GPU ACCELERATED POLYNOMIAL SPECTRAL TRANSFORMATION METHODS

By:

JARED L. AURENTZ

A dissertation submitted in fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
Department of Mathematics

AUGUST 2014

To the faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of JARED L. AURENTZ find it satisfactory and recommend that it be accepted.

David S. Watkins, Ph.D., Chair

Kevin D. Cooper, Ph.D.

Sandra C. Cooper, Ph.D.

Acknowledgements

To my parents,

thank you for teaching me how to work.

To my advisor,

thank you for helping me find something to work for.

GPU ACCELERATED POLYNOMIAL SPECTRAL

TRANSFORMATION METHODS

Abstract

by Jared L. Aurentz, Ph.D.
Washington State University
August 2014

Chair: David S. Watkins

A new class of methods for accelerating linear system solving and eigenvalue computations for positive definite matrices using GPUs is presented. This method makes use of techniques from polynomial approximation theory to construct new types of polynomial spectral transformations that are easy to parallelize and when combined with GPUs can give a factor of 100 reduction in run times for certain matrices. These methods also require significantly less memory than traditional methods, making it possible to solve large problems on an average workstation.

Contents

List of Figures

List of Tables

Abbreviations

CG	**C**onjugate **G**radient Method
CPU	**C**entral **P**rocessing **U**nit
CUBLAS	**CU**DA **B**asic **L**inear **A**lgebra **S**ubprograms
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
DCT	**D**iscrete **C**osine **T**ransform
DFT	**D**iscrete **F**ourier **T**ransform
GMRES	**G**eneralized **M**inimum **RES**idual
GPU	**G**raphics **P**rocessing **U**nit
ILU	**I**ncomplete **LU** factorization
IRLM	**I**mplicitly **R**estarted **L**anczos **M**ethod
LAPACK	**L**inear **A**lgebra **PACK**age

Symbols

$\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{R}^{m\times n}$	set of real numbers, vectors, and matrices
$\mathbb{C}$, $\mathbb{C}^n$, $\mathbb{C}^{m\times n}$	set of complex numbers, vectors, and matrices
$\|\cdot\|_2$	vector or matrix 2-norm
$\|\cdot\|_F$	vector or matrix Frobenius norm
$\|\cdot\|_\infty$	supremum norm of a function
$\kappa_2(A)$	2-norm condition number of a matrix
A^T, A^H	transpose and conjugate transpose
$\operatorname{span}\{v_1, \ldots, v_n\}$	space spanned by vectors
$\dim(\mathcal{U})$	dimension of a vector space
I, I_n	arbitrary and $n \times n$ identity matrix
$\sigma(A)$	spectrum of a matrix
$\rho(A)$	spectral radius of a matrix
$\mathrm{d}(\mathcal{U}, \mathcal{V})$	distance between two subspaces
$\lambda_{\max}$	the largest eigenvalue of a positive definite matrix
$\lambda_{\min}$	the smallest eigenvalue of a positive definite matrix

Chapter 1

Introduction

1.1 Motivation

Linear algebra computations involving large sparse positive definite matrices are ubiquitous in science and engineering. These problems arise naturally in the approximation of elliptic PDEs in fields like structural engineering and quantum physics. Solving these problems approximately typically involves computing the solution to a linear system $Ax = b$ or computing the eigenvalues and eigenvectors $Ax = \lambda x$ for a small subset of the spectrum.

For large sparse positive definite matrices a wide variety of fast and efficient algorithms have been developed. Many of these methods have been around for decades and were not designed to make efficient use of emerging parallel architectures like multi-core CPUs and GPUs. As these architectures become more prevalent, the inefficiencies become more apparent. The time has come to rethink the way in which numerical computations are performed.

1.1.1 Linear Systems

Methods for computing the solution to $Ax = b$ typically fall into two categories: direct or iterative [10, 25, 33, 37, 36, 43, 52, 67, 69, 72, 74, 77]. For direct methods one typically computes some variant of an LU or Cholesky factorization and then performs a sequence of back substitutions to compute x. For large sparse matrices these methods typically give factors with significantly more non-zeros, making them difficult or impossible to store. Even if the factors could be stored, the computation of the factors is difficult to parallelize when the sparsity pattern is irregular.

For iterative methods, a sequence of approximate solutions are generated in a way such that each new iterate is closer to the correct solution than the last. Two major classes of iterative methods are Richardson and Krylov type methods. Both of these types replace factorizations with matrix-vector multiplications. For large sparse systems this is often easy to parallelize. However, these methods only work well when the matrices are well conditioned. When the matrices are ill-conditioned the system must be transformed, typically by a preconditioner, to one that is easier to solve. The preconditioner must be applied at every iteration and thus the iterative method is only parallelizable if the preconditioner is. Unfortunately most preconditioners are related to direct methods and suffer from the same drawbacks.

1.1.2 Eigenvalues and Eigenvectors

Methods for computing a subset of the spectrum of a large sparse matrix are necessarily iterative and are especially adept at computing eigenvalues that are well separated and near the periphery of the spectrum [18, 25, 36, 46, 56, 62, 67, 70, 72, 76, 77, 78]. In order to calculate eigenvalues that are clustered or inside the spectral radius one typically adopts a shift and invert strategy

$A \to (A - \sigma I)^{-1}$. This involves the solution of a linear system at each iteration and means that the eigensolver is only as parallel as the linear solver.

The shift and invert strategy can be viewed as a rational spectral transformation via the function $f(z) = (z-\sigma)^{-1}$. This transforms the original system into a new system with the same eigenvectors but a new eigenvalue distribution. If the shift σ is close to the desired eigenvalues of the original system, these eigenvalues will be mapped to the periphery in the new transformed system.

1.2 Polynomial Spectral Transformations

For both linear system solving and eigenvalue computations the transformation $f(A) = (A-\sigma I)^{-1}$ can be thought of as a spectral transformation via the rational function $f(z) = (z - \sigma)^{-1}$, with $\sigma = 0$ for linear systems. The term spectral transformation is used since for every eigenvalue λ of A, $f(\lambda)$ is an eigenvalue of $f(A)$, thus f transforms the spectrum of A.

Using this analogy other spectral transformations can be constructed using more general rational functions, $f(z) = p(z)/q(z)$ where p and q are polynomials. The difficulty of solving linear systems remains if q is not constant. If q is constant f would be a polynomial and applying $f(A)$ to a vector would only require matrix-vector multiplication. The question is: *Can useful spectral transformations be constructed using only polynomials?*

1.2.1 Polynomial Preconditioners for Linear Systems

Some partial answers already exist for linear systems in the area of polynomial preconditioners [7, 8, 6, 29, 32, 60, 61]. Here the goal was always to approximate $A^{-1} \approx p(A)$ via a polynomial in

order to accelerate the convergence of iterative methods. These methods differ in the construction of the polynomial p. The best polynomials typically require the most spectral information.

The need for accurate approximations of the spectrum initially made polynomial preconditioners unpopular. Despite this difficulty interest in these methods has renewed due to their intrinsic parallelism. In the past few years implementations of polynomial preconditioners on multi-core CPUs [48] and GPUs [80] have shown significant speed ups in runtime versus more traditional preconditioning methods.

1.2.2 Polynomial Preconditioners for Eigenvalues and Eigenvectors

More general rational spectral transformations for accelerating eigenvalue computations were perhaps first proposed by Ruhe [58]. These so-called rational Krylov methods still require the solution of linear systems whenever the rational function has poles. The idea of strictly using polynomials to accelerate eigenvalue computations was first proposed by Saad [62, p. 200]. In this implementation the polynomial preconditioner performed poorly when compared to a traditional shift and invert strategy. Recently Saad and others have been successfully developing polynomial filters for isolating interior eigenvalues of Hermitian matrices using techniques from digital signal processing [64].

1.3 Graphics Processing Units

The title of this thesis begins with the acronym GPU, short for Graphics Processing Unit. GPUs have been around nearly as long as the personal computer. These devices were initially designed to offload the computationally expensive task of rendering graphics. Graphics computations are often

very easy to parallelize and the GPUs themselves have become highly parallel platforms. Some GPUs today have compute cores that number in the thousands and dedicated memory measuring in gigabytes, all in a device that fits neatly inside an average workstation.

In the last decade or so these devices have caught the attention of numerical analysts and have yielded new parallel algorithms and software packages to take advantage of GPUs [2, 42, 41]. This thesis presents a class of methods that take advantage of this technology in order to efficiently solve some fundamental problems in numerical linear algebra.

1.4 Outline of the Thesis

The outline of the thesis is as follows. In Chapter 2 the basic facts about spectral transformations are reviewed and the connection with linear system solving and eigenvalue computations is made. New techniques for constructing polynomial spectral transformations using techniques from Chebyshev interpolation are also presented. In Chapter 3 details for constructing polynomial approximate inverses are discussed, with special attention on the importance of spectral information. In Chapter 4 novel techniques for constructing polynomial spectral transformations for accelerating eigenvalue computations are presented. These methods are very flexible and require only approximate knowledge of the spectrum. In Chapter 5 experiments are performed to test the accuracy and efficiency of these new methods using a GPU.

1.5 Notation

Unless otherwise noted, lower case Greek letters α, β, γ, will be used to denote scalars in $\mathbb{R}$ or $\mathbb{C}$, lower case Roman letters x, b, f, g, will be used to denote vectors in $\mathbb{R}^n$ or $\mathbb{C}^n$ or scalar functions

mapping $\mathbb{R} \to \mathbb{R}$ or $\mathbb{C} \to \mathbb{C}$, and upper case roman letters A, B, Q, will be used to denote matrices in $\mathbb{R}^{m\times n}$ or $\mathbb{C}^{m\times n}$. The main results of this thesis apply to positive definite matrices and unless otherwise stated square matrices $A \in \mathbb{R}^{n\times n}$ or $\mathbb{C}^{n\times n}$ will be assumed to be positive definite. The *transpose* and *conjugate transpose* of a matrix A will be denoted as A^T and A^H respectively. For $A \in \mathbb{R}^{n\times n}$ or $\mathbb{C}^{n\times n}$ positive definite $\rho(A)$ will correspond to the *spectral radius* and $\lambda_{\min}$ and $\lambda_{\max}$ will correspond to the largest and smallest eigenvalues of A respectively.

Chapter 2

Spectral Transformations and Chebyshev Polynomial Approximation

2.1 Basics of spectral transformations

A spectral transformation of a semi-simple matrix A is any complex valued function $f(z)$ defined on its spectrum [40]. There is no requirement that $f(z)$ be differentiable or even continuous.

Definition 2.1. Let $n > 0$ and $A \in \mathbb{C}^{n\times n}$ be semi-simple. For any $f := \mathbb{C} \to \mathbb{C}$, $f(A)$ is defined as,

$$f(A) = V \begin{bmatrix} f(\lambda_1) & & \\ & f(\lambda_2) & \\ & & \ddots \end{bmatrix} V^{-1},$$

where,

$$A = V \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \end{bmatrix} V^{-1},$$

is the spectral decomposition of A.

This definition makes for a great theoretical tool but has little practical significance in general. Take for example $f(z) = |z|$. This function is defined everywhere in the complex plane yet computing $f(A)$ is only possible if the spectrum of A already known. For a spectral transformation to be useful it needs to transform the spectrum of A using little to no spectral information.

One class of functions for which this definition is useful is the class of polynomials. If $p(z)$ is a polynomial,

$$p(z) = \sum_{i=0}^{n} c_i z^i, \ c_i \in \mathbb{C}, \ 0 \leq i \leq n,$$

then $p(A)$ can be computed via,

$$p(A) = \sum_{i=0}^{n} c_i A^i.$$

It can easily be shown that,

$$\sum_{i=0}^{n} c_i A^i = V \begin{bmatrix} p(\lambda_1) & & \\ & p(\lambda_2) & \\ & & \ddots \end{bmatrix} V^{-1}.$$

Every spectral transformation of a semi-simple matrix is equivalent to a polynomial spectral transformation. Let $p(z)$ be any polynomial such that $p(\lambda_i) = f(\lambda_i)$, $i = 1, \ldots, n$ then from Definition 2.1

$p(A) = f(A)$. Of course the eigenvalues would have to be known in order to construct such a polynomial.

For the remainder of this chapter it is assumed that for some $n > 0$, $A \in \mathbb{C}^{n \times n}$ is positive definite and the eigenvalues of A are labeled such that $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. If a polynomial could be constructed such that $p(z) = f(z)$, $z \in [\lambda_1, \lambda_n]$ then $p(A) = f(A)$ would still hold. In general this will not be possible for all f, i.e. f discontinuous, but for certain well behaved functions it will be possible to construct a polynomial that is a good approximation to f on $[\lambda_1, \lambda_n]$. If p is close to f then $p(A)$ will to be close to $f(A)$.

2.1.1 Spectral Transformations for Linear Systems

For linear systems there is really only one spectral transformation that is relevant $f(z) = z^{-1}$. Here f is a rational function with a pole at the origin. Since A is positive definite, $\lambda_1 > 0$ and f is analytic on $[\lambda_1, \lambda_n]$. Having the interval $[\lambda_1, \lambda_n]$ requires exact knowledge of part of the spectrum, something that might not be available. More generally the spectral interval will have to be relaxed even to an interval $[\alpha, \beta]$ such that $[\lambda_1, \lambda_n] \subseteq [\alpha, \beta]$. This relaxation is more realistic to the knowledge that might be obtained from spectral localization theorems or rough approximations that were computed numerically.

In the case where $\alpha > 0$, f is analytic on $[\alpha, \beta]$. In the case where the best guess for α is 0, f will no longer be analytic and the approximation becomes more challenging. In fact no polynomial will ever be able to accurately approximate f near $z = 0$. In Chapter 3 some solutions will be suggested for dealing with the pole at the origin.

2.1.2 Spectral Transformations for Eigenvalues and Eigenvectors

For eigenvalue computations, choosing the right spectral transformation is less obvious. If for example, the interest is in computing the smallest j eigenvalues of A then a function f such that,

$$\max_{j<i\leq n}\{|f(\lambda_i)|\} < \min_{1\leq i\leq j}\{|f(\lambda_i)|\}, \tag{2.1}$$

is desired. If $\lambda_j < \lambda_{j+1}$ then this is always possible and there are infinitely many functions that will guarantee (2.1). Unfortunately, without accurate spectral information many of these functions will be difficult to approximate. One way to avoid such complications is to use transformations with more "global" properties. For example, z^{-1} decreases monotonically on the positive real axis and this behavior guarantees that (2.1) is satisfied. If the spectrum of A is contained in the interval $[\alpha, \beta]$ then any function that decreased monotonically on $[\alpha, \beta]$ would guarantee (2.1). Moreover the steeper the slope the better the separation of the transformed eigenvalues which gives faster convergence.

Good spectral transformations do not end at z^{-1}. For example $e^{-(z-\alpha)}$, $e^{-(z-\alpha)^2}$ and $(z-\alpha+.01)^{-100}$ all decrease monotonically on $[\alpha, \beta]$. A polynomial p could be constructed to be an accurate approximation of any of these and be used as a spectral transformation.

2.2 Chebyshev approximation on the real line

Whether for linear systems or eigenvalue computations, the question of constructing polynomial spectral transformations has now become a question of polynomial approximation for functions on a subset of the real line. One set of polynomials which are well suited to this task are the

so-called Chebyshev polynomials. The following draws heavily from [71] but similar results have also appeared in [14, 49, 34, 75].

Chebyshev approximation comes in several flavors. The focus of this discussion will be polynomial interpolation using Chebyshev polynomials of the 1^{st} kind. The reason for this choice is the intimate and useful connection of these polynomials and the Discrete Fourier Transform (DFT) [15, 21]. This connection allows for the fast construction of accurate approximations that are easy to use. The following is restricted to functions on $[-1, 1]$ which is minor as any interval $[\alpha, \beta]$ can be easily mapped to $[-1, 1]$.

Definition 2.2. Let $T_0(z) = 1$, $T_1(z) = z$ and

$$T_{k+1}(z) = 2zT_k(z) - T_{k-1}(z),\ k \geq 1.$$

The set $\{T_i(z)\}_{i=0}^{\infty}$ are called the *Chebyshev polynomials of the* 1^{st} *kind.*

The Chebyshev polynomials of the 1^{st} kind enjoy many useful properties on $[-1, 1]$: orthogonality, equi-oscillation, aliasing, most of which come directly from their connection to Fourier. It is these properties that make them so useful.

Definition 2.3. For $m > 0$, let

$$z_k = \cos\left(\frac{\pi k}{m}\right),\ 0 \leq k \leq m.$$

The set $\{z_i\}_{i=0}^{m}$ are called the $(m+1)$ *Chebyshev extreme points* or *Chebyshev points of the* 1^{st} *kind.*

The $(m+1)$ Chebyshev extreme points are exactly the points such that $T_m(z_k) = (-1)^k$, $0 \leq k \leq m$. These points correspond to the $2m$ roots of unity used in the DFT. Just as with Fourier the Chebyshev extreme points will be used as sampling points when constructing polynomial approximations. Unlike Fourier the Chebyshev extreme points are not uniformly spaced in $[-1, 1]$. Instead they cluster at the ends of the interval, a fact that cannot be overlooked when trying to compute interior eigenvalues.

Definition 2.4. Let $m > 0$ and f be defined on $[-1, 1]$. The m^{th} *degree Chebyshev interpolant of* f is the polynomial p_m,

$$p_m(z) = \sum_{i=0}^{m} a_i T_i(z),$$

such that $p_m(z_k) = f(z_k)$, for $0 \leq k \leq m$, where $\{z_k\}_{k=0}^{m}$ are the $(m+1)$ Chebyshev extreme points.

The construction of p_m is straightforward. First set $m = 8$ and sample f at the $(m+1)$ Chebyshev extreme points. Then compute the Discrete Cosine Transform (DCT) [3] of $\{f(z_k)\}_{k=0}^{m}$ to obtain the coefficients $\{a_i\}_{i=0}^{m}$. If the highest degree coefficients, $a_m, a_{m-1}, a_{m-2}, \ldots$ are smaller than some tolerance set them to 0 and use a lower degree approximation. If none of them are small increase m to 16 and try again. This process is continued with $m = 32, 64, 128, \ldots$ until a suitably accurate approximation has been found. Algorithm 1 outlines this process and the reader is referred to [71, Chapter 3] for more details.

Each iteration is on the order of $m \log(m)$. For general functions it will not be known in advance how high the degree of the approximating polynomial will have to be. However, for differentiable functions a good estimate can found using the knowledge from the next two theorems.

Algorithm 1 Chebyshev Interpolation

Require: f, ϵ, MaxPow2
  $M \leftarrow \|f\|_\infty$
  for $i = 3$ to MaxPow2 **do**
    $m \leftarrow 2^i$
    $\{f_k\}_{k=0}^m \leftarrow \{f\left(\cos(\pi k/m)\right)\}_{k=0}^m$
    $\{a_k\}_{k=0}^m \leftarrow \mathrm{DCT}\left(\{f_k\}_{k=0}^m\right)$
    for $j = 0$ to m **do**
      if $|a_{m-j}| > \epsilon M$ **and** $j > 0$ **then**
        $m \leftarrow m - j$
        return $\{a_k\}_{k=0}^m$
      else if $|a_{m-j}| \leq \epsilon M$ **and** $j = 0$ **then**
        increase i and try again
      else
        increase i and try again
      end if
    end for
  end for

Definition 2.5. Given a real interval $[\alpha, \beta]$, the *infinity norm* between two functions f and g from $[\alpha, \beta]$ to $\mathbb{C}$ is defined as,

$$||f - g||_\infty = \sup_{z \in [\alpha,\beta]} |f(z) - g(z)|.$$

Theorem 2.6 (Convergence for differentiable functions, [71, p. 53])**.** *For some integer $k \geq 1$ assume f has $k-1$ absolutely continuous derivatives and a k^{th} derivative of bounded variation V on $[-1, 1]$, then for $m > k$,*

$$||p_m - f||_\infty \leq \frac{4V}{\pi k (m-k)^k}.$$

Definition 2.7. Let $\rho > 0$, then the *open Bernstein ellipse* $\mathcal{E}_\rho$ is the interior of the ellipse whose foci are ± 1 and whose half axes sum to ρ.

Theorem 2.8 (Convergence for analytic functions, [71, p. 57])**.** *For some $\rho > 1$ assume f has an analytic continuation in $\mathcal{E}_\rho$ then,*

$$||p_m - f||_\infty \leq 4 \max_{z \in \overline{\mathcal{E}}_\rho} |f(z)| \frac{\rho^{-m}}{\rho - 1}.$$

Theorems 2.6 and 2.8 say that smooth functions are the easiest to approximate. The smoother the function the lower the degree of the polynomial approximation. The next example makes use of *Chebfun* [73] a package written in MATLAB [50] for constructing and manipulating Chebyshev polynomial interpolants to illustrate the details of the construction process.

Example 2.1. *In this example the* 128^{th} *degree Chebyshev interpolants for three different functions with varying amounts of differentiability in the interval* $[-1, 1]$ *are computed. The three functions are* $|\sin(.5z)|^3$, $\left(1+4z^2\right)^{-1}$ *and* e^z. *The function* $|\sin(.5z)|^3$ *has* 2 *absolutely continuous derivatives and a* 3^{rd} *derivative of bounded variation. Theorem 2.6 says that the Chebyshev coefficients* $\{a_k\}_{k=0}^m$ *of the polynomial interpolant of* $|\sin(.5z)|^3$ *should decay at a rate of* $O(k^{-3})$. *The function* $\left(1+4z^2\right)^{-1}$ *is analytic in the Bernstein ellipse* $\mathcal{E}_\rho$ *with* $\rho = 1+\sqrt{5}$. *Theorem 2.8 says that the Chebyshev coefficients* $\{a_k\}_{k=0}^m$ *of the polynomial interpolant of* $\left(1+4z^2\right)^{-1}$ *should decay at a rate of* $O(\rho^{-k})$. *The last function* e^z *is entire and therefore analytic in every Bernstein ellipse* $\mathcal{E}_\rho$ *with* $\rho > 1$. *Theorem 2.8 says that the Chebyshev coefficients* $\{a_k\}_{k=0}^m$ *of the polynomial interpolant of* $\left(1+4z^2\right)^{-1}$ *should decay at a rate of* $O(\rho^{-k})$ *and since this holds for every* $\rho > 1$ *the convergence is actually super geometric.*

Figure 2.1 plots the $\log|a_k|$ *for each of these functions. The coefficients of* $|\sin(.5z)|^3$ *give sub-linear decay on a semi-logarthmic plot. The coefficients of* $\left(1+4z^2\right)^{-1}$ *give a linear decay on a semi-logarthmic plot. The coefficients of* e^z *give super-linear decay on a semi-logarthmic plot.*

2.3 Computing with Chebyshev Polynomials

The results in the previous section suggest that constructing polynomial approximations for smooth functions via interpolation through Chebyshev points of the 1^{st} kind can be efficient and yield

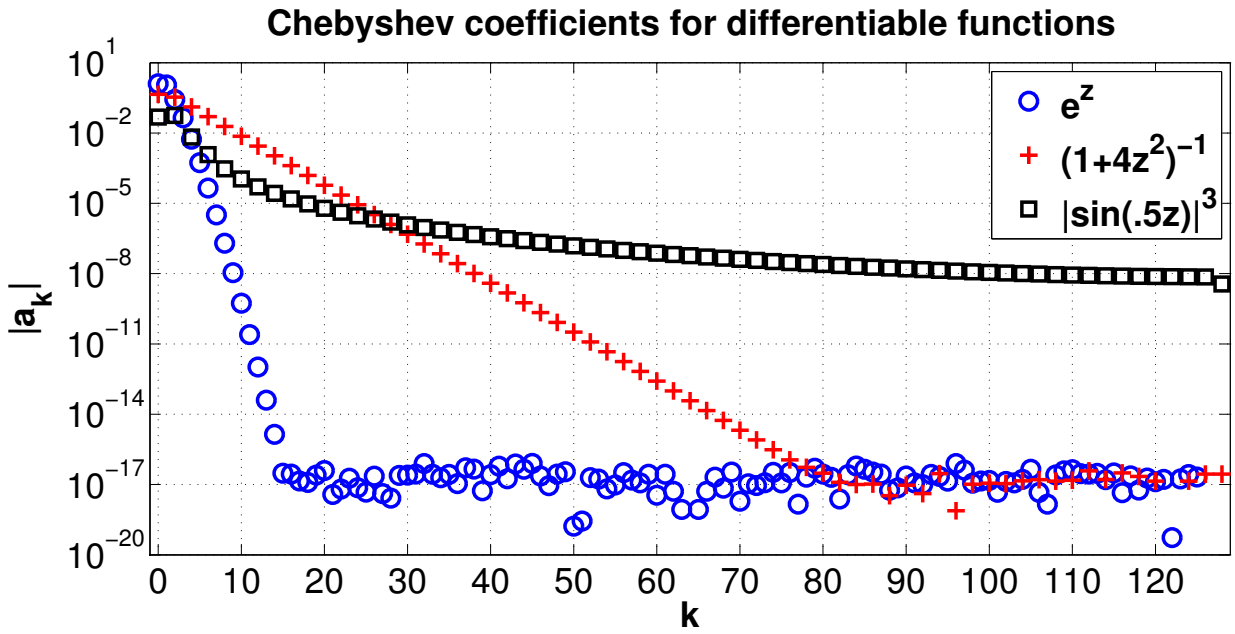

FIGURE 2.1: Convergence of polynomial interpolants constructed by interpolating through 129 Chebyshev points of the 1^{st} kind in the interval $[-1, 1]$. The three choices of functions $|\sin(.5z)|^3$, $\left(1+4z^2\right)^{-1}$ and e^z illustrate sub-linear, linear and super-linear convergence respectively on a semi-logarithmic plot.

accurate approximations. Once the desired polynomial has been constructed there is still the matter of making use of them in practice. For most iterative methods, including Krylov methods, the only user input required is the action of the matrix on a vector, $w = Av$. Since the matrix being used is not A but $p(A)$ the action of $w = p(A)v$ must be computed. If p is written in a Chebyshev basis then the natural algorithm is the Clenshaw algorithm for 3-term recurrences [20].

2.3.1 The Clenshaw Algorithm

The Clenshaw algorithm is a natural extension of Horner's method for evaluating linear combinations of basis functions that are defined by a 3-term recurrence.

Definition 2.9. For some integer $m \geq 0$ let $\{T_i(z)\}_{i=0}^m$ be the Chebyshev polynomials from Definition 2.2. If $p(z) = \sum_{i=0}^m c_i T_i(z)$ for some collection of coefficients $\{c_i\}_{i=0}^m$ taking values in $\mathbb{C}$ then the *Clenshaw algorithm* to compute $p(z)$ is defined by Algorithm 2.

Algorithm 2 Scalar Clenshaw Algorithm for Chebyshev Polynomials

$b_0 = c_m$
$b_1 = c_{m-1} + 2zb_0$
for $k = 2$ to $m - 1$ **do**
  $b_k = c_{m-k} + 2zb_{k-1} - b_{k-2}$
end for
$p(z) = b_m = c_0 + zb_{m-1} - b_{m-2}$

A simple modification to Algorithm 2 computes $p(A)v$ recursively using only the action of Av.

Definition 2.10. For some integers $m \geq 0$ and $n > 0$ let $A \in \mathbb{C}^{n\times n}$, $v \in \mathbb{C}^n$ and $\{T_i(z)\}_{i=0}^m$ be the Chebyshev polynomials from Definition 2.2. If $p(z) = \sum_{i=0}^m c_i T_i(z)$ for some collection of coefficients $\{c_i\}_{i=0}^m$ taking values in $\mathbb{C}$ then the *vector Clenshaw algorithm* to compute $p(A)v$ is defined by Algorithm 3.

Algorithm 3 Vector Clenshaw Algorithm for Chebyshev Polynomials

$b_0 = c_m v$
$b_1 = c_{m-1} v + 2Ab_0$
for $k = 2$ to $m - 1$ **do**
  $b_k = c_{m-k} v + 2Ab_{k-1} - b_{k-2}$
end for
$p(A)v = b_m = c_0 v + Ab_{m-1} - b_{m-2}$

It is important to note that Algorithm 3 only requires matrix-vector multiplication and scaled vector addition. If both of these operations are parallelizable then Algorithm 3 will also be parallelizable.

There have been concerns that recurrences like Algorithm 3 can be unstable when the polynomial degree is high [48]. Indeed the stability analysis for the scalar case [65, 39] shows that Algorithm 2 can have a forward error growing exponentially in the degree. These results can be extended to a norm-wise error analysis of Algorithm 3. As with many useful algorithms these results tend to be overly pessimistic. The experiments in Chapter 5 suggest that even for difficult examples the output of Algorithm 3 can be accurate.

Chapter 3

Solving Linear Systems

Modern methods for solving large sparse linear systems typically rely on iteratively constructing polynomials that approximate z^{-1} [37, 52, 61, 74]. Some of the most popular methods like Conjugate Gradient (CG) [38] and Generalized Minimum Residual (GMRES) [63] as well as their restarted variants [53, 57], make use of Krylov spaces to adaptively compute these polynomials. Convergence of these methods depends on the conditioning of the system. For positive definite matrices this corresponds to the length of the interval $[\lambda_{\min}, \lambda_{\max}]$ and how the rest of the spectrum is located within [27].

The distribution of the eigenvalues can usually be improved by using preconditioners [9]. A wide variety of preconditioners have been developed over the years, some of which can be applied efficiently in parallel. One such example is the class of polynomial preconditioners [6, 7, 8, 17, 28, 29, 32, 44, 48, 60, 80]. These methods attempt to construct low degree polynomial approximations to z^{-1}. The degree is kept low so that the spectral transformation can be applied efficiently. For ill-conditioned systems keeping the degree low prevents the construction of good spectral transformations giving little to no convergence acceleration.

In this chapter the techniques from Chapter 2 will be used to construct polynomial approximations to z^{-1} explicitly without the need for Krylov spaces. The novelty in this approach is that the degree of the polynomial approximation will not be limited directly. Instead the focus will be on how accurately the polynomial spectral transformation approximates z^{-1}. This will significantly improve the quality of the transforming polynomial and in some cases give approximations that are accurate enough to be viewed as direct solvers.

3.1 Constructing Polynomial Approximate Inverses

When constructing polynomial approximate inverses for positive definite matrices two different cases that depend on the amount of available spectral information must be considered. The first case is that $\sigma(A) \subset [\alpha, \beta]$ for $\alpha > 0$. This means that z^{-1} is analytic on $[\alpha, \beta]$ and can be approximated directly using polynomials. The second case is that $\sigma(A) \subset [\alpha, \beta]$ for $\alpha = 0$. Here z^{-1} is not defined and something must be done to regularize near the pole if a polynomial approximation is to be used.

3.1.1 Approximating z^{-1} away from the origin

In the first case where $f(z) = z^{-1}$ is being approximated over an interval $[\alpha, \beta]$ where f is analytic, Theorem 2.8 says that f will be easy to approximate. The following example illustrates the accuracy and degree of the polynomial approximations for various choices of $[\alpha, \beta]$.

Example 3.1. *In this example the degree and accuracy of polynomial approximations of z^{-1} for various choices of the interval $[\alpha, \beta]$ are compared. The polynomials were constructed using Algorithm 1 with a relative tolerance of $\epsilon = 10^{-15}$. The quality of the approximation $p(z) \approx z^{-1}$ is*

measured by taking the supremum norm of $1 - q$ *where* $q(z) = p(z)z$*. Table 3.1 shows this error as well as the degree of the approximating polynomial* p*.*

$[\alpha, \beta]$	degree of p	$\|1 - q\|_\infty$
$[10^{-1}, 1]$	56	0.00×10^{-00}
$[10^{-2}, 1]$	179	1.91×10^{-15}
$[10^{-3}, 1]$	504	3.35×10^{-13}
$[10^{-4}, 1]$	1626	9.93×10^{-13}
$[10^{-5}, 1]$	4011	2.00×10^{-09}
$[10^{-6}, 1]$	13251	9.51×10^{-10}
$[10^{-7}, 1]$	31363	4.51×10^{-06}

TABLE 3.1: Comparison of polynomial approximate inverses for various interval lengths.

Example 3.1 illustrates the accuracy of polynomials constructed by interpolation at Chebyshev points for various intervals. The ratio β/α can be thought of as an upper bound for the condition number of A. The results in Table 3.1 suggests that the more ill-conditioned the matrix is the more difficult it is to construct good polynomial spectral transformations.

3.1.2 Approximating z^{-1} near the origin

The more difficult case is when the best guess for the interval $[\alpha, \beta]$ includes the origin. In this case z^{-1} is no longer analytic (or even defined at $z = 0$) and something must be done to regularize near the origin. Since the system is positive definite polynomial spectral transformations can be constructed that are good approximations to z^{-1} in some region away from 0 but not so good (necessarily) near 0.

This can be accomplished by replacing $f(z) = z^{-1}$ with $f(z) = h(z)z^{-1}$ where $h(z)$ is some smooth function that removes the pole at the origin and makes f easy to approximate. The following example illustrates that such a function is possible to construct.

Example 3.2. *Consider the following one parameter family of functions,*

$$S(z,\tau) = \frac{1 - e^{-\tau z}}{z}.$$

It is easy to check that the singularity at the origin is removable and that $S(z,\tau)$ *is entire for every* $\tau \in \mathbb{C}$. *Figure 3.1 compares the shape of* $S(z,\tau)$ *versus* z^{-1} *for various values of* τ *on the interval* $[0,1]$.

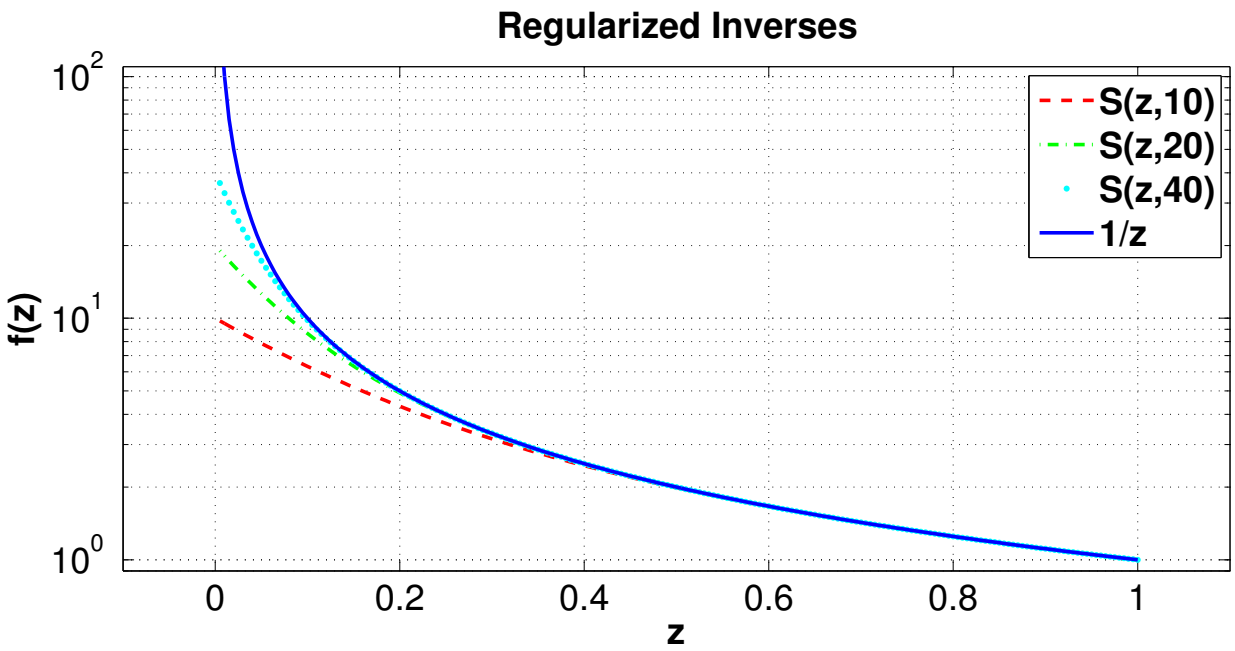

FIGURE 3.1: Comparison of $S(z,\tau)$ and z^{-1} for various values of τ.

Example 3.2 illustrates how z^{-1} might be regularized when the only available lower bound on the spectrum is 0. The point of the regularization is to create a function that is close to z^{-1} but still easy to approximate. The difficulty is that the approximation might not be very good if τ is poorly chosen. The next example illustrates this.

Example 3.3. *Consider the case where the spectrum of a positive definite matrix* A *is contained in the interval* $[\lambda_{\min}, \lambda_{\max}] = [10^{-3}, 1]$ *but the best guess for* $[\alpha, \beta]$ *is the interval* $[0,1]$. *Polynomial*

spectral transformations p are constructed by interpolating $S(z,\tau)$ on the interval $[0,1]$. Table 3.2 compares the norm of the error $\|1-q\|_\infty$, $q(z)=p(z)z$ on the interval $[10^{-3},1]$ for various values of τ.

τ	degree of p	$\|1-q\|_\infty$
10^1	22	9.90×10^{-01}
10^2	59	9.05×10^{-01}
10^3	177	3.68×10^{-01}
10^4	509	4.54×10^{-05}
10^5	1620	4.73×10^{-11}
10^6	4771	6.17×10^{-10}
10^7	16257	2.97×10^{-10}

TABLE 3.2: Comparison of polynomial approximations of the function $S(z,\tau)$ for various values of τ on the interval $[10^{-3},1]$.

Example 3.3 illustrates the accuracy of polynomial approximate inverses when using the function $S(z,\tau)$. Table 3.2 suggests that in order to guarantee an accurate approximation τ should be quite large. This means that the degree of the spectral transformation might also be large. If matrix vector multiplication is significantly cheaper than estimating the spectrum the higher degree polynomials may still perform well.

3.2 Solving $Ax = b$

The matrix $p(A)$ can be interpreted as either a direct approximation to A^{-1} or as a preconditioner for use with a general iterative solver. The following results give some links between the quality of the approximation and the accuracy of the approximate solutions.

3.2.1 $p(A)$ as a direct method

The following lemma gives a bound on the absolute error of approximating A^{-1} using polynomial approximate inverses.

Lemma 3.1. *For some positive integer n, let $A \in \mathbb{C}^{n\times n}$ be positive definite with eigenvalues contained in $[\alpha, \beta] \subset \mathbb{R}$, $\alpha > 0$. Let p be a polynomial approximation of z^{-1} on $[\alpha, \beta]$ then,*

$$\|I - p(A)A\|_2 \leq \|1 - q\|_\infty,$$

where $q(z) = p(z)z$.

Proof. Since A is normal, there exists $Q \in \mathbb{C}^{n\times n}$ and $\Lambda \in \mathbb{R}^{n\times n}$ such that $QQ^H = Q^HQ = I$, $\Lambda = \text{diag}\{\lambda_1, \ldots, \lambda_n\}$ and $A = Q\Lambda Q^H$, where the $\{\lambda_i\}_{i=1}^n$ are the eigenvalues of A. Using the Definition 2.1 we have,

$$I - p(A)A = QQ^H - Qp(\Lambda)Q^HQ\Lambda Q^H = Q(I - p(\Lambda)\Lambda)Q^H.$$

Since the matrix 2-norm is invariant under unitary transformations we have,

$$\|I - p(A)A\|_2 = \|I - p(\Lambda)\Lambda\|_2,$$

and since $p(\Lambda)\Lambda$ is diagonal we have,

$$\|I - p(A)A\|_2 = \max_{1\leq i\leq n} |1 - p(\lambda_i)\lambda_i| \leq \|1 - q\|_\infty.$$

□

Lemma 3.1 says that the error between $p(A)$ and A^{-1} only depends on how well $p(z)$ approximates z^{-1} on $[\alpha, \beta]$ when A is positive definite.

Given a vector b an approximate solution to $Ax = b$ can be computed by $x \approx \tilde{x} = p(A)b$. If x is known exactly a relative error $\|x - \tilde{x}\|_2 / \|x\|_2$ could be computed. The following corollary says that an *a priori* upper bound on this error can be computed if the accuracy of the polynomial approximation is known.

Corollary 3.2. *For some positive integer n, let $A \in \mathbb{C}^{n \times n}$ be positive definite with eigenvalues contained in $[\alpha, \beta] \subset \mathbb{R}$, $\alpha > 0$. For some $0 \neq b \in \mathbb{C}^n$, let $x \in \mathbb{C}^n$ be the solution to $Ax = b$. If $p(z)$ is an approximation to z^{-1} on $[\alpha, \beta]$ and $q(z) = p(z)z$ then,*

$$\frac{\|x - \tilde{x}\|_2}{\|x\|_2} \leq \|1 - q\|_\infty,$$

where $\tilde{x} = p(A)b$.

Proof. By definition,

$$x - \tilde{x} = A^{-1}b - p(A)b = A^{-1}Ax - p(A)Ax = (I - p(A)A)x.$$

Taking norms and applying Lemma 3.1 gives,

$$\|x - \tilde{x}\|_2 \leq \|I - p(A)A\|_2 \|x\|_2 \leq \|1 - q\|_\infty \|x\|_2.$$

□

Corollary 3.2 says that the approximate solution $\tilde{x}$ computed using $p(A)$ is accurate if $p(z)$ is an accurate approximation of z^{-1}.

3.2.2 $p(A)$ as a preconditioner

An alternative interpretation of $p(A)$ is as a preconditioner for some iterative method. The original system $Ax = b$ is replaced by the transformed system, $\hat{A}x = \hat{b}$, $\hat{A} = p(A)A$ and $\hat{b} = p(A)b$, with the hope that the new system is easier to solve. For methods like CG and GMRES, easier to solve means that the eigenvalues of $\hat{A}$ are tightly clustered.

Definition 3.3. For some positive integer n, let $A \in \mathbb{C}^{n\times n}$ be invertible. The 2-*norm condition number* of A is defined as,

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2.$$

For positive definite matrices $\kappa_2(A) = \lambda_{\max}/\lambda_{\min}$ where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of A respectively. For CG a condition number of $\kappa_2(A) = 1$ gives convergence in one step [68]. Even if $\kappa_2(A) \approx 1$ the convergence will still be rapid. When using polynomial preconditioners an upper bound on the condition number of the transformed system can be computed if the accuracy of the approximating polynomial is known.

Corollary 3.4. *For some positive integer n, let $A \in \mathbb{C}^{n\times n}$ be positive definite with eigenvalues contained in $[\alpha, \beta] \subset \mathbb{R}$, $a > 0$. Let $p(z)$ be an approximation to z^{-1} on $[\alpha, \beta]$ and let $q(z) = zp(z)$. If $\{\hat{\lambda}_i\}_{i=1}^n$ are the eigenvalues of $p(A)A$ then,*

$$1 - \epsilon \leq \hat{\lambda}_i \leq 1 + \epsilon, \quad i = 1, \ldots, n,$$

where $\epsilon = \|1 - q\|_\infty$.

Proof. Since A is positive definite, $p(A)A$ and $I - p(A)A$ are normal. If $\hat{\lambda}_i$ is an eigenvalue of $p(A)A$ then $1 - \hat{\lambda}_i$ is an eigenvalue of $I - p(A)A$. Since the spectral radius and 2-norm are equivalent for normal matrices the following holds,

$$|1 - \hat{\lambda}_i| \leq \|I - p(A)A\|_2 \leq \|1 - q\|_\infty, \quad i = 1, \ldots, n.$$

□

Corollary 3.4 says that the eigenvalues of $p(A)A$ will be tightly clustered if $p(z)$ is a good approximation to z^{-1}.

Chapter 4

Accelerating Eigenvalue Computations

Most methods for computing eigenvalues and eigenvectors of large sparse matrices involve iteratively constructing polynomial spectral transformations. Some notable examples of this are the Lanczos [47] and Arnoldi methods [5]. These methods construct polynomials implicitly by computing Krylov subspaces in a stable way. The degree of the transforming polynomials in these methods is limited by the number of vectors that can be stored. To get around this restarted variants are used to construct polynomials with higher degrees by taking the product of polynomials with low degree [66]. Restarts often work well because they construct polynomials using the most current spectral information. Unfortunately, since the degree of each restart polynomial is fixed there are still many cases where these methods fail to converge in a reasonable amount of time. See [11] for a detailed analysis of the restarted Arnoldi method.

The goal of this chapter is to show that good polynomial spectral transformations can be constructed

explicitly using very little spectral information and most importantly without the need to construct Krylov spaces. Recently the authors in [31] explicitly construct polynomials for the same purpose using techniques from digital signal processing. The novelty in the method presented here is the way in which the polynomials are constructed.

4.1 Polynomial Spectral Transformations for Eigenvalue Computations

A polynomial spectral transformation that is used for accelerating eigenvalue computations is one that accentuates the desired spectrum while keeping the degree low. These two ideas are often in conflict with one another. For simplicity the spectrum of A is assumed to be contained in the interval $[0, 1]$. The following two functions are considered as reasonable candidates for transforming the spectrum. The first candidate is the well known Runge function [59],

$$R(z, \sigma, \tau) = \frac{1}{1 + \tau(z - \sigma)^2}. \tag{4.1}$$

The second candidate is the normal distribution or bell curve [24, 35],

$$N(z, \sigma, \tau) = e^{-\tau(z-\sigma)^2}. \tag{4.2}$$

The parameter $\sigma \in [0, 1]$ is a shift near the desired eigenvalues and $\tau > 0$ is a shape parameter that controls the steepness of the function near the shift. These two functions were chosen because of their similar shape which makes them well suited for accentuating interior eigenvalues. They were also chosen because of their smoothness properties. The Runge function has poles at $\sigma \pm i/\sqrt{\tau}$

which makes it analytic inside the Bernstein ellipse that passes through the poles. The bell curve is also analytic, in fact it is entire making it especially easy to approximate.

When computing the polynomial approximations for these two functions a relative tolerance $\epsilon > 0$ will have to be chosen such that,

$$||p - f||_\infty \leq \epsilon ||f||_\infty.$$

For double precision, floating point calculations the best tolerance is $\epsilon \approx 10^{-16}$. However if the tolerance is increased the degree of the approximating polynomial will likely be lower leading to faster computations.

Example 4.1. *In this example the polynomial approximations of* $R(z, \sigma, \tau)$ *and* $N(z, \sigma, \tau)$ *for* $\sigma = 0.05$ *and* $\tau = 1000$ *are compared. Figure 4.1 shows a plot of* $p(z)$ *for* z *in* $[0, 1]$*. Both polynomials were constructed using the relative tolerance* $\epsilon = 10^{-16}$*. The polynomial approximation of* $R(z, \sigma, \tau)$ *and* $N(z, \sigma, \tau)$ *have degrees* 262 *and* 126 *respectively. Notice that the bell curve does a better job of separating values near* σ*. This is due to the exponential decay. It also has a lower degree approximation which is due to the fact that* $N(z, \sigma, \tau)$ *is entire. The Runge function could be made steeper by increasing* τ *but it would require a much higher degree polynomial for a good approximation.*

Example 4.2. *In this example the polynomial approximations of* $R(z, \sigma, \tau)$ *and* $N(z, \sigma, \tau)$ *for* $\sigma = .5$ *and* $\tau = 1000$ *are compared. Figure 4.2 shows a plot* $p(z)$ *for* z *in* $[0, 1]$*. Both polynomials were constructed using the relative tolerance* $\epsilon = 10^{-16}$*. The polynomial approximation of* $R(z, \sigma, \tau)$ *and* $N(z, \sigma, \tau)$ *have degrees* 572 *and* 192 *respectively. The key difference between this example and the previous one is the price paid for being in the center of the interval. Since the Chebyshev extreme points cluster at the ends of the interval there is less sampling in the interior leading to higher degree approximations for shifts near the center.*

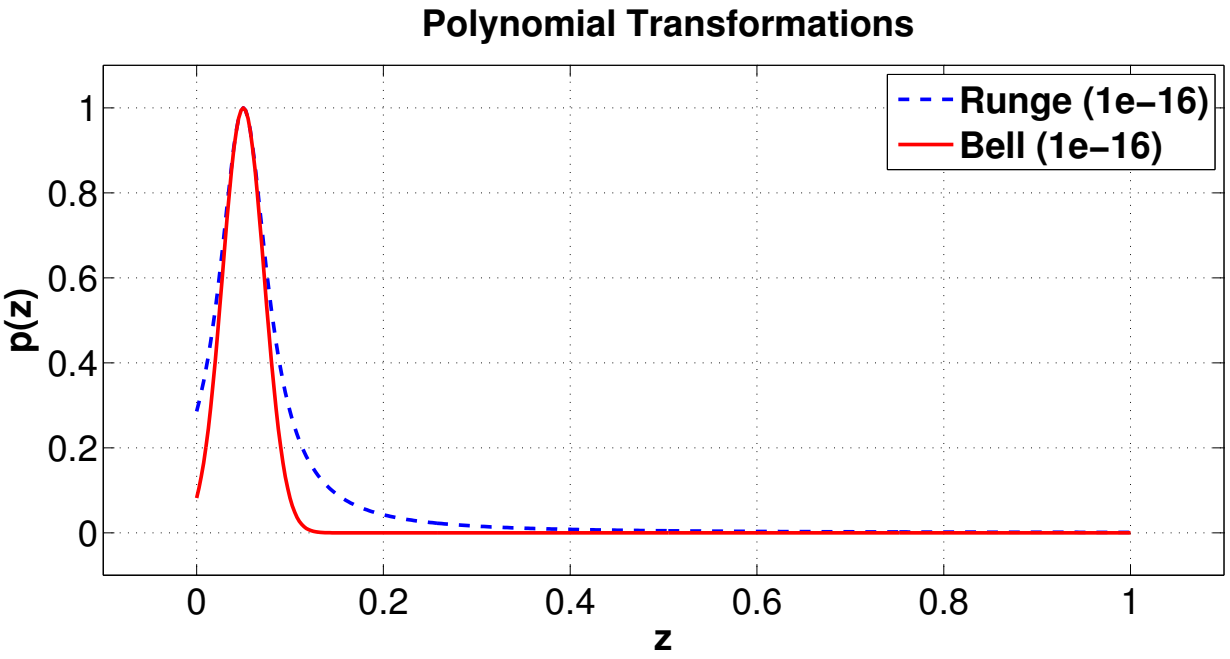

FIGURE 4.1: For $\epsilon = 10^{-16}$ the degrees of the approximations to $R(z, \sigma, \tau)$ and $N(z, \sigma, \tau)$ are 262 and 126 respectively.

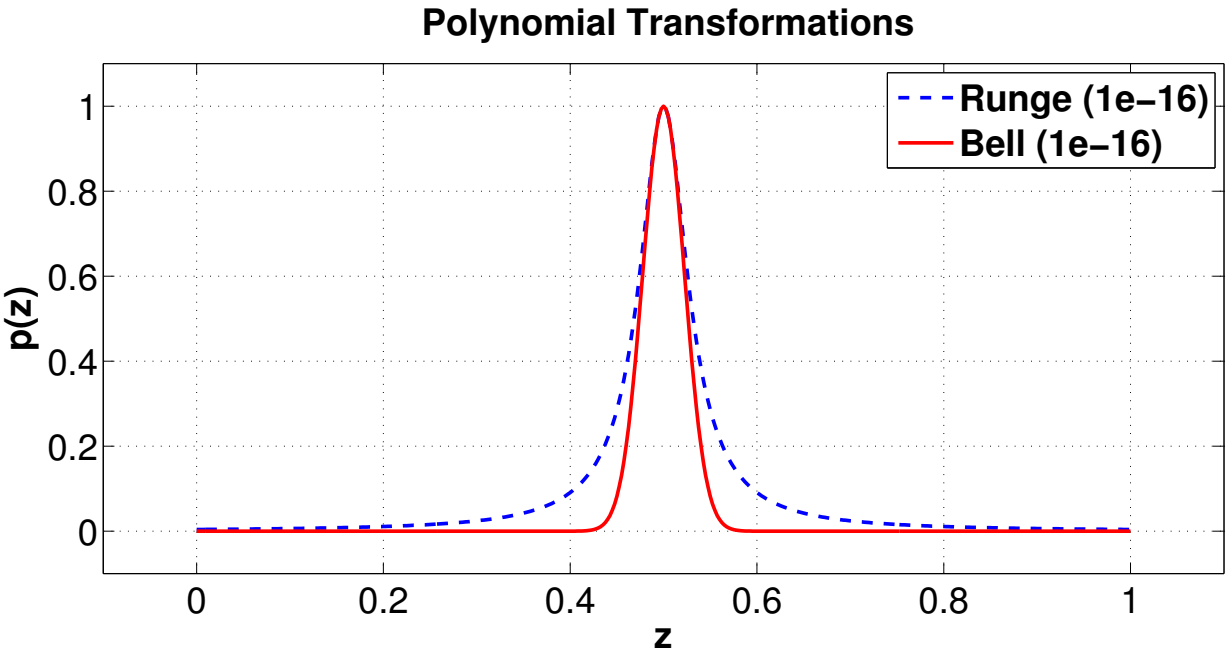

FIGURE 4.2: For $\epsilon = 10^{-16}$ the degrees of the approximations to $R(z, \sigma, \tau)$ and $N(z, \sigma, \tau)$ are 572 and 192 respectively.

Examples 4.1 and 4.2 were meant to illustrate the use of Chebyshev interpolation to construct good polynomial spectral transformations. This is by no means the only way to do so. The key to

polynomial spectral transformations is their incredible flexibility. It should also be noted that this choice of ϵ is based on double precision arithmetic. There is no implication that these are the best choices. Taking $\epsilon = 10^{-3}$ may work equally well with a dramatic reduction in polynomial degree and overall runtime. The next example illustrates this.

Example 4.3. *In this example the polynomial approximations of* $N(z, \sigma, \tau)$ *for* $\sigma = 0.05$, $\tau = 1000$ *and two different tolerances* $\epsilon = 10^{-2}$ *and* $\epsilon = 10^{-16}$ *are compared. Figure 4.3 shows a plot of* $p(z)$ *for* z *in* $[0, 1]$. *The polynomial approximations have degrees* 22 *and* 126 *respectively. The two different choices give qualitatively similar polynomials. The advantage of the higher tolerance is a dramatic reduction in the degree.*

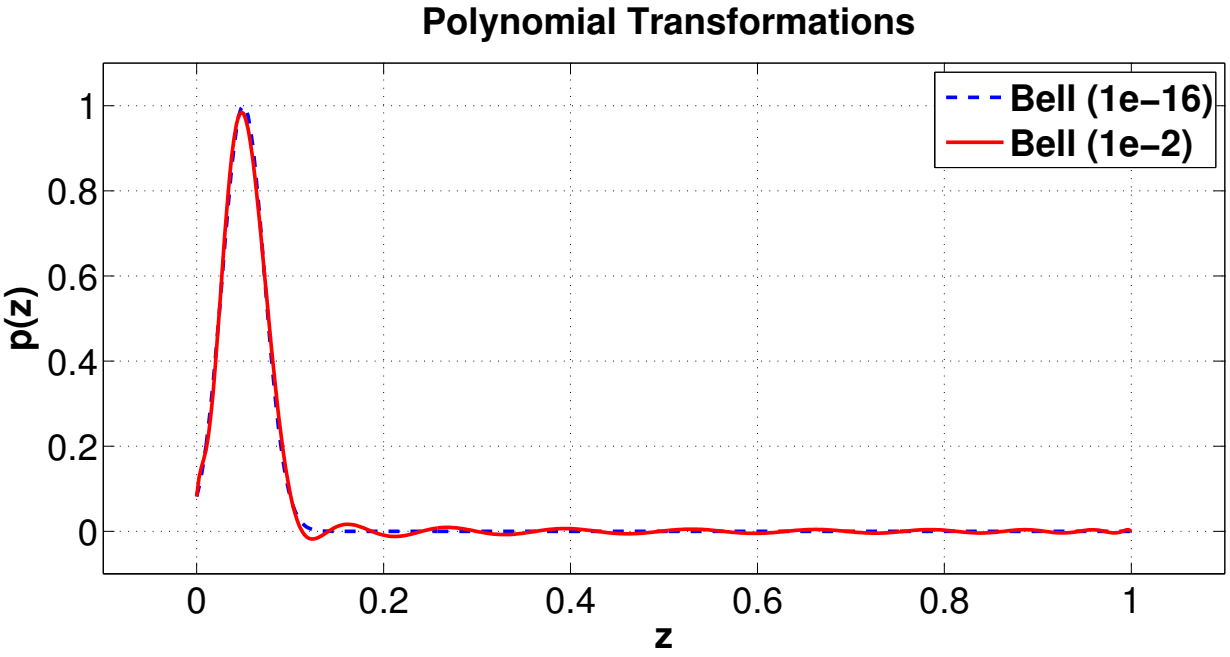

FIGURE 4.3: For $\epsilon = 10^{-16}$ the degree of the approximation to $N(z, \sigma, \tau)$ is 126 and for $\epsilon = 10^{-2}$ the degree of the approximation to $N(z, \sigma, \tau)$ is 22.

4.2 Eigenvalues and Eigenvectors of $p(A)$

With any method for computing eigenvalues and eigenvectors there are two important questions that arise when using polynomial spectral transformations: *How accurately can invariant subspaces of A be computed using $p(A)$?* and *How are the eigenvalues of A recovered using the computed invariant subspaces?*

4.2.1 Invariant Subspaces of $p(A)$

The goal of spectral transformations is to bring the desired eigenvalues of A to the exterior of the spectrum of $p(A)$ so that the corresponding invariant subspace can be computed with relative ease. Under some reasonable assumptions this invariant subspace will be the same for both A and $p(A)$. Even though the invariant subspace is the same, the condition number of the subspace may be different for A and $p(A)$.

Definition 4.1. Let $B \in \mathbb{C}^{n\times n}$ be a normal matrix and let $\Lambda_1 = \{\lambda_1, \ldots, \lambda_k\}$ and $\Lambda_2 = \{\lambda_{k+1}, \ldots, \lambda_n\}$ be a disjoint partition of the spectrum of B. Let $\mathcal{U}$ be the k-dimensional invariant subspace of B corresponding to Λ_1. The *spectral gap* of $\mathcal{U}$ with respect to B is defined as,

$$\text{gap}_B(\mathcal{U}) = \min\{|\lambda_i - \lambda_j| \mid \lambda_i \in \Lambda_1 \text{ and } \lambda_j \in \Lambda_2\}.$$

Definition 4.2. Let $\mathcal{U}$ and $\mathcal{V}$ be two k-dimensional subspaces of $\mathbb{C}^n$. The *distance* between $\mathcal{U}$ and $\mathcal{V}$ is defined as,

$$d(\mathcal{U}, \mathcal{V}) = \max_{\substack{\|u\|_2=1 \\ u\in\mathcal{U}}} \min_{v\in\mathcal{V}} \|u - v\|_2.$$

Definition 4.3. For some positive integers n and k, let $B \in \mathbb{C}^{n\times n}$ and let $\mathcal{U}$ be a k-dimensional subspace of $\mathbb{C}^n$. Let $Q \in \mathbb{C}^{n\times k}$ be a matrix with orthonormal columns that span $\mathcal{U}$. The *residual*

$R \in \mathbb{C}^{n \times k}$ of $\mathcal{U}$ with respect to B is defined as,

$$R = BQ - QH,$$

where $H = Q^* BQ$.

Eigenvalue methods for large sparse matrices attempt to compute a subspace $\hat{\mathcal{U}}$ that is a good approximation to a true invariant subspace $\mathcal{U}$ of A. If $\text{gap}_A(\mathcal{U})$ is large then it will be relatively easy to compute a $\hat{\mathcal{U}}$ that is close to $\mathcal{U}$. This fact is illustrated by the following theorem.

Theorem 4.4 ([76, p. 105]). *Let $B \in \mathbb{C}^{n \times n}$ be a normal matrix and let $\Lambda_1 = \{\lambda_1, \ldots, \lambda_k\}$ and $\Lambda_2 = \{\lambda_{k+1}, \ldots, \lambda_n\}$ be a disjoint partition of the spectrum of B. Let $\mathcal{U}$ be the k-dimensional invariant subspace of B corresponding to Λ_1. If $\hat{B} \in \mathbb{C}^{n \times n}$ is a matrix such that,*

$$\|B - \hat{B}\|_F < 2\,\text{gap}_B(\mathcal{U}),$$

then there exists a k-dimensional invariant subspace $\hat{\mathcal{U}}$ of $\hat{B}$ such that,

$$d(\mathcal{U}, \hat{\mathcal{U}}) < 2\frac{\|B - \hat{B}\|_F}{\text{gap}_B(\mathcal{U})}.$$

The quantity $\|B - \hat{B}\|_F$ from Theorem 4.4 is equal to the norm of the residual from Definition 4.3,

$$\hat{R} = B\hat{Q} - \hat{Q}H,$$

where the range of $\hat{Q}$ equals $\hat{\mathcal{U}}$, $\hat{Q}^*\hat{Q} = I$ and,

$$\hat{B} = B + \hat{R}\hat{Q}^*.$$

Thus a tiny residual and a large spectral gap guarantee a good approximation. Even if the $\text{gap}_A(\mathcal{U})$ is small, a good choice of spectral transformation $p(z)$ can give a much larger $\text{gap}_{p(A)}(\mathcal{U})$ making the approximation of $\mathcal{U}$ using $p(A)$ much less sensitive to error. This means that not only should $p(z)$ move the desired eigenvalues to the exterior, it should do it with as much separation from the unwanted spectrum as possible.

4.2.2 Effects of Rounding Errors

It should be noted that all of these results are true in exact arithmetic. In practice A will not be known exactly and the computations will be carried out with a matrix $\tilde{A} = A + E$ where hopefully $\|E\|_F$ is small. If the invariant subspaces are computed with $p(\tilde{A})$ what can be said about the invariant subspaces of A? Unfortunately, in the worst case separated eigenvalues of A can merge under the perturbation E and produce a higher dimensional invariant subspace. When this happens there is no polynomial that can separate the eigenvalues. Such a perturbation will almost never happen. A more likely scenario is that several eigenvalues that are already close will become closer making the corresponding invariant subspaces even more difficult to compute. This is a danger that every eigenvalue method faces. One solution is to always go after a larger invariant subspace that contains the desired poorly separated ones and if there really is a danger of repeated eigenvalues a block method should be used instead (See [23] for example).

4.2.3 Recovering the Eigenvalues of A

For all but the worst cases it should be expected that a good choice of $p(z)$ will greatly improve the quality of the approximate invariant subspaces. What is not so clear is how the eigenvalues of $p(A)$ can be used to recover the eigenvalues of A. With shift and invert strategies the recovery of the

eigenvalues of A from the eigenvalues of $(A-\sigma I)^{-1}$ is a simple matter of inverting $(z-\sigma)^{-1}$. When using polynomial spectral transformations it may be very difficult or even impossible to invert $p(z)$ to recover the eigenvalues of A. An alternative to this approach is to use the approximate invariant subspace $\hat{\mathcal{U}}$ of $p(A)$ to compute the Rayleigh quotients of A. For normal matrices, a good approximate invariant subspace guarantees that the Rayleigh quotients are good approximations to the desired eigenvalues. The next theorem makes this precise.

Theorem 4.5 ([56, p. 222])**.** *Let $B \in \mathbb{C}^{n\times n}$ be normal and let (λ_1, v_1), $\|v_1\|_2 = 1$ be an eigenpair of B. Let q_1, $\|q_1\|_2 = 1$ be an approximation to v_1 such that,*

$$\|q_1 - v_1\|_2 = \epsilon.$$

If $\rho_1 = q_1^* B q_1$, then

$$|\lambda_1 - \rho_1| = O(\epsilon^2).$$

Theorem 4.5 and Definition 4.2 tell us that an accurate invariant subspace guarantees a tiny absolute error in the Rayleigh quotients. This means there is no need to invert $p(z)$ as the Rayleigh quotients will likely be much more accurate.

Chapter 5

Numerical Experiments

To illustrate the effectiveness of combining polynomial spectral transformations with parallel computer architectures these methods are implemented on a Graphics Processing Unit (GPU). This is done using the programming language CUDA [54, 55], which is a collection of C libraries for interfacing with NVIDIA brand GPUs. Unless otherwise stated all of the experiments in this chapter will be performed using IEEE double precision arithmetic on the same workstation, which has an AMD FX-4100 Quad-Core Processor with 12GB of CPU memory and an NVIDIA GeForce GTX 650 GPU with 384 compute cores and 1024MB of GPU memory.

5.1 Linear System Solving

The experiments in the next two sections will use polynomial spectral transformations to solve large sparse positive definite linear systems, $Ax = b$. The first set will consider the matrix $p(A)$ as a direct approximation to A^{-1}. The second set will consider $p(A)$ as a preconditioner for the Conjugate Gradient Method (CG).

5.1.1 Direct Method

In the first experiment of this section a polynomial spectral transformation is used to solve the 3D Poisson equation using finite differences.

Definition 5.1. Let $n > 0$, the *discrete Laplace operator* $\Delta_n \in \mathbb{R}^{n \times n}$ is defined as,

$$\Delta_n = \begin{bmatrix} 2 & -1 & & \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 2 \end{bmatrix}.$$

In Definition 5.1 the usual $-h^{-2}$ term is dropped since it scales the eigenvalues without changing their relative distribution.

Example 5.1. *In this example a finite difference approximation is used to solve the Poisson problem on a cube,*

$$-(u_{xx} + u_{yy} + u_{zz}) = f, \quad \Omega = [0,1]^3, \quad u\,|_{\partial\Omega} = 0.$$

The discretized Laplacian has the form,

$$\hat{\Delta}_{n_x n_y n_z} = I_{n_y n_z} \otimes \Delta_{n_x} + I_{n_z} \otimes (\Delta_{n_y} \otimes I_{n_x}) + \Delta_{n_z} \otimes I_{n_x n_y},$$

where Δ is the same as in Definition 5.1 and n_x, n_y and n_z are the number of grid points in the x, y and z directions respectively.

Since the eigenvalues of $\hat{\Delta}_{n_x n_y n_z}$ are known explicitly the polynomial spectral transformation can be constructed by approximating $f(z) = z^{-1}$ for $z \in [\lambda_{\min}, \lambda_{\max}]$. Figure 5.1 plots $\|1 - q\|_\infty$,

$q(z) = p(z)z$ for $z \in [\lambda_{\min}, \lambda_{\max}]$. Table 5.1 gives the degree of the spectral transformation, the relative residual of the approximate solution, the time to compute the polynomial, the time to perform the solve and the total time. The right-hand-side b was taken to be the all ones vector $[1, \ldots, 1]^T$ and the number of grid points was taken to be $n_x = n_y = n_z = 10^2$ giving a total problem dimension of 10^6.

degree	$\|Ax - b\|_2 / \|b\|_2$	pre-proc time	solve time	total time
987	1.44×10^{-11}	0.22 s	10.6 s	10.8 s

TABLE 5.1: Accuracy of polynomial spectral transformation for approximating the inverse of the 3D discrete Laplace operator using precise spectral information.

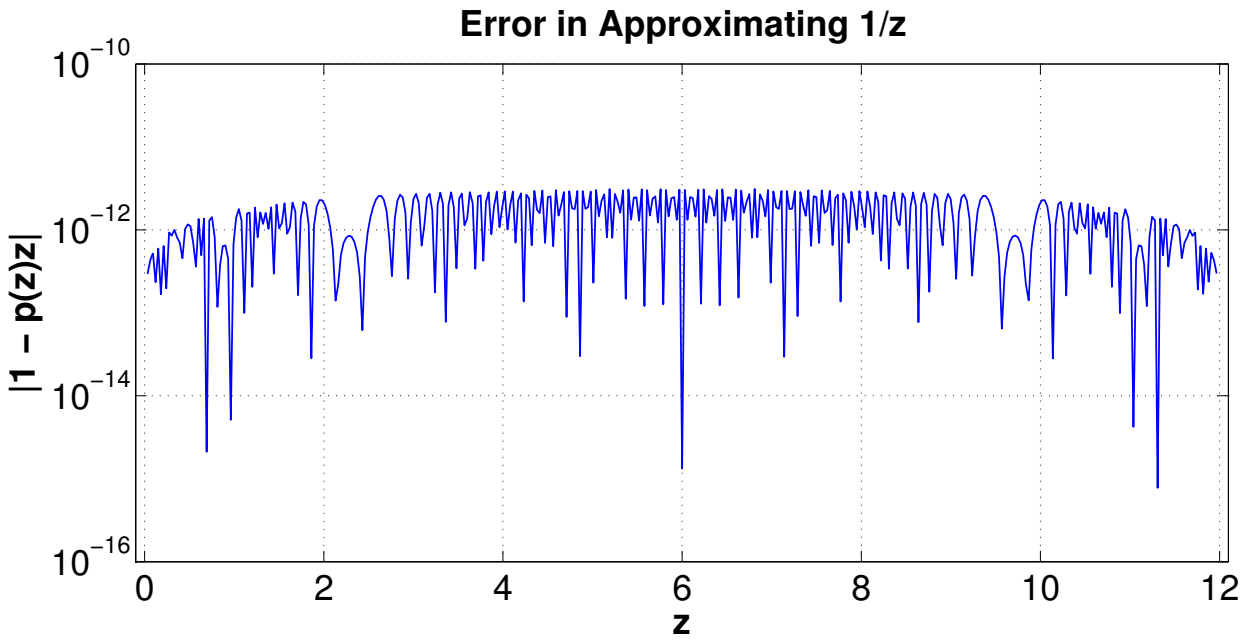

FIGURE 5.1: Error in approximating z^{-1} for the discrete Laplace operator using a 987 degree polynomial.

The results of Example 5.1 suggest that polynomial spectral transformations can be used to solve linear systems when accurate spectral information is known. The next example repeats the experiment in Example 5.1 using less precise spectral information.

Example 5.2. *In this example the experiment in Example 5.1 is repeated using only an upper bound on the spectral radius* $\rho(\hat{\Delta}_{n_x n_y n_z}) < M$ *to construct the polynomial approximate inverse. The polynomial spectral transformation is constructed by approximating the function* $f(z) = \left(1 - e^{-\tau z/M}\right)/z$ *on the interval* $[0, M]$ *with* $\tau = 10^4, 10^5$ *and* 10^6.

Table 5.2 gives the degree of the spectral transformation, the relative residual of the approximate solution, the time to compute the polynomial and M*, the time to perform the solve and the total time. The right-hand-side* b *was taken to be the all ones vector* $[1, \ldots, 1]^T$ *and the number of grid points was taken to be* $n_x = n_y = n_z = 10^2$ *giving a total problem dimension of* 10^6.

τ	degree	$\|Ax - b\|_2 / \|b\|_2$	pre-proc time	solve time	total time
10^4	495	8.20×10^{-02}	3.10 s	5.32 s	8.42 s
10^5	1531	2.67×10^{-10}	3.10 s	16.4 s	19.5 s
10^6	4692	1.01×10^{-09}	3.10 s	50.4 s	53.5 s

TABLE 5.2: Accuracy of polynomial spectral transformations for approximating the inverse of the 3D discrete Laplace operator using inexact spectral information.

Example 5.2 illustrates that a good polynomial approximate inverse can still be constructed using minimal spectral information. The price paid for having less information is the increased possibility of high degree polynomials and lower accuracy.

The next experiment compares polynomial approximate inverses to computing a Cholesky factorization. To the best of this author's knowledge there are no available packages that make use of the GPU to compute Cholesky factorizations. The comparison will be made using CHOLMOD [19] which runs entirely on the CPU.

Example 5.3. *In this example the experiment in Example 5.1 is repeated using only an upper bound on the spectral radius* $\rho(\hat{\Delta}_{n_x n_y n_z}) < M$ *to construct the polynomial approximate inverse. The polynomial spectral transformation is constructed by approximating the function* $f(z) = \left(1 - e^{-\tau z/M}\right)/z$

on the interval $[0, M]$ *with* $\tau = 10^5$. *The polynomial approximate inverse is compared with a Cholesky factorization run on the CPU.*

Table 5.3 and Table 5.4 give the problem dimension, the memory requirements to store the inverse, the relative residual of the approximate solution, the time to compute the polynomial or factorization, the time to perform the solve and the total time for CHOLMOD and polynomial approximate inverses respectively. The right-hand-side b was taken to be the all ones vector $[1, \ldots, 1]^T$.

It is important to note that the special structure of $\hat{\Delta}_{n_x n_y n_z}$ *makes it easy to perform matrix-vector multiplication without ever storing the entries of the matrix explicitly. Since polynomial approximate inverses only require matrix-vector multiplication the solution can be computed with very little memory overhead. The memory requirement in Table 5.3 corresponds to the memory required to store the Cholesky factor. The memory requirement in Table 5.4 corresponds to the memory required to store the coefficients of the polynomial. The N/As in the last row of Table 5.3 correspond to CHOLMOD running out of memory.*

$n_x n_y n_z$	Memory	$\|Ax - b\|_2 / \|b\|_2$	pre-proc time	solve time	total time
30^3	50.0 MB	5.38×10^{-14}	0.96 s	0.02 s	0.98 s
40^3	176 MB	1.02×10^{-13}	5.11 s	0.06 s	5.17 s
50^3	516 MB	1.92×10^{-13}	25.1 s	0.15 s	25.2 s
60^3	1.25 GB	2.94×10^{-13}	89.6 s	0.35 s	90.0 s
70^3	2.54 GB	4.37×10^{-13}	260 s	0.70 s	261 s
80^3	4.44 GB	5.48×10^{-13}	548 s	1.23 s	549 s
90^3	7.98 GB	7.03×10^{-13}	1485 s	50.3 s	1535 s
100^3	N/A	N/A	N/A	N/A	N/A

TABLE 5.3: Direct solution of a 3D Poisson equation using a Cholesky factorization.

In Example 5.3 the Cholesky factorizations computed using CHOLMOD gave better residuals than the polynomial approximate inverses. The increased accuracy required considerably more memory and longer runtimes for a wide range of problem sizes. In some instances the polynomial

$n_x n_y n_z$	Memory	$\|Ax-b\|_2/\|b\|_2$	pre-proc time	solve time	total time
30^3	12.2 KB	4.34×10^{-11}	0.34 s	0.63 s	0.97 s
40^3	12.2 KB	4.68×10^{-11}	0.36 s	1.58 s	1.94 s
50^3	12.2 KB	5.82×10^{-11}	0.54 s	2.53 s	3.07 s
60^3	12.3 KB	7.00×10^{-11}	0.76 s	3.83 s	4.59 s
70^3	12.2 KB	9.21×10^{-11}	1.16 s	5.76 s	6.92 s
80^3	12.2 KB	1.14×10^{-10}	1.58 s	8.55 s	10.1 s
90^3	12.2 KB	1.40×10^{-10}	2.27 s	11.9 s	14.2 s
100^3	12.3 KB	2.65×10^{-10}	2.90 s	16.5 s	19.4 s

TABLE 5.4: Direct solution of a 3D Poisson equation using a polynomial spectral transformation.

computed an approximate solution 100 times faster and for the largest problem size CHOLMOD failed outright.

5.1.2 Polynomial Preconditioning

In this section polynomial spectral transformations are used as preconditioners for the Conjugate Gradient Method (CG). The CG solver is part of the package CULA Sparse developed and distributed by EM Photonics [30, 42]. The first experiment in this section constructs polynomial approximate inverses of different degrees by varying the approximation tolerance.

Example 5.4. *In this example the experiment in Example 5.1 is repeated using a polynomial spectral transformation as a preconditioner to the Conjugate Gradient Method (CG). The polynomial is constructed by approximating the function* $f(z) = z^{-1}$ *on the interval* $[\lambda_{\min}, \lambda_{\max}]$. *To keep the degree low a larger relative tolerance* ϵ *is used when the polynomial* p *is constructed,*

$$\|f - p\|_\infty < \epsilon \|f\|_\infty.$$

Table 5.5 gives the degree of the spectral transformation for various choices of ϵ. *It also records the number of CG iterations until convergence, the time to compute the polynomial, the time to perform*

the solve and the total time. The CG iterations halted when the relative residual $\|Ax - b\|_2 / \|b\|_2$ *was less than* 10^{-13}*. The right-hand-side* b *was taken to be the all ones vector* $[1, \ldots, 1]^T$ *and the number of grid points was taken to be* $n_x = n_y = n_z = 10^2$ *giving a total problem dimension of* 10^6*.*

ϵ	degree	iterations	pre-proc time	solve time	total time
10^{-04}	184	11	.001 s	21.9 s	21.9 s
10^{-05}	258	6	.001 s	16.7 s	16.7 s
10^{-06}	332	4	.02 s	14.4 s	14.4 s
10^{-07}	406	3	.02 s	13.2 s	13.2 s
10^{-08}	486	3	.02 s	15.7 s	15.7 s
10^{-09}	554	2	.02 s	12.0 s	12.0 s
10^{-10}	628	2	.02 s	13.5 s	13.5 s

TABLE 5.5: Solution of a 3D Poisson equation using polynomial approximate inverses constructed using various tolerances ϵ to precondition CG.

Example 5.4 illustrates that even a large approximation tolerance can reduce the total number of CG iterations. The next example repeats the previous experiment using less precise spectral information.

Example 5.5. *In this example the experiment in Example 5.1 is repeated using a polynomial spectral transformation as a preconditioner to the Conjugate Gradient Method (CG). The polynomial is constructed by first computing an upper bound on the spectral radius* $\rho(\hat{\Delta}_{n_x n_y n_z}) < M$ *and then approximating the function* $f(z) = (1 - e^{-\tau z/M})/z$*,* $\tau = 10^5$ *on the interval* $[0, M]$*. To keep the degree low a larger relative tolerance* ϵ *is used when the polynomial* p *is constructed,*

$$\|f - p\|_\infty < \epsilon \|f\|_\infty.$$

Table 5.6 gives the degree of the spectral transformation for various choices of ϵ*. It also records the number of CG iterations until convergence, the time to compute the upper bound* M *and construct the polynomial, the time to perform the solve and the total time. The CG iterations halted when*

the relative residual $\|Ax - b\|_2 / \|b\|_2$ *was less than* 10^{-13}*. The right-hand-side* b *was taken to be the all ones vector* $[1, \ldots, 1]^T$ *and the number of grid points was taken to be* $n_x = n_y = n_z = 10^2$ *giving a total problem dimension of* 10^6*.*

The N/As in the first row of Table 5.6 correspond to CG failing to converge in 500 *iterations.*

ϵ	degree	iterations	pre-proc time	solve time	total time
10^{-04}	481	N/A	N/A	N/A	N/A
10^{-05}	686	17	2.92 s	118 s	121 s
10^{-06}	785	7	2.93 s	59.1 s	62.0 s
10^{-07}	907	5	2.91 s	48.8 s	52.7 s
10^{-08}	1016	4	2.93 s	43.7 s	46.6 s
10^{-09}	1116	3	2.93 s	36.0 s	38.9 s
10^{-10}	1208	3	2.92 s	39.0 s	41.9 s

TABLE 5.6: Solution of a 3D Poisson equation using polynomial approximate inverses constructed using various tolerances ϵ to precondition CG.

Examples 5.4 and 5.5 illustrate that the amount of available spectral information can have a dramatic effect of the quality of the preconditioner. The next experiment compares the use of several different preconditioners to solve the same problem. The preconditioners available in CULA Sparse are a simple Jacobi preconditioner [1], an incomplete LU preconditioner (ILU) [13], a sparse approximate inverse preconditioner (Ainv) [12, 26] and an incomplete Cholesky preconditioner (IChol) [51, 45].

Example 5.6. *In this example the experiment in Example 5.1 is repeated using several different preconditioners available in CULA Sparse. Standard CG is compared to preconditioned CG using a Jacobi preconditioner, an incomplete LU preconditioner (ILU), an approximate inverse precondtioner (Ainv), an incomplete Cholesky preconditioner (IChol) and a polynomial approximate inverse preconditioner (Poly). The polynomial approximate inverse is constructed by first computing an upper bound on the spectral radius* $\rho(\hat{\Delta}_{n_x n_y n_z}) < M$ *and approximating the function* $f(z) = \left(1 - e^{-\tau z / M}\right) / z$ *on the interval* $[0, M]$ *with* $\tau = 10^5$*.*

Table 5.7 gives the type of preconditioner, the number of CG iterations, the time to compute the preconditioner, the time to perform the solve and the total time. The right-hand-side b was taken to be the all ones vector $[1, \ldots, 1]^T$ *and the number of grid points was taken to be* $n_x = n_y = n_z = 10^2$ *giving a total problem dimension of* 10^6*.*

It is important to note that the special structure of $\hat{\Delta}_{n_x n_y n_z}$ *makes it easy to perform matrix-vector multiplication without ever storing the entries of the matrix explicitly. Since polynomial approximate inverses only require matrix-vector multiplication the solution can be computed with very little memory overhead. The memory requirement for the polynomial preconditioner corresponds to the memory required to store the coefficients of the polynomial.*

Prec	Iterations	pre-proc time	solve time	total time
none	330	0 s	3.29 s	3.29 s
Jacobi	330	0 s	3.78 s	3.78 s
ILU	149	0.20 s	5.92 s	6.12 s
Ainv	250	4.16 s	3.41 s	7.57 s
IChol	236	0.66 s	3.84 s	4.50 s
Poly	3	2.90 s	36.0 s	38.9 s

TABLE 5.7: Solution of a 3D Poisson equation using various preconditioners with CG.

Example 5.6 illustrates that even though polynomial approximate inverses can drastically lower the number of CG iterations, inexact spectral information can lead to high degree polynomials and larger run times.

5.2 Eigenvalues and Eigenvectors via Lanczos

The experiments in this section will use polynomial spectral transformations to accelerate the Implicitly Restarted Lanczos Method (IRLM) for computing eigenvalues and eigenvectors [66, 16, 47].

The implementation of IRLM follows the algorithm outlined in [76, p. 374]. This algorithm performs full reorthogonalization in order to ensure accurate Lanczos vectors. Thick implicit restarts [79] are performed via the Krylov–Schur algorithm [70] following the implementation outlined in [76, p. 365].

Whenever possible computations are performed using only the GPU in order to avoid expensive memory transfers between the CPU and GPU. The algorithm makes use of the package CUDA BLAS (CUBLAS) [22] to perform the reorthogonalizations and LAPACK [4] to perform the Krylov–Schur restarts.

The first experiment in this section illustrates how polynomial spectral transformations can be used to accurately approximate the invariant subspace corresponding to the smallest eigenvalues of the scaled discrete Laplace operator.

Example 5.7. *In this example a high degree polynomial spectral transformation is used to accurately approximate the invariant subspace of* Δ_n *corresponding to the* 10 *smallest eigenvalues for* $n = 2^{10}$*. The eigenvalues and eigenvectors of* Δ_n *are known explicitly,*

$$\lambda_i = 4 \sin\left(\frac{\pi i}{2(n+1)}\right)^2, \quad i = 1, \ldots, n, \tag{5.1}$$

$$v_{i,j} = \sqrt{\frac{2}{n+1}} \sin\left(\frac{\pi i j}{n+1}\right), \quad i, j = 1, \ldots, n. \tag{5.2}$$

A 1054 *degree polynomial approximation of* $f(z) = e^{-\tau(z/M)^2}$*,* $\tau = 10^7$*,* $\rho(\Delta_n) < M$ *and IRLM are used to compute* 10 *approximate eigenvectors* q_i*,* $\|q_i\|_2 = 1$*,* $i = 1, \ldots, 10$ *of* Δ_n*. These approximate eigenvectors are then used to compute Rayleigh quotients* $\mu_i = q_i^H \Delta_n q_i$*,* $i = 1, \ldots, 10$*. Since the eigenvalues and eigenvectors are known a relative error for the eigenvalues* $|\lambda_i - \mu_i|/|\lambda_i|$ *and the*

distance between eigenvectors $d(v_i, q_i)$ *can be computed. A relative residual is also computed,* $r_i = \|\Delta_n q_i - \rho_i q_i\|_2 / \|\Delta_n\|_2$. *These quantities are recorded in Table 5.8.*

Using a spectral transformation with degree greater than the size of the matrix illustrates the accuracy of using Algorithm 3 to compute the product $w = p(A)v$. *In theory, if the eigenvalues are known a polynomial of degree at most* n *could be constructed so that it transforms the spectrum identically. In practice the spectrum is not known. Much lower degree polynomials can be constructed by choosing a larger approximation tolerance. Figure 5.2 shows how the polynomial spectral transformation shifts the the first* 20 *eigenvalues of* Δ_n. *Notice how much the separation in the* 10^{th} *and* 11^{th} *eigenvalues increases after the transformation.*

Computing the upper bound of the spectral radius M *took* 0.33 *seconds, constructing the polynomial spectral transformation took* 0.14 *seconds and running IRLM took* 0.77 *seconds. The length of a single Lanczos run was* 60 *and convergence happened in the first run. The total number of matrix-vector multiplications was* $63,240$.

i	λ_i	r_i	$\lvert\mu_i - \lambda_i\rvert / \lvert\lambda_i\rvert$	$d(v_i, q_i)$
1	9.39×10^{-06}	3.08×10^{-15}	1.17×10^{-14}	8.88×10^{-16}
2	3.76×10^{-05}	2.76×10^{-15}	2.34×10^{-15}	2.22×10^{-16}
3	8.45×10^{-05}	2.49×10^{-15}	4.81×10^{-16}	4.44×10^{-16}
4	1.50×10^{-04}	3.27×10^{-15}	3.07×10^{-15}	9.99×10^{-16}
5	2.35×10^{-04}	3.26×10^{-15}	1.15×10^{-16}	6.66×10^{-16}
6	3.38×10^{-04}	3.61×10^{-15}	1.12×10^{-15}	2.22×10^{-16}
7	4.60×10^{-04}	3.59×10^{-15}	1.53×10^{-15}	0.00×10^{-00}
8	6.01×10^{-04}	3.97×10^{-15}	5.41×10^{-16}	2.22×10^{-16}
9	7.61×10^{-04}	4.62×10^{-15}	7.12×10^{-16}	2.22×10^{-16}
10	9.39×10^{-04}	5.50×10^{-15}	1.15×10^{-16}	3.33×10^{-16}

TABLE 5.8: Computed errors for the 10 smallest eigenvalues of the discrete Laplace operator.

A difficult question when constructing a polynomial spectral transformation using the function $f(z) = e^{-\tau(z/M)^2}$ is the choice of the shape parameter τ. In Example 5.7 a good τ was easy to

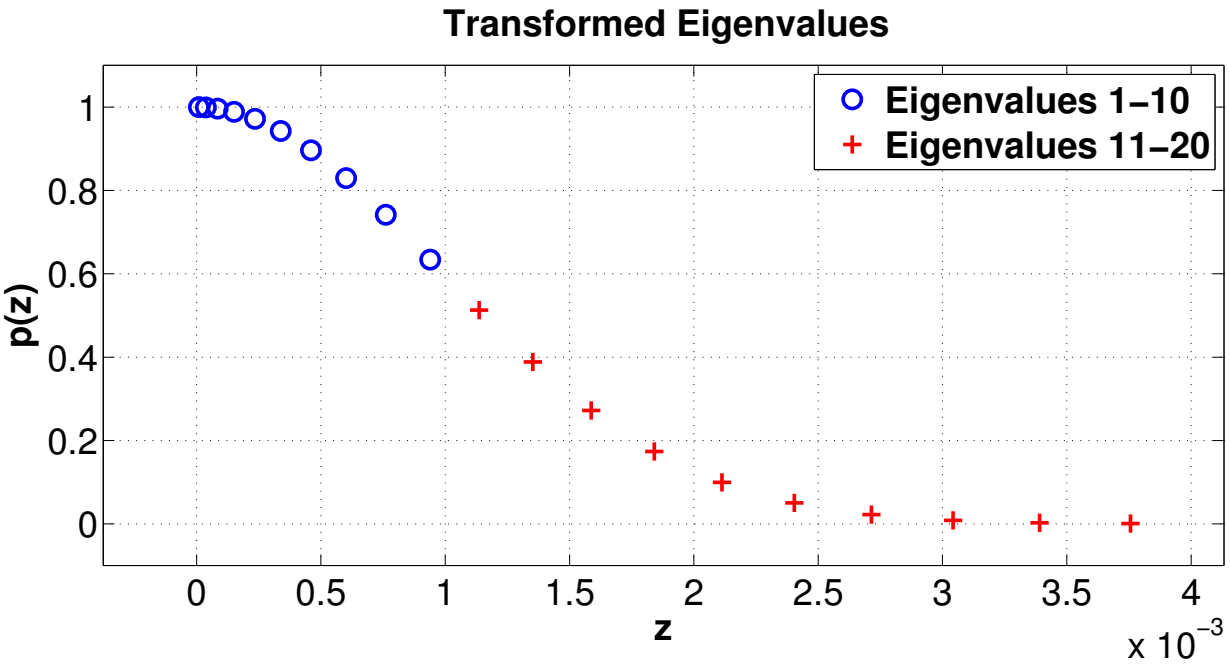

FIGURE 5.2: Transformed eigenvalues of the discrete Laplace operator using a 1054 degree polynomial spectral transformation.

choose because the spectrum was already known. The next example repeats the experiment in Example 5.7 using several different values of τ.

Example 5.8. *In this example the experiment in Example 5.7 is repeated using various values of* τ*. Table 5.9 records the chosen* τ*, degree of the polynomial spectral transformation, the maximum error in the computed eigenvalues, the number of restarts, the number of matrix-vector multiplications and the time to run IRLM. The length of a single Lanczos run was* 60 *and the maximum number of restarts was* 50*. Since the spectral radius was the same throughout an upper bound* M *was computed once.*

The results in Table 5.9 give some idea of the sensitivity of the method to the choice of τ*. For the smallest value* $\tau = 10^5$*, IRLM did not converge after* 50 *restarts. This is because the spectral transformation was not steep enough near the desired eigenvalues. For* $\tau = 10^6, 10^7$*, and* 10^8 *the desired eigenvalues were computed accurately. For* $\tau = 10^9$*, and* 10^{10}*, IRLM converged but some*

of the computed eigenvalues were incorrect. This is because the transformation was so steep that some of the desired spectrum was mapped to near 0 *giving a high dimensional null space. Figure 5.3 shows the transformed spectrum for various values of* τ*.*

τ	degree	$\max \lvert\mu_i - \lambda_i\rvert / \lvert\lambda_i\rvert$	restarts	matvecs	IRLM time (s)
10^{05}	336	$1.06 \times 10^{+03}$	50	1028160	17.26
10^{06}	594	1.62×10^{-15}	0	35640	0.50
10^{07}	1052	1.53×10^{-14}	0	63120	0.77
10^{08}	1883	1.14×10^{-14}	0	112980	1.25
10^{09}	3557	$3.36 \times 10^{+01}$	0	213420	2.23
10^{10}	7496	$9.71 \times 10^{+01}$	0	449760	4.51

TABLE 5.9: Sensitivity of the shape parameter τ when computing the 10 smallest eigenvalues of the discrete Laplace operator.

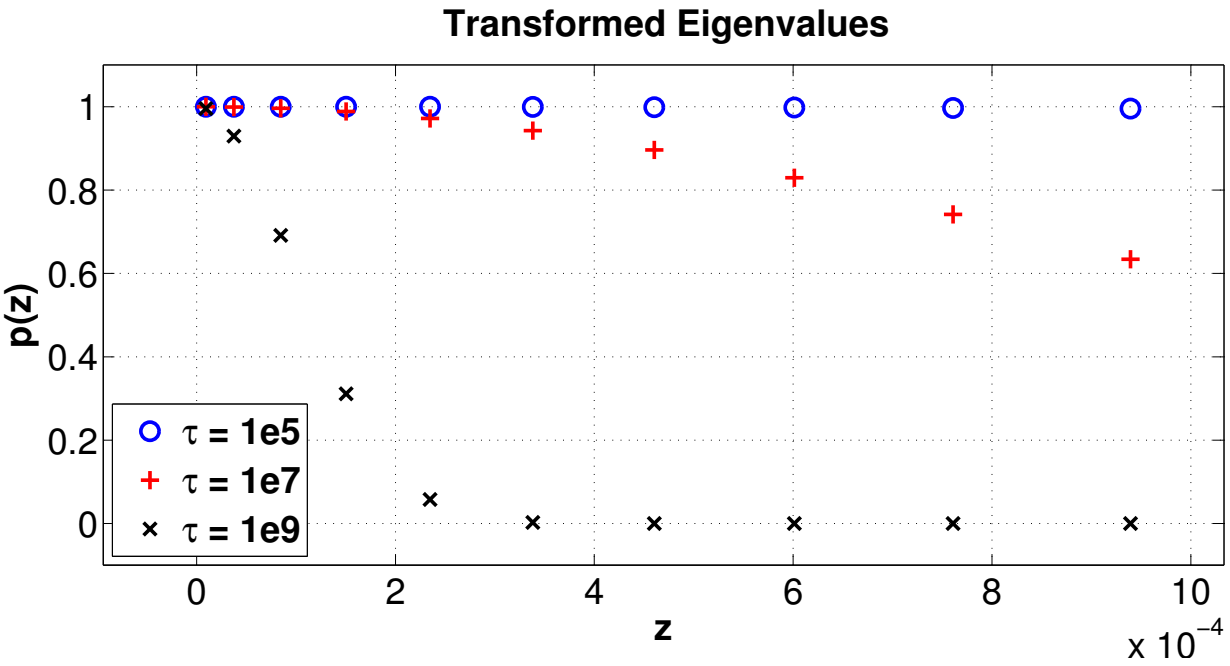

FIGURE 5.3: 10 smallest eigenvalues of the discrete Laplace operator transformed using the values $\tau = 10^5, 10^7$ and 10^9.

Examples 5.7 and 5.8 illustrate how high degree polynomial spectral transformations can be used to accurately compute invariant subspaces corresponding to clustered eigenvalues. The next experiment shows that low degree polynomials also work well, often with much lower run times.

Example 5.9. *In this example a finite difference approximation is used to compute the* 4 *smallest eigenvalues and corresponding eigenvectors of a weighted, 3D Laplace operator,*

$$-(u_{xx} + 2u_{yy} + 3u_{zz}) = \lambda u, \quad \Omega = [0,1]^3, \quad u\,|_{\partial\Omega} = 0.$$

The discretized Laplacian has the form,

$$\hat{\Delta}_{n_x n_y n_z} = I_{n_y n_z} \otimes \Delta_{n_x} + I_{n_z} \otimes (2\Delta_{n_y} \otimes I_{n_x}) + 3\Delta_{n_z} \otimes I_{n_x n_y},$$

where Δ *is the same as in Definition 5.1 and* n_x*,* n_y *and* n_z *are the number of grid points in the* x*,* y *and* z *directions respectively.*

The polynomial spectral transformations were constructed by first approximating an upper bound on the spectral radius $\rho(\hat{\Delta}_{n_x n_y n_z}) < M$ *and approximating the function* $f(z) = e^{-\tau(z/M)^2}$ *on the interval* $[0, M]$ *with* $\tau = 10^6$*. To keep the degree low a larger relative tolerance* ϵ *is used when the polynomial* p *is constructed,*

$$\|f - p\|_\infty < \epsilon \|f\|_\infty.$$

The polynomial spectral transformation and IRLM are used to compute 4 *approximate eigenvectors* q_i*,* $\|q_i\|_2 = 1$*,* $i = 1, \ldots, 4$ *of* $\hat{\Delta}_{n_x n_y n_z}$*. These approximate eigenvectors are then used to compute Rayleigh quotients* $\mu_i = q_i^H \Delta_n q_i$*,* $i = 1, \ldots, 4$*. Since the eigenvalues and eigenvectors are known a relative error for the eigenvalues* $|\lambda_i - \mu_i|/|\lambda_i|$ *and the distance between the true and approximate eigenvectors* $d(v_i, q_i)$ *can be computed. A relative residual is also computed,* $r_i = \|\Delta_n q_i - \rho_i q_i\|_2 / \|\Delta_n\|_2$*. These quantities are recorded in Table 5.10 for various choices of the tolerance* ϵ*.*

The number of grid points was taken to be $n_x = n_y = n_z = 10^2$ *giving a total problem dimension of* 10^6*. The length of a single Lanczos run was* 60 *and every choice of* ϵ *converged in one Lanczos run. Since the spectral radius was the same throughout an upper bound* M *was computed only once and took* 3.01 *seconds to compute.*

ϵ	degree	$\max \lvert\mu_i - \lambda_i\rvert / \lvert\lambda_i\rvert$	$\max r_i$	$\max d(v_i, q_i)$	IRLM time
10^{-02}	37	4.48×10^{-16}	3.38×10^{-14}	1.24×10^{-13}	26.5 s
10^{-03}	98	3.99×10^{-16}	4.52×10^{-14}	1.25×10^{-13}	65.8 s
10^{-04}	146	1.99×10^{-16}	4.52×10^{-14}	1.24×10^{-13}	96.7 s
10^{-05}	212	5.98×10^{-16}	4.64×10^{-14}	1.23×10^{-13}	139 s
10^{-06}	255	2.99×10^{-16}	4.46×10^{-14}	1.23×10^{-13}	167 s
10^{-07}	294	4.48×10^{-16}	4.48×10^{-14}	1.25×10^{-13}	192 s
10^{-08}	330	2.56×10^{-16}	4.50×10^{-14}	1.24×10^{-13}	215 s

TABLE 5.10: Sensitivity of the approximation tolerance ϵ when computing the 4 smallest eigenvalues of the weighted, 3D, discrete Laplace operator.

Example 5.9 illustrates the relative insensitivity of the computed eigenvalues and eigenvectors to the choice of approximation tolerance ϵ. The accuracy was essentially the same for all choices. The main difference was the degree of the polynomial and the overall runtime.

Example 5.10. *In this example the experiment in Example 5.9 is repeated using two different types of spectral transformations. The first type is a shift and invert spectral transformation using the preconditioned Conjugate Gradient method (CG). The preconditioners and the CG are supplied in the package CULA Sparse [30, 42] and performed on the GPU. For added comparison two different preconditioners are used. The first is a simple Jacobi preconditioner [1] and the second is an incomplete Cholesky preconditioner (IChol) [51, 45]. The CG iterations halted when the relative residual was below* 10^{-14}*. The shift was taken to be* 0*.*

The second type is a polynomial spectral transformation (Poly) that is constructed by computing an upper bound on the spectral radius $\rho(\hat{\Delta}_{n_x n_y n_z}) < M$ *and approximating the function* $f(z) = e^{-\tau(z/M)^2}$*,* $\tau = 10^6$ *on the interval* $[0, M]$ *using a relative tolerance of* $\epsilon = 10^{-2}$*.*

The number of grid points was taken to be $n_x = n_y = n_z = 10^2$ *giving a total problem dimension of* 10^6. *The length of a single Lanczos run was* 60 *and all the methods converged in one Lanczos run.*

Method	$\max \lvert\mu_i - \lambda_i\rvert / \lvert\lambda_i\rvert$	$\max r_i$	$\max d(v_i, q_i)$	IRLM time
Jacobi + CG	1.99×10^{-16}	1.37×10^{-14}	1.24×10^{-13}	468 s
IChol + CG	3.99×10^{-16}	1.14×10^{-14}	1.24×10^{-13}	427 s
Poly	4.48×10^{-16}	2.86×10^{-14}	1.24×10^{-13}	29.4 s

TABLE 5.11: Comparison of spectral transformations when computing the 4 smallest eigenvalues of the weighted, 3D, discrete Laplace operator.

Example 5.10 illustrates the use of polynomial spectral transformations to accelerate IRLM. A good choice of polynomial can give over a factor of 10 speedup over a GPU accelerated shift and invert strategy with essentially the same accuracy.

Example 5.11. *In this example the experiment in Example 5.10 is repeated using* $n_x = n_y = n_z = 2 \times 10^2$ *giving a total problem dimension of* 8×10^6. *Since the GPU used in the previous examples does not have enough memory this experiment is run on a different machine. This new workstation has an AMD FX-8320 eight core CPU with 16GB of memory and an NVIDIA GeForce GTX Titan GPU with 2688 cores and 6GB of onboard GPU memory.*

Even with the increased memory the shift and invert spectral transformations using preconditioned CG were not able to execute due to insufficient memory. To perform shift and invert CG was used without a preconditioner. The CG iterations halted when the relative residual was below 10^{-10}. *The shift was taken to be* 0.

The polynomial spectral transformation (Poly) is constructed by computing an upper bound on the spectral radius $\rho(\hat{\Delta}_{n_x n_y n_z}) < M$ *and approximating the function* $f(z) = e^{-\tau(z/M)^2}$, $\tau = 10^8$ *on the interval* $[0, M]$ *using a relative tolerance of* $\epsilon = 5 \times 10^{-3}$. *The length of a single Lanczos run was* 60 *and both methods converged in one Lanczos run.*

Method	$\max \lvert\mu_i - \lambda_i\rvert / \lvert\lambda_i\rvert$	$\max r_i$	$\max d(v_i, q_i)$	IRLM time
CG	7.89×10^{-16}	1.74×10^{-11}	1.43×10^{-13}	1178 s
Poly	7.40×10^{-16}	2.42×10^{-13}	1.41×10^{-13}	125 s

TABLE 5.12: Comparison of spectral transformations when computing the 4 smallest eigenvalues of the weighted, 3D, discrete Laplace operator.

Example 5.11 illustrates the use of polynomial spectral transformations to accelerate IRLM when shift and invert strategies are difficult to use.

Chapter 6

Conclusions

A novel technique for computing polynomial spectral transformations for the acceleration of linear algebra computations for large sparse positive definite matrices was presented. These methods were implemented on a GPU and for certain problems gave significant speed ups over traditional methods.

6.1 Linear System Solving

The experiments in Chapter 6 gave evidence that using a GPU to accelerate polynomial approximate inverses can be effective. When viewed as a direct solver and compared with Cholesky factorizations this combination can give speedups of over 100 for certain problems and allows large problems to be solved on a workstation. When viewed as preconditioners for the conjugate gradient method polynomial approximate inverses were not as efficient as more classical preconditioners that have been implemented on the GPU and there is still work needed to close the gap.

6.2 Eigenvalue and Eigenvector Computations

For eigenvalue computations, the combination of GPUs and polynomial spectral transformations was able to successfully accelerate the implicitly restarted Lanczos method for certain problems. The results in Chapter 5 give some examples where these techniques outperform current GPU methods by a factor 10. These methods also require minimal memory and allow large problems to be solved on a standard workstation.

Bibliography

[1] L. Adams, *M-step preconditioned conjugate gradient methods*, SIAM Journal on Scientific and Statistical Computing **6** (1985), 452–463.

[2] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov, *Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects*, Journal of Physics: Conference Series **180** (2009), 012037.

[3] N. Ahmed, T. Natarajan, and K. R. Rao, *Discrete cosine transform*, IEEE Transactions on Computers **C-23** (1974), 90–93.

[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK users' guide*, Third ed., SIAM, Philadelphia, PA, USA, 1999.

[5] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics **9** (1951), 17–29.

[6] S. F. Ashby, *Minimax polynomial preconditioning for hermitian linear systems*, SIAM Journal on Matrix Analysis and Applications **12** (1991), 766–789.

[7] S. F. Ashby, T. A. Manteuffel, and J. S. Otto, *A comparison of adaptive chebyshev and least squares polynomial preconditioning for hermitian positive definite linear systems*, SIAM Journal on Scientific and Statistical Computing **13** (1992), 1–29.

[8] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, *Adaptive polynomial preconditioning for hermitian indefinite linear systems*, BIT Numerical Mathematics **29** (1989), 583–609.

[9] O. Axelsson, *A generalized SSOR method*, BIT Numerical Mathematics **12** (1972), 443–467.

[10] ———, *Iterative solution methods*, Cambridge University Press, Cambridge, UK, 1996.

[11] C. Beattie, M. Embree, and D. C. Sorensen, *Convergence of polynomial restart krylov methods for eigenvalue computations*, SIAM Review **47** (2005), 492–515.

[12] M. Benzi, C. D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM Journal on Scientific Computing **17** (1996), 1135–1149.

[13] M. Bollhöfer and Y. Saad, *Multilevel preconditioners constructed from inverse-based ILUs*, SIAM Journal on Scientific Computing **27** (2006), 1627–1650.

[14] J. P. Boyd, *Chebyshev and fourier spectral methods*, Second ed., Courier Dover Publications, 2013.

[15] W. L. Briggs et al., *The DFT: An owners' manual for the discrete fourier transform*, SIAM, Philadelphia, PA, USA, 1995.

[16] D. Calvetti, L. Reichel, and D. C. Sorensen, *An implicitly restarted lanczos method for large symmetric eigenvalue problems*, Electronic Transactions on Numerical Analysis **2** (1994), 1–21.

[17] L. Cesari, *Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive*, Atti Accad. Naz. Lincei Rend. Cl. Sci. Fis. Mat. Nat. **25** (1937), 422–428.

[18] Françoise Chatelin, *Eigenvalues of matrices: Revised edition*, vol. 71, SIAM, Philadelphia, PA, USA, 2012.

[19] Y. Chen, T. A Davis, W. W. Hager, and S. Rajamanickam, *Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate*, ACM Transactions on Mathematical Software **35** (2008), 22.

[20] C. W. Clenshaw, *A note on the summation of chebyshev series*, Math. Tab. Wash. **9** (1955), 118–120.

[21] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, *The fast fourier transform and its applications*, IEEE Transactions on Education **12** (1969), 27–34.

[22] NVIDIA Corporation, *CUDA toolkit documentation*, July 2013.

[23] J. K. Cullum and W. E. Donath, *A block lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace for large, sparse symmetric matrices*, Proceedings of the 1974 IEEE Conference on Decision and Control (Piscataway, NJ), IEEE Press, 1974, pp. 505–509.

[24] A. de Moivre, *Miscellanea analytica de seriebus et quadraturis*, J. Tonson & J. Watts, 1730.

[25] J. W. Demmel, *Applied numerical linear algebra*, SIAM, Philadelphia, PA, USA, 1997.

[26] P. Dewilde and E. F. Deprettere, *Approximative inversion of positive matrices with applications to modelling*, Modelling, Robustness and Sensitivity Reduction in Control Systems, Springer, 1987, pp. 211–238.

[27] T. Driscoll, K. Toh, and L. N. Trefethen, *From potential theory to matrix iterations in six steps*, SIAM Review **40** (1998), 547–578.

[28] P. F. Dubois, A. Greenbaum, and G. H. Rodrigue, *Approximating the inverse of a matrix for use in iterative algorithms on vector processors*, Computing **22** (1979), 257–268.

[29] S. Eisenstat, J. Ortega, and C. Vaughan, *Efficient polynomial preconditioning for the conjugate gradient method*, SIAM Journal on Scientific and Statistical Computing **11** (1990), 859–872.

[30] EM Photonics, *CULA Sparse Programmer's Guide*, 2013, `http://www.culatools.com/cula_sparse_programmers_guide/`.

[31] H. Fang and Y. Saad, *A filtered lanczos procedure for extreme and interior eigenvalue problems*, SIAM Journal on Scientific Computing **34** (2012), A2220–A2246.

[32] B. Fischer and R. Freund, *On adaptive weighted polynomial preconditioning for hermitian positive definite matrices*, SIAM Journal on Scientific Computing **15** (1994), 408–426.

[33] G. E. Forsythe and C. B. Moler, *Computer solution of linear algebraic systems*, vol. 7, Prentice-Hall, Englewood Cliffs, NJ, USA, 1967.

[34] L. Fox and I. B. Parker, *Chebyshev polynomials in numerical analysis*, Oxford mathematical handbooks, Oxford Univ. Press, Oxford, UK, 1968.

[35] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*, 1809.

[36] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3, JHU Press, Baltimore, MD, USA, 2012.

[37] A. Greenbaum, *Iterative methods for solving linear systems*, vol. 17, SIAM, Philadelphia, PA, USA, 1997.

[38] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stand. **49** (1952), 409–436.

[39] N. J. Higham, *Accuracy and stability of numerical algorithms*, Second ed., SIAM, Philadelphia, PA, USA, 2002.

[40] ———, *Functions of matrices: Theory and computation*, SIAM, Philadelphia, PA, USA, 2008.

[41] J. Hoberock and N. Bell, *Thrust: A parallel template library*, 2010, `http://thrust.github.io/`.

[42] J. R. Humphrey, D. K. Price, K. E. Spagnoli, A. L. Paolini, and E. J. Kelmelis, *CULA: hybrid GPU accelerated linear algebra routines*, SPIE Defense, Security, and Sensing, International Society for Optics and Photonics, 2010, pp. 770502–770502.

[43] I. C. F. Ipsen, *Numerical matrix analysis: Linear systems and least squares*, SIAM, PHiladelphia, PA, USA, 2009.

[44] O. Johnson, C. Micchelli, and G. Paul, *Polynomial preconditioners for conjugate gradient calculations*, SIAM Journal on Numerical Analysis **20** (1983), 362–376.

[45] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings I. theory*, SIAM Journal on Matrix Analysis and Applications **14** (1993), 45–58.

[46] D. Kressner, *Numerical methods for general and structured eigenvalue problems*, Tech. report, Springer, 2005.

[47] 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), 255–282.

[48] Y. Liang, *The use of parallel polynomial preconditioners in the solution of systems of linear equations*, Ph.D. thesis, University of Ulster, 2005.

[49] G. G. Lorentz, *Approximation of functions*, Second ed., Chelsea Publishing Co, New York, NY, USA, 1986.

[50] MATLAB, *version 7.14.0 (r2012a)*, The MathWorks Inc., Natick, MA, USA, 2012.

[51] J. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of computation **31** (1977), 148–162.

[52] G. Meurant, *The lanczos and conjugate gradient algorithms: from theory to finite precision computations*, vol. 19, SIAM, Philadelphia, PA, USA, 2006.

[53] R. Morgan, *A restarted GMRES method augmented with eigenvectors*, SIAM Journal on Matrix Analysis and Applications **16** (1995), 1154–1171.

[54] J. Nickolls, I. Buck, M. Garland, and K. Skadron, *Scalable parallel programming with CUDA*, Queue **6** (2008), 40–53.

[55] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*, July 2013, `http://docs.nvidia.com/cuda/cuda-c-programming-guide/`.

[56] B. N. Parlett, *The symmetric eigenvalue problem*, Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1980.

[57] M. J. D. Powell, *Restart procedures for the conjugate gradient method*, Mathematical Programming **12** (1977), 241–254.

[58] A. Ruhe, *Rational krylov sequence methods for eigenvalue computation*, Linear Algebra and its Applications **58** (1984), 391 – 405.

[59] C. Runge, *Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten*, Z. Math. Phys. **46** (1901), 224–243.

[60] Y. Saad, *Practical use of polynomial preconditionings for the conjugate gradient method*, SIAM Journal on Scientific and Statistical Computing **6** (1985), 865–881.

[61] ———, *Iterative methods for sparse linear systems*, Second ed., SIAM, Philadelphia, PA, USA, 2003.

[62] ———, *Numerical methods for large eigenvalue problems*, Second ed., SIAM, Philadelphia, PA, USA, 2011.

[63] Y. Saad and M. Schultz, *GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM Journal on Scientific and Statistical Computing **7** (1986), 856–869.

[64] G. Schofield, J. R. Chelikowsky, and Y. Saad, *A spectrum slicing method for the kohn–sham problem*, Computer Physics Communications **183** (2012), 497 – 505.

[65] A. Smoktunowicz, *Backward stability of clenshaw's algorithm*, BIT Numerical Mathematics **42** (2002), 600–610.

[66] D. C. Sorensen, *Implicit application of polynomial filters in a k-step arnoldi method*, SIAM Journal on Matrix Analysis and Applications **13** (1992), 357–385.

[67] G. W. Stewart, *Introduction to matrix computations*, (1973).

[68] ———, *The convergence of the method of conjugate gradients at isolated extreme points of the spectrum*, Numerische Mathematik **24** (1975), 85–93.

[69] G. W. Stewart, *Matrix algorithms volume 2: Basic decompositions*, Cambridge University Press, Cambridge, UK, 1998.

[70] G. W. Stewart, *Matrix algorithms volume 1: Eigensystems*, SIAM, Philadelphia, PA, USA, 2001.

[71] L. N. Trefethen, *Approximation theory and approximation practice*, Applied mathematics, SIAM, Philadelphia, PA, USA, 2013.

[72] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50, SIAM, Philadelphia, PA, USA, 1997.

[73] L. N. Trefethen et al., *Chebfun Version 4.2*, The Chebfun Development Team, 2011, `http://www.chebfun.org/`.

[74] H. A. Van der Vorst, *Iterative krylov methods for large linear systems*, vol. 13, Cambridge University Press, Cambridge, UK, 2003.

[75] J. L. Walsh, *Interpolation and approximation by rational functions in the complex domain*, American Mathematical Society, Providence, RI, USA, 1965.

[76] D. S. Watkins, *The matrix eigenvalue problem: GR and Krylov subspace methods*, SIAM, Philadelphia, PA, USA, 2007.

[77] ———, *Fundamentals of Matrix Computations*, Third ed., John Wiley and Sons, Hoboken, NJ, USA, 2010.

[78] J. H. Wilkinson, *The algebraic eigenvalue problem*, vol. 87, Clarendon Pres, Oxford, UK, 1965.

[79] K. Wu and H. Simon, *Thick-restart lanczos method for large symmetric eigenvalue problems*, SIAM Journal on Matrix Analysis and Applications **22** (2000), 602–616.

[80] J. Zhang and L. Zhang, *Efficient CUDA polynomial preconditioned conjugate gradient solver for finite element computation of elasticity problems*, Mathematical Problems in Engineering **2013** (2013).