha}_j \hat{\mathbf{x}}_j - \hat{\beta}_{j-1} \hat{\mathbf{x}}_{j-1}$
$\quad \hat{\mathbf{p}}_j = \hat{\mathbf{t}}_j - \hat{\alpha}_j \hat{\mathbf{y}}_j - \hat{\beta}_{j-1} \hat{\mathbf{y}}_{j-1}$
end

Then, we have $\hat{X}_K = \begin{bmatrix} \hat{\mathbf{x}}_1 & \cdots & \hat{\mathbf{x}}_K \end{bmatrix}$ and $\hat{T}_K$ obtained from storage of $\hat{\alpha}_1, \ldots, \hat{\alpha}_K$ and $\hat{\beta}_1, \ldots, \hat{\beta}_{K-1}$. Finally, we obtain the approximation

$$\mathbf{u}^H f(A)\mathbf{v} = \sum_{j=1}^{K} f(t_j) w_j + R[f] \tag{2.11}$$

where the nodes $t_1, \ldots, t_K$ are the eigenvalues of T_K, and the weights $w_1, \ldots, w_K$ are equal to $\beta_0 = \mathbf{u}^T \mathbf{v}$ times the squares of the first components of the normalized eigenvectors of $\hat{T}_K$. The error $R[f]$ is

$$R[f] = \frac{f^{(2K)}(\eta)}{(2K)!} \int_a^b \prod_{j=1}^{K} (\lambda - t_j)^2 \, d\alpha(\lambda). \tag{2.12}$$

For general $\mathbf{u} \neq \mathbf{v}$, the tridiagonal matrix T_K is not necessarily symmetric because $\hat{\beta}_{j-1}$ might not be positive [10]. This outcome can lead to negative weights, which numerically destabilizes the quadrature rule (2.11) [1]. However, for $\mathbf{u}$ and $\mathbf{v}$ sufficiently close, the dot product $\hat{\mathbf{p}}_{j-1}^T \hat{\mathbf{r}}_{j-1} \geq 0$ would yield positive weights, and T_K would be symmetric.

2.5 Motivation for Perturbation

Quadratic forms are approximated by running the symmetric Lanczos algorithm which guarantees positive weights. When approximating a bilinear form, we can simply run unsymmetric Lanczos; however, this is not a good idea because the initial vectors $\mathbf{u}$ and $\mathbf{v}$ can be arbitrary. Therefore, we are not guaranteed positive weights, which can lead to

a bad approximation [10]. So, for $\mathbf{u} \neq \mathbf{v}$, we consider the following methods for a good approximation of bilinear forms involving matrix functions (2.6) yielding positive weights:

- For initial vectors $\mathbf{u}$ and $\mathbf{v}$ *not* close enough, one can apply the unsymmetric Lanczos algorithm to A using the block approach described in [10], with initial block $\begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix}$, which is similar to two full Lanczos runs,

- For initial vectors $\mathbf{u}$ and $\mathbf{v}$ sufficiently close, one can apply the symmetric Lanczos algorithm to A twice, $\mathbf{u}+\mathbf{v}$ and $\mathbf{u}-\mathbf{v}$; however, the subtraction involved in both formulas can produce cancellation error,

- For initial vectors $\mathbf{u}$ and $\mathbf{v}$ close enough, one can apply our *Perturbation Approach* algorithm to A with a right-side initial vector that is a perturbation of the left, $\mathbf{u}^T f(A)(\mathbf{u}+d\mathbf{v})$ as in (1.4).

In Chapter 3, we will discuss the discoveries made through the investigation of how to efficiently approximate bilinear forms involving matrix functions implementing our *Perturbation Approach*.

This approximation method, using a right-side initial vector that is a perturbation of the left, is motivated by the work of Lambers, *Fourier Spectral Methods for Variable Coefficient PDEs*. KSS Methods treats $\mathbf{u}^T f(A)\mathbf{v}$ (1.1) as a perturbation, but the challenge was how to get from the tridiagonal matrix T_K (2.4) to $\hat{T}_K$ (2.10) with as little work as possible [17]. It could be done analytically using formulas and brute force. The dilemma of KSS Methods with using unsymmetric Lanczos for Krylov subspace spectral (KSS) methods involved computing $\mathbf{u}^T f(A)\mathbf{v}$ where $\mathbf{u}$ is varying over a fixed set ($e^{i\omega x}$), where omega is an integer parameter), and $\mathbf{v}$ changing from one time-step to the next. For each case, he would have to compute $\mathbf{u}^T f(A)\mathbf{u}$, $\mathbf{u}^T f(A)(\mathbf{u}+d\mathbf{v})$ as in (1.4), and then subtract and divide to get $\mathbf{u}^T f(A)\mathbf{v}$. Since he had an easier way of handling the quadratic form case, was there a way to get the Jacobi matrix (2.10) more directly for $\mathbf{u}^T f(A)(\mathbf{u}+d\mathbf{v})$ as in (1.4) without applying Lanczos each and every time?

Approximating in the "regular" way using unsymmetric Lanczos would generate an overwhelming number of Krylov subspaces. As discussed in [17], if $\mathbf{u}$ is varying over a set of size N, it is necessary to compute $2N$ Krylov subspaces. The Perturbation Approach lets us compute only *one* Krylov subspace, which comes from generating all the $\tilde{\mathbf{v}}_j$ (detailed in Chapter 3). Every time we generate a new $\tilde{\mathbf{v}}$, we have to multiply by A. Multiplying by A is the operation we want to carryout as rarely as possible, because matrix vector multiplication is the most expensive. An additional benefit of our study is we are able to express bilinear forms using quantities we have already computed for quadratic forms, and information reuse

is efficient. Under normal circumstances, if we simply approximate $\mathbf{u}^T f(A)\mathbf{u}$ as in (1.3) and then approximate $\mathbf{u}^T f(A)\mathbf{v}$ as in (1.1), there is no information reuse.

Chapter 3

PERTURBATION METHOD

We began the pursuit of developing a more efficient algorithm for approximating bilinear forms with matrix functions (1.1) by attempting to derive Closed-form expressions for $\hat{\alpha}_j$ and $\hat{\beta}_j$, the entries of the modified Jacobi matrix (2.10). Following the unsymmetric Lanczos iteration outlined in Section 2.4, we did succeed for lower indicies without general relationship. However, simplifying the perturbed quantities beyond the third iteration became extremely difficult. As displayed in the tables below, the sum of terms in each hatted quantity compounds with each iteration. The outcome of this work, in the simplest expressions achieved, is presented here utilizing the 3-term recurrence relation $\tilde{\mathbf{v}}_j$,

$$\tilde{\mathbf{v}}_j = \begin{cases} 0 & j = -1, \\ \mathbf{v} & j = 0, \\ A\tilde{\mathbf{v}}_{j-1} - \hat{\alpha}_j \tilde{\mathbf{v}}_{j-1} - \hat{\beta}_{j-1}^2 \tilde{\mathbf{v}}_{j-2} & j \geq 1. \end{cases} \tag{3.1}$$

Table 3.1: First Iteration to Discover Closed-form Expressions

Symmetric Quantities	**Unsymmetric (Perturbed) Quantities**
$\mathbf{r}_0 = \mathbf{u}$ $(initial\,vector)$	$\hat{\mathbf{p}}_0 = \mathbf{r}_0 \qquad (left - initial\,vector)$ $\tilde{\mathbf{v}}_0 = \mathbf{v}$ $\hat{\mathbf{r}}_0 = \mathbf{u} + d\tilde{\mathbf{v}}_0 \qquad (right - initial\,vector)$
$\beta_0^2 = \mathbf{r}_0^T \mathbf{r}_0$	$\hat{\beta}_0^2 = \hat{\mathbf{p}}_0^T \hat{\mathbf{r}}_0$ $= \beta_0^2 + d\mathbf{r}_0^T \tilde{\mathbf{v}}_0$
$\mathbf{x}_1 = \dfrac{\mathbf{r}_0}{\beta_0}$	$\hat{\mathbf{x}}_1 = \dfrac{\hat{\mathbf{r}}_0}{\hat{\beta}_0}$ $= \dfrac{\mathbf{r}_0}{\hat{\beta}_0} + \dfrac{d}{\hat{\beta}_0}\tilde{\mathbf{v}}_0$ $\hat{\mathbf{y}}_1 = \dfrac{\hat{\mathbf{p}}_0}{\hat{\beta}_0}$ $= \dfrac{\mathbf{r}_0}{\hat{\beta}_0}$
$\alpha_1 = \mathbf{x}_1^T A \mathbf{x}_1$ $= \dfrac{\mathbf{r}_0^T A \mathbf{r}_0}{\beta_0^2}$	$\hat{\alpha}_1 = \hat{\mathbf{y}}_1^T A \hat{\mathbf{x}}_1$ $= \alpha_1 + \dfrac{\beta_0 d}{\hat{\beta}_0^2}\mathbf{r}_1{}^T \tilde{\mathbf{v}}_0$

Table 3.2: Second Iteration to Discover Closed-form Expressions

Symmetric Quantities	**Unsymmetric (Perturbed) Quantities**
	$\hat{\mathbf{p}}_1 = (A-\hat{\alpha}_1 I)\hat{\mathbf{y}}_1 - \hat{\beta}_0\hat{\mathbf{y}}_0$ $= \frac{\beta_0}{\hat{\beta}_0}\mathbf{r}_1 + \frac{(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0}\mathbf{r}_0$
	$\tilde{\mathbf{v}}_1 = (A-\hat{\alpha}_1 I)\tilde{\mathbf{v}}_0$ $= A\tilde{\mathbf{v}}_0 - \hat{\alpha}_1\tilde{\mathbf{v}}_0$
$\mathbf{r}_1 = (A-\alpha_1 I)\mathbf{x}_1$ $= \frac{A\mathbf{r}_0 - \alpha_1\mathbf{r}_0}{\beta_0}$	$\hat{\mathbf{r}}_1 = (A-\hat{\alpha}_1 I)\hat{\mathbf{x}}_1 - \hat{\beta}_0\hat{\mathbf{x}}_0$ $= \frac{\beta_0}{\hat{\beta}_0}\mathbf{r}_1 + \frac{(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0}\mathbf{r}_0 + \frac{d}{\hat{\beta}_0}\mathbf{v}_1 + \frac{d(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0}\mathbf{v}_0$ $= \frac{\beta_0}{\hat{\beta}_0}\mathbf{r}_1 + \frac{(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0}\mathbf{r}_0 + \frac{d}{\hat{\beta}_0}\tilde{\mathbf{v}}_1$
$\beta_1^2 = \mathbf{r}_1^T\mathbf{r}_1$	$\hat{\beta}_1^2 = \hat{\mathbf{p}}_1^T\hat{\mathbf{r}}_1$ $= \frac{\beta_0^2\beta_1^2}{\hat{\beta}_0^2} + \frac{\beta_0 d}{\hat{\beta}_0^2}\mathbf{r}_1^T\mathbf{v}_1 + \frac{\beta_0 d(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0^2}\mathbf{r}_1^T\mathbf{v}_0$ $= \frac{\beta_0^2\beta_1^2}{\hat{\beta}_0^2} + \frac{\beta_0 d}{\hat{\beta}_0^2}\mathbf{r}_1^T\tilde{\mathbf{v}}_1$
$\mathbf{x}_2 = \frac{\mathbf{r}_1}{\beta_1}$	$\hat{\mathbf{x}}_2 = \frac{\hat{\mathbf{r}}_1}{\hat{\beta}_1}$ $= \frac{\beta_0}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_1 + \frac{(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_0 + \frac{d}{\hat{\beta}_0\hat{\beta}_1}\tilde{\mathbf{v}}_1$
	$\hat{\mathbf{y}}_2 = \frac{\hat{\mathbf{p}}_1}{\hat{\beta}_1}$ $= \frac{\beta_0}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_1 + \frac{(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_0$
$\alpha_2 = \mathbf{x}_2^T A\mathbf{x}_2$ $= \frac{\mathbf{r}_1^T A\mathbf{r}_1}{\beta_1^2}$	$\hat{\alpha}_2 = \hat{\mathbf{y}}_2^T A\hat{\mathbf{x}}_2$ $= \alpha_2 + (\alpha_1-\hat{\alpha}_1) + \frac{\beta_0\beta_1 d}{\hat{\beta}_0^2\hat{\beta}_1^2}\mathbf{r}_2{}^T\mathbf{v}_1 + \frac{\beta_0\beta_1 d(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0^2\hat{\beta}_1^2}\mathbf{r}_2{}^T\mathbf{v}_0$ $= \alpha_2 + (\alpha_1-\hat{\alpha}_1) + \frac{\beta_0\beta_1 d}{\hat{\beta}_0^2\hat{\beta}_1^2}\mathbf{r}_2{}^T\tilde{\mathbf{v}}_1$

Table 3.3: Third Iteration to Discover Closed-form Expressions

Symmetric Quantities	**Unsymmetric (Perturbed) Quantities**
	$\hat{\mathbf{p}}_2 = (A-\hat{\alpha}_1 I)\hat{\mathbf{y}}_1 - \hat{\beta}_0\hat{\mathbf{y}}_0$ $= \frac{\beta_0}{\hat{\beta}_0}\mathbf{r}_1 + \frac{(\alpha_1-\hat{\alpha}_1)}{\hat{\beta}_0}\mathbf{r}_0$
	$\tilde{\mathbf{v}}_2 = (A-\hat{\alpha}_2 I))\tilde{\mathbf{v}}_1 - \hat{\beta}_1^2\tilde{\mathbf{v}}_0$ $= A\tilde{\mathbf{v}}_1 - \hat{\alpha}_2\hat{\mathbf{v}}_1 - \hat{\beta}_1^2\tilde{\mathbf{v}}_0$
$\mathbf{r}_2 = (A-\alpha_2 I)\mathbf{x}_2 - \beta_1\mathbf{x}_1$ $= \frac{A\mathbf{r}_1-\alpha_2\mathbf{r}_1}{\beta_1} - \frac{\beta_1\mathbf{r}_0}{\beta_0}$	$\hat{\mathbf{r}}_2 = (A-\hat{\alpha}_2 I)\hat{\mathbf{x}}_2 - \hat{\beta}_1\hat{\mathbf{x}}_1$ $= \frac{\beta_0\beta_1}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_2 + \frac{(\alpha_1-\hat{\alpha}_1)\beta_0}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_1 + \frac{(\alpha_2-\hat{\alpha}_2)\beta_0}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_1$ $+ \frac{(\beta_1^2-\hat{\beta}_1^2)}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_0 + \frac{(\alpha_1-\hat{\alpha}_1)(\alpha_1-\hat{\alpha}_2)}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_0 + \frac{d}{\hat{\beta}_0\hat{\beta}_1}\tilde{\mathbf{v}}_2$
$\beta_2^2 = \mathbf{r}_2^T\mathbf{r}_2$	$\hat{\beta}_2^2 = \hat{\mathbf{p}}_1^T\hat{\mathbf{r}}_1$ $= \frac{\beta_0^2\beta_1^2\beta_2^2}{\hat{\beta}_0^2\hat{\beta}_1^2} + \frac{(\alpha_1-\hat{\alpha}_1)^2(\alpha_1-\hat{\alpha}_2)^2\beta_0^2}{\hat{\beta}_0^2\hat{\beta}_1^2} + \frac{(\beta_1^2-\hat{\beta}_1^2)^2\beta_0^2}{\hat{\beta}_0^2\hat{\beta}_1^2}$ $+ \frac{2(\alpha_1-\hat{\alpha}_1)(\alpha_1-\hat{\alpha}_2)(\beta_1^2-\hat{\beta}_1^2)\beta_0^2}{\hat{\beta}_0^2\hat{\beta}_1^2} + \frac{\beta_0^4\beta_1^4 d^2}{\hat{\beta}_0^6\hat{\beta}_1^6}\mathbf{r}_2{}^T\tilde{\mathbf{v}}_1\mathbf{r}_2^T\tilde{\mathbf{v}}_1$ $- \frac{\beta_0^2\beta_1 d^2}{\hat{\beta}_0^4\hat{\beta}_1^4}\mathbf{r}_2{}^T\tilde{\mathbf{v}}_1\mathbf{r}_1^T\tilde{\mathbf{v}}_2 + \frac{\beta_0\beta_1 d}{\hat{\beta}_0^2\hat{\beta}_1^2}\mathbf{r}_2{}^T\tilde{\mathbf{v}}_2$ $+ \frac{(\alpha_1-\hat{\alpha}_1)(\alpha_1-\hat{\alpha}_2)d}{\hat{\beta}_0^2\hat{\beta}_1^2}\mathbf{r}_0^T\tilde{\mathbf{v}}_2$ $+ \frac{(\beta_1^2-\hat{\beta}_1^2)d}{\hat{\beta}_0\hat{\beta}_1}\mathbf{r}_0^T\tilde{\mathbf{v}}_2$
$\mathbf{x}_3 = \frac{\mathbf{r}_2}{\beta_2}$	$\hat{\mathbf{x}}_3 = \frac{\hat{\mathbf{r}}_1}{\hat{\beta}_1}$ $= \frac{\beta_0\beta_1}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_2 + \frac{(\alpha_1-\hat{\alpha}_1)\beta_0}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_1 + \frac{(\alpha_2-\hat{\alpha}_2)\beta_0}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_1$ $+ \frac{(\beta_1^2-\hat{\beta}_1^2)}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_0 + \frac{(\alpha_1-\hat{\alpha}_1)(\alpha_1-\hat{\alpha}_2)}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_0 + \frac{d}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\tilde{\mathbf{v}}_2$

Table 3.4: Third Iteration to Discover Closed-form Expressions

	$\hat{\mathbf{y}}_3 = \dfrac{\hat{\mathbf{p}}_2}{\hat{\beta}_2}$ $= \dfrac{\beta_0\beta_1}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_2 + \dfrac{(\alpha_1-\hat{\alpha}_1)\beta_0}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_1 + \dfrac{(\alpha_2-\hat{\alpha}_2)\beta_0}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_1$ $+ \dfrac{(\alpha_1-\hat{\alpha}_1)(\alpha_1-\hat{\alpha}_2)\beta_0}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_0\dfrac{(\beta_1^2-\hat{\beta}_1^2)}{\hat{\beta}_0\hat{\beta}_1\hat{\beta}_2}\mathbf{r}_1$
$\alpha_3 = \mathbf{x}_3^T A\mathbf{x}_3$ $= \dfrac{\mathbf{r}_2^T A\mathbf{r}_2}{\beta_2^2}$	$\hat{\alpha}_3 = \hat{\mathbf{y}}_2^T A\hat{\mathbf{x}}_2$ $= \alpha_2 - \dfrac{\beta_0 d}{\hat{\beta}_0^2}{\mathbf{r}_1}^T\tilde{\mathbf{v}}_0 + \dfrac{\beta_0\beta_1 d}{\hat{\beta}_0^2\hat{\beta}_1^2}{\mathbf{r}_2}^T\tilde{\mathbf{v}}_1 + \dfrac{\beta_0\beta_1(\alpha_1-\hat{\alpha}_1)d}{\hat{\beta}_0^2\hat{\beta}_1^2}{\mathbf{r}_2}^T\tilde{\mathbf{v}}_0$

3.1 Relations, Definitions, and Notation

After many months of tedious examination and pain staking manipulation of the perturbed quantities and relations in the tables above to derive Closed-form expressions for $\hat{\alpha}_j$ and $\hat{\beta}_j$, the need for a different approach to achieve this goal became abundantly clear. As an alternative, we define the relations needed to generate the perturbed quantities utilized in our *Perturbation Approach* algorithm to compute the modified Jacobi matrix $\hat{T}_K$ (2.10). We will refer to these definitions and relations for the remainder of Chapter 3 and throughout Chapter 4.

We see it is possible to generate $\hat{\alpha}_j$ and $\hat{\beta}_j$ by relying on those previously computed plus $\mathbf{r}_j$ and $\tilde{\mathbf{v}}_j$. The coefficients for a change of basis, $\mathbf{C}_j$, are how $\hat{\mathbf{r}}_j$ (3.11) relate to $\mathbf{r}_j$. $\mathbf{C}_j$ are the intermediate quantities that help to compute the perturbed α_j and β_j. Furthermore, $\mathbf{C}_j$ and $\mathbf{F}_{j,j}$, make generating these quantities simpler. So, we define the recursive relations for the coefficients $\mathbf{C}_j$ and $\mathbf{F}_{j,j}$. For $i, j \geq 0$ and $i \leq j$, we have

$$\sum_{i=0}^{j}\mathbf{C}_{i,j}\mathbf{r}_i = (A-\hat{\alpha}_j I)\sum_{i=0}^{j-1}\frac{\mathbf{C}_{i,j-1}}{\hat{\beta}_{j-1}}\mathbf{r}_i - \frac{\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}}\sum_{i=0}^{j-2}\mathbf{r}_i \tag{3.2}$$

where

$$A\mathbf{r}_i = \alpha_{i+1}\mathbf{r}_i + \beta_i\mathbf{r}_{i+1} + \frac{\beta_i^2}{\beta_{i-1}}\mathbf{r}_{i-1}. \tag{3.3}$$

After substitution of (3.3) into (3.2) and setting all terms equal to zero, we shift the indicies

on each summation so that all $\mathbf{r}$ terms have an index of i such that

$$\begin{aligned} 0 &= \sum_{i=0}^{j-1} \frac{\mathbf{C}_{i,j-1}\alpha_{i+1}}{\hat{\beta}_{j-1}}\mathbf{r}_i + \sum_{i=1}^{j} \frac{\mathbf{C}_{i-1,j-1}\beta_{i-1}}{\hat{\beta}_{j-1}}\mathbf{r}_i + \sum_{i=0}^{j-2} \frac{\mathbf{C}_{i+1,j-1}\beta_{i+1}^2}{\hat{\beta}_{j-1}\beta_i}\mathbf{r}_i - \sum_{i=0}^{j-1} \frac{\mathbf{C}_{i,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}\mathbf{r}_i \\ &\quad - \sum_{i=0}^{j-2} \frac{\mathbf{C}_{i,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}}\mathbf{r}_i - \sum_{i=0}^{j} \mathbf{C}_{i,j}\mathbf{r}_i. \end{aligned}$$

The indicies on each summation in (3.4) are modified ($i = 0$ to $i = 1$, $i = j$ and $i = j-1$ to $i = j-2$) to consolidate the terms into one large sum. As a consequence, the outlier $\mathbf{r}$ terms with an index other than i are appended to the large summation as follows

$$\begin{aligned} 0 &= \sum_{i=1}^{j-2} \left(\frac{\mathbf{C}_{i,j-1}\alpha_{i+1}}{\hat{\beta}_{j-1}}\mathbf{r}_i + \frac{\mathbf{C}_{i-1,j-1}\beta_{i-1}}{\hat{\beta}_{j-1}}\mathbf{r}_i + \frac{\mathbf{C}_{i+1,j-1}\beta_{i+1}^2}{\hat{\beta}_{j-1}\beta_i}\mathbf{r}_i - \frac{\mathbf{C}_{i,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}\mathbf{r}_i - \frac{\mathbf{C}_{i,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}}\mathbf{r}_i \right. \\ &\quad \left. - \mathbf{C}_{i,j}\mathbf{r}_i \right) + \frac{\mathbf{C}_{0,j-1}\alpha_1}{\hat{\beta}_{j-1}}\mathbf{r}_0 + \frac{\mathbf{C}_{j-1,j-1}\alpha_j}{\hat{\beta}_{j-1}}\mathbf{r}_{j-1} + \frac{\mathbf{C}_{j-1,j-1}\beta_{j-1}}{\hat{\beta}_{j-1}}\mathbf{r}_j + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}}\mathbf{r}_{j-1} \\ &\quad + \frac{\mathbf{C}_{1,j-1}\beta_1^2}{\hat{\beta}_{j-1}\beta_0}\mathbf{r}_0 - \frac{\mathbf{C}_{0,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}\mathbf{r}_0 - \frac{\mathbf{C}_{j-1,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}\mathbf{r}_{j-1} - \frac{\mathbf{C}_{0,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}}\mathbf{r}_0 - \mathbf{C}_{0,j}\mathbf{r}_0 - \mathbf{C}_{j,j}\mathbf{r}_j \\ &\quad - \mathbf{C}_{j-1,j}\mathbf{r}_{j-1}. \end{aligned} \tag{3.4}$$

This is important since we are now able to identify the coefficients for each vector $\mathbf{r}$ in terms of $\mathbf{C}_{(}j)$, α_j, and β_j. Combining all like terms for each index of $\mathbf{r}$ yields the recurrence relation,

$$\begin{aligned} 0 &= \sum_{i=1}^{j-2} \left(\frac{\mathbf{C}_{i,j-1}\alpha_{i+1}}{\hat{\beta}_{j-1}} + \frac{\mathbf{C}_{i-1,j-1}\beta_{i-1}}{\hat{\beta}_{j-1}} + \frac{\mathbf{C}_{i+1,j-1}\beta_{i+1}^2}{\hat{\beta}_{j-1}\beta_i} - \frac{\mathbf{C}_{i,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}} - \frac{\mathbf{C}_{i,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}} - \mathbf{C}_{i,j} \right) \mathbf{r}_i \\ &\quad + \left(\frac{\mathbf{C}_{0,j-1}\alpha_1}{\hat{\beta}_{j-1}} + \frac{\mathbf{C}_{1,j-1}\beta_1^2}{\hat{\beta}_{j-1}\beta_0} - \frac{\mathbf{C}_{0,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}} - \frac{\mathbf{C}_{0,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}} - \mathbf{C}_{0,j} \right) \mathbf{r}_0 \\ &\quad + \left(\frac{\mathbf{C}_{j-1,j-1}\alpha_j}{\hat{\beta}_{j-1}} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}} - \frac{\mathbf{C}_{j-1,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}} - \mathbf{C}_{j-1,j} \right) \mathbf{r}_{j-1} \\ &\quad + \left(\frac{\mathbf{C}_{j-1,j-1}\beta_{j-1}}{\hat{\beta}_{j-1}} - \mathbf{C}_{j,j} \right) \mathbf{r}_j \end{aligned}$$

Thus, we are able to compute the coefficients for $\mathbf{r}_1, \ldots, \mathbf{r}_j$. Since $\mathbf{r}_j$ is orthogonal, their

coefficients must be zero and determined by the following recurrence relations

$$\begin{aligned}
\sum_{i=1}^{j-2}\left(\frac{\mathbf{C}_{i,j-1}\alpha_{i+1}}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{i-1,j-1}\beta_{i-1}}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{i+1,j-1}\beta_{i+1}^2}{\hat{\beta}_{j-1}\beta_i}-\frac{\mathbf{C}_{i,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}-\frac{\mathbf{C}_{i,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}}-\mathbf{C}_{i,j}\right) &= 0,\\
\left(\frac{\mathbf{C}_{0,j-1}\alpha_1}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{1,j-1}\beta_1^2}{\hat{\beta}_{j-1}\beta_0}-\frac{\mathbf{C}_{0,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}-\frac{\mathbf{C}_{0,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}}-\mathbf{C}_{0,j}\right) &= 0,\\
\left(\frac{\mathbf{C}_{j-1,j-1}\alpha_j}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}}-\frac{\mathbf{C}_{j-1,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}-\mathbf{C}_{j-1,j}\right) &= 0,\\
\left(\frac{\mathbf{C}_{j-1,j-1}\beta_{j-1}}{\hat{\beta}_{j-1}}-\mathbf{C}_{j,j}\right) &= 0.
\end{aligned}$$

We solve each case of the recurrence relation for the indicies 0-apart ($\mathbf{C}_{j,j}$), 1-apart ($\mathbf{C}_{j-1,j}$), and the 2 or more apart ($\mathbf{C}_{i,j}$). Then we have

$$\mathbf{C}_{i,j} = \frac{\mathbf{C}_{i,j-1}\alpha_{i+1}}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{i-1,j-1}\beta_{i-1}}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{i+1,j-1}\beta_{i+1}^2}{\hat{\beta}_{j-1}\hat{\beta}_i}-\frac{\mathbf{C}_{i,j-1}\alpha_j}{\hat{\beta}_{j-1}}-\frac{\mathbf{C}_{i,j-2}\beta_{j-1}}{\hat{\beta}_{j-2}}, \tag{3.5}$$

$$\mathbf{C}_{0,j} = \frac{\mathbf{C}_{0,j-1}\alpha_1}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{1,j-1}\beta_1^2}{\hat{\beta}_{j-1}\beta_0}-\frac{\mathbf{C}_{0,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}-\frac{\mathbf{C}_{0,j-2}\hat{\beta}_{j-1}}{\hat{\beta}_{j-2}}, \tag{3.6}$$

$$\mathbf{C}_{j-1,j} = \frac{\mathbf{C}_{j-1,j-1}\alpha_j}{\hat{\beta}_{j-1}}+\frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}}-\frac{\mathbf{C}_{j-1,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}}, \tag{3.7}$$

$$\mathbf{C}_{j,j} = \frac{\mathbf{C}_{j-1,j-1}\beta_{j-1}}{\hat{\beta}_{j-1}}, \tag{3.8}$$

and

$$\mathbf{F}_{j,j} = \frac{\mathbf{F}_{j-1,j-1}}{\hat{\beta}_{j-1}}. \tag{3.9}$$

We now examine the computation of the folllowing recursive relations for $j \geq 1$,

- Recurrence relation for $\hat{\mathbf{p}}_j$:

$$\begin{aligned}
\hat{\mathbf{p}}_j &= (A-\hat{\alpha}_j I)\hat{\mathbf{y}}_j-\hat{\beta}_{j-1}\hat{\mathbf{y}}_{j-1}\\
&= \sum_{i=0}^{j}\mathbf{C}_{i,j}\mathbf{r}_i
\end{aligned} \tag{3.10}$$

- Representation of $\hat{\mathbf{r}}_j$ in terms of the defined quantities $\mathbf{C}_{i,j}$, $\mathbf{F}_{j,j}$, $\mathbf{r}_j$ and $\tilde{\mathbf{v}}_j$:

$$\begin{aligned}
\hat{\mathbf{r}}_j &= (A-\hat{\alpha}_j I)\hat{\mathbf{x}}_j-\hat{\beta}_{j-1}\hat{\mathbf{x}}_{j-1}\\
&= \sum_{k=0}^{j}\mathbf{C}_{k,j}\mathbf{r}_k+\mathbf{F}_{j,j}\tilde{\mathbf{v}}_j
\end{aligned}$$

- Recurrence relation for $\hat{\beta}_{j-1}^2$:

$$\begin{aligned}\hat{\beta}_{j-1}^2 &= \hat{\mathbf{p}}_{j-1}^T \hat{\mathbf{r}}_{j-1} \\ &= \left(\sum_{i=0}^{j-1} \mathbf{C}_{i,j-1}\mathbf{r}_i\right)^T \left(\sum_{i=0}^{j-1} \mathbf{C}_{i,j-1}\mathbf{r}_i + \mathbf{F}_{j-1,j-1}\tilde{\mathbf{v}}_{j-1}\right) \\ &= \sum_{i=0}^{j-1} \hat{\beta}_i^2 \mathbf{C}_{i,j-1}^2 + \sum_{i=0}^{j-1} \mathbf{C}_{i,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}_i^T \tilde{\mathbf{v}}_{j-1}\end{aligned} \tag{3.11}$$

- Recurrence relation for $\hat{\alpha}_j$:

$$\begin{aligned}\hat{\alpha}_j &= \frac{1}{\hat{\beta}_{j-1}^2} \hat{\mathbf{p}}_{j-1}^T A \hat{\mathbf{r}}_{j-1} \\ &= \frac{1}{\hat{\beta}_{j-1}^2} \left(\sum_{i=0}^{j-1} \mathbf{C}_{i,j-1}\mathbf{r}_i\right)^T A \left(\sum_{k=0}^{j-1} \mathbf{C}_{k,j-1}\mathbf{r}_k + \mathbf{F}_{j-1,j-1}\tilde{\mathbf{v}}_{j-1}\right) \\ &= \frac{1}{\hat{\beta}_{j-1}^2} \left(\sum_{i=0}^{j-1} \mathbf{C}_{i,j-1}\mathbf{C}_{k,j-1}\mathbf{r}_i^T A \mathbf{r}_k + \sum_{k=0}^{j-1} \mathbf{C}_{i,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}_i^T A \tilde{\mathbf{v}}_{j-1}\right) \\ &= \frac{1}{\hat{\beta}_{j-1}^2} \left(\sum_{\substack{i=0, \\ |i-k|\leq 1}}^{j-1} \mathbf{C}_{i,j-1}\mathbf{C}_{k,j-1}\beta_i\beta_k \mathbf{T}_{i+1,k+1} + \sum_{k=0}^{j-1} \mathbf{C}_{i,j-1}\mathbf{F}_{j-1,j-1}(\mathbf{A}\mathbf{r}_i)^T \tilde{\mathbf{v}}_{j-1}\right)\end{aligned} \tag{3.12}$$

where $\mathbf{T}_{i+1,i+1} = \alpha_{i+1}$ and $\mathbf{T}_{i,i+1} = \mathbf{T}_{i+1,i} = \beta_i$.

3.2 Perturbation Approach Algorithm

The Perturbation Approach is based on the unsymmetric Lanczos algorithm. Lanczos lends itself to the kind of optimization we want to do. Let's remember, our goal has always been to get from unperturbed α_j and β_j to the perturbed ones $\hat{\alpha}_j$ and $\hat{\beta}_j$ as direCtly as possible. We approximate the perturbed bilinear form $\mathbf{u}^T f(A)(\mathbf{u}+\delta\mathbf{v})$ from (3.11) for $\mathbf{u} \neq \mathbf{v}$ using Gaussian quadrature, and we apply *The Perturbation Approach* algorithm to A with initial vectors $\mathbf{u}$ and $(\mathbf{u}+\delta\mathbf{v})$. Putting the relations (3.1), (3.5), (3.8), (3.9), (3.11), (3.12) together and attaining α_j and β_j from the Jacobi matrix T_K (2.4) after carrying out the symmetric Lanczos iteration yields the following algorithm.

Let $\hat{\mathbf{p}}_0 = \mathbf{u}$, $\tilde{\mathbf{v}}_0 = \mathbf{v}$ an $N \times 1$ given matrix, $\hat{\mathbf{r}}_0 = (\mathbf{u}+d\mathbf{v})$, $\mathbf{C}_{0,0} = 1$, and $\mathbf{F}_{0,0} = d$. Then, for $i, j = 1, 2, \ldots$, we compute

$\mathbf{C}_{1,1} = 1$

$\mathbf{F}_{1,1} = d$

$\tilde{\mathbf{v}}_1 = \mathbf{v}$

for $j = 1, \ldots, K$

$\hat{\beta}^2_{j-1} = \mathbf{C}^2_{j-1,j-1}\beta_{j-1} + \mathbf{C}_{j-1,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}_{j-1}\tilde{\mathbf{v}}_{j-1}$

$\mathbf{F}_{j,j} = \mathbf{F}_{j-1,j-1}/\hat{\beta}_{j-1}$

$\hat{\alpha}_j = (\mathbf{C}_{j,j-1}\mathbf{C}_{j,j-1}\beta_{j-1}\beta_{j-1}\hat{\alpha}_{j-1} + \mathbf{C}_{j,j-1}\mathbf{F}_{j-1,j-1}A\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1})/\hat{\beta}^2_{j-1}$

$\tilde{\mathbf{v}}_j = A\tilde{\mathbf{v}}_{j-1} - \hat{\alpha}_j\tilde{\mathbf{v}}_{j-1} - \hat{\beta}^2_{j-1}\tilde{\mathbf{v}}_{j-2}$

$\mathbf{C}_{i,j} = \frac{\mathbf{C}_{i,j-1}\alpha_{i+1}}{\hat{\beta}_{j-1}} + \frac{\mathbf{C}_{i-1,j-1}\beta_{i-1}}{\hat{\beta}_{j-1}} + \frac{\mathbf{C}_{i+1,j-1}\beta^2_{i+1}}{\hat{\beta}_{j-1}\hat{\beta}_i} - \frac{\mathbf{C}_{i,j-1}\alpha_j}{\hat{\beta}_{j-1}} - \frac{\mathbf{C}_{i,j-2}\beta_{j-1}}{\hat{\beta}_{j-2}}$

end

Finally, we obtain the modified Jacobi matrix $\hat{T}_K$ with the recursion coefficients $\hat{\alpha}_1, \ldots, \hat{\alpha}_K$ and $\hat{\beta}_1, \ldots, \hat{\beta}_K$ computed by the *Perturbation Approach* algorithm,

$$\hat{T}_K = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & & & \\ \hat{\beta}_1 & \hat{\alpha}_2 & \hat{\beta}_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \hat{\beta}_{K-2} & \hat{\alpha}_{K-1} & \hat{\beta}_{K-1} & \\ & & & \hat{\beta}_{K-1} & \hat{\alpha}_K & \end{bmatrix}. \tag{3.13}$$

It is worth noting that each $\hat{\mathbf{p}}_j$ depends on all $\mathbf{r}_j$ generated by the symmetric Lanczos algorithm. Additionally, each $\hat{\mathbf{r}}_j$ depends on all the $\mathbf{r}_j$ generated by the symmetric Lanczos algorithm, the $\tilde{\mathbf{v}}_j$ generated by the *Perturbation Approach* algorithm, $\mathbf{C}_j$ and $\mathbf{F}_{j,j}$. We utilize the "unhatted" quanitities (α_j, β_j, and $\mathbf{r}_j$) from the symmetric Lanczos iteration together with C_j and $F_{j,j}$ to compute the perturbed quantities $\hat{\alpha}_j$ and $\hat{\beta}_j$. As a fortunate consequence, $\hat{\mathbf{p}}_j$ and $\hat{\mathbf{r}}_j$ are **never generated**, **never stored**. Now, we are interested in $\mathbf{r}_j$ expressed as a linear combination of the inverse of $\mathbf{C}_j$ with the columns of matrix $\hat{\mathbf{P}}_j$. This relationship is given in the following lemma.

Lemma 3.2.1. *For $j = 0, 1, 2, \ldots$ there exists a $(j+1)\times(j+1)$ nonsingular upper triangular matrix $\tilde{\mathbf{C}}$ such that*

$$\mathbf{r}_j \;=\; \sum_{i=0}^{j} \tilde{\mathbf{C}}_{i,j}\hat{\mathbf{p}}_i$$

Proof. We define the $N \times (j+1)$ matrices $\hat{\mathbf{P}}_j$ and $\mathbf{R}_j$ by

$$\hat{\mathbf{P}}_j = \begin{bmatrix} \hat{\mathbf{p}}_0 & \hat{\mathbf{p}}_1 & \hat{\mathbf{p}}_2 & \cdots \end{bmatrix}, \qquad \mathbf{R}_j = \begin{bmatrix} \mathbf{r}_0 & \mathbf{r}_1 & \mathbf{r}_2 & \cdots \end{bmatrix}.$$

We define $\mathbf{C}_j$ in the following way,

$$\mathbf{C}_j = \begin{bmatrix} \mathbf{C}_{0,0} & \mathbf{C}_{0,1} & \cdots & \cdots & \mathbf{C}_{0,j} \\ 0 & \mathbf{C}_{1,1} & \mathbf{C}_{1,2} & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & 0 & \ddots & \mathbf{C}_{i,j} \\ 0 & 0 & \cdots & 0 & \mathbf{C}_{j,j} \end{bmatrix}, \tag{3.14}$$

which has nonzero elements on the diagonal by (3.8). Thus, by definition, the inverse of $\mathbf{C}_j$ exists and is upper triangular. We let $\tilde{\mathbf{C}}_j$ be the inverse of $\mathbf{C}_j$, with entries

$$\tilde{\mathbf{C}}_j = \begin{bmatrix} \frac{1}{\mathbf{C}_{0,0}} & \tilde{\mathbf{C}}_{0,1} & \cdots & \cdots & \tilde{\mathbf{C}}_{0,j} \\ 0 & \frac{1}{\mathbf{C}_{1,1}} & \tilde{\mathbf{C}}_{1,2} & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & 0 & \ddots & \tilde{\mathbf{C}}_{i,j} \\ 0 & 0 & \cdots & 0 & \frac{1}{\mathbf{C}_{j,j}} \end{bmatrix}. \tag{3.15}$$

Each column of $\hat{\mathbf{P}}_j$ can be expressed as a linear combination of the columns of $\mathbf{R}_j$ as defined in (3.10) such that

$$\begin{bmatrix} \hat{\mathbf{p}}_0 & \hat{\mathbf{p}}_1 & \hat{\mathbf{p}}_2 & \cdots \end{bmatrix} = \begin{bmatrix} \mathbf{r}_0 & \mathbf{r}_1 & \mathbf{r}_2 & \cdots \end{bmatrix} \begin{bmatrix} \mathbf{C}_{0,0} & \mathbf{C}_{0,1} & \cdots & \cdots & \mathbf{C}_{0,j} \\ 0 & \mathbf{C}_{1,1} & \mathbf{C}_{1,2} & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & 0 & \ddots & \mathbf{C}_{i,j} \\ 0 & 0 & \cdots & 0 & \mathbf{C}_{j,j} \end{bmatrix}. \tag{3.16}$$

Since $\mathbf{C}_j$ is invertible, with some manipulations we can write $\mathbf{R}_j$ as linear combinations of the columns of $\hat{\mathbf{P}}_j$ and $\tilde{\mathbf{C}}_j$. We solve (3.16) for $\mathbf{R}_j$ in terms of $\tilde{\mathbf{C}}_j$:

$$\begin{bmatrix} \mathbf{r}_0 & \mathbf{r}_1 & \mathbf{r}_2 & \cdots \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{p}}_0 & \hat{\mathbf{p}}_1 & \hat{\mathbf{p}}_2 & \cdots \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{C}}_{0,0} & \tilde{\mathbf{C}}_{0,1} & \cdots & \cdots & \tilde{\mathbf{C}}_{0,j} \\ 0 & \tilde{\mathbf{C}}_{1,1} & \tilde{\mathbf{C}}_{1,2} & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & 0 & \ddots & \tilde{\mathbf{C}}_{i,j} \\ 0 & 0 & \cdots & 0 & \tilde{\mathbf{C}}_{j,j} \end{bmatrix}.$$

□

Theorem 1. For $j \geq 0$,

$$\hat{\beta}_j^2 = \mathbf{C}_{j,j}^2 \beta_j^2 + \mathbf{C}_{j,j} \mathbf{F}_{j,j} \mathbf{r}_j^T \tilde{\mathbf{v}}_j$$

Proof. By Lemma 3.2.1,

$$\begin{aligned} \mathbf{r}_j &= \sum_{i=0}^{j} \tilde{\mathbf{C}}_{i,j} \hat{\mathbf{p}}_i \\ &= \sum_{i=0}^{j-1} \tilde{\mathbf{C}}_{i,j} \hat{\mathbf{p}}_i + \tilde{\mathbf{C}}_{j,j} \hat{\mathbf{p}}_j \\ &= \sum_{i=0}^{j-1} \tilde{\mathbf{C}}_{i,j} \hat{\mathbf{p}}_i + \frac{1}{\mathbf{C}_{j,j}} \hat{\mathbf{p}}_j. \end{aligned}$$

As a consequence,

$$\hat{\mathbf{p}}_j = \mathbf{C}_{j,j} \mathbf{r}_j - \sum_{i=0}^{j-1} \mathbf{C}_{j,j} \tilde{\mathbf{C}}_{i,j} \hat{\mathbf{p}}_i.$$

It follows from recurrence (3.10) that

$$\begin{aligned} \hat{\beta}_j^2 &= \hat{\mathbf{p}}_j^T \hat{\mathbf{r}}_j \\ &= \left(\mathbf{C}_{j,j} \mathbf{r}_j - \sum_{i=0}^{j-1} \mathbf{C}_{j,j} \tilde{\mathbf{C}}_{i,j} \hat{\mathbf{p}}_i \right)^T \hat{\mathbf{r}}_j. \\ &= \mathbf{C}_{j,j} \mathbf{r}_j^T \hat{\mathbf{r}}_j - \sum_{i=0}^{j-1} \mathbf{C}_{j,j} \tilde{\mathbf{C}}_{i,j} \hat{\mathbf{p}}_i^T \hat{\mathbf{r}}_j. \end{aligned}$$

By orthogonality and recurrence 3.11, we conclude

$$\begin{aligned} \hat{\beta}_j^2 &= \mathbf{C}_{j,j} \mathbf{r}_j^T \hat{\mathbf{r}}_j \\ &= \mathbf{C}_{j,j} \mathbf{r}_j^T \left(\sum_{k=0}^{j} \mathbf{C}_{k,j} \mathbf{r}_k + \mathbf{F}_{j,j} \tilde{\mathbf{v}}_j \right) \\ &= \mathbf{C}_{j,j} \mathbf{C}_{j,j} \mathbf{r}_j^T \mathbf{r}_j + \mathbf{C}_{j,j} \mathbf{r}_j^T \left(\sum_{k=0}^{j-1} \mathbf{C}_{k,j} \mathbf{r}_k \right) + \mathbf{C}_{j,j} \mathbf{F}_{j,j} \mathbf{r}_j^T \tilde{\mathbf{v}}_j \\ &= \mathbf{C}_{j,j}^2 \beta_j^2 + \mathbf{C}_{j,j} \mathbf{F}_{j,j} \mathbf{r}_j^T \tilde{\mathbf{v}}_j. \end{aligned}$$

□

Theorem 2. For $j \geq 1$,

$$\hat{\alpha}_j = \alpha_{\mathbf{j}} - \mathbf{C}_{j-1,j-1} \mathbf{F}_{j-1,j-1} \mathbf{r}_{j-1}^T \tilde{\mathbf{v}}_{j-2} + \mathbf{C}_{j,j} \mathbf{F}_{j,j} \mathbf{r}_j^T \tilde{\mathbf{v}}_{j-1}$$

Proof. In recurrence relation (3.7), we isolate $\hat{\alpha}_{\mathbf{j}}$:

$$\begin{aligned}
\mathbf{C}_{j-1,j} &= \frac{\mathbf{C}_{j-1,j-1}\alpha_j}{\hat{\beta}_{j-1}} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}} - \frac{\mathbf{C}_{j-1,j-1}\hat{\alpha}_j}{\hat{\beta}_{j-1}} \\
&= \frac{\mathbf{C}_{j-1,j-1}}{\hat{\beta}_{j-1}}(\alpha_j - \hat{\alpha}_{\mathbf{j}}) + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}} \\
\frac{\mathbf{C}_{j-1,j-1}}{\hat{\beta}_{j-1}}(\alpha_j - \hat{\alpha}_{\mathbf{j}}) &= \mathbf{C}_{j-1,j} - \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}} \\
(\alpha_j - \hat{\alpha}_{\mathbf{j}}) &= \mathbf{C}_{j-1,j}\left(\frac{\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}}\right) - \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\hat{\beta}_{j-1}}\left(\frac{\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}}\right) \\
\hat{\alpha}_{\mathbf{j}} &= \alpha_j - \frac{\mathbf{C}_{j-1,j}\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\mathbf{C}_{j-1,j-1}}. \qquad (3.17)
\end{aligned}$$

Utilizing recurrence relation (3.11), we rewrite $\mathbf{C}_{j-1,j}$ in terms of $\mathbf{F}_{j,j}$ and $\tilde{\mathbf{v}}_j$ as follows,

$$\begin{aligned}
\hat{\mathbf{r}}_j &= \sum_{i=0}^{j} \mathbf{C}_{i,j}\mathbf{r}_i + \mathbf{F}_{j,j}\tilde{\mathbf{v}}_j \\
&= \sum_{i=0}^{j-3} \mathbf{C}_{i,j}\mathbf{r}_i + \mathbf{C}_{j,j}\mathbf{r}_j + \mathbf{C}_{j-1,j}\mathbf{r}_{j-1} + \mathbf{C}_{j-2,j}\mathbf{r}_{j-2} + \mathbf{F}_{j,j}\tilde{\mathbf{v}}_j
\end{aligned}$$

$$\begin{aligned}
\hat{\mathbf{r}}_j^T(\mathbf{r}_{j-1}) &= \sum_{i=0}^{j-3} \mathbf{C}_{i,j}\mathbf{r}_i^T(\mathbf{r}_{j-1}) + \mathbf{C}_{j,j}\mathbf{r}_j^T(\mathbf{r}_{j-1}) + \mathbf{C}_{j-1,j}\mathbf{r}_{j-1}^T(\mathbf{r}_{j-1}) \\
&\quad + \mathbf{C}_{j-2,j}\mathbf{r}_{j-2}^T(\mathbf{r}_{j-1}) + \mathbf{F}_{j,j}\tilde{\mathbf{v}}_j^T(\mathbf{r}_{j-1}) \\
&= \mathbf{C}_{j-1,j}\mathbf{r}_{j-1}^T(\mathbf{r}_{j-1}) + \mathbf{F}_{j,j}\tilde{\mathbf{v}}_j^T(\mathbf{r}_{j-1}) \\
&= \mathbf{C}_{j-1,j}\beta_{j-1}^2 + \mathbf{F}_{j,j}\tilde{\mathbf{v}}_j^T(\mathbf{r}_{j-1}) \\
\mathbf{C}_{j-1,j}\beta_{j-1}^2 &= \hat{\mathbf{r}}_j^T(\mathbf{r}_{j-1}) - \mathbf{F}_{j,j}\tilde{\mathbf{v}}_j^T(\mathbf{r}_{j-1}).
\end{aligned}$$

Thus,

$$\mathbf{C}_{j-1,j} = \frac{\hat{\mathbf{r}}_j^T(\mathbf{r}_{j-1})}{\beta_{j-1}^2} - \frac{\mathbf{F}_{j,j}\tilde{\mathbf{v}}_j^T(\mathbf{r}_{j-1})}{\beta_{j-1}^2}. \qquad (3.18)$$

Reducing the indicies by 1 in the above relation for $\mathbf{C}_{j-1,j}$ also yields

$$\mathbf{C}_{j-2,j-1} = \frac{\hat{\mathbf{r}}_{j-1}^T(\mathbf{r}_{j-2})}{\beta_{j-2}^2} - \frac{\mathbf{F}_{j-1,j-1}\tilde{\mathbf{v}}_{j-1}^T(\mathbf{r}_{j-2})}{\beta_{j-2}^2}. \qquad (3.19)$$

We substitute (3.18) into $\hat{\alpha}_j$ (3.17) and obtain

$$\begin{aligned}
\hat{\alpha}_j &= \alpha_j - \frac{\mathbf{C}_{j-1,j}\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\mathbf{C}_{j-1,j-1}} \\
&= \alpha_j - \left(\frac{\hat{\mathbf{r}}_j^T(\mathbf{r}_{j-1})}{\beta_{j-1}^2} - \frac{\mathbf{F}_{j,j}\tilde{\mathbf{v}}_j^T(\mathbf{r}_{j-1})}{\beta_{j-1}^2} \right) \frac{\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\mathbf{C}_{j-1,j-1}} \\
&= \alpha_j - \frac{\hat{\mathbf{r}}_j^T\mathbf{r}_{j-1}\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{F}_{j,j}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_j\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\mathbf{C}_{j-1,j-1}}. \qquad (3.20)
\end{aligned}$$

As a consequence of Lemma 3.2.1, reduCing each index of $\mathbf{r}_j$ by 1 we have

$$\mathbf{r}_{j-1} = \sum_{i=0}^{j-2} \tilde{\mathbf{C}}_{i,j-1}\hat{\mathbf{p}}_i + \tilde{\mathbf{C}}_{j-1,j-1}\hat{\mathbf{p}}_{j-1} \qquad (3.21)$$

Now we substitute relation (3.21) and formula (3.19) into $\hat{\alpha}_j$ (3.17):

$$\begin{aligned}
\hat{\alpha}_j &= \alpha_j - \frac{\hat{\mathbf{r}}_j^T\mathbf{r}_{j-1}\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{F}_{j,j}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_j\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\mathbf{C}_{j-1,j-1}} \\
&= \alpha_j - \frac{\hat{\mathbf{r}}_j^T(\sum_{i=0}^{j-2}\tilde{\mathbf{C}}_{i,j-1}\hat{\mathbf{p}}_i + \tilde{\mathbf{C}}_{j-1,j-1}\hat{\mathbf{p}}_{j-1})\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{F}_{j,j}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_j\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} \\
&\quad + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\mathbf{C}_{j-1,j-1}} \\
&= \alpha_j + \frac{\mathbf{F}_{j,j}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_j\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{C}_{j-2,j-1}\beta_{j-2}}{\mathbf{C}_{j-1,j-1}} \\
&= \alpha_j + \frac{\mathbf{F}_{j,j}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_j\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \left(\frac{\hat{\mathbf{r}}_{j-1}^T\mathbf{r}_{j-2}}{\beta_{j-2}^2} - \frac{\mathbf{F}_{j-1,j-1}\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\beta_{j-2}^2} \right) \frac{\beta_{j-2}}{\mathbf{C}_{j-1,j-1}} \\
&= \alpha_j + \frac{\mathbf{F}_{j,j}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_j\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{F}_{j-1,j-1}\beta_{j-2}\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\beta_{j-2}^2\mathbf{C}_{j-1,j-1}}
\end{aligned}$$

Using recurrence relation (3.1):

$$\begin{aligned}
\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_j &= \mathbf{r}_{j-1}(A\tilde{\mathbf{v}}_{j-1} - \hat{\alpha}_j\tilde{\mathbf{v}}_{j-1} - \hat{\beta}_{j-1}^2\tilde{\mathbf{v}}_{j-2}) \\
&= (A\mathbf{r}_{j-1})^T\tilde{\mathbf{v}}_{j-1} - \hat{\alpha}_j\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1} - \hat{\beta}_{j-1}^2\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2} \\
&= \beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1} + \alpha_j\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1} + \frac{\beta_{j-1}^2\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\beta_{j-2}} - \hat{\alpha}_j\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1} \\
&\quad - \hat{\beta}_{j-1}^2\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}
\end{aligned}$$

Substitute (3.22) into $\hat{\alpha}_j$ (3.22) yields

$$\begin{aligned}
\hat{\alpha}_j &= \alpha_j + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}\left(\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1} + \alpha_j\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1} + \frac{\beta_{j-1}^2\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\beta_{j-2}} - \hat{\alpha}_j\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}\right. \\
&\quad \left. -\hat{\beta}_{j-1}^2\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}\right) - \frac{\mathbf{F}_{j-1,j-1}\beta_{j-2}\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\beta_{j-2}^2\mathbf{C}_{j-1,j-1}} \\
&= \alpha_j + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\alpha_j\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\beta_{j-1}^2\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2\beta_{j-2}} \\
&\quad - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\hat{\alpha}_j\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{F}_{j-1,j-1}\beta_{j-2}\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-2}^2} \\
&= \alpha_j + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}(\alpha_j - \hat{\alpha}_j) + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{F}_{j-1,j-1}\hat{\beta}_{j-1}\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-2}\hat{\beta}_{j-1}} \\
&\quad - \frac{\mathbf{F}_{j-1,j-1}\mathbf{r}_{j-2}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-2}} - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} \\
&= \alpha_j + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}(\alpha_j - \hat{\alpha}_j) + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}
\end{aligned}$$

Subtracting α_j from both sides of (3.22), using the formulas for $\mathbf{C}_{j,j}$ (3.8) and $\mathbf{F}_{j,j}$ (3.9) yields the desired result

$$\begin{aligned}
\hat{\alpha}_{\mathbf{j}} - \alpha_j - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}(\alpha_j - \hat{\alpha}_j) &= \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} \\
-(\alpha_{\mathbf{j}} - \hat{\alpha}_j)\left(1 + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}\right) &= \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} \\
-(\alpha_{\mathbf{j}} - \hat{\alpha}_j)\left(\frac{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}\right) &= \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} \\
&\quad - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} \\
-(\alpha_{\mathbf{j}} - \hat{\alpha}_j)\left(\frac{\mathbf{C}_{j-1,j-1}^2\beta_{j-1}^2}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} + \right. & \\
\left. \frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}\right) &= \frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\hat{\beta}_{\mathbf{j-1}}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} \\
&\quad - \frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2},
\end{aligned}$$

and by *Theorem 1*

$$-(\alpha_{\mathbf{j}}-\hat{\alpha}_j)\left(\frac{\hat{\beta}_{j-1}^2}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}\right) = \frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\hat{\beta}_{\mathbf{j-1}}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2},$$

isolating $-(\alpha_{\mathbf{j}}-\hat{\alpha}_j)$ we have

$$\begin{aligned}
-(\alpha_{\mathbf{j}}-\hat{\alpha}_j) &= \frac{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}{\hat{\beta}_{j-1}^2}\left(\frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\hat{\beta}_{\mathbf{j-1}}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}\right)\\
-\alpha_{\mathbf{j}}+\hat{\alpha}_j &= \frac{\mathbf{C}_{j-1,j-1}^2\beta_{j-1}^2}{\hat{\beta}_{j-1}^2}\left(\frac{\mathbf{F}_{j,j}\hat{\beta}_{\mathbf{j-1}}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2} - \frac{\mathbf{F}_{j,j}\hat{\beta}_{j-1}^3\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}}{\mathbf{C}_{j-1,j-1}\beta_{j-1}^2}\right)\\
-\alpha_{\mathbf{j}}+\hat{\alpha}_j &= \frac{\mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\beta_{j-1}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}}{\hat{\beta}_{j-1}} - \mathbf{C}_{j-1,j-1}\mathbf{F}_{j,j}\hat{\beta}_{j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2}\\
\hat{\alpha}_j &= \alpha_{\mathbf{j}} - \mathbf{C}_{j-1,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}_{j-1}^T\tilde{\mathbf{v}}_{j-2} + \mathbf{C}_{j,j}\mathbf{F}_{j,j}\mathbf{r}_j^T\tilde{\mathbf{v}}_{j-1}
\end{aligned}$$

□

Thus, *Closed-form* expressions for $\hat{\alpha}_j$ and $\hat{\beta}_j$ are realized and proven. Iterations that follow make use of these expressions to generate an optimal algorithm to approximate the perturbed bilinear form (1.4).

Table 3.5: First Iteration with Closed-form Expressions

Symmetric Quantities	**Unsymmetric Quantities** (In terms of $\mathbf{C}_{j,j}$, $\mathbf{F}_{j,j}$, $\tilde{\mathbf{v}}$)
$\mathbf{r}_0 = \mathbf{u}$ $(initial\,vector)$	$\hat{\mathbf{p}}_0 = \mathbf{r}_0 \qquad (left - initial\,vector)$ $= \mathbf{C}_{(0,0)}\mathbf{r}_0$ $\hat{\mathbf{r}}_0 = \mathbf{u} + d\mathbf{v}_0 \qquad (right - initial\,vector)$ $= \mathbf{C}_{(0,0)}\mathbf{r}_0 + \mathbf{F}_{(0,0)}\tilde{\mathbf{v}}_0$
$\beta_0^2 = \mathbf{r}_0^T\mathbf{r}_0$	$\hat{\beta}_0^2 = \hat{\mathbf{p}}_0^T\hat{\mathbf{r}}_0$ $= \beta_0^2 + \mathbf{F}_{(1,1)}\mathbf{r}_0^T\tilde{\mathbf{v}}_0$
$\mathbf{x}_1 = \dfrac{\mathbf{r}_0}{\beta_0}$	$\hat{\mathbf{x}}_1 = \dfrac{\hat{\mathbf{r}}_0}{\hat{\beta}_0}$ $= \dfrac{\mathbf{r}_0}{\hat{\beta}_0} + \mathbf{F}_{(1,1)}\tilde{\mathbf{v}}_0$ $\hat{\mathbf{y}}_1 = \dfrac{\hat{\mathbf{p}}_0}{\hat{\beta}_0}$ $= \dfrac{\mathbf{r}_0}{\hat{\beta}_0}$
$\alpha_1 = \mathbf{x}_1^T A\mathbf{x}_1$ $= \dfrac{\mathbf{r}_0^T A\mathbf{r}_0}{\beta_0^2}$	$\hat{\alpha}_1 = \hat{\mathbf{y}}_1^T A\hat{\mathbf{x}}_1$ $= \alpha_1 + \mathbf{C}_{(1,1)}\mathbf{F}_{(1,1)}{\mathbf{r}_1}^T\tilde{\mathbf{v}}_0$

Table 3.6: Second Iteration with Closed-form Expressions

Symmetric Quantities	**Unsymmetric Quantities** (In terms of $\mathbf{C}_{j,j}$, $\mathbf{F}_{j,j}$, $\tilde{\mathbf{v}}$)
	$\hat{\mathbf{p}}_1 = (A-\hat{\alpha}_1 I)\hat{\mathbf{y}}_1 - \hat{\beta}_0\hat{\mathbf{y}}_0$ $= \mathbf{C}_{(1,1)}\mathbf{r}_1 + \mathbf{C}_{(0,1)}\mathbf{r}_0$
$\mathbf{r}_1 = (A-\alpha_1 I)\mathbf{x}_1$ $= \dfrac{A\mathbf{r}_0 - \alpha_1\mathbf{r}_0}{\beta_0}$	$\hat{\mathbf{r}}_1 = (A-\hat{\alpha}_1 I)\hat{\mathbf{x}}_1 - \hat{\beta}_0\hat{\mathbf{x}}_0$ $= \mathbf{C}_{(1,1)}\mathbf{r}_1 + \mathbf{C}_{(0,1)}\mathbf{r}_0 + \mathbf{F}_{(1,1)}\tilde{\mathbf{v}}_1$
$\beta_1^2 = \mathbf{r}_1^T\mathbf{r}_1$	$\hat{\beta}_1^2 = \hat{\mathbf{p}}_1^T\hat{\mathbf{r}}_1$ $= \mathbf{C}_{(1,1)}^2\beta_1^2 + \mathbf{C}_{(1,1)}\mathbf{F}_{(1,1)}\mathbf{r}_1^T\tilde{\mathbf{v}}_1$
$\mathbf{x}_2 = \dfrac{\mathbf{r}_1}{\beta_1}$	$\hat{\mathbf{x}}_2 = \dfrac{\hat{\mathbf{r}}_1}{\hat{\beta}_1}$ $= \dfrac{\mathbf{C}_{(1,1)}}{\hat{\beta}_1}\mathbf{r}_1 + \dfrac{\mathbf{C}_{(0,1)}}{\hat{\beta}_1}\mathbf{r}_0 + \dfrac{\mathbf{F}_{(1,1)}}{\hat{\beta}_1}\tilde{\mathbf{v}}_1$
	$\hat{\mathbf{y}}_2 = \dfrac{\hat{\mathbf{p}}_1}{\hat{\beta}_1}$ $= \dfrac{\mathbf{C}_{(1,1)}}{\hat{\beta}_1}\mathbf{r}_1 + \dfrac{\mathbf{C}_{(0,1)}}{\hat{\beta}_1}\mathbf{r}_0$
$\alpha_2 = \mathbf{x}_2^T A\mathbf{x}_2$ $= \dfrac{\mathbf{r}_1^T A\mathbf{r}_1}{\beta_1^2}$	$\hat{\alpha}_2 = \hat{\mathbf{y}}_2^T A\hat{\mathbf{x}}_2$ $= \alpha_2 + \mathbf{C}_{(2,2)}\mathbf{F}_{(2,2)}{\mathbf{r}_2}^T\tilde{\mathbf{v}}_1 - \mathbf{C}_{(1,1)}\mathbf{F}_{(1,1)}\mathbf{r}_1^T\tilde{\mathbf{v}}_0$

Table 3.7: Third Iteration with Closed-form Expressions

Symmetric Quantities	**Unsymmetric Quantities** (In terms of $\mathbf{C}_{j,j}$, $\mathbf{F}_{j,j}$, $\tilde{\mathbf{v}}$)
	$\hat{\mathbf{p}}_2 = (A-\hat{\alpha}_2 I)\hat{\mathbf{y}}_2 - \hat{\beta}_1\hat{\mathbf{y}}_1$ $= \mathbf{C}_{(2,2)}\mathbf{r}_2 + \mathbf{C}_{(1,2)}\mathbf{r}_1 + \mathbf{C}_{(0,2)}\mathbf{r}_0$
$\mathbf{r}_2 = (A-\alpha_2 I)\mathbf{x}_2$ $= \dfrac{A\mathbf{r}_1 - \alpha_2\mathbf{r}_1}{\beta_1}$	$\hat{\mathbf{r}}_2 = (A-\hat{\alpha}_2 I)\hat{\mathbf{x}}_2 - \hat{\beta}_1\hat{\mathbf{x}}_1$ $= \mathbf{C}_{(2,2)}\mathbf{r}_2 + \mathbf{C}_{(1,2)}\mathbf{r}_1 + \mathbf{C}_{(0,2)}\mathbf{r}_0 + \mathbf{F}_{(2,2)}\tilde{\mathbf{v}}_2$
$\beta_2^2 = \mathbf{r}_2^T\mathbf{r}_2$	$\hat{\beta}_2^2 = \hat{\mathbf{p}}_2^T\hat{\mathbf{r}}_2$ $= \mathbf{C}_{(2,2)}^2\beta_2^2 + \mathbf{C}_{(2,2)}\mathbf{F}_{(2,2)}\mathbf{r}_2^T\tilde{\mathbf{v}}_2$
$\mathbf{x}_3 = \dfrac{\mathbf{r}_2}{\beta_2}$	$\hat{\mathbf{x}}_3 = \dfrac{\hat{\mathbf{r}}_2}{\hat{\beta}_2}$ $= \dfrac{\mathbf{C}_{(2,2)}}{\hat{\beta}_2}\mathbf{r}_2 + \dfrac{\mathbf{C}_{(1,2)}}{\hat{\beta}_2}\mathbf{r}_1 + \dfrac{\mathbf{C}_{(0,2)}}{\hat{\beta}_2}\mathbf{r}_0 + \dfrac{\mathbf{F}_{(2,2)}}{\hat{\beta}_2}\tilde{\mathbf{v}}_1$
	$\hat{\mathbf{y}}_3 = \dfrac{\hat{\mathbf{p}}_2}{\hat{\beta}_2}$ $= \dfrac{\mathbf{C}_{(2,2)}}{\hat{\beta}_2}\mathbf{r}_2 + \dfrac{\mathbf{C}_{(1,2)}}{\hat{\beta}_2}\mathbf{r}_1 + \dfrac{\mathbf{C}_{(0,2)}}{\hat{\beta}_2}\mathbf{r}_0$
$\alpha_3 = \mathbf{x}_3^T A\mathbf{x}_3$ $= \dfrac{\mathbf{r}_2^T A\mathbf{r}_2}{\beta_2^2}$	$\hat{\alpha}_3 = \hat{\mathbf{y}}_3^T A\hat{\mathbf{x}}_3$ $= \alpha_3 + \mathbf{C}_{(3,3)}\mathbf{F}_{(3,3)}{\mathbf{r}_3}^T\tilde{\mathbf{v}}_2 - \mathbf{C}_{(2,2)}\mathbf{F}_{(2,2)}\mathbf{r}_2^T\tilde{\mathbf{v}}_1$

3.3 The Closed-form Perturbation Method: An Efficient Algorithm

Because we have the algorithm for $\hat{\alpha}_j$ and $\hat{\beta}_j$, we no longer need to explicitly use the unsymmetric Lanczos algorithm. It is preferable to avoid it because it suffers from serious breakdown which occurs when $\hat{\mathbf{p}}_j$ and $\hat{\mathbf{r}}_j$ are orthogonal; a terrible condition for unsymmetric Lanczos. Fortunately, symmetric Lanczos does not have a problem with this condition.

So, by separating the work a little bit and having the bilinear form related to the quadratic form we now have an algorithm that is more efficient. In addition, not having to compute $\hat{\mathbf{r}}_j$

at all saves expense. Let $\hat{\mathbf{p}}_0 = \mathbf{u}$, $\tilde{\mathbf{v}}_0 = \mathbf{v}$ be an $N \times 1$ given matrix, $\hat{\mathbf{r}}_0 = (\mathbf{u} + d\mathbf{v})$, $\mathbf{C}_{0,0} = 1$, and $\mathbf{F}_{0,0} = d$. Then, for $i, j = 1, 2, \ldots$ and $\mathbf{C}_{i,j} = 0$ for $i > j$, we compute

$\mathbf{C}_{1,1} = 1$
$\mathbf{F}_{1,1} = d$
$\tilde{\mathbf{v}}_1 = \mathbf{v}$
for $j = 1, \ldots, K$
$\quad \hat{\beta}^2_{j-1} = \mathbf{C}^2_{j-1,j-1}\beta^2_{j-1} + \mathbf{C}_{j-1,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}^T_{j-1}\tilde{\mathbf{v}}_{j-1}$
$\quad \mathbf{F}_{j,j} = \mathbf{F}_{j-1,j-1}/\hat{\beta}_{j-1}$
$\quad \mathbf{C}_{j,j} = \frac{\mathbf{C}_{j-1,j-1}\beta_{j-1}}{\hat{\beta}_{j-1}}$
$\quad \hat{\alpha}_j = \alpha_j + \mathbf{C}_{j,j}\mathbf{F}_{j,j}\mathbf{r}^T_j\tilde{\mathbf{v}}_{j-1} - \mathbf{C}_{j-1,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}^T_{j-1}\tilde{\mathbf{v}}_{j-2}$
$\quad \tilde{\mathbf{v}}_j = A\tilde{\mathbf{v}}_{j-1} - \hat{\alpha}_{j-1}\tilde{\mathbf{v}}_{j-1} - \hat{\beta}^2_{j-1}\tilde{\mathbf{v}}_{j-2}$
end

Finally, we obtain the modified Jacobi matrix $\hat{T}_K$ with the recursion coefficients $\hat{\alpha}_1, \ldots, \hat{\alpha}_K$ and $\hat{\beta}_1, \ldots, \hat{\beta}_K$ computed by the *Perturbation Approach* algorithm,

$$\hat{T}_K = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & & & \\ \hat{\beta}_1 & \hat{\alpha}_2 & \hat{\beta}_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \hat{\beta}_{K-2} & \hat{\alpha}_{K-1} & \hat{\beta}_{K-1} & \\ & & & \hat{\beta}_{K-1} & \hat{\alpha}_K & \end{bmatrix}. \tag{3.22}$$

3.4 Formulas for Derivatives: Derivatives of the Jacobi Matrix and Quadratic Forms

Taking the derivative of all the quantities in the Perturbation Approach algorithm with respect to d, and then evaluating at $d = 0$ gives us a sensitivity measure. We take the algorithm and differentiate at every step, then the output will be the derivatives of every quantity.

To obtain the derivative of the matrix (3.22) produced by Perturbation algorithm, it will be helpful to first obtain the Jacobi matrix (2.4) produced by symmetric Lanczos applied to A with initial vector $\mathbf{u}$, and also its derivatives. The following algorithm does so; its derivation mainly relies on applying the product rule to the steps of the Lanczos algorithm.

function $[X, X', J, J', \beta_0, \beta_0']$ = **deriv_lanczos**$(A, A', \mathbf{r}_0, \mathbf{r}_0', n)$

$\mathbf{x}_0 = \mathbf{0}$
$\mathbf{x}'_0 = \mathbf{0}$
for $j = 1\ldots,n$
$\beta_{j-1} = \|\mathbf{r}_{j-1}\|_2$
$\beta'_{j-1} = \mathrm{Re}(\mathbf{r}^T_{j-1}\mathbf{r}'_{j-1})/\beta_{j-1}$
$\mathbf{x}_j = \mathbf{r}_{j-1}/\beta_{j-1}$
$\mathbf{x}'_j = (\beta_{j-1}\mathbf{r}'_{j-1} - \mathbf{r}_{j-1}\beta'_{j-1})/\beta^2_{j-1}$
$\mathbf{v}_j = A\mathbf{x}_j$
$\mathbf{v}'_j = A'\mathbf{x}_j + A\mathbf{x}'_j$
$\alpha_j = \mathbf{x}^T_j\mathbf{v}_j$
$\alpha'_j = (\mathbf{x}'_j)^T\mathbf{v}_j + \mathbf{x}^T_j\mathbf{v}'_j$
$\mathbf{r}_j = \mathbf{v}_j - \alpha_j\mathbf{x}_j - \beta_{j-1}\mathbf{x}_{j-1}$
$\mathbf{r}'_j = \mathbf{v}'_j - \alpha'_j\mathbf{x}_j - \alpha_j\mathbf{x}'_j - \beta'_{j-1}\mathbf{x}_{j-1} - \beta_{j-1}\mathbf{x}'_{j-1}$
end

The derivation of the recursion coefficients $\alpha_1,\ldots,\alpha_K$ and $\beta_1,\ldots,\beta_K$ yields the Jacobi matrix $(T_K)'$

$$(T_K)' = \begin{bmatrix} \alpha'_1 & \beta'_1 & & & \\ \beta'_1 & \alpha'_2 & \beta'_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta'_{K-2} & \alpha'_{K-1} & \beta'_{K-1} \\ & & & \beta'_{K-1} & \alpha'_K \end{bmatrix}. \tag{3.23}$$

Additionally, the derivation of the "hatted" quantities are computed in the *Perturbation Approach Closed-form* algorithm. A helpful consequence is that in every perturbed (hatted) quantity, the $\tilde{\mathbf{v}}_j$ and $\mathbf{C}_j$ all depend on d. But, when $d = 0$, each of the perturbed quantities equal the "unhatted" case. Because we are performing the product rule, many of our quantities simplify quite a bit when $d = 0$. So, when we take the derivatives at each step of the algorithm and take the simplifications into account, we get the simplest possible formulas for the derivatives. In general, the algorithm to compute the derivatives of every quantity of unsymmetric Lanczos follows.

function $[X,X',J,J',\beta_0,\beta'_0]$ = **deriv_unsymlanczos**$(A,A',\hat{\mathbf{p}}_0,\hat{\mathbf{p}}'_0,\hat{\mathbf{r}}_0,\hat{\mathbf{r}}'_0,n)$
$\hat{\mathbf{p}}_0 = \mathbf{u}$
$\hat{\mathbf{r}}_0 = \mathbf{v}$

$\hat{\mathbf{x}}_0 = \mathbf{0}$
$\hat{\mathbf{x}}'_0 = \mathbf{0}$
$\hat{\mathbf{y}}_0 = \mathbf{0}$
$\hat{\mathbf{y}}'_0 = \mathbf{0}$
for $j = 1\ldots,n$
$\hat{\beta}^2_{j-1} = \hat{\mathbf{p}}^T_{j-1}\hat{\mathbf{r}}_{j-1}$
$\hat{\beta}'_{j-1} = (\hat{\mathbf{p}}'^T_{j-1}\hat{\mathbf{r}}_{j-1} + \hat{\mathbf{p}}^T_{j-1}\hat{\mathbf{r}}'_{j-1})/(2\hat{\beta}_{j-1})$
$\hat{\mathbf{x}}_j = \hat{\mathbf{r}}_{j-1}/\hat{\beta}_{j-1}$
$\hat{\mathbf{x}}'_j = (\hat{\beta}_{j-1}\hat{\mathbf{r}}'_{\mathbf{j-1}} - \hat{\mathbf{r}}_{\mathbf{j-1}}\hat{\beta}'_{\mathbf{j-1}})/(\hat{\beta}^2_{j-1})$
$\hat{\mathbf{y}}_j = \hat{\mathbf{p}}_{j-1}/\hat{\beta}_{j-1}$
$\hat{\mathbf{y}}'_j = (\hat{\beta}_{j-1}\hat{\mathbf{p}}'_{\mathbf{j-1}} - \hat{\mathbf{p}}_{\mathbf{j-1}}\hat{\beta}'_{\mathbf{j-1}})/(\hat{\beta}^2_{j-1})$
$\hat{\mathbf{w}}_j = A\hat{\mathbf{x}}_j$
$\hat{\mathbf{w}}'_j = A'\hat{\mathbf{x}}_j + A\hat{\mathbf{x}}'_j$
$\hat{\mathbf{t}}_j = A\hat{\mathbf{y}}_j$
$\hat{\mathbf{t}}'_j = A'\hat{\mathbf{y}}_j + A\hat{\mathbf{y}}'_j$
$\hat{\alpha}_j = \hat{\mathbf{y}}^T_j\hat{\mathbf{w}}_j$
$\hat{\alpha}'_j = \hat{\mathbf{y}}'^T_j\hat{\mathbf{w}}_j + \hat{\mathbf{y}}^T_j\hat{\mathbf{w}}'_j$
$\hat{\mathbf{r}}_j = \hat{\mathbf{w}}_j - \hat{\alpha}_j\hat{\mathbf{x}}_j - \hat{\beta}_{j-1}\hat{\mathbf{x}}_{j-1}$
$\hat{\mathbf{r}}'_j = \hat{\mathbf{w}}'_j - \hat{\alpha}'_j\hat{\mathbf{x}}_j - \hat{\alpha}_j\hat{\mathbf{x}}'_j - \hat{\beta}'_{j-1}\hat{\mathbf{x}}_{j-1} - \hat{\beta}_{j-1}\hat{\mathbf{x}}'_{j-1}$
$\hat{\mathbf{p}}_j = \hat{\mathbf{t}}_j - \hat{\alpha}_j\hat{\mathbf{y}}_j - \hat{\beta}_{j-1}\hat{\mathbf{y}}_{j-1}$
$\hat{\mathbf{p}}_j = \hat{\mathbf{t}}'_j - \hat{\alpha}'_j\hat{\mathbf{y}}_j - \hat{\alpha}_j\hat{\mathbf{y}}'_j - \hat{\beta}'_{j-1}\hat{\mathbf{y}}_{j-1} - \hat{\beta}_{j-1}\hat{\mathbf{y}}'_{j-1}$
end

The algorithm to compute the derivatives of every quantity of the *Perturbation Approach* follows.

Let $\hat{\mathbf{p}}_0 = \mathbf{u}$, $\tilde{\mathbf{v}}_0 = \mathbf{v}$ be $N \times 1$ randomly chosen matrices, $\hat{\mathbf{r}}_0 = (\mathbf{u} + d\mathbf{v})$, $\mathbf{C}_{0,0} = 1$, and $\mathbf{F}_{0,0} = d$. Then, for $i, j = 1, 2, \ldots$ and $\mathbf{C}_{i,j} = 0$ for $i > j$, we compute

$\mathbf{C}_{1,1} = 1$
$\mathbf{C}'_{1,1} = 0$
$\mathbf{F}_{1,1} = d$
$\mathbf{F}'_{1,1} = 1$
$\tilde{\mathbf{v}}_1 = \mathbf{v}$
$\tilde{\mathbf{v}}_1 = 0 * \mathbf{v}$
for $j = 1,\ldots,K$
$\hat{\beta}^2_{j-1} = \mathbf{C}^2_{j-1,j-1}\beta^2_{j-1} + \mathbf{C}_{j-1,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}^T_{j-1}\tilde{\mathbf{v}}_{j-1}$

$(\hat{\beta}^2_{j-1})' = (\mathbf{C}^2_{j-1,j-1})'\beta^2_{j-1} + (\mathbf{F}'_{j-1,j-1}\mathbf{r}^T_{j-1}\tilde{\mathbf{v}}_{j-1})/2\beta_{j-1}$
$\mathbf{F}_{j,j} = \mathbf{F}_{j-1,j-1}/\hat{\beta}_{j-1}$
$\mathbf{F}'_{j,j} = \mathbf{F}'_{j-1,j-1}/\beta_{j-1}$
$\mathbf{C}_{j,j} = (\mathbf{C}_{j-1,j-1}\beta_{j-1})/\beta_{j-1}$
$\mathbf{C}'_{j,j} = \mathbf{C}'_{j-1,j-1} - \hat{\beta}_{j-1}{}'\hat{\beta}_{j-1}$
$\hat{\alpha}_j = \alpha_j + \mathbf{C}_{j,j}\mathbf{F}_{j,j}\mathbf{r}^T_j\tilde{\mathbf{v}}_{j-1} - \mathbf{C}_{j-1,j-1}\mathbf{F}_{j-1,j-1}\mathbf{r}^T_{j-1}\tilde{\mathbf{v}}_{j-2}$
$\hat{\alpha}'_j = \mathbf{F}'_{j,j}\mathbf{r}^T_j\tilde{\mathbf{v}}_{j-1} - \mathbf{C}_{j-1,j-1}\mathbf{F}'_{j-1,j-1}\mathbf{r}^T_{j-1}\tilde{\mathbf{v}}_{j-2}$
$\tilde{\mathbf{v}}_j = A\tilde{\mathbf{v}}_{j-1} - \hat{\alpha}_{j-1}\tilde{\mathbf{v}}_{j-1} - \hat{\beta}^2_{j-1}\tilde{\mathbf{v}}_{j-2}$
$\tilde{\mathbf{v}}_j = A\tilde{\mathbf{v}}'_{j-1} - \alpha_{j-1}\tilde{\mathbf{v}}'_{j-1} - \hat{\alpha}'_{j-1}\tilde{\mathbf{v}}_{j-1} - 2\beta_{j-1}\hat{\beta}'_{j-1}\tilde{\mathbf{v}}_{j-2}$

end

When we make the perturbation $\mathbf{r}_0 + d\mathbf{v}_0$, the derivatives give us an idea of how much α_j and β_j are going to change to get $\hat{\alpha}_j$ and $\hat{\beta}_j$. So, they could tell us if even a small perturbation in $\mathbf{r}_0$ could lead to an unexpectedly large change in α_j and β_j. This is the kind of information our sensitivity analysis will provide in Chapter 4.

Chapter 4

NUMERICAL RESULTS

In this chapter, we will compare the output of the modified Jacobi matrix generated by the unsymmetric Lanczos algorithm and the *Perturbation Approach* algorithm. Having an understanding of the $\hat{\alpha}_j$ and $\hat{\beta}_j$ and how they relate to the "unhatted" quantities can give us an understanding of sensitivity. Given a certain $\mathbf{v}$, we get an understanding of how much the Jacobi matrix (2.4) will change. When we take a derivative of the "hatted" quantites with respect to the parameter d, it is with a perturbation in the direction of $\mathbf{v}_j$ in mind. We think of the derivatives of each perturbed quantity as a *condition number*, which helps to determine whether perturbing $\mathbf{r}_0$ by $d\mathbf{v}_0$ is going to cause a small change or a large change in α_j and β_j. So, if we change $\mathbf{v}_j$, we change the derivatives. For testing, we use a random choice $\mathbf{v}$, because it gives us a fuller picture of the changes in the output when we change the initial vector. We will also use Newton's Method to aid in choosing a "good" value for our parameter d to ensure positive weights. All numerical experiments are conducted in MATLAB R2015b.

4.1 Error

We will investigate the error to determine how stable the Closed-form Perturbation algorithm is compared to the unsymmetric Lanczos algorithm. Our first study is without reorthogonalization within the unsymmetric Lanczos algorithm. The error behaves differently with or without reorthogonalization. The error without reorthogonalization grows fairly quickly and then tails off again, in a cyclical pattern. The error with reothogonilization puts off the error growth for a time, but when it grows it gets really large.

Error: Closed-form vs. Unsymmetric Lan. w/o Reorthogonalization			
N	$d = 0.1$	$d = 0.01$	$d = 0.001$
5	1.3282e-15	5.5992e-16	5.6936e-16
10	2.9060e-14	8.5553e-15	3.8532e-14
20	0.0902	0.0126	0.2009
50	1.4121	0.9617	0.3497
100	2.2320	1.0005	1.0128
Error: Closed-form vs. Unsymmetric Lan. w/ Reorthogonalization			
N	$d = 0.1$	$d = 0.01$	$d = 0.001$
5	1.9853e-16	1.6430e-16	2.2773e-16
10	2.6188e-16	2.2473e-16	1.5626e-16
20	4.3206e-11	1.2749e-12	3.5855e-13
50	1.1616e+04	1.0000	0.7788
100	NAN	NAN	NAN

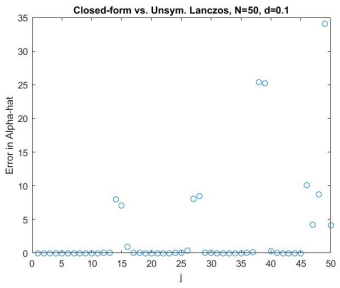
Table 4.1: Accuracy of Perturbation Approach: Norms of the error between the Closed-form Perturbation Approach algorithm with Derivatives (Coeffs2CFwDeriv) and the Unsymmetric Lanczos Algorithm without reorthogonalization (UnsymLanczosPHD) with random symmetric $N \times N$ matrix A for N=5,10,20,50,100.

We will investigate the large error for $N = 50$, $N = 100$ and $N = 250$ by computing the error in blocks to determine how long the algorithms remain stable. We will also look at the error in $\hat{\alpha}_j$ and $\hat{\beta}_j$ to provide insight as to why instability occurs.

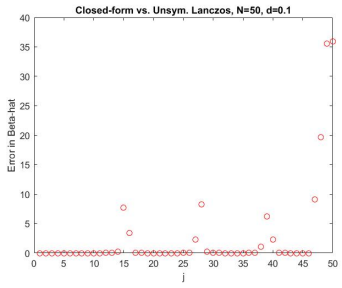
Error: CF - Leading principal submatrix (N=50) w/o Reorth.			
$k \times k$	d = 0.1	$d = 0.01$	d = 0.001
5	2.6630e-16	5.1023e-16	1.2203e-15
10	7.5104e-12	6.1025e-12	1.1542e-11
20	0.2233	0.1743	0.1768
35	0.2341	0.5790	0.1768
50	1.4121	0.9617	0.3497

Table 4.2: Accuracy of Perturbation Approach: Error in leading principal submatrix ($k \times k$ block) for N = 50 to determine when the large error occurs. Closed-form (CF) algorithm with Derivatives (Coeffs2CFwDeriv) vs. Unsymmetric Lanczos Algorithm with and without reorthogonalization (UnsymLanczosPHD)

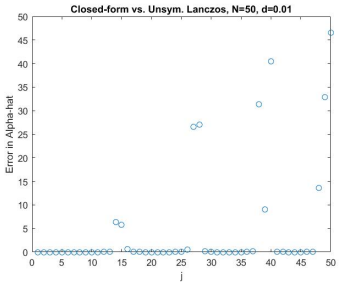
(a) ($\hat{\alpha}_j$): N=50, d=0.1



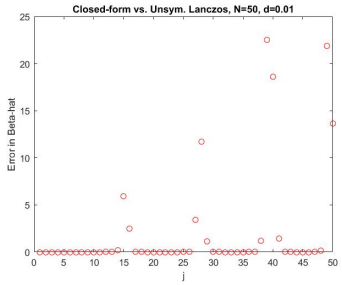
(b) ($\hat{\beta}_j$): N=50, d=0.1



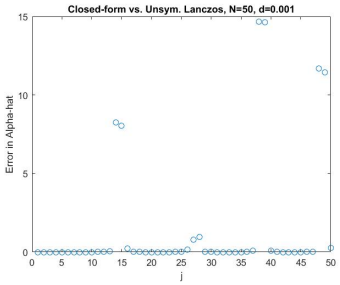
(c) ($\hat{\alpha}_j$): N=50, d=0.01



(d) ($\hat{\beta}_j$): N=50, d=0.01



(e) ($\hat{\alpha}_j$): N=50, d=0.001



(f) ($\hat{\beta}_j$): N=50, d=0.001

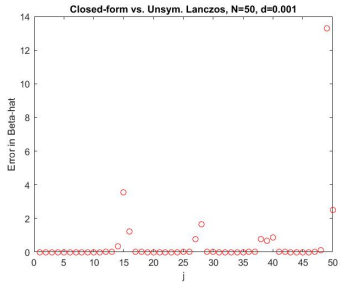
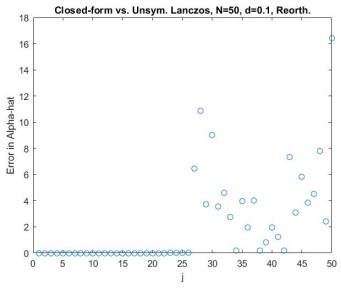


Figure 4.1: Order of Magnitude: $\hat{\alpha}_j$, $\hat{\beta}_j$ for $N = 50$

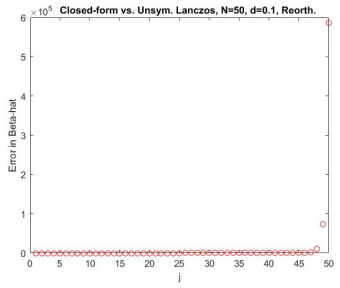
Error: CF - Leading principal submatrix (N=50) w/ Reorth.			
$k \times k$	$d = 0.1$	$d = 0.01$	$d = 0.001$
5	1.4441e-16	2.0029e-16	6.1336e-16
10	1.6331e-16	2.0029e-16	6.1336e-16
20	1.0251e-11	5.1538e-13	2.0967e-13
35	0.4935	0.9879	0.7702
50	1.1616e+04	0.9879	0.7702

Table 4.3: Accuracy of Perturbation Approach: Error in leading principal submatrix ($k \times k$ block) for $N = 50$ to determine when the large error occurs. Closed-form algorithm with Derivatives (Coeffs2CFwDeriv) vs. Unsymmetric Lanczos Algorithm without reorthogonalization (UnsymLanczosPHD)

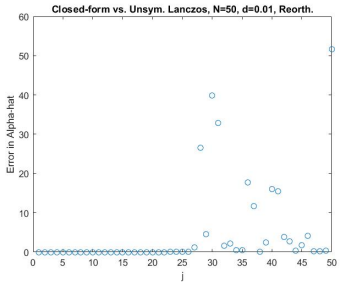
(a) ($\hat{\alpha}_j$): N=50, d=0.1



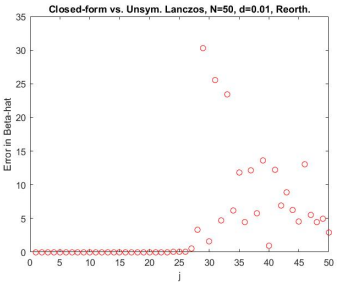
(b) ($\hat{\beta}_j$): N=50, d=0.1



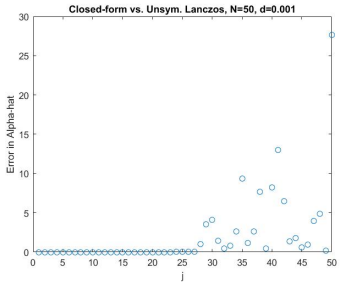
(c) ($\hat{\alpha}_j$): N=50, d=0.01



(d) ($\hat{\beta}_j$): N=50, d=0.1



(e) ($\hat{\alpha}_j$): N=50, d=0.001



(f) ($\hat{\beta}_j$): N=50, d=0.1

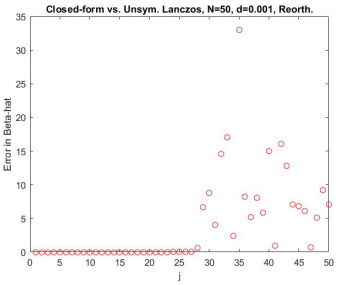
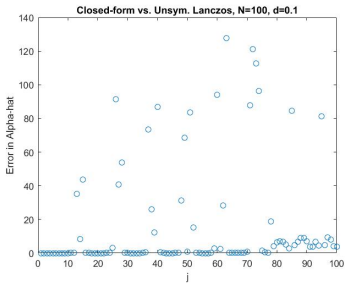


Figure 4.2: Order of Magnitude: $\hat{\alpha}_j$, $\hat{\beta}_j$ for $N = 50$

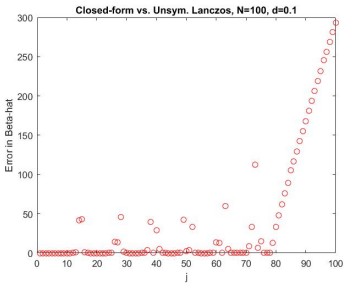
Error: CF - Leading principal submatrix (N=100) w/o Reorth.			
$k \times k$	$d = 0.1$	$d = 0.01$	$d = 0.001$
5	3.9907e-16	2.6991e-16	2.9719e-16
10	1.6617e-09	6.3712e-11	9.2793e-10
20	0.7324	0.5112	0.7105
50	0.9779	1.0175	0.9807
100	2.2320	1.0005	1.0128

Table 4.4: Accuracy of Perturbation Approach: Error in leading principal submatrix ($k \times k$ block) for $N = 100$ to determine when the large error occurs. Closed-form (CF) algorithm with Derivatives (Coeffs2CFwDeriv) vs. Unsymmetric Lanczos Algorithm with and without reorthogonalization (UnsymLanczosPHD)

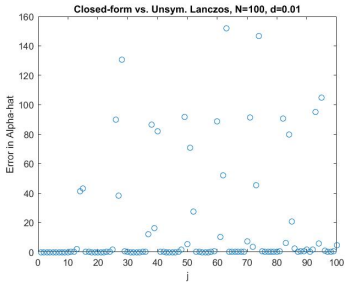
(a) ($\hat{\alpha}_j$): N=100, d=0.1



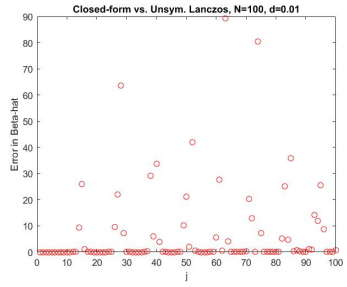
(b) ($\hat{\beta}_j$): N=100, d=0.1



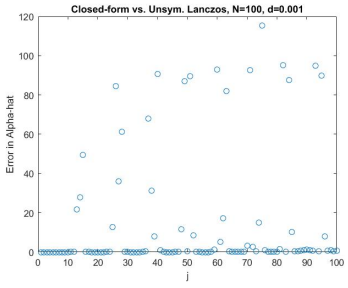
(c) ($\hat{\alpha}_j$): N=100, d=0.01



(d) ($\hat{\beta}_j$): N=100, d=0.01



(e) ($\hat{\alpha}_j$): N=100, d=0.001



(f) ($\hat{\beta}_j$): N=100, d=0.001

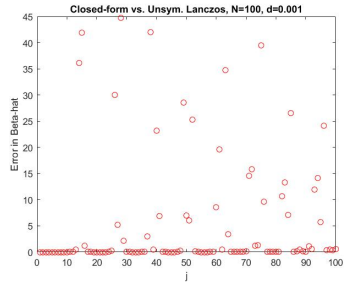
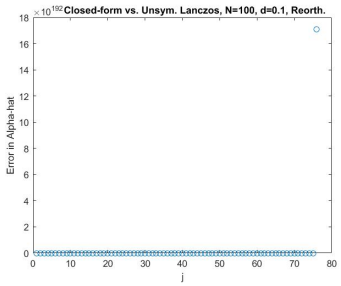


Figure 4.3: Order of Magnitude: $\hat{\alpha}_j$, $\hat{\beta}_j$ for $N = 100$

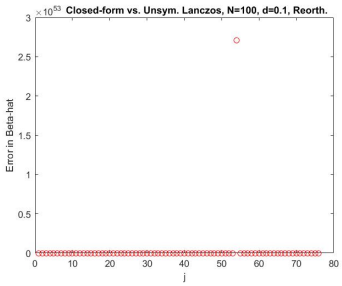
Error: CF - Leading principal submatrix (N=100) w/ Reorth.			
$k \times k$	$d = 0.1$	$d = 0.01$	$d = 0.001$
5	1.4327e-16	1.9934e-16	1.4327e-16
10	1.4327e-16	1.9934e-16	1.4327e-16
20	1.4316e-09	1.6350e-11	1.4682e-11
50	1.0000	1.0000	1.0000
100	NaN	NaN	NaN

Table 4.5: Accuracy of Perturbation Approach: Error in leading principal submatrix ($k \times k$ block) for $N = 100$ to determine when the large error occurs. Closed-form algorithm with Derivatives (Coeffs2CFwDeriv) vs. Unsymmetric Lanczos Algorithm without reorthogonalization (UnsymLanczosPHD)

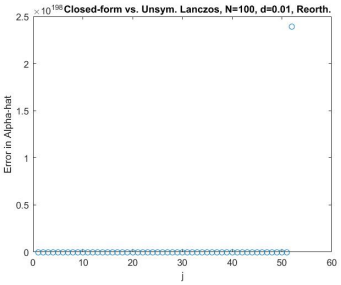
(a) ($\hat{\alpha}_j$): N=100, d=0.1



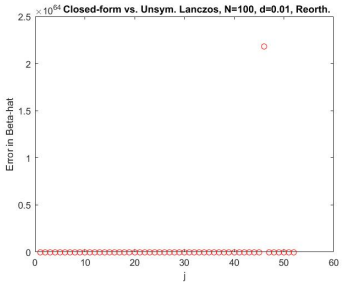
(b) ($\hat{\beta}_j$): N=100, d=0.1



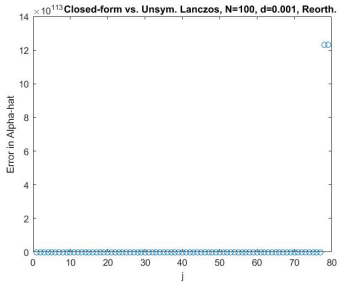
(c) ($\hat{\alpha}_j$): N=100, d=0.01



(d) ($\hat{\beta}_j$): N=100, d=0.1



(e) ($\hat{\alpha}_j$): N=100, d=0.001



(f) ($\hat{\beta}_j$): N=100, d=0.1

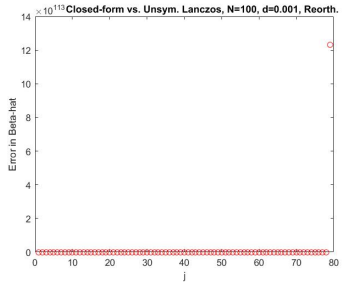


Figure 4.4: Order of Magnitude: $\hat{\alpha}_j$, $\hat{\beta}_j$ for $N = 100$

Error: Closed-form vs. Unsymmetric Lan. w/ Reorthogonalization				
N	$d = 0.1$	$d = 0.001$	$d = 0.00001$	d = 0.1e-6
50 (R)	1.0000	1.0000	0.5698	0.5166
50 (RSp)	1.1196e+21	2.9247e-04	4.0151e-06	3.3946e-08

Table 4.6: Accuracy of Perturbation Approach: Error in leading principal submatrix ($k \times k$ block) for $N = 50$ to determine when the large error occurs. Closed-form algorithm with Derivatives (Coeffs2CFwDeriv) vs. Unsymmetric Lanczos Algorithm without reorthogonalization (UnsymLanczosPHD)

In general, the sensitivity study seems to favor the perturbation parameter d = 0.1e-6, since the error is better longer for this value of d. However, as discussed in **??**, the results of our study failed to show the accuracy of our Closed-form Perturbation algorithm due to the growth in rounding errors in the algorithms. A consequence of the Lanczos algorithms is the loss of orthogonality of the computed Lanczos vectors, $\mathbf{r}_j$ for symmetric Lanczos and $\hat{\mathbf{r}}_j$, $\hat{\mathbf{p}}_j$ for unsymmetric Lanczos. We added reorthogonalization code to both Lanczos algorithms to correct this loss at each iteration. However, there is currently no reorthogonalization counterpart to correct the loss in the Closed-form Perturbation algorithm.

4.2 Newton's Method

Newton's method works well in general for solving $f(x) = 0$ where f is analytic. Suppose x_k is an approximate solution of $f(x) = 0$ and $f'(x_k) \neq 0$, by definition as in [1], the next approximation is given by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

When running unsymmetric Lanczos, if $\hat{\beta}_j^2 = \hat{\mathbf{p}}_j^T \hat{\mathbf{r}}_j = 0$, the algorithm can experience *break-down*. Let $G(d) = \hat{\beta}_j^2$ where $G(d)$ is some function that depends on d. We would like to solve this equation for d, but we are unable to do that because the equation is too complicated. But, we do not have to solve it exactly; we estimate a solution since d is small using Newton's method. However, we are just performing one iteration where the previous iterate is zero. If $d = 0$, there is no perturbation at all which yields β_j^2, the dot product of $\mathbf{r}_j$ and $\mathbf{r}_j$. Then $G'(0)$ is the derivative, $(\hat{\beta}_j^2)'$. Therefore, one iteration of Newton's Method is

$$d = -\frac{G(0)}{G'(0)}, \tag{4.1}$$

where

$$G(0) = \beta_j^2 \tag{4.2}$$

and

$$G'(0) = (\hat{\beta}_j^2)'. \tag{4.3}$$

Thus, as each iteration computes $\hat{\beta}_j$ and its derivative $(\hat{\beta}_j)'$, a d is generated. But, we want to choose a d that is safe. We want to choose a d small enough that *break-down* does not occur, thus taking care not to perturb by too much. This parameter d we will call d^* which

is computed as follows,

$$\begin{aligned} d^* &= -\frac{\hat{\beta}_j^2}{(\hat{\beta}_j^2)'} \\ &= -\frac{\beta_j^2}{(\hat{\beta}_j^2)'} \\ &= -\frac{\beta_j^2}{2\hat{\beta}_j\hat{\beta}_j'} \\ &= -\frac{\beta_j^2}{2\beta_j\hat{\beta}_j'} \\ &= -\frac{\beta_j}{2\hat{\beta}_j'}. \end{aligned} \tag{4.4}$$

We take the minimum of d^* over j as a guideline, because we are only solving $G(d) = 0$ approximately. A "good" d is smaller than the absolute value of the minimum of d^*. We decide to be conservative and cut d^* by a factor of 10 to achieve a safe d,

$$\text{"Good" } d = \frac{min|\ d^*\ |}{10}. \tag{4.5}$$

The thing to keep in mind is that the generated d from Newton's Method is a very crude estimate for a safe d for each $\hat{\beta}_j$. There is a trade-off. If we choose d too large we get negative weights. If we pick d too small we get cancellation error from subtraction (round-off). Since the matrix A is symmetric, the weights will be positive. We want to choose a d smaller than the computed d.

Choosing a "Good" d		
N	min $\|d^*\|$	"Good" d
10	0.0198	0.0020
20	0.1277	0.0128
50	0.0463	0.0046
100	0.0594	0.0059

Table 4.7: Using Newton's Method to choose a "Good" d to ensure positive weights.

4.3 Sensitivity

The perturbation parameter d is useful for computing bilinear forms using our perturbation method. The d values that are produced are independent of the d chosen to execute the algorithm. If we choose a d value larger than the largest d produced for a "good" d, then T_K is complex. The 'MatGen' code (A.1) generates and saves a random $N \times N$ symmetric matrix A, initial vector **u** and initial vector **v** for numerical results testing.

If any derivative is particularly large, then we know those quantities are extra sensitive. Our function produces the Jacobi matricies $\hat{T}_K$ and $(\hat{T}_K)'$ that we take the norm of to determine if our Lanczos process is ill-conditioned or not. We test the accuracy of the derivatives by comparing the computed derivatives of the modified Jacobi matrix using the *Perturbation Approach Closed-form algorithm* with its equivalent difference quotient $(\hat{T}_K - T_K)/d$ derived by the difference of the Closed-form modified Jacobi matrix $\hat{T}_K$ and Jacobi matrix T_K generated by the Lanczos algorithm all divided by d for a small symmetric matrix A.

Accuracy of Derivative, $N = 5$, $d = .001$					
Derivative	Derivative of Modified $\hat{T}_K$				
$\frac{\hat{T}_K - T_K}{d}$	0.5056	0.1901	0	0	0
	0.1901	0.3714	0.0237	0	0
	0	0.0237	0.0626	0.0296	0
	0	0	0.0296	0.0441	0.0112
	0	0	0	0.0112	0.0275
$(\hat{T}_K)$'	0.5058	0.1901	0	0	0
	0.1901	0.3716	0.0238	0	0
	0	0.0238	0.0626	0.0296	0
	0	0	0.0296	0.0441	0.0112
	0	0	0	0.0112	0.0276
Error = 4.4815e-04					

Table 4.8: Accuracy of derivatives: Compare the computed derivative of the modified Jacobi matrix $\hat{T}_K$ using the Perturbation Approach Closed-form algorithm with its equivalent difference quotient $(\hat{T}_K - T_K)/d$ for $N = 5$, $d = .001$

Convergence of Finite Difference Approximation of $(\hat{T}_K)$'				
N	$d = 0.01$	d = 0.001	d = 0.0001	d = 0.00001
10	0.0072	7.2221e-04	7.2270e-05	7.2275e-06
20	0.0733	0.0518	0.0670	0.0571
50	0.1904	0.0699	0.0412	0.0182
100	16.5746	7.3038	0.0465	0.0263

Table 4.9: Confirming the accuracy of the derivatives using the same MATLAB codes as Accuracy of Derivative study.

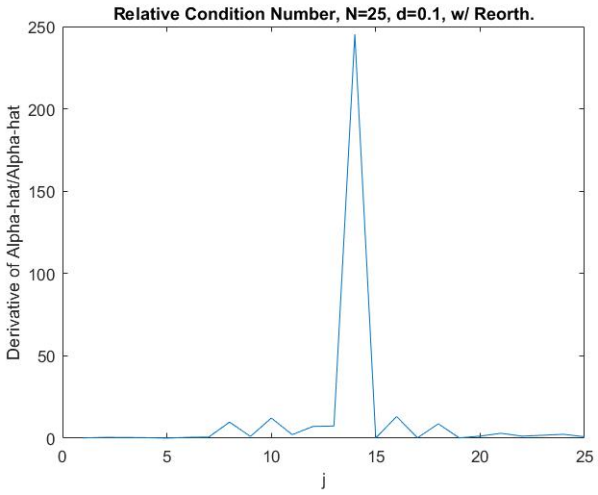
The condition number is an application of the derivative since it is a ratio of the change in output to the change in input. The absolute value of each derivative is the absolute condition number. As defined in [18], the relative condition number for the derivatives of $\hat{\alpha}_j$ are computed to determine the sensitivity of these quantities as follows,

$$K_{rel} = \left| \frac{(\hat{\alpha}_j)'}{\hat{\alpha}_j} \right|. \tag{4.6}$$

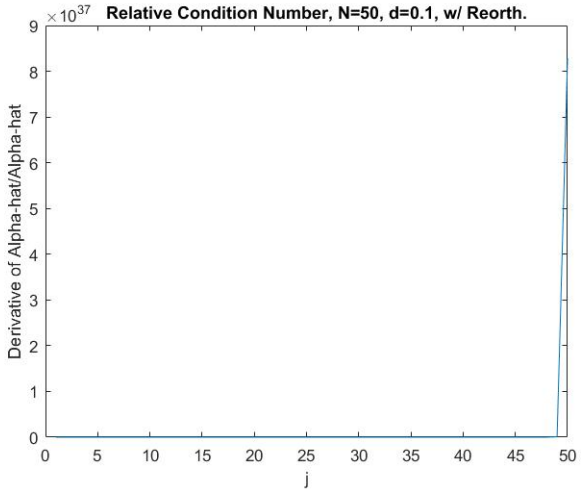
In other aspects of numerical linear algebra, it is expensive to compute a condition number because it is a difficult process. However, if needed, computing the condition number of all of our quantities is attained easily with little expense. The following graphs are plots of the computed condition numbers of $\hat{\alpha}_j$ for N=25, N=50, N=100 and d=0.1 with reorthogonalization.

The first graph shows that one of the $\hat{\alpha}_j$ quantities is far more sensitive than the others. The second and third graph show that the $\hat{\alpha}_j$ quantities are insensitive for a longer period of time then spike for a particular $\hat{\alpha}_j$, which illustrates the care needed in choosing a small enough perturbation parameter d. Thus, to arbitrarily choose a parameter d value to achieve an optimal approximation of a bilinear form (1.1) would not be wise.

(a) ($\hat{\alpha}_j$): N=25, d=0.1



(b) ($\hat{\alpha}_j$): N=50, d=0.1



(c) ($\hat{\alpha}_j$): N=100, d=0.1

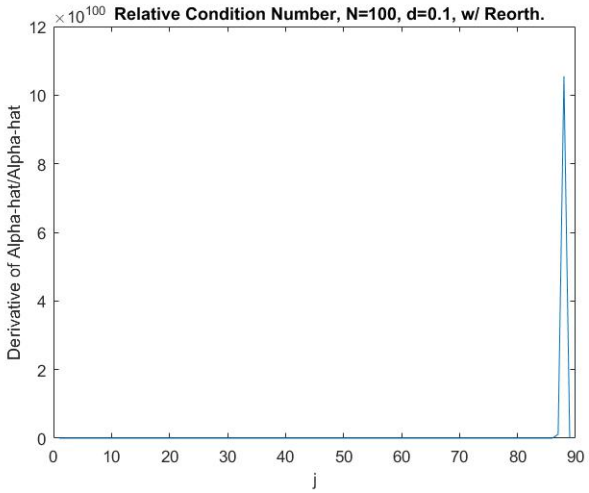


Figure 4.5: Relative Condition Number: $\hat{\alpha}_j$ for $N = 25$, $N = 50$, $N = 100$ and $d = 0.1$ with reorthogonalization

Chapter 5

CONCLUSION

In this dissertation, we developed an algorithm to efficiently approximate bilinear forms expressing every step in the algorithm of approximating bilinear forms in terms of the corresponding steps of approximating quadratic forms. Although the number of steps to approximate in our perturbation method are basically the same as those used in the unsymmetric Lanczos method, we did achieve approximation utilizing less storage by attaining closed-form expressions for the recursion coefficients needed to develop our *Closed-form Perturbation Approach* algorithm. Algorithms that are more efficient in terms of computation and storage for certain applications by which bilinear forms are approximated are needed. Thinking about approximation in this way opens the door for people to look at other situations that have not been thought of yet, instead of using the more common algorithms. A process could be broken down to meet the needs of the application. In this dissertation, we develop algorithms that mimic the unsymmetric Lanczos algorithm with efficiency due to information sharing. When we have these standard methods like Lanczos and others, we can always use them. But, when we have an application where the needs are a little different, it's helpful to understand how the processes like symmetric Lanczos and unsymmetric Lanczos relate to one another so that we can find a way to break things down to accommodate specific cases like Lambers' case. Let's not just settle for the straightforward methods, because sometimes gains can be made from getting a deeper understanding of all these algorithms and how they relate to one another.

The unsymmetric Lanczos algorithm suffers horribly from rounding error due to loss of orthogonality, and studies have been conducted to correct this issue. Future work would be to develop code to amend the Closed-form Perturbation algorithm to improve its stability and accuracy, because reorthogonalization at every iteration comes at great cost. In addition, reorthogonalizing the Lanczos vectors achieved stability for a longer length of time but did not achieve complete stability. Testing developed reothogonalization methods would benefit the study as well.

Appendix A

MATLAB CODES

A.1 'MatGen.m'

We write the MATLAB function 'MatGen.m' to produce and save symmetric matrix A and randomly choose initial vectors **u** and **v** for numerical analysis.

```
%Generate matrices for sensitivity testing
n=50;
A=rand(n);
%Symmetric
A=A+A';
u=rand(n,1);
v=rand(n,1);
save matrices A u v n
```

A.2 'LanczosPHD.m'

We write the MATLAB function 'LanczosPHD.m' which implements the Lanczos algorithm on a randomly generated symmetric $N \times N$ matrix A with initial vector **u**, where n is the number of iterations to produce the Jacobi matrix T_k (2.4) that contains the quantities α_j and β_j computed by the algorithm.

```
function [R,Al,B] = Lanczos_PHD(A,u,n)
T = zeros(n);
N = length(u);
R = zeros(N,n);
Al = zeros(n,1);
B = zeros(n-1,1);

% Variable assignments
r = u;
x = 0;
```

```
%Initialize storage of x's
X = [];

% Lanczos Algorithm: Calculations of alpha & beta
%(recur. relation coeffs.)
for j = 1:n
    %Reorthogonalization
    if j>1
        r=r-X*X'*r;
    end
    Beta = norm(r);
    B(j) = Beta;

    old_x = x;
    x = r/Beta;
    %Store new x
    X=[ X x ];
    R(:,j) = r;

    Alpha = x'*A*x;
    % store alpha in T Matrix
    T(j,j) = Alpha;
    Al(j) = Alpha;

    r = (A*x - Alpha*x)- Beta*old_x;
end
R(:,n+1) = r;
```

A.3 'UnsymLanczosPHD.m'

We write the MATLAB function 'UnsymLanczosPHD.m' which implements the unsymmetric Lanczos algorithm on a randomly generated symmetric $N \times N$ matrix A with initial vectors **u** and **v** where n is the number of iterations to produce the modified Jacobi matrix $\hat{T}_k$ (2.10) that contains the hatted quantities $\hat{\alpha}_j$ and $\hat{\beta}_j$ computed by the algorithm.

```
function [ThU,Rh,AhU,BhU] = UnsymLanczos_PHD(A,u,v,n)
ThU = zeros(n);
```

```
N = length(u);
Rh = zeros(N,n);
AhU = zeros(n,1);
BhU = zeros(n,1);
Ph = zeros(N,n);

% Variable assignments
ph = u;
rh = v;
xh = 0;
yh = 0;

% Initialize storage of x's, y's
X=[];
Y=[];

% Unsymmetric Lanczos Algorithm: Calculations of alpha-hat,
%beta-hat recur relation coeffs.
for j = 1:n
    %Reorthogonalization
    if j>1
        rh=rh-X*Y'*rh;
        ph=ph-Y*X'*ph;
    end
    Betah = sqrt(ph'*rh);
    if j>1
       ThU(j-1,j) = Betah;
       ThU(j,j-1) = Betah;
    end
    BhU(j) = Betah;

    old_x = xh;
    xh = rh/Betah;
    Rh(:,j) = rh;

    %Store new x
```

```
    X=[ X xh ];

    old_y = yh;
    yh = ph/Betah;
    Ph(:,j) = ph;

    %Store new y
    Y=[ Y yh ];

    Alphah = yh'*A*xh;
    % store alpha in T Matrix
    ThU(j,j) = Alphah;
    AhU(j) = Alphah;

    rh = (A*xh - Alphah*xh)- Betah*old_x;
    ph = (A'*yh - Alphah*yh)- Betah*old_y;
end

ThU2 = Y'*A*X;
err = norm(ThU-ThU2(1:n,1:n))/norm(ThU)
```

A.4 'Coeffs2.m'

We write the MATLAB function 'Coeffs2.m' to compute the approximation of the perturbed bilinear form $\mathbf{u}^T f(A)(\mathbf{u}+d\mathbf{v})$ (1.4) by generating $\mathbf{C}_{i,j}$, $\mathbf{F}_{j,j}$, and $\tilde{\mathbf{v}}_j$ using the recurrence relation definitions to compute the modified Jacobi matrix $\hat{T}_k$ 2.10. The unhatted quantities computed by 'LanczosPHD.m' are used to implement this algorithm.

```
function [Th2,C,RhC] = Coeffs2(A,Al,B,R,v,d,n)
C = zeros(n);
F = zeros(n,1);
Th2 = zeros(n);
Ah = zeros(n,1);
Bh = zeros(n-1,1);
RhC = zeros(size(R));

% Variable assignments
```

```
C(1,1) = 1;
F(1,1) = d;
vt(:,1) = v;
Alpha = Al;
Beta = B;

for j = 2:n+1
%Generate Beta_hats
BCpart = 0;
Fpart1 = 0;
for m = 1:j-1
BCpart = (BCpart+(Beta(m))^2*(C(m,j-1))^2);
Fpart1 = Fpart1+(C(m,j-1)*F(j-1,j-1)*(R(:,m))'*vt(:,j-1));
end
Betah = sqrt(BCpart + Fpart1);
if j>2
Th2(j-1,j-2) = Betah;   %Betahat stored in Th2,  sub & super diag
Th2(j-2,j-1) = Betah;
end
Bh(j-1) = Betah;     %Allows me to call Beta_hats as a vector
%Generate F Coefficients
FCoeff = (F(j-1,j-1))/Bh(j-1);
F(j,j) = FCoeff;         %Store F Coefficeints in F Matrix

%Generate Alpha_hats
Alphah = 0;
Fpart2 = 0;
for h = 1:j-1
for k = 1:j-1
if h==k
Alphah = (Alphah +(C(h,j-1)*C(k,j-1)*Beta(h)*Beta(k)*Alpha(h)));

elseif h-k==1
Alphah = (Alphah +(C(h,j-1)*C(k,j-1)*Beta(h)*Beta(k)*Beta(k+1)));

elseif h-k==-1
```

```
Alphah = (Alphah +(C(h,j-1)*C(k,j-1)*Beta(h)*Beta(k)* Beta(h+1)));
end
end
Fpart2 = Fpart2+(C(h,j-1)*F(j-1,j-1)*(A*R(:,h))'*vt(:,j-1));
end
Alphah = (1/(Bh(j-1))^2)*(Alphah+Fpart2);
Th2(j-1,j-1) = Alphah;   %Store  Alphahat, Th Matrix as diag entries
Ah(j-1) = Alphah;          %Allows me to call Alphahats as a vector

if j==2                          %Generating v~s
vt(:,j) = (A*vt(:,j-1)) - (Ah(j-1)*vt(:,j-1));
else
vt(:,j) = (A*vt(:,j-1))- Ah(j-1)*vt(:,j-1))-((Bh(j-1))^2)*(vt(:,j-2));
end

for i = 1:j
if i==1 && j==2
C(1,2) = (Alpha(1)-Ah(1))/(Bh(1));

%Coeff Case 2 of recurrence;
elseif i==1 && j>=3
Coeff = ((C(1,j-1)*Alpha(1))/Bh(j-1))...
            +((C(2,j-1)*(Beta(2))^2)/(Bh(j-1)*(Beta(1))))...
            -((C(1,j-1)*(Ah(j-1)))/(Bh(j-1)))-((C(1,j-2)...
            *Bh(j-1)))/(Bh(j-2));
C(i,j) = Coeff;

%Coeff Case 3 of recurrence;
elseif i==j-1 && j>2
Coeff = ((C(j-1,j-1)*Alpha(j-1))/(Bh(j-1)))+(C(j-2,j-1)...
        *(Beta(j-2))/((Bh(j-1))))-(C(j-1,j-1)*Ah(j-1))/Bh(j-1);
C(i,j) = Coeff;

%Coeff Case 4 of recurrence;
elseif i==j && j>=2
Coeff = ((C(j-1,j-1)*Beta(j-1))/Bh(j-1));
```

```
C(i,j) = Coeff;

else
%i
%j
Coeff = ((C(i,j-1)*Alpha(i))/Bh(j-1))...
       +((C(i-1,j-1)*Beta(i-1))/Bh(j-1))...
       +((C(i+1,j-1)*(Beta(i+1))^2)/(Bh(j-1)*Beta(i)))...
       -((C(i,j-1)*Ah(j-1))/Bh(j-1))...
       -((C(i,j-2)*Bh(j-1))/Bh(j-2));

%Store each coefficient in C Matrix
C(i,j) = Coeff;
end
end

%Generate r_hats using Cs
RhC_Cpart = 0;
for p = 1:j-1
RhC_Cpart = (C(p,j-1)*(R(:,p)));
end
RhC_Fpart = F(j-1,j-1)*vt(:,j-1);
RhC(:,j-1) = RhC_Cpart+RhC_Fpart;
 end
```

A.5 'Coeffs2CFwDeriv.m'

We write the MATLAB function 'Coeffs2CFwDeriv.m' to compute the approximation of the perturbed bilinear form $\mathbf{u}^T f(A)(\mathbf{u}+d\mathbf{v})$ (1.4) by generating $\mathbf{C}_{j,j}$, $\mathbf{F}_{j,j}$, and $\tilde{\mathbf{v}}_j$ using their closed-form definitions to compute the modified Jacobi matrix $\hat{T}_k$ (2.10) as discussed in Chapter 3. The unhatted quantities from the quadratic form computed by 'LanczosPHD.m' are used to implement this algorithm we call *The Closed-form Perturbation Approach* algorithm. We also compute the derivatives of each quantity for our sensitivity study.

```
function [ThCF,ThCFdrv,Cs,Ah,Ahdrv,Bh,Bhdrv] = Coeffs2CFwDeriv

%Space Allocation for speed
```

```
Cs = zeros(n);
Csdrv = zeros(n);
F = zeros(n,1);
Fdrv = zeros(n,1);
ThCF = zeros(n);
ThCFdrv = zeros(n);
Ah = zeros(n,1);
Ahdrv = zeros(n,1);
Bh = zeros(n-1,1);
Bhdrv = zeros(n-1,1);


% Variable assignments
Cs(1,1) = 1;
Csdrv(1,1) = 0;
Cs0(1,1) = 1;
F(1,1) = d;
Fdrv(1,1) = 1;
vt(:,1) = v;
vt0(:,1) = v;
vtdrv(:,1) = 0*v;
Alpha = Al;
Beta = B;

for j = 2:n+1
%Generate Beta_hats
Betah = sqrt((Cs(j-1,j-1))^2*(Beta(j-1))^2
       +(Cs(j-1,j-1))* F(j-1,j-1)*(R(:,j-1))'*vt(:,j-1));
if j>2
ThCF(j-1,j-2) = Betah;  %Betahat stored in Th
ThCF(j-2,j-1) = Betah;
end
Bh(j-1) = Betah; %Allows me to call Betahats as a vector

%Generate Derivatives of Beta_hats
Betahdrv = (Csdrv(j-1,j-1)*Beta(j-1))+(Fdrv(j-1,j-1)*(R(:,j-1))'...
```

```
              *vt0(:,j-1))/(2*Beta(j-1));
if j>2
ThCFdrv(j-1,j-2) = Betahdrv; %Derivative of Betahats stored in Th
ThCFdrv(j-2,j-1) = Betahdrv;
end
Bhdrv(j-1) = Betahdrv;  %To call the derivs of Betahats as a vector

%Generate F Coefficients
FCoeff = (F(j-1,j-1))/Bh(j-1);
F(j,j) = FCoeff;   %Store F Coefficeints in F Matrix

FCoeffdrv = Fdrv(j-1,j-1)/Beta(j-1);
Fdrv(j,j) = FCoeffdrv;

%Generate C Coefficients;
Coeff = ((Cs(j-1,j-1)*Beta(j-1))/Bh(j-1));
Cs(j,j) = Coeff;    %Store each coefficient in Cs Matrix
Cs0(j,j) = Cs0(j-1,j-1);

%Generate Derivatives of C Coefficients;
Coeffdrv = Csdrv(j-1,j-1)-(Bhdrv(j-1)*Cs0(j-1,j-1))/(Beta(j-1));
Csdrv(j,j) = Coeffdrv;

%Generating Alpha_hats
%Betahat stored in Th
if j==2
Alphah = (Alpha(j-1) + Cs(j,j)*F(j,j)*(R(:,j))'*vt(:,j-1));
else
Alphah = (Alpha(j-1) + Cs(j,j)*F(j,j)*(R(:,j))'*vt(:,j-1)...
           - Cs(j-1,j-1)*(F(j-1,j-1)*(R(:,j-1))'*vt(:,j-2)));
end

ThCF(j-1,j-1) = Alphah;  %Store each Alphahat in Th as diag entries
Ah(j-1) = Alphah;     %Allows me to call Alphahats as a vector

%Betahat stored in Th
```

```
%Generate Derivatives of Alpha_hats
if j==2
Alphahdrv = (Cs0(j,j)*Fdrv(j,j)*(R(:,j))'*vt0(:,j-1));
else
Alphahdrv = (Cs0(j,j)*Fdrv(j,j)*(R(:,j))'*vt0(:,j-1)-Cs0(j-1,j-1)...
          *(Fdrv(j-1,j-1)*(R(:,j-1))'*vt0(:,j-2)));
end

ThCFdrv(j-1,j-1) = Alphahdrv; %Store Alphahat in Th as diag entries
Ahdrv(j-1) = Alphahdrv;  %To call Alphahats as a vector

%Generate v~s
if j==2
vt(:,j) = (A*vt(:,j-1)) - (Ah(j-1)*vt(:,j-1));
vt0(:,j) = (A*vt0(:,j-1)) - (Alpha(j-1)*vt0(:,j-1));
else
vt(:,j) = (A*vt(:,j-1))-(Ah(j-1)*vt(:,j-1))-((Bh(j-1))^2)*(vt(:,j-2));
vt0(:,j) = (A*vt0(:,j-1))-(Alpha(j-1)*vt0(:,j-1))-((Beta(j-1))^2)...
         *(vt0(:,j-2));
end

%Generate Derivatives of v~s
if j==2
vtdrv(:,j) = (A*vtdrv(:,j-1)) - (Alpha(j-1)*vtdrv(:,j-1)...
           - Ahdrv(j-1)*vt0(:,j-1));
else
vtdrv(:,j) = (A*vtdrv(:,j-1))-(Alpha(j-1)*vtdrv(:,j-1))-(Ahdrv(j-1)...
           *vt0(:,j-1))-(2*Beta(j-1)*Bhdrv(j-1))*vt0(:,j-2)...
           +(Beta(j-1))^2*vtdrv(:,j-2);
end
end
```

A.6 'TestPHD.m'

We write the MATLAB code *'TestPHD.m'* to generate the outputs for 'LanczosPHD.m', 'UnsymLanczosPHD', 'Coeffs2',and 'Coeffs2CFwDeriv'. This code also computes our

"Good" *d* and the error to show code accuracy discussed in Chapter 4.

```
load matrices
d=0.001;    %make smaller if T-hat has any complex entries
Bh2 = zeros(n,1);

[R,Al,B] = Lanczos_PHD(A,u,n);
[ThU,Rh,AhU,BhU] = UnsymLanczos_PHD(A,u,u+d*v,n);
[Th2,C,RhC] = Coeffs2(A,Al,B,R,v,d,n);
[ThCF,ThCFdrv,Cs,Ah,Ahdrv,Bh,Bhdrv] = Coeffs2CFwDeriv
(A,Al,B,R,v,d,n);
T = diag(Al)+diag(B(2:end),1)+diag(B(2:end),-1);

%ThFD (Th Forward difference/Finite difference)
%Should be approx equal to Thdrv
ThFD = (ThCF-T)/d;

%Newton's Method for a "Good" d
ds = abs(B ./(2*Bhdrv));
MINd = min(ds);
Gd = MINd/10;

%Testing for Error
%Coeffs2wDeriv vs Unsymmetric Lanczos
ThError1 = norm(ThCF - ThU)/norm(ThCF);
ThError1_10 = norm(ThCF(1:10,1:10) - ThU(1:10,1:10))...
/norm(ThCF(1:10,1:10));
ThError1_25 = norm(ThCF(1:25,1:25) - ThU(1:25,1:25))...
/norm(ThCF(1:25,1:25));
ThError1_50 = norm(ThCF(1:50,1:50) - ThU(1:50,1:50))...
/norm(ThCF(1:50,1:50));
ThError1_100 = norm(ThCF(1:100,1:100) - ThU(1:100,1:100))...
/norm(ThCF(1:100,1:100));
ThError1_150 = norm(ThCF(1:150,1:150) - ThU(1:150,1:150))...
/norm(ThCF(1:150,1:150));
```

```
%Coeffs2 vs Unsymmetric Lanczos
ThError2 = norm(Th2 - ThU)/norm(Th2);
ThError2_10 = norm(Th2(1:10,1:10) - ThU(1:10,1:10))...
/norm(Th2(1:10,1:10));
ThError2_25 = norm(Th2(1:25,1:25) - ThU(1:25,1:25))...
/norm(Th2(1:25,1:25));
ThError2_50 = norm(Th2(1:50,1:50) - ThU(1:50,1:50))...
/norm(Th2(1:50,1:50));
ThError2_100 = norm(Th2(1:100,1:100) - ThU(1:100,1:100))...
/norm(Th2(1:100,1:100));
ThError2_150 = norm(Th2(1:150,1:150) - ThU(1:150,1:150))...
/norm(Th2(1:150,1:150));
```

BIBLIOGRAPHY

[1] K. Atkinson, *An Introduction to Numerical Analysis*, 2nd ed. Wiley, Hoboken (1989).

[2] Z. Bai and G. H. Golub, "Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices", *Annals Numer. Math.* **4** (1997), p. 29-38.

[3] R. H. Bartels and G. W. Stewart, "Solution of the matrix equation $AX + XB = C$", *Communications of the ACM* **15**(9) (1972), p. 820-826.

[4] D. Calvetti, G. H. Golub and L. Reichel, "Estimation of the L-curve via Lanczos bidiagonalization", *BIT* **39**(4) (1999), p. 603-619.

[5] D. Calvetti, S.-M. Kim and L. Reichel, "Quadrature Rules based on the Arnoldi Process", *SIAM J. Matrix. Anal. Appl.* **26**(3) (2005), p. 765-781.

[6] A. Cibotarica, J. V. Lambers, and E. M. Palchak, "Solution of Nonlinear Time-Dependent PDE Through Componentwise Approximation of Matrix Functions", *Journal of Computational Physics* **321** (2016), p. 1120-1143.

[7] R. G. Dúran, "Galerkin Approximations and Finite Element Methods", (2005).

[8] P. Fika, M. Mitrouli, "Aitken's Method for Estimating Bilinear Forms Arising in Applications", *Calcolo 54*, (2017), p. 455–470.

[9] K. E. Gilbert and M. J. White, "Application of the parabolic equation to sound propagation in a refracting atmosphere", *J. Acoust. Soc. Am.* **85**(2) (1989), p. 630-637.

[10] G. H. Golub and G. Meurant, *Matrices, Moments and Quadrature with Applications*, Princeton University Press, 2009.

[11] G. H. Golub, M. Stoll and A. Wathen, *Approximation of the scattering amplitude*, *Elec. Trans. Numer. Anal.* **31** (2008), p. 178-203.

[12] G. H. Golub and R. Underwood, "The Block Lanczos Method for Computing Eigenvalues", *Mathematical Software III*, J. R. Rice, ed. (1977), p. 361-377.

[13] G. H. Golub and J. Welsch, "Calculation of Gauss Quadrature Rules", *Mathematics of Computation* **23** (1969), p. 221-230.

[14] J. V. Lambers, "Derivation of High-Order Spectral Methods for Time-Dependent PDEs Using Modified Moments", *Electronic Transactions on Numerical Analysis* **28** (2008), p. 114-135.

[15] J. V. Lambers, "Enhancement of Krylov Subspace Spectral Methods through Block Lanczos Iteration", *Electronic Transactions on Numerical Analysis* **31** (2008), p. 86-109.

[16] J. V. Lambers, "Krylov Subspace Spectral Methods for Variable-Coefficient Initial Boundary Value Problems", *Electronic Transactions on Numerical Analysis* **20** (2005), p. 212-234.

[17] J. V. Lambers, “Practical Implementation of Krylov Subspace Spectral Methods”, *J Science Computation* **32** (2007), p. 451-459.

[18] J. V. Lambers and A. C. Sumner, *Explorations In Numerical Analysis*, World Scientific, 2018.

[19] C. Lanczos, “An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators", *Journal of Research of the National Bureau of Standards* **45** (1950), p. 255-269.

[20] G. Meurant, “The computation of bounds for the norm of the error in the conjugate gradient algorithm”, *Numer. Algo.* **16** (1997), p. 77-87.

[21] C. D. Meyer, , *Matrix Analysis and Applied Linear Algebra*, 2nd ed. Society for Industrial and Applied Mathematics, (2000).

[22] P. E. Saylor and D. C. Smolarski, “Why Gaussian quadrature in the complex plane?”, *Numer. Algorithms* **26** (2001), p. 251-280.

[23] G. E. Shilov, *Linear Algebra*, Dover Publications Inc. (1978).