The Dissertation Committee for Jack Lesly Poulson
certifies that this is the approved version of the following dissertation:

Fast Parallel Solution of Heterogeneous 3D Time-harmonic Wave Equations

Committee:

Lexing Ying, Supervisor

Björn Engquist

Sergey Fomel

Omar Ghattas

Robert van de Geijn

Fast Parallel Solution of Heterogeneous 3D Time-harmonic Wave Equations

by

Jack Lesly Poulson, B.S.As.E.; M.S.E.; M.S.C.A.M.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2012

Acknowledgments

I would like to begin by thanking Lexing Ying for his immense amount of help and advice. He has always made me feel comfortable discussing my conceptual stumbling blocks, and I am now convinced that there are few things he cannot teach with five minutes and a blackboard. Looking back, it is hard to express how much he has helped me learn over the past three years. Modern numerical analysis is far more intellectually satisfying that I could have ever possibly known, and I look forward to continually expanding upon the many insights he has been kind enough to share with me.

I also owe a large amount of thanks to Robert van de Geijn. My master's thesis was on parallel dense linear algebra, and most of the insights it contained were due to our many conversations. We both have an appreciation for continually refining and expanding core concepts and notation, and so we have continually collaborated towards this goal. I would have never expected so many insights from dense linear algebra to play a role in this work.

I would also like to thank Sergey Fomel for introducing me to seismic applications and for helping to teach me some of the language of the field. I am also grateful for his help in interfacing the Parallel Sweeping Preconditioner with Madagascar, and for his and Siwei Li's in working with the Overthrust model. Computational geophysics is an exciting field and I look forward to working in the area for years to come.

I would also like to thank Paul Tsuji for sharing his hard-earned experience in extending sweeping preconditioners to such a large class of problems and discretizations. He has been a good friend and a better colleague, and I look forward to many fruitful future collaborations.

Fast Parallel Solution of Heterogeneous 3D Time-harmonic Wave Equations

Publication No. ____________

Jack Lesly Poulson, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Lexing Ying

Several advancements related to the solution of 3D time-harmonic wave equations are presented, especially in the context of a parallel *moving-PML* sweeping preconditioner for problems without large-scale resonances. The main contribution of this dissertation is the introduction of an efficient parallel sweeping preconditioner and its subsequent application to several challenging velocity models. For instance, 3D seismic problems approaching a billion degrees of freedom have been solved in just a few minutes using several thousand processors. The setup and application costs of the sequential algorithm were also respectively refined to $O(\gamma^2 N^{4/3})$ and $O(\gamma N \log N)$, where N denotes the total number of degrees of freedom in the 3D volume and $\gamma(\omega)$ denotes the modestly frequency-dependent number of grid points per Perfectly Matched Layer discretization.

Furthermore, high-performance parallel algorithms are proposed for performing multifrontal triangular solves with many right-hand sides, and a custom compression scheme is introduced which builds upon the translation invariance of free-space Green's functions in order to justify the replacement of each dense matrix within a certain modified multifrontal method with the sum of a small number of Kronecker products. For the sake of reproducibility, every algorithm exercised within this dissertation is made available as part of the open source packages *Clique* and *Parallel Sweeping Preconditioner (PSP)*.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The holy grail of exploration geophysics is to be able to take a boat out to sea, fire off air guns near the surface of the water, and to use measurements of the resulting echoes as a means of quickly constructing an accurate model for the wave propagation characteristics throughout the portion of the earth below the boat. This is an important example of an *inverse problem*, and a large percentage of the seismic community is dedicated to at least finding qualitative solutions to questions such as, “Do the material properties imply a rock formation which is likely to contain hydrocarbons?”.

1.1 Fundamental problem

The name *inverse* problem is a reference to the usual method of attack: if we assume the necessary material properties throughout the relevant portion of the domain then we may use a *forward model*, such as the 3D *elastic wave equation*,

$$-\nabla \cdot (\mathbf{C}(x)\nabla \mathbf{U}(x,t)) + \rho(x)\frac{\partial^2}{\partial t^2}\mathbf{U}(x,t) = \mathbf{F}(x,t), \tag{1.1}$$

to model the evolution of the displacement field, $\mathbf{U}(x,t)$, given the forcing function $\mathbf{F}(x,t)$, which is perhaps the result of a blast from an air gun or stick of dynamite. The difference between this simulation and a physical experiment can then be used to drive an iterative method which hopefully converges towards the “true” mass density, $\rho(x)$, and fourth-order stiffness tensor, $\mathbf{C}(x)$.

While the details of effective inversion techniques are well beyond the scope of this document, it is important to recognize a few common characteristics:

- the forward model must be used many times,
- the same forward model is typically used for many experiments,
- many researchers prefer to work in the temporal-frequency domain [115], and, perhaps most importantly,
- the domain often spans large numbers of wavelengths, which requires a fine-scale discretization (and therefore a large-scale linear system).

In order to drive the last point home, let us recognize that the speed of sound in water is roughly 1.5 km/sec, which implies that it would take roughly 10 seconds for sound to propagate across a distance of 15 km. If a sound wave oscillates at a frequency of 30 Hz, then, throughout the course of the 10 seconds it takes to travel this distance, it will oscillate 300 times.

In order to capture these oscillations over a 15 km $\times$ 15 km region of the surface, significantly more than 300 grid points would be required in each of the two horizontal directions. We might say that such a domain *spans 300 wavelengths* and describe the quality of the discretization in terms of the number of *points per wavelength* (eight to ten points per wavelength is quite common in industry).[1]

[1]It is important to combat so-called *pollution* effects [15], for instance using high-order spectral elements [123, 124], but this issue is somewhat orthogonal to the focus of this dissertation.

1.2 Time-harmonic wave equations

Consider the scalar analogue of the elastic wave equation, the 3D *acoustic wave equation* [121],

$$\left[-\Delta + \frac{1}{c^2(x)}\frac{\partial^2}{\partial t^2}\right] U(x,t) = F(x,t), \tag{1.2}$$

where $U(x,t)$ is the time and spatially-varying *pressure field*, $F(x,t)$ is the *forcing function*, and $c(x)$ is the sound speed. If we were to perform a formal *Fourier transform* [84] in the time variable of each side of this equation, we would decouple the solution $U(x,t)$ into a sum of solutions of the form $u(x)e^{-i\omega t}$, each driven by a forcing function of the form $f(x)e^{-i\omega t}$, where ω is the *temporal frequency*, which usually has units of radians per second.[2] Since

$$\frac{\partial^2}{\partial t^2}\left[u(x)e^{-i\omega t}\right] = -\omega^2\left[u(x)e^{-i\omega t}\right],$$

for each frequency ω, we can instead pose a *Helmholtz equation*,

$$\left[-\Delta - \frac{\omega^2}{c^2(x)}\right] u(x) = f(x). \tag{1.3}$$

Because of the implicit $e^{-i\omega t}$ time-dependence of the solution and forcing function, the Helmholtz equation is (the prototypical example of) a *time-harmonic wave equation.*

It is important to recognize that the negative Laplacian operator, $-\Delta$, when discretized over a finite domain with homogeneous boundary conditions, results in a real, symmetric, *positive-definite* matrix, with eigenvalues on the positive real-line which become arbitrarily close to the origin and approach

[2]Different communities have different conventions for Fourier transforms. It is also common to use a time-dependence of $e^{i\omega t}$, but this essentially only requires conjugating the appropriate terms in the following discussion.

infinity as the discretization is refined. The Helmholtz operator, $-\Delta - \omega^2$, therefore has the same eigenvalues, but shifted left by a distance of ω^2 in the complex plane. Thus, for any nonzero frequency ω and sufficiently fine discretization, its eigenvalues span both sides of the origin on the real line (and are often quite close to the origin). This admittedly crude interpretation of the spectrum of the Helmholtz operator explains the essential difficulty with solving Helmholtz equations via iterative methods. For instance, because of the shape of the spectrum, it is not possible to construct an ellipse in the complex plane which contains the spectrum but not the origin (see Definition B.4.4).

An analogous, though significantly more complex, technique can be used to reduce Maxwell's equations to time-harmonic form [123]. Consider the constant-coefficient Maxwell's equations:

$$\nabla \times \mathbf{E}(x,t) = -\dot{\mathbf{H}}(x,t), \tag{1.4}$$

$$\nabla \times \mathbf{H}(x,t) = \dot{\mathbf{E}}(x,t) + \mathbf{J}(x,t), \tag{1.5}$$

$$\nabla \cdot \mathbf{E}(x,t) = \frac{i}{\omega}\nabla \cdot \mathbf{J}(x,t), \text{ and} \tag{1.6}$$

$$\nabla \cdot \mathbf{H}(x,t) = 0, \tag{1.7}$$

where $\mathbf{E}(x,t)$ and $\mathbf{H}(x,t)$ respectively model the three-dimensional electric and magnetic fields, and $\mathbf{J}(x,t)$ is the current distribution, which is also a three-dimensional vector quantity. If we again perform a formal Fourier transform in the time variable of each set of equations, then we may represent the solutions $\mathbf{E}(x,t)$ and $\mathbf{H}(x,t)$ as sums of terms of the form $\mathcal{E}(x)e^{-i\omega t}$ and $\mathcal{H}(x)e^{-i\omega t}$, which are each driven by a harmonic component $\mathcal{J}(x)e^{-i\omega t}$ of the current distribution, $\mathbf{J}(x,t)$. Substituting these harmonic quantities into Maxwell's equations yields

$$\nabla \times \mathcal{E}(x) = i\omega\mathcal{H}(x), \tag{1.8}$$

$$\nabla \times \mathcal{H}(x) = -i\omega\mathcal{E}(x) + \mathcal{J}(x), \tag{1.9}$$

$$\nabla \cdot \mathcal{E}(x) = \frac{i}{\omega}\nabla \cdot \mathcal{J}(x), \text{ and} \tag{1.10}$$

$$\nabla \cdot \mathcal{H}(x) = 0. \tag{1.11}$$

The last step in deriving the time-harmonic form of Maxwell's equations involves the vector identity

$$\nabla \times \nabla \times \mathbf{a} = \nabla(\nabla \cdot \mathbf{a}) - \Delta\mathbf{a}, \tag{1.12}$$

which may be readily proved with so-called *index notation* [104] via the product rule and knowledge of the *Levi-Civita symbol*, which is also known as the *alternating symbol*. In particular, if we take the curl of both sides of Equation (1.8), then our vector identity allows us to write

$$\nabla(\nabla \cdot \mathcal{E}) - \Delta\mathcal{E} = i\omega\nabla \times \mathcal{H}.$$

We may then substitute Equations (1.10) and (1.11) in order to arrive at the result

$$\left[-\Delta - \omega^2\right]\mathcal{E}(x) = f_{\mathcal{E}}(\mathcal{J}) \equiv i\omega\mathcal{J} - \frac{i}{\omega}\nabla(\nabla \cdot \mathcal{J}), \tag{1.13}$$

which was purposely written in a form similar to that of Equation (1.3). We may similarly take the curl of Equation (1.9) in order to find

$$\left[-\Delta - \omega^2\right]\mathcal{H}(x) = f_{\mathcal{H}}(\mathcal{J}) \equiv \nabla \times \mathcal{J}, \tag{1.14}$$

which is clearly also similar in structure to the Helmholtz equation. Note that these time-harmonic equations for the electric and magnetic fields, which are ostensibly decoupled, will typically interact through an appropriate choice of boundary conditions.

Before we move on, it is important to emphasize that the equations for 3D time-harmonic linear elasticity are essentially an *anisotropic* form of the Helmholtz equation, which roughly means that waves do not propagate radially outwards equally in each direction. In particular, a formal Fourier transform of the elastic wave equation implies the *time-harmonic elastic wave equation*,

$$-(C_{ijkl}U_{k,l})_{,j} - \omega^2 \rho U_i = F_i, \tag{1.15}$$

which we have written in the usual index notation.[3] The basic idea is that $\mathbf{x}_i$ refers to the i'th coordinate of the vector $\mathbf{x}$, $\mathbf{x}_{i,j}$ refers to the derivative of the i'th coordinate in the j'th direction, and a repeated index implies that it is a dummy summation variable, e.g., $A_{ik}B_{kj}$ is an expression of the (i,j) entry of the matrix AB, and $A_{ki}B_{kj}$ represents the (i,j) entry of $A^T B$.

1.3 Solving Helmholtz equations

Until recently, doubling the frequency of Equation (1.3) not only increased the size of the linear system by at least a factor of two in each dimension, it also doubled the number of iterations required for convergence with preconditioned Krylov methods [23, 45, 47]. Thus, denoting the number of degrees of freedom in a three-dimensional finite-element or finite-difference discretization as $N = \Omega(\omega^3)$, every linear solve required $\Omega(\omega^4)$ work with iterative techniques. Engquist and Ying recently introduced two classes of *sweeping* pre-

[3]For an introduction to index notation and linear elasticity, please consult a book on continuum mechanics, such as [104].

conditioners for Helmholtz equations without internal resonance [43, 44]. Both approaches approximate a block LDL^T factorization of the Helmholtz operator in block tridiagonal form in a manner which exploits a *radiation boundary condition* [78]. The first approach performs a block tridiagonal factorization algorithm in $\mathcal{H}$-matrix arithmetic [59, 66], while the second approach approximates the Schur complements of the factorization using auxiliary problems with artificial radiation boundary conditions. Though the $\mathcal{H}$-matrix sweeping preconditioner has theoretical support for two-dimensional problems [43, 92], there is not yet justification for three-dimensional problems.

This dissertation therefore focuses on the second approach, which relies on multifrontal factorizations [41, 54, 91, 112] of approximate auxiliary problems in order to achieve an $O(\gamma^2 N^{4/3})$ setup cost and an $O(\gamma N \log N)$ application cost, where $\gamma(\omega)$ denotes the number of grid points used to discretize each radiation boundary condition (in our case, a Perfectly Matched Layer [19, 29, 78]), and we note that γ typically grows logarithmically with frequency. While the sweeping preconditioner is competitive with existing techniques even for a single right-hand side, its main advantage is for problems with large numbers of right-hand sides, as the preconditioner appears to converge in $O(1)$ iterations for problems without *internal resonance* [44]. Thus, after setting up the preconditioner, typically only $O(\gamma N \log N)$ work is required for each solution. As mentioned at the beginning of this chapter, frequency-domain seismic inversion requires the solution of large numbers of time-harmonic wave equations posed over domains spanning large numbers of wavelengths, and so we will focus on the efficient implementation of a parallel sweeping preconditioner in order to achieve solution costs closer to $O(\omega^3)$ than $O(\omega^4)$.

1.4 Background material

A more detailed discussion of the sweeping preconditioner requires a thorough knowledge of sparse-direct methods and at least a passing familiarity with iterative methods and wave equations. Chapter 2 gives a detailed introduction to multifrontal Cholesky factorization and triangular solves under the assumption that the reader is well-versed in numerical linear algebra. If this is not the case, it is recommended that the reader work through the proofs in Appendix A leading up to the Hermitian spectral decomposition (Corollary A.2). In addition, Appendix B is provided for those who would like to familiarize themselves with the theory behind the Generalized Minimum Residual method (GMRES) [108] used in Chapters 3 and 4. Lastly, Appendix C is provided for readers who are interested in the (parallel) dense linear algebra algorithms used at the core of the multifrontal algorithms discussed in Chapter 2. Their implementations are all part of the Elemental library [99], which is written in a style derived from FLAME notation [60]. It is also strongly recommended that the reader familiarize themselves with the fundamentals behind BLAS [37], LAPACK [5], and MPI [39]. In particular, readers interested in gaining a better understanding of *collective communication* algorithms should consult [28].

1.5 Contributions

There are essentially four significant contributions in this dissertation:

1. two high-performance algorithms for multifrontal triangular solves are introduced,

2. a parallelization of the sweeping preconditioner is introduced and applied to large-scale challenging heterogeneous 3D Helmholtz equations,

3. a compressed variant of the sweeping preconditioner is introduced which attempts to exploit the (approximate) translation invariance of free-space Green's functions, and

4. a high-performance algorithm for applying the compressed frontal matrices in their Kronecker-product form is proposed.

Together, these advancements allow for the solution of large-scale 3D Helmholtz equations at unprecedented rates. In particular, it will be shown in Chapter 3 that, after spending a few minutes setting up the sweeping preconditioner, 3D systems of equations approaching a billion degrees of freedom can be solved in a matter of seconds.

1.6 Outline

The remainder of this dissertation is organized as follows:

- Chapter 2 provided a detailed introduction to sequential and parallel algorithms for multifrontal Cholesky factorization from the point of view of reordering the operations in a dense Cholesky factorization. Readers who are not concerned with the inner-workings of sparse-direct solvers may simply want to familiarize themselves with Table 2.1.

- Chapter 3 provides an introduction to the *moving PML* sweeping preconditioner, describes a careful parallelization, and then discusses results for several challenging velocity models.

- Chapter 4 describes how the translation invariance of free-space Green's functions can be used to compress certain half-space Green's functions associated with the diagonal blocks of a modified multifrontal factorization, as well as how these compressed matrices may be efficiently applied as part of a multifrontal triangular solve.

- And, lastly, Chapter 5 gives a brief review of the contributions made throughout this dissertation and then concludes with possible future work and the websites of the software produced as part of this dissertation.

As previously mentioned, Appendix A is meant to provide a brief introduction to finite-dimensional spectral theory, Appendix B gives a short introduction to GMRES, and Appendix C is meant to provide a concise description of the dense linear algebra algorithms used throughout this dissertation.

Chapter 2

Multifrontal methods

This chapter attempts to give a self-contained, elementary introduction to efficient distributed-memory algorithms for *multifrontal* [41, 91] Cholesky factorization, as later chapters will rely upon an essentially identical approach for indefinite sparse LDL^T factorization as part of a *sweeping preconditioner* [44]. Many numerical analysts will undoubtedly feel uneasy about unpivoted LDL^T factorizations, but this ostensibly naïve approach is at least partially justified for the following reasons:

- sparse-direct solvers will only be used to solve approximate auxiliary problems as part of a preconditioner,
- the spectra are typically well-separated from the origin, and
- no instabilities have been encountered, even for large-scale models.

Nevertheless, scalable pivoted variants of the algorithms discussed in this chapter will be investigated as part of future work. Readers who are not interested in the inner-workings of multifrontal algorithms may safely proceed to Chapter 3 after familiarizing themselves with the asymptotic complexities listed in Table 2.1.

Although a major portion of this chapter will be a review of well-known results, we will deviate from standard approaches to scalable multifrontal Cholesky factorization and solution [56, 63, 64, 113] in the following two ways:

1. element-wise matrix distributions [71, 93, 99] will be used for parallelizing dense linear algebra operations, and
2. distributed sparse-direct triangular solve algorithms built upon dense distributed *TRSM* and *GEMM* [36, 99] kernels will be introduced, where the latter exploits *selective inversion* [102, 103, 114].

Before we start, it is also important to briefly mention the context of multifrontal methods within the world of sparse solvers. By the end of this chapter, we will have shown that a multifrontal factorization is essentially a reordering of the operations within a right-looking dense Cholesky factorization, where the ability to reorder said operations is precisely due to the sparsity of the matrix being factored. It is, of course, possible to modify other dense Cholesky factorization algorithms in order to produce alternative sparse factorization algorithms, such as the *column fan-in* [10] approach. Also, a short note on the name *multifrontal* itself: it is a reference to the earlier *frontal* scheme of Irons [75], which was an improvement to *banded* Cholesky factorization. Rather than discussing this progression in detail, we will instead transition from dense to sparse matrices through the class of (block) arrowhead matrices.

2.1 Dense Cholesky factorization

Definition 2.1.1 (Hermitian positive (semi-)definite)**.** A matrix is called *Hermitian positive semi-definite* (HPSD) if it is Hermitian and every eigenvalue is non-negative, and *Hermitian positive-definite* (HPD) when every eigenvalue is positive.

Theorem 2.1.1 (semi-definite Cholesky decomposition)**.** *Every Hermitian positive semi-definite matrix has a* Cholesky decomposition*,*

$$A = R^H R,$$

where R is upper-triangular.

Proof. As a consequence of Corollary A.2, every HPSD matrix $A \in M_n$, where M_n is the space of $n \times n$ complex matrices, has a *spectral decomposition*

$$A = V \Lambda V^H,$$

where $V \in M_n$ is unitary and, by assumption, every eigenvalue along the diagonal of Λ is non-negative. We may thus define the square-root of the diagonal matrix Λ entry-wise along its diagonal, and we denote this result as $\sqrt{\Lambda}$. Then the operator

$$A^{1/2} = V \sqrt{\Lambda} V^H$$

is also HPSD, and $A^{1/2}A^{1/2} = A$, which justifies the choice of notation.

Now, let us apply Theorem A.9 to $A^{1/2}$ to show there exists some unitary matrix $Q \in M_n$ and upper-triangular matrix $R \in M_n$ such that $A^{1/2} = QR$. But, $A^{1/2}$ is Hermitian, and so $A^{1/2} = R^H Q^H$, and we find that

$$A = R^H Q^H Q R = R^H R.$$

□

Remark 2.1.1. We have now justified an interpretation of a Cholesky factor of a Hermitian positive semi-definite matrix A as the square-root of A: it may always be chosen to be the upper-triangular factor, R, of a QR decomposition of $A^{1/2}$. We will now strengthen our assumptions and give a constructive proof of uniqueness which will serve as the starting point for the remainder of this chapter.

Lemma 2.1.2. *If $A \in M_n$ is Hermitian positive-definite, and $X \in M_{n,k}$ has linearly independent columns, then $X^H A X$ is also Hermitian positive-definite.*

Proof.

$$(y, X^H A X y) = ((Xy), A(Xy)) \geq 0,$$

with equality if and only if $Xy = 0$. But, because the columns of X are linearly independent, Xy must be nonzero. $\square$

Theorem 2.1.3 (Cholesky factorization)**.** *Every Hermitian positive-definite matrix has the unique Cholesky factorization*

$$A = LL^H,$$

where L is lower-triangular with positive diagonal entries.

Proof. If $A \in M_n$ is HPD, then Theorem 2.1.1 guarantees the existence of some upper-triangular matrix $R \in M_n$ such that $A = R^H R$. Let us partition this matrix expression as

$$\begin{pmatrix} \alpha_{1,1} & a_{2,1}^H \\ a_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} \overline{\rho_{1,1}} & 0 \\ r_{1,2} & R_{2,2}^H \end{pmatrix} \begin{pmatrix} \rho_{1,1} & r_{1,2}^H \\ 0 & R_{2,2} \end{pmatrix} = \begin{pmatrix} |\rho_{1,1}|^2 & \overline{\rho_{1,1}} r_{1,2}^H \\ \rho_{1,1} r_{1,2} & r_{1,2} r_{1,2}^H + R_{2,2}^H R_{1,1} \end{pmatrix}.$$

Since A is nonsingular, R must also be nonsingular, and thus Theorem A.1 guarantees that the diagonal entries of R are nonzero. We can thus recognize that the top-left entry of A, $\alpha_{0,0} = |\rho_{0,0}|^2$, is positive, and so we can legally take its square root and divide by it. Consider the partitioned matrix expression

$$\begin{pmatrix} \alpha_{1,1} & a_{2,1}^H \\ a_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} \lambda_{1,1} & 0 \\ \ell_{2,1} & L_{2,2} \end{pmatrix} \begin{pmatrix} \lambda_{1,1} & \ell_{1,0}^H \\ 0 & L_{2,2}^H \end{pmatrix} = \begin{pmatrix} \lambda_{1,1}^2 & \lambda_{1,1}\ell_{2,1}^H \\ \lambda_{1,1}\ell_{2,1}^H & \ell_{2,1}\ell_{2,1}^H + L_{2,2} \end{pmatrix},$$

where $L = \begin{pmatrix} \lambda_{1,1} & 0 \\ \ell_{2,1} & L_{2,2} \end{pmatrix}$ is the lower-triangular Cholesky factor we wish to show can be chosen to have a positive diagonal. Clearly we may set $\lambda_{1,1} = \sqrt{\alpha_{1,1}}$ and $\ell_{2,1} := a_{2,1}/\lambda_{1,1}$, and then find that $L_{2,2}^H L_{2,2} = A_{2,2} - \ell_{2,1}\ell_{2,1}^H$, but, in order to recurse on the updated bottom-right quadrant, we must first show that the *Schur complement* $A_{2,2} - \ell_{2,1}\ell_{2,1}^H$ inherits the positive-definiteness of A. But this Schur complement is the result of the product $X^H A X$, where

$$X = \begin{pmatrix} -a_{2,1}/\alpha_{1,1} \\ I \end{pmatrix}$$

has linearly independent columns. The preceding lemma therefore yields the result. □

Remark 2.1.2. The recursive procedure for computing the lower-triangular Cholesky factor of a Hermitian positive-definite matrix given in the previous theorem is called a *right-looking* Cholesky factorization, and its pseudocode is given in Algorithm C.16. Furthermore, it can easily be generalized into a so-called *blocked algorithm* [5], which expresses most of the factorization in terms of matrix-matrix multiplication in order to minimize data movement between different levels of the memory hierarchy. The pseudocode for a blocked right-looking Cholesky factorization is given in Algorithm C.17, though the upcoming subsection will work with the decomposition

$$\begin{pmatrix} A_{0,0} & A_{1,0}^H & A_{2,0}^H \\ A_{1,0} & A_{1,1} & A_{2,1}^H \\ A_{2,0} & A_{2,1} & A_{2,2} \end{pmatrix} = \begin{pmatrix} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{pmatrix} \begin{pmatrix} L_{0,0}^H & L_{1,0}^H & L_{2,0}^H \\ 0 & L_{1,1}^H & L_{2,1}^H \\ 0 & 0 & L_{2,2}^H \end{pmatrix},$$

which can be directly computed with Algorithm 2.1 using the notation

$$L_{1:2,0} \equiv \begin{pmatrix} L_{1,0} \\ L_{2,0} \end{pmatrix}, \quad A_{1:2,1:2} \equiv \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix}.$$

Algorithm 2.1: unrolled 3×3 block Cholesky factorization

```
1 $L_{0,0} := \text{Cholesky}(A_{0,0})$
2 $L_{1:2,0} := A_{1:2,0} L_{0,0}^{-H}$
3 $A_{1:2,1:2} := A_{1:2,1:2} - L_{1:2,0} L_{1:2,0}^H$
4 $L_{1,1} := \text{Cholesky}(A_{1,1})$
5 $L_{2,1} := A_{2,1} L_{1,1}^{-H}$
6 $A_{2,2} := A_{2,2} - L_{2,1} L_{2,1}^H$
7 $L_{2,2} := \text{Cholesky}(A_{2,2})$
```

2.2 Arrowhead matrices

Now suppose that $A_{1,0} = A_{0,1}^H = 0$ so that we may write

$$A = \begin{pmatrix} A_{0,0} & 0 & A_{2,0}^H \\ 0 & A_{1,1} & A_{2,1}^H \\ A_{2,0} & A_{2,1} & A_{2,2} \end{pmatrix}. \tag{2.1}$$

We say that such a matrix has *arrowhead* [133] form.[1] This class of matrices is of interest because it allows us to easily motivate many of the basic ideas behind the multifrontal method. In particular, the fact that $A_{1,0} = 0$ allows us to decouple and reorganize many of the operations in Algorithm 2.1.

[1]We should perhaps call A a *block-arrowhead* matrix for the same reason that a block-diagonal matrix should not be called diagonal. Despite their usefulness for Hermitian eigenvalue problems, we will not make use of unblocked arrowhead matrices in this dissertation, and so we will drop the *block* qualifier.

2.2.1 Factorization

Because $A_{1,0} = 0$, it is easy to see that Step 2 of Algorithm 2.1 will compute $L_{1,0} := 0$ (as well as $L_{2,0} := A_{2,0}L_{0,0}^{-H}$), so that the Cholesky factor, L, has the block structure

$$L = \begin{pmatrix} L_{0,0} & 0 & 0 \\ 0 & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{pmatrix}. \quad (2.2)$$

Once we recognize that $A_{1,0} = L_{1,0} = 0$, we may easily modify Algorithm 2.1 into Algorithm 2.2.

Algorithm 2.2: arrowhead Cholesky

```
1 $L_{0,0} := \text{Cholesky}(A_{0,0})$
2 $L_{2,0} := A_{2,0}L_{0,0}^{-H}$
3 $A_{2,2} := A_{2,2} - L_{2,0}L_{2,0}^{H}$
4 $L_{1,1} := \text{Cholesky}(A_{1,1})$
5 $L_{2,1} := A_{2,1}L_{1,1}^{-H}$
6 $A_{2,2} := A_{2,2} - L_{2,1}L_{2,1}^{H}$
7 $L_{2,2} := \text{Cholesky}(A_{2,2})$
```

It is important to notice that Steps 4-5 of Algorithm 2.2 are independent of Steps 1-3, and thus we may execute Steps 1-2 in parallel with Steps 4-5. In addition, the vast majority of the work involved in Steps 3 and 6 lies in the matrix-matrix multiplication required for the formation of the Hermitian updates, say $U_j \equiv -L_{2,j}L_{2,j}^{H}$, $j = 0, 1$, as their subsequent addition to $A_{2,2}$ has a lower-order cost. For the sake of parallelism, it is then natural to form and store these updates for later use. The result is Algorithm 2.3.

We can now recognize that Steps 2-4 of Algorithm 2.3 are equivalent to the partial Cholesky factorization of the *original frontal matrix*

$$\hat{F}_j = \begin{pmatrix} A_{j,j} & \star \\ A_{2,j} & 0 \end{pmatrix},$$

Algorithm 2.3: reorganized arrowhead Cholesky

```
1 foreach $j \in \{0,1\}$ do
2     $L_{j,j} := \text{Cholesky}(A_{j,j})$
3     $L_{2,j} := A_{2,j} L_{j,j}^{-H}$
4     $U_j := -L_{2,j} L_{2,j}^H$
5 end
6 $A_{2,2} := A_{2,2} + U_0 + U_1$
7 $L_{2,2} := \text{Cholesky}(A_{2,2})$
```

where we have used a $\star$ to denote a quadrant that will not be accessed due to implicit symmetry (i.e., $A_{2,j} = A_{j,2}^H$). If a right-looking Cholesky factorization algorithm applied to $\hat{F}_j$ is stopped just before beginning the factorization of the bottom-right quadrant, then the resulting *factored frontal matrix* will be

$$F_j = \begin{pmatrix} L_{j,j} & \star \\ L_{2,j} & U_j \end{pmatrix}.$$

We will refer this partial-factorization process as *processing* the front. We can also define the initial state of the *root front* to be $\hat{F}_2 = A_{2,2}$, which must be updated with U_0 and U_1 and then completely factored. It is useful to interpret the root front in the same manner as the previous fronts, but with a 0×0 bottom-right quadrant. Processing the root front is then equivalent to computing its full Cholesky factorization. Algorithm 2.3 can now be concisely summarized in the language of fronts (see Algorithm 2.4).

The scheme we have just described is essentially a non-recursive version of *the multifrontal method* [41, 91], which derives its name from the fact that several fronts are simultaneously maintained. The next subsection will give an overview of how to perform triangular solves using the processed fronts, and then we will dive into the details behind applying the multifrontal method to more general classes of matrices. We also note that all of the above (and

Algorithm 2.4: "multifrontal" arrowhead Cholesky

```
1 foreach $j \in \{0,1\}$ do
2     $F_j := \hat{F}_j$
3     Process $F_j$
4 end
5 $F_2 := \hat{F}_2$
6 Add bottom-right quadrants of $F_0$ and $F_1$ onto $F_2$
7 Process $F_2$
```

proceeding) algorithms may be trivially modified to perform unpivoted LDL^T and LDL^H factorizations, which allows us to handle a significant subset of complex-symmetric and Hermitian arrowhead matrices.

2.2.2 Triangular solves

The first thing to notice is that the lower-triangular matrix L, expressed in the form of Equation (2.2), is completely represented by the top-left and bottom-left quadrants of the processed fronts, $\{F_0, F_1, F_2\}$. It is then reasonable to expect to modify dense triangular solve algorithms for applying L^{-1} and L^{-H} so that they exploit arrowhead structure and access L through the processed fronts.

Before we consider the arrowhead triangular solve

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix} := \begin{pmatrix} L_{0,0} & 0 & 0 \\ 0 & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix}, \tag{2.3}$$

let us recognize that the dense triangular solve

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix} := \begin{pmatrix} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix} \tag{2.4}$$

can be performed with Algorithm 2.5, which simply executes a matrix version of *forward substitution* (see Algorithm C.6). Clearly $L_{1,0} = 0$ implies that Step 2 of Algorithm 2.5 can be replaced with $X_2 := X_2 - L_{2,0}X_0$, and the resulting optimized algorithm is given in Algorithm 2.6. Just as we introduced temporary storage in order to decouple the updates within an arrowhead factorization, temporarily storing the update matrices $Z_j \equiv -L_{2,j}X_j$, $j = 0, 1$, allows us to effectively decouple Steps 1-2 and 3-4 within Algorithm 2.6, resulting in Algorithm 2.7.

Algorithm 2.5: unrolled 3×3 block triangular solve

```
1 $X_0 := L_{0,0}^{-1} X_0$
2 $X_{1:2} := X_{1:2} - L_{1:2,0} X_0$
3 $X_1 := L_{1,1}^{-1} X_1$
4 $X_2 := X_2 - L_{2,1} X_1$
5 $X_2 := L_{2,2}^{-1} X_2$
```

Algorithm 2.6: arrowhead solve

```
1 $X_0 := L_{0,0}^{-1} X_0$
2 $X_2 := X_2 - L_{2,0} X_0$
3 $X_1 := L_{1,1}^{-1} X_1$
4 $X_2 := X_2 - L_{2,1} X_1$
5 $X_2 := L_{2,2}^{-1} X_2$
```

Algorithm 2.7: reorganized arrowhead solve

```
1 foreach $j \in \{0, 1\}$ do
2     $X_j := L_{j,j}^{-1} X_j$
3     $Z_j := -L_{2,j} X_j$
4 end
5 $X_2 := X_2 + Z_0 + Z_1$
6 $X_2 := L_{2,2}^{-1} X_2$
```

Applying the inverse of L^H, say,

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix} := \begin{pmatrix} L_{0,0}^H & 0 & L_{2,0}^H \\ 0 & L_{1,1}^H & L_{2,1}^H \\ 0 & 0 & L_{2,2}^H \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \end{pmatrix}, \tag{2.5}$$

is an equally straight-forward modification of dense *backward substitution* (Algorithm C.11). The resulting steps are given by Algorithm 2.8 and show that, if we interpret the processed fronts, $\{F_0, F_1, F_2\}$, as a tree, where F_2 is the root and F_0 and F_1 are the leaves, then we see that applying L^{-H} requires a downward traversal of the frontal tree, while applying L^{-1} requires an upward traversal. In the following subsection we will see that, for the full multifrontal method, factorizations and forward substitution will both correspond to an upward traversal of a frontal tree, while backward substitution will correspond to a downward traversal.

Algorithm 2.8: reorganized arrowhead adjoint solve

```
  Input: $L$,$X$
  Output: $X := L^{-H}X$
1 $X_2 := L_{2,2}^{-H} X_2$
2 foreach $j \in \{0,1\}$ do
3     $X_j := X_j - L_{2,j}^H X_2$
4     $X_j := L_{j,j}^{-H} X_j$
5 end
```

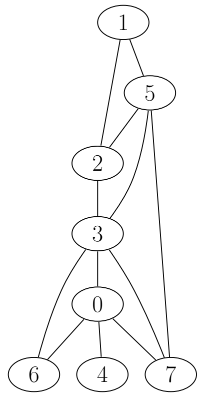
2.2.3 Separators

There is an important interpretation of arrowhead matrices in the language of *graphs*. Let us define the *graph*, $G = (V, E)$, of a matrix $A \in M_n$ to be the pairing of a set of n *vertices*, $V = \{0, 1, \ldots, n-1\}$, and a set of *edges*, $E \subset V \times V$, such that the graph G contains an edge $(i, j) \in E$ if and only if the (i, j) entry of A is nonzero (it is therefore common to associate vertex j with the j'th diagonal entry of A). Because $A_{1,0} = A_{0,1}^H = 0$ within a Hermitian 3×3 arrowhead matrix, we see that no edges connect the vertices associated with the diagonal entries of $A_{0,0}$ and $A_{1,1}$, and so it is natural to say that the remaining vertices (those of $A_{2,2}$) *separate* the graph associated with the matrix A.

Example 2.2.1. An example of a graph associated with a 8×8 symmetric sparse matrix is shown in Figure 2.1. In its current state, this matrix is

$$\begin{pmatrix}
\textcircled{0} & & & X & X & & X & X \\
 & \textcircled{1} & X & & & X & & \\
 & X & \textcircled{2} & X & & X & & \\
X & & X & \textcircled{3} & & & X & X \\
X & & & & \textcircled{4} & & & \\
 & X & X & & & \textcircled{5} & & X \\
X & & & X & & & \textcircled{6} & \\
X & & & X & & X & & \textcircled{7}
\end{pmatrix}$$

(a) symmetric sparsity pattern



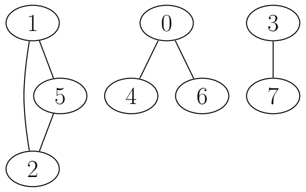
(b) associated graph

Figure 2.1: An 8×8 structurally-symmetric sparse matrix and its associated graph

clearly not in arrowhead form, but it turns out that investigating its graph leads to a way to permute the matrix into the desired form: If we clipped all external connections to vertices {3,7}, we would end up with the graph shown in Figure 2.2b. It should be clear from the clipped graph that the remaining set of vertices can be grouped into two subsets, $V_0 = \{1, 2, 5\}$ and $V_1 = \{0, 4, 6\}$, which do not directly interact; we therefore refer to vertices $V_2 = \{3, 7\}$ as a *separator*. If we now reorder the sparse matrix so that vertex set V_0 appears

$$\left(\begin{array}{ccc|ccc|cc}
\textcircled{1} & X & X & & & & & \\
X & \textcircled{2} & X & & & & X & \\
X & X & \textcircled{5} & & & & & X \\
\hline
& & & \textcircled{0} & X & X & X & X \\
& & & X & \textcircled{4} & & & \\
& & & X & & \textcircled{6} & X & \\
\hline
& X & & X & & X & \textcircled{3} & X \\
& & X & X & & & X & \textcircled{7}
\end{array}\right)$$

(a) reordered sparsity pattern



(b) clipped graph

Figure 2.2: (a) The arrowhead sparsity pattern of an 8×8 matrix reordered via graph partitioning and (b) the associated graph with the connections to the separator clipped

first, followed by V_1, and finally the separator V_2, then the result, shown in Figure 2.2a, is in arrowhead form.

Remark 2.2.1. The fact that the top-left and bottom-right blocks of the matrix in Figure 2.2a are dense corresponds to their associated sets of vertices, V_0 and V_2, being fully connected subsets of the induced subgraph.

Definition 2.2.1 (clique)**.** A subset of vertices of a graph such that every vertex in the subset is directly connected to every other vertex in the subset is referred to as a *clique*.

Definition 2.2.2 (induced subgraph)**.** Given a graph $G = (V, E)$ and a subset of vertices, say $U \subset V$, the *induced subgraph* $G|_U = (U, E|_U)$, the restriction of the graph G onto the vertex set U, is defined by setting

$$E|_U = \{\{v_i, v_j\} \in E : v_i, v_j \in U\}.$$

Definition 2.2.3 (complete graph)**.** A graph $G = (V, E)$ is called *complete*, or *fully connected*, if every vertex is directly connected to every other vertex.

Remark 2.2.2. We can now exercise our new terminology: the vertex subsets V_0 and V_2 from the previous example are both cliques, and thus their induced subgraphs, $G|_{V_0}$ and $G|_{V_2}$, are complete/fully-connected.

Definition 2.2.4 (separator)**.** We say that a vertex subset $V_2 \subset V$ of a graph $G = (V, E)$ *separates* the graph G into vertex subsets V_0 and V_1 if the induced graph $G|_{V \backslash V_2} = (V \backslash V_2, E|_{V \backslash V_2})$ is the union of two disconnected sets of vertices, i.e.:

1. $V \backslash V_2 = V_0 \cup V_1$, and
2. $\big\{\{v_0, v_1\} \in E|_{V \backslash V_2} : v_0 \in V_0, v_1 \in V_1\big\} = \emptyset$.

We will delay discussion of the important problem of how to compute graph partitions until much later in this chapter. For now, it will suffice to mention that it is an *NP-complete* problem [53] that is usually attacked with multi-level heuristics [69, 70, 81].

2.3 Introduction to multifrontal methods

It should not be surprising that our next step is to recurse on the graph partitioning approach described in the previous section. That is, given a partitioning $V = V_0 \cup V_1 \cup V_2$, where V_2 separates V_0 and V_1, *nested dissection* [54] recursively partitions the restricted graphs $G|_{V_0}$ and $G|_{V_1}$ in order to expose a recursive arrowhead structure (see Figure 2.3 for a two-level example).

If we now revisit Algorithm 2.3, then, for a two-level arrowhead matrix, the dense Cholesky factorization in step 2 should be a recursive call to another arrowhead Cholesky, and step 3, which requires an application of the inverse of the factored matrix, should use the arrowhead solve algorithms, i.e.,

$$\begin{pmatrix} A_{0,0} & & A_{0,2} & & & & A_{0,6} \\ & A_{1,1} & A_{1,2} & & & & A_{1,6} \\ A_{2,0} & A_{2,1} & A_{2,2} & & & & A_{2,6} \\ & & & A_{3,3} & & A_{3,5} & A_{3,6} \\ & & & & A_{4,4} & A_{4,5} & A_{4,6} \\ & & & A_{5,3} & A_{5,4} & A_{5,5} & A_{5,6} \\ A_{6,0} & A_{6,1} & A_{6,2} & A_{6,3} & A_{6,4} & A_{6,5} & A_{6,6} \end{pmatrix} \quad \begin{pmatrix} L_{0,0} & & & & & & \\ & L_{1,1} & & & & & \\ L_{2,0} & L_{2,1} & L_{2,2} & & & & \\ & & & L_{3,3} & & & \\ & & & & L_{4,4} & & \\ & & & L_{5,3} & L_{5,4} & L_{5,5} & \\ L_{6,0} & L_{6,1} & L_{6,2} & L_{6,3} & L_{6,4} & L_{6,5} & L_{6,6} \end{pmatrix}$$

Figure 2.3: A two-level arrowhead matrix and its Cholesky factor, L

Algorithms 2.7 and 2.8. It turns out that it is also possible to exploit sparsity *within* the off-diagonal blocks, but it will help to introduce more notation before tackling this issue. Note that our following discussions will assume that an $n \times n$ Hermitian positive-definite matrix A has already been reordered in some beneficial manner, typically via nested dissection.

2.3.1 Elimination forests

Definition 2.3.1 (supernode [11, 13, 91])**.** For any $n \times n$ Hermitian positive-definite matrix A, if the n vertices are partitioned into m contiguous subsets, say $\mathcal{D} = (\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{m-1})$, such that $j < k$ implies that every member of $\mathcal{D}_j$ is less than every member of $\mathcal{D}_k$, then each set $\mathcal{D}_i$ is referred to as a (relaxed) *supernode*.[2]

Remark 2.3.1. If possible, these supernodes should be chosen such that each corresponding lower-triangular diagonal block of the Cholesky factor of A, say $L(\mathcal{D}_s, \mathcal{D}_s)$, is sufficiently dense. We will frequently refer to supernodes by their indices, e.g., by speaking of "supernode s" rather than of "supernode $\mathcal{D}_s$". In all of the calculations performed within this dissertation, these supernodes

[2]The usual graph-theoretic definition of a supernode is much more restrictive, as the subset must be a clique whose edges satisfy a certain property. We must therefore make it clear that our *relaxed* [11] definition is different.

correspond to the degrees of freedom of the separators produced during nested dissection. Please glance ahead to Figures 2.5 and 2.6 for a visual interpretation of supernodes (and their associated elimination tree, which will soon be introduced).

Definition 2.3.2 (original structure)**.** The (lower) *original structure* of supernode s, denoted by $\hat{\mathcal{L}}_s$, is the set of all indices of rows of A which have nonzero entries directly below the diagonal block $A(\mathcal{D}_s, \mathcal{D}_s)$. We will also define $\hat{\mathcal{L}}$ to be the tuple of the original structures, that is, $\hat{\mathcal{L}} = (\hat{\mathcal{L}}_0, \hat{\mathcal{L}}_1, \ldots, \hat{\mathcal{L}}_{m-1})$.

Definition 2.3.3 (factored structure)**.** If we use the notation $\mathcal{D}_{0:s} = \cup_{0\le j\le s}\mathcal{D}_j$, then the (lower) *factored structure* of supernode s can be inductively defined as

$$\mathcal{L}_s = \hat{\mathcal{L}}_s \cup \bigcup_{\substack{0\le j<s \\ \mathcal{L}_j\cap\mathcal{D}_s\neq\emptyset}} \mathcal{L}_j \setminus \mathcal{D}_{0:s}. \tag{2.6}$$

Each contribution in the right term, say $\mathcal{L}_j \setminus \mathcal{D}_{0:s}$, is meant to model the effects of an outer-product update

$$A(\mathcal{L}_j, \mathcal{L}_j) := A(\mathcal{L}_j, \mathcal{L}_j) - L(\mathcal{L}_j, \mathcal{D}_j)L(\mathcal{L}_j, \mathcal{D}_j)^H$$

on the portion of A below supernode s, and clearly the columns with indices $\mathcal{D}_s$ can only be effected when $\mathcal{L}_j$ intersects $\mathcal{D}_s$.

We will also define $\mathcal{L}$ to be the tuple of the factored structures, i.e., $\mathcal{L} = (\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_{m-1})$.

Remark 2.3.2. When the supernodes each consist of a single vertex, this definition yields the precise nonzero structure of the Cholesky factor of A, assuming that no exact numerical cancellation took place. When the supernodes contain multiple vertices, this formula is meant to return a relatively tight superset of the true nonzero structure.

Definition 2.3.4 (ancestors, parent, c.f. [112])**.** We may define the set of *ancestors* of supernode s as all supernodes which have a nonzero intersection with the factored structure of supernode s, that is, supernodes

$$\mathcal{A}(s) = \{j : \mathcal{L}_s \cap \mathcal{D}_j \neq \emptyset\}. \tag{2.7}$$

When this set is non-empty, we may define the *parent* of supernode s as

$$\text{parent}(s) = \min \mathcal{A}(s). \tag{2.8}$$

Definition 2.3.5 (descendants, children)**.** The *descendants* of a supernode are simply those which have a particular supernode as an ancestor, i.e.,

$$\mathcal{A}^{-1}(s) = \{j : s \in \mathcal{A}(j)\} = \{j : \mathcal{L}_j \cap \mathcal{D}_s \neq \emptyset\}. \tag{2.9}$$

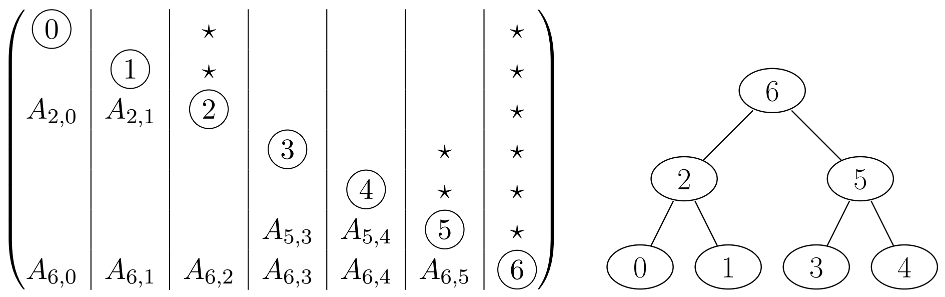
We also say that the *children* of supernode s are the supernodes with $\mathcal{D}_s$ as their parent, i.e.,

$$\mathcal{C}(s) = \text{parent}^{-1}(s) = \{c : \text{parent}(c) = s\}. \tag{2.10}$$

Definition 2.3.6 (elimination forest, c.f. [97, 112])**.** The structure we have been describing is extremely important to multifrontal methods and was first formalized in [112] in the context of supernodes composed of single vertices, or, more simply, *nodes*. Simply connecting each supernode with its parent supernode, when it exists, implies a graph, and parent$(s) > s$ further implies that this graph is actually a collection of trees. This collection of trees is known as the *elimination forest*, which we will denote by $\mathcal{E}(A, \mathcal{D})$, and it can be used to guide a multifrontal factorization.

Remark 2.3.3. The elimination forest, which becomes an *elimination tree* when the underlying matrix is irreducible, will be at the heart of our further discussions of the multifrontal method. We will begin by showing that a much simpler formula for the factored structure can be constructed if we think in terms of the elimination forest.

Example 2.3.1. A non-trivial elimination tree for a two-level arrowhead matrix is shown in Figure 2.4.



(a) two-level arrowhead matrix (b) elimination tree

Figure 2.4: (a) A symmetric two-level arrowhead matrix and (b) its elimination tree

Proposition 2.3.1 (optimal symbolic factorization [55]). *The factored structure of a supernode s, $\mathcal{L}_s$, can be directly formed from the original structure of supernode s, $\hat{\mathcal{L}}_s$, and the factored structure of its children. In particular,*

$$\mathcal{L}_s = \hat{\mathcal{L}}_s \cup \bigcup_{c \in \mathcal{C}(s)} \mathcal{L}_c \setminus \mathcal{D}_s. \tag{2.11}$$

This result immediately implies Algorithm 2.9.

Proof. By definition of the descendants of supernode s, we may rewrite Equation (2.3.1) as

$$\mathcal{L}_s = \hat{\mathcal{L}}_s \cup \bigcup_{j \in \mathcal{A}^{-1}(s)} \mathcal{L}_j \setminus \mathcal{D}_{0:s}.$$

Now consider recursively defining the related sets

$$\mathcal{G}_s = \hat{\mathcal{L}}_s \cup \bigcup_{j \in \mathcal{A}^{-1}(s)} \mathcal{G}_j,$$

which satisfy the property that $j \in \mathcal{A}^{-1}(c) \implies \mathcal{G}_j \subset \mathcal{G}_c$, and thus

$$\bigcup_{j \in \mathcal{A}^{-1}(s)} \mathcal{G}_j = \bigcup_{c \in \mathcal{C}(s)} \mathcal{G}_c.$$

On the other hand, $\mathcal{L}_s = \mathcal{G}_s \setminus \mathcal{D}_{0:s}$, so we may write

$$\mathcal{L}_s = \hat{\mathcal{L}}_s \cup \bigcup_{c \in \mathcal{C}(s)} \mathcal{L}_c \setminus \mathcal{D}_{0:s}.$$

We now have only to show that, for each $c \in \mathcal{C}(s)$, $\mathcal{L}_c \setminus \mathcal{D}_{0:s} = \mathcal{L}_c \setminus \mathcal{D}_s$. But this is equivalent to the requirement that, for each j such that $c < j < s$, $\mathcal{L}_c \cap \mathcal{D}_j = \emptyset$, which follows from the fact that $s = \text{parent}(c)$. □

Algorithm 2.9: Optimal symbolic factorization

```
  Input: supernodes, $\mathcal{D}$, initial structure, $\hat{\mathcal{L}}$, and root node, $s$
  Output: the factored structure, $\mathcal{L}$
1 foreach $c \in \mathcal{C}(s)$ do Recurse($\mathcal{D}$,$\hat{\mathcal{L}}$,$c$)
2 $\mathcal{L}_s := \hat{\mathcal{L}}_s \cup \bigcup_{c \in \mathcal{C}(s)} \mathcal{L}_c \setminus \mathcal{D}_s$
```

2.3.2 Factorization

When we first introduced the notion of structure, we recognized that an update $A(\mathcal{L}_j, \mathcal{L}_j) := A(\mathcal{L}_j, \mathcal{L}_j) - L(\mathcal{L}_j, \mathcal{D}_j) L(\mathcal{L}_j, \mathcal{D}_j)^H$ will only effect the columns assigned to supernode s if $\mathcal{L}_j \cap \mathcal{D}_s \neq \emptyset$, and we can now recognize this as equivalent to supernode s being an ancestor of supernode j. Thus, a supernode's diagonal block will be ready for factorization as soon as each of its descendants' updates has been applied, and so a factorization algorithm must work "up" the elimination forest. Since each elimination tree is independent, from now on we will assume that all matrices are irreducible, with the understanding that the following algorithms should be run on each tree in the elimination forest.

Algorithm 2.10 is simply a blocked right-looking Cholesky factorization (where the blocksize of the j'th step is the size of the j'th supernode) which makes use of the elimination tree in order to expose different possible execution orders (but unfortunately no trivial parallelism). Just as we reorganized Algorithm 2.2 into Algorithm 2.3, we can also expose latent parallelism in Algorithm 2.10 by storing temporary copies of update matrices. The key difference is that, because a particular supernode's portion of the matrix will be effected by all of its descendants update matrices, it is beneficial to take updates which came from grandchildren, great-grandchildren, etc., and to add them into the update matrices of the direct descendants. This process is justified by Proposition 2.3.1 and demonstrated by Algorithm 2.11, which makes use of the so-called *extend-add operator* [91], $\updownarrow\!\!\!\!\leftrightarrow$, to represent the process of adding the entries of the update matrix U_c into the appropriate rows and columns of a larger matrix. In particular, U_c is a square matrix meant to update the $(\mathcal{L}_c, \mathcal{L}_c)$ submatrix of A, and $\mathcal{L}_c \subset \mathcal{D}_s \cup \mathcal{L}_s$ implies that U_c can be extended by zero into a matrix meant to update the $(\mathcal{D}_s \cup \mathcal{L}_s, \mathcal{D}_s \cup \mathcal{L}_s)$ submatrix of A. Thus, in this context, $\updownarrow\!\!\!\!\leftrightarrow$ can be interpreted as extending its right argument by zero and then performing standard matrix addition.

Algorithm 2.11 is actually the classical multifrontal Cholesky factorization algorithm in disguise. If, instead of working directly with the matrix A, we instead initialize a *frontal tree*, say $\mathcal{F}(A, \mathcal{D})$, which associates each node s

Algorithm 2.10: Cholesky factorization via an elimination tree

```
    Input: HPD matrix, $A$, supernodes, $\mathcal{D}$, and root node, $s$
    Output: a lower triangular matrix, $L$, such that $A = LL^H$
1   foreach $c \in \mathcal{C}(s)$ do  Recurse($A$,$\mathcal{D}$,$c$)
2   $L(\mathcal{D}_s, \mathcal{D}_s) :=$ Cholesky$(A(\mathcal{D}_s, \mathcal{D}_s))$
3   $L(\mathcal{L}_s, \mathcal{D}_s) := A(\mathcal{L}_s, \mathcal{D}_s) L(\mathcal{D}_s, \mathcal{D}_s)^{-H}$
4   $A(\mathcal{L}_s, \mathcal{L}_s) := A(\mathcal{L}_s, \mathcal{L}_s) - L(\mathcal{L}_s, \mathcal{D}_s) L(\mathcal{L}_s, \mathcal{D}_s)^H$
```

Algorithm 2.11: Verbose multifrontal Cholesky factorization

```
    Input: HPD matrix, $A$, elimination tree, $\mathcal{E}$, and root node, $s$
    Output: a lower triangular matrix, $L$, such that $A = LL^H$
1   foreach $c \in \mathcal{C}(s)$ do  Recurse($A$,$\mathcal{E}$,$c$)
2   $U_s :=$ zeros$(|\mathcal{L}_s|, |\mathcal{L}_s|)$
3   foreach $c \in \mathcal{C}(s)$ do
        $\begin{pmatrix} A(\mathcal{D}_s, \mathcal{D}_s) & A(\mathcal{D}_s, \mathcal{L}_s) \\ A(\mathcal{L}_s, \mathcal{D}_s) & U_s \end{pmatrix} := \begin{pmatrix} A(\mathcal{D}_s, \mathcal{D}_s) & A(\mathcal{D}_s, \mathcal{L}_s) \\ A(\mathcal{L}_s, \mathcal{D}_s) & U_s \end{pmatrix} \oplus U_c$
4   $L(\mathcal{D}_s, \mathcal{D}_s) :=$ Cholesky$(A(\mathcal{D}_s, \mathcal{D}_s))$
5   $L(\mathcal{L}_s, \mathcal{D}_s) := A(\mathcal{L}_s, \mathcal{D}_s) L(\mathcal{D}_s, \mathcal{D}_s)^{-H}$
6   $U_s := U_s - L(\mathcal{L}_s, \mathcal{D}_s) L(\mathcal{L}_s, \mathcal{D}_s)^H$
```

in the elimination tree with the *original frontal matrix*

$$\hat{F}_s = \begin{pmatrix} A(\mathcal{D}_s,\mathcal{D}_s) & A(\mathcal{D}_s,\mathcal{L}_s) \\ A(\mathcal{L}_s,\mathcal{D}_s) & 0 \end{pmatrix}, \tag{2.12}$$

then we can use the same language as Algorithm 2.4 to condense Algorithm 2.11 into Algorithm 2.12, which produces the *factored frontal tree*, $\mathcal{F}$, with each front F_s equal to

$$F_s = \begin{pmatrix} L(\mathcal{D}_s,\mathcal{D}_s) & \star \\ L(\mathcal{L}_s,\mathcal{D}_s) & \star \end{pmatrix}, \tag{2.13}$$

where the irrelevant quadrants have been marked with a $\star$.[3]

Algorithm 2.12: Multifrontal Cholesky factorization

```
   Input: original frontal tree, $\hat{\mathcal{F}}$, and root node, $s$
   Output: factored frontal tree, $\mathcal{F}$
1  foreach $c \in \mathcal{C}(s)$ do  Recurse($\hat{\mathcal{F}}$,$c$)
2  $F_s := \hat{F}_s$
3  foreach $c \in \mathcal{C}(s)$ do  $F_s := F_s \Updownarrow\!\!\!\Leftrightarrow U_c$
4  Process $F_s$
```

2.3.3 Triangular solves

We will now suppose that a multifrontal factorization of our HPD matrix A has already been performed and that the factored frontal tree $\mathcal{F}$ is available. Since each factored front F_s is simply a compact representation of the nonzero entries within supernode s's set of columns of the Cholesky factor L, we will now introduce analogues of Algorithms 2.7 and 2.8 which apply to arbitrary frontal trees. Just as with multifrontal factorization, the key idea is to accumulate updates through several generations of an elimination tree so

[3]In fact, a practical implementation will free the memory required for these quadrants the last time they are used within the multifrontal factorization.

that each supernode must only interact with its children and its parent (when it exists). We will see that multifrontal lower triangular solves ($X := L^{-1}X$) work up the elimination tree making use of extend-add operations, but the adjoint operation ($X := L^{-H}X$) works *down* the elimination tree and requires more care. We will begin by explaining the simple case and will then move on to adjoint solves.

Consider performing the triangular solve $X := L^{-1}X$, where the lower triangular matrix is the Cholesky factorization of a two-level arrowhead matrix (see the right side of Figure 2.3). It is then natural to conformally partition X to express the solve as the operation

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix} := \begin{pmatrix} L_{0,0} & & & & & & \\ & L_{1,1} & & & & & \\ L_{2,0} & L_{2,1} & L_{2,2} & & & & \\ & & & L_{3,3} & & & \\ & & & & L_{4,4} & & \\ & & & L_{5,3} & L_{5,4} & L_{5,5} & \\ L_{6,0} & L_{6,1} & L_{6,2} & L_{6,3} & L_{6,4} & L_{6,5} & L_{6,6} \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix}.$$

It should be apparent that, for any block row s of L which is only nonzero at its diagonal block, $L_{s,s}$, the solution for that block row may be found by simply setting $X_s := L_{s,s}^{-1} X_s$. Since this condition on block row s holds precisely when supernode s has no descendants in the elimination tree (such a supernode is called a *leaf* of the tree), we see that a multifrontal lower-triangular solve can start working from all of the leaves of the elimination tree at once rather than just the first set of degrees of freedom as in dense forward substitution.

Once the solution X_ℓ has been computed for each leaf ℓ in the elimination tree, each set of degrees of freedom can be eliminated via the update

$$X(\mathcal{L}_\ell, :) := X(\mathcal{L}_\ell, :) - L(\mathcal{L}_\ell, \mathcal{D}_\ell) X(\mathcal{D}_\ell, \mathcal{D}_\ell),$$

which we see, by definition, only effects the portions of X assigned to the ancestors of leaf ℓ. As was mentioned earlier, our goal is to find a way to accumulate these updates up the elimination tree so that, for instance, when we reach the root node, its updates from the entire rest of the tree have all been merged into the update matrices of its children. Such an approach is demonstrated by Algorithm 2.13, which is quite similar to the verbose description of multifrontal factorization in Algorithm 2.11, which can likewise be condensed using the language of fronts. In particular, if we introduce a *right-hand side tree*, say $\mathcal{Y}$, which assigns to supernode s the matrix

$$Y_s = \begin{pmatrix} X(\mathcal{D}_s,:) \\ \text{zeros}(|\mathcal{L}_s|,k) \end{pmatrix},$$

where k is the number of columns of X, then we can express the solve using the compact language of Algorithm 2.14. The term *trapezoidal elimination* [61] simply refers to viewing the last two steps of Algorithm 2.13 as a partial elimination process with the matrices

$$F_s = \begin{pmatrix} L(\mathcal{D}_s,\mathcal{D}_s) & \star \\ L(\mathcal{L}_s,\mathcal{D}_s) & \star \end{pmatrix}, \quad \text{and} \quad Y_s = \begin{pmatrix} X(\mathcal{D}_s,:) \\ Z_s \end{pmatrix}.$$

Notice that the nonzero structure of the left half of F_s is trapezoidal: $L(\mathcal{D}_s,\mathcal{D}_s)$ is lower-triangular and $L(\mathcal{L}_s,\mathcal{D}_s)$ is, in general, dense.

Now consider the operation, $X := L^{-H}X$, which, for a two-level arrowhead matrix, can written as

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix} := \begin{pmatrix} L_{0,0}^H & & L_{2,0}^H & & & & L_{6,0}^H \\ & L_{1,1}^H & L_{2,1}^H & & & & L_{6,1}^H \\ & & L_{2,2}^H & & & & L_{6,2}^H \\ & & & L_{3,3}^H & & L_{5,3}^H & L_{6,3}^H \\ & & & & L_{4,4}^H & L_{5,4}^H & L_{6,4}^H \\ & & & & & L_{5,5}^H & L_{6,5}^H \\ & & & & & & L_{6,6}^H \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{pmatrix}.$$

```
Algorithm 2.13: Verbose multifrontal lower solve
    Input: lower triangular matrix, $L$, right-hand side matrix of
           width $k$, $X$, elimination tree, $\mathcal{E}$, and root node, $s$
    Output: $X := L^{-1}X$
  1 foreach $c \in \mathcal{C}(s)$ do Recurse($L$,$X$,$\mathcal{E}$,$c$)
  2 $Z_s :=$ zeros($|\mathcal{L}_s|, k$)
  3 foreach $c \in \mathcal{C}(s)$ do $\begin{pmatrix} X(\mathcal{D}_s,:) \\ Z_s \end{pmatrix} := \begin{pmatrix} X(\mathcal{D}_s,:) \\ Z_s \end{pmatrix} \overset{\updownarrow}{\leftrightarrow} Z_c$
  4 $X(\mathcal{D}_s,:) := L(\mathcal{D}_s,\mathcal{D}_s)^{-1}X(\mathcal{D}_s,:)$
  5 $Z_s := Z_s - L(\mathcal{L}_s,\mathcal{D}_s)X(\mathcal{D}_s,:)$
```

```
Algorithm 2.14: Multifrontal lower solve
    Input: frontal tree, $\mathcal{F}$, right-hand side tree, $\mathcal{Y}$, and root node, $s$
  1 foreach $c \in \mathcal{C}(s)$ do Recurse($\mathcal{F}$,$\mathcal{Y}$,$c$)
  2 foreach $c \in \mathcal{C}(s)$ do $Y_s := Y_s \overset{\updownarrow}{\leftrightarrow} Z_c$
  3 Trapezoidal forward elimination of $Y_s$ with $F_s$
```

This time, the only part of the solution which we can immediately compute corresponds to the root node: $X_s := L_{s,s}^{-H} X_s$ (in this case, $s = 6$). These degrees of freedom can then be eliminated from each of the descendants $d \in \mathcal{A}^{-1}(s)$ via the update $X_d := X_d - L_{s,d}^H X_s$, and we can recognize that the sparsity in L implies that the update may be simplified to

$$X(\mathcal{D}_d,:) := X(\mathcal{D}_d,:) - L(\mathcal{L}_d \cap \mathcal{D}_s, \mathcal{D}_d)^H X(\mathcal{L}_d \cap \mathcal{D}_s,:).$$

One possible approach would be to update all of the descendants in this manner and then to recurse on each of the children of the root node.

On the other hand, suppose that we were interest interested in simultaneously applying all of the updates to a particular supernode from the entire set of its ancestors, i.e.,

$$X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:) - \sum_{a \in \mathcal{A}(s)} L(\mathcal{L}_s \cap \mathcal{D}_a, \mathcal{D}_s)^H X(\mathcal{L}_s \cap \mathcal{D}_a,:).$$

Since the set of ancestors of node s is precisely equal to the set of supernodes which intersect its lower structure, this update simplifies to

$$X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:) - L(\mathcal{L}_s,\mathcal{D}_s)^H X(\mathcal{L}_s,:),$$

which implies the top-down approach of Algorithm 2.15. The downside of this approach is that there is no clear notion of data locality: each supernode continually indexes out of the global right-hand side matrix, X.

Algorithm 2.15: Verbose multifrontal lower adjoint solve

```
    Input: lower triangular matrix, $L$, right-hand sides, $X$,
           elimination tree, $\mathcal{E}$, and root node, $s$
    Output: $X := L^{-H}X$
1   if $\mathcal{A}(s) \neq \emptyset$ then $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:) - L(\mathcal{L}_s,\mathcal{D}_s)^H X(\mathcal{L}_s,:)$
2   $X(\mathcal{D}_s,:) := L(\mathcal{D}_s,\mathcal{D}_s)^{-H} X(\mathcal{D}_s,:)$
3   foreach $c \in \mathcal{C}(s)$ do Recurse($L$,$X$,$\mathcal{E}$,$c$)
```

On the other hand, if the parent of supernode s is supernode p, then $\mathcal{L}_s \subset \mathcal{D}_p \cup \mathcal{L}_p$, and so if each supernode s pulls $X(\mathcal{L}_s,:)$ down from its parent and then performs all necessary computation on $X(\mathcal{D}_s,:)$, it will have locally computed all of the pieces of X needed for its children to repeat the process. A recursive procedure based on this idea is given in Algorithm 2.16, where initially

$$Y_s = \begin{pmatrix} X(\mathcal{D}_s,:) \\ \text{zeros}(|\mathcal{L}_s|,k) \end{pmatrix},$$

and Extract(Y_s, Y_p) refers to the process of extracting $X(\mathcal{L}_s,:)$ out of $Y_p = X(\mathcal{D}_p \cup \mathcal{L}_p,:)$ and placing it into the bottom section of Y_s.

2.3.4 Computational complexity

We have now derived the main algorithms required for solving sparse Hermitian positive-definite linear systems with the multifrontal algorithm (with

Algorithm 2.16: Multifrontal lower adjoint solve

```
   Input: frontal tree, $\mathcal{F}$, right-hand side tree, $\mathcal{Y}$, and root node, $s$
1  if $\mathcal{A}(s) \neq \emptyset$ then Extract($Y_s, Y_{\text{parent}(s)}$)
2  Trapezoidal backward elimination of $Y_s$ with $F_s^H$
3  foreach $c \in \mathcal{C}(s)$ do Recurse($\mathcal{F}$,$\mathcal{Y}$,$c$)
```

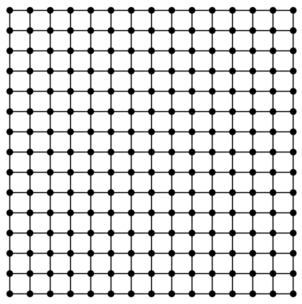


Figure 2.5: A 15×15 grid graph. Each node is connected to its nearest horizontal and vertical neighbors.

the notable exception of graph partitioning approaches, which we will discuss at the end of the chapter) and are ready to begin discussing storage and work requirements. We will begin by mirroring George's argument [54] that, if the graph of a Hermitian positive-definite matrix is an $n \times n$ *grid graph* [90] (see Figure 2.5), then nested dissection may be combined with the multifrontal algorithm in order to factor the sparse matrix with only $O(n^3)$ work (as opposed to the $O(n^6)$ work required for a dense factorization). This sort of sparsity pattern frequently arises for two-dimensional finite-difference equations, and an example of an elimination tree for a 15×15 grid is shown in Figure 2.6.

While the separators chosen in Figure 2.6 are what we would choose in practice, in order to keep our analysis simple we will choose so-called *cross separators* [61], which are approximately twice as large as necessary but do not change the asymptotics of the method. Figure 2.7 shows an elimination

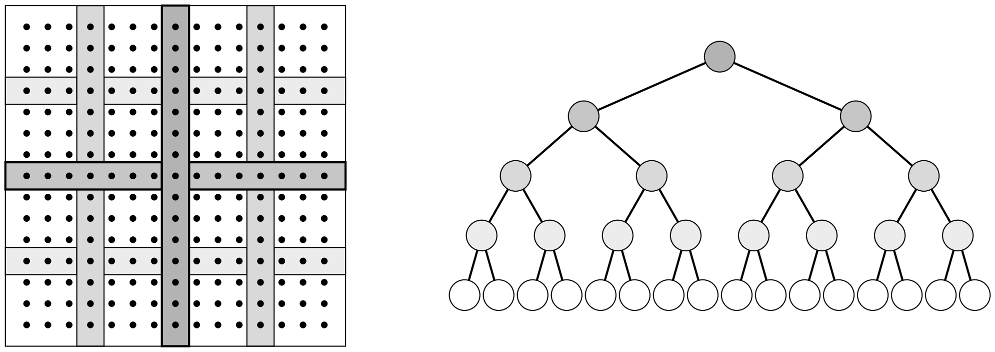


Figure 2.6: A separator-based elimination tree (right) for a 15×15 grid graph (left)

tree for a 15×15 grid graph using cross separators. In particular, we will be considering $n \times n$ grids where n is one less than a power of two, which result in the cross separators on level ℓ of the tree (counting down from zero from the root node) having heights and widths of

$$w_\ell = \frac{n+1}{2^\ell} - 1.$$

We thus know that, if $s \in \text{level}(\ell)$ (that is, supernode s is on the ℓ'th level of the elimination tree), then $|\mathcal{D}_s| = 2w_l - 1$. We can also easily bound $|\mathcal{L}_s|$ from above by $4w_\ell$ (the maximum number of nodes that can border the $w_\ell \times w_\ell$ box spanned by the cross separator and its descendants).

We can easily see that the work required for processing front F_s is approximately equal to

$$\text{Work}(F_s) = \frac{1}{3}|\mathcal{D}_s|^3 + |\mathcal{D}_s|^2|\mathcal{L}_s| + |\mathcal{D}_s||\mathcal{L}_s|^2, \tag{2.14}$$

where the first term corresponds to the Cholesky factorization of the $|\mathcal{D}_s| \times |\mathcal{D}_s|$ diagonal block, the second term corresponds to a triangular solve of the diagonal block against the $|\mathcal{L}_s| \times |\mathcal{D}_s|$ submatrix $A(\mathcal{L}_s, \mathcal{D}_s)$, and the third term

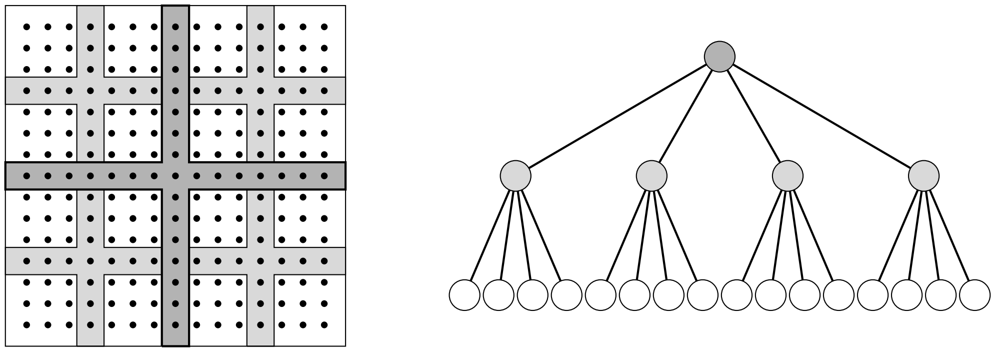


Figure 2.7: An elimination tree based on cross separators (right) of a 15×15 grid graph (left)

corresponds to a rank-$|\mathcal{D}_s|$ Hermitian update of an $|\mathcal{L}_s| \times |\mathcal{L}_s|$ matrix. We can now plug the previously discussed level-dependent values and bounds for $|\mathcal{D}_s|$ and $|\mathcal{L}_s|$ into a summation over all of the fronts to find the total work for the multifrontal factorization (ignoring the lower-order extend-add costs):

$$\begin{aligned}\text{Work}(\mathcal{F}) &\le \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{2\ell} \left(\frac{1}{3}(2w_\ell)^3 + (2w_\ell)^2(4w_\ell) + (2w_\ell)(4w_\ell)^2 \right) \\ &\le C \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{2\ell} w_\ell^3 \\ &\le C(n+1)^3 \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{-\ell} \in O(n^3).\end{aligned}$$

We have just shown that nested dissection of regular 2d grids leads only requires $O(n^3) = O(N^{3/2})$ work, where $N = n^2$ is the total number of degrees of freedom. Soon after George showed this result [54], Lipton and Tarjan generalized the approach to the entire class of *planar* graphs [90], which is simply the class of graphs which may be drawn on paper without any overlapping edges (that is, *embedded in the plane*).

Before we discuss more abstract classes of graphs, we will first extend our analysis to d-dimensional grid graphs with n vertices in each direction, for a total of $N = n^d$ vertices, which only interact with their nearest neighbors in each of the d cardinal directions. This time, the cross separators will consist of d overlapping hyperplanes of dimension $d-1$ (i.e., in 3D, the cross separators consist of three planes), and we can bound $|\mathcal{D}_s|$ from above by dw_ℓ^{d-1}, which is exact other than ignoring the overlap of the d hyperplanes. Also, a d-dimensional cube has $2d$ faces, and so our best bound for $|\mathcal{L}_s|$ is $2dw_\ell^{d-1}$. The resulting bound for d-dimensional nested dissection is

$$\begin{aligned}
\text{Work}(\mathcal{F}) &\leq \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{d\ell} \left(\frac{1}{3}(dw_\ell^{d-1})^3 + 2(dw_\ell^{d-1})^3 + 4(dw_\ell^{d-1})^3 \right) \\
&= C \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{d\ell} w_\ell^{3(d-1)} \\
&\leq C(n+1)^{3(d-1)} \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{\ell(3-2d)}.
\end{aligned}$$

When we set $d = 1$, we find that the cost is $O(n)$, as it should be for a tridiagonal matrix, when we set $d = 2$, we recover our previous result, and when we set $d = 3$ we find that the cost is $O(n^6) = O(N^2)$. We can also see that, for $d \geq 2$, the cost is $O(n^{3(d-1)})$, which is the same asymptotic complexity as a dense factorization over a root separator of size $n^{d-1} \times n^{d-1}$.

Now let us recall that, after factorization, all that must be kept from each frontal matrix F_s is its top-left and bottom-left quadrants, i.e., $L(\mathcal{D}_s, \mathcal{D}_s)$ and $L(\mathcal{L}_s, \mathcal{D}_s)$, which respectively require $\frac{1}{2}|\mathcal{D}_s|^2$ and $|\mathcal{D}_s||\mathcal{L}_s|$ units of storage. We therefore write

$$\text{Mem}(F_s) = \frac{1}{2}|\mathcal{D}_s|^2 + |\mathcal{D}_s||\mathcal{L}_s|. \qquad (2.15)$$

If we make use of the same bounds on $|\mathcal{D}_s|$ and $|\mathcal{L}_s|$ as before, then we find that, for a d-dimensional grid graph, the memory required for storing the Cholesky

d	work	storage
1	$O(N)$	$O(N)$
2	$O(N^{3/2})$	$O(N\log N)$
3	$O(N^2)$	$O(N^{4/3})$
$d \geq 3$	$O(N^{3(1-1/d)})$	$O(N^{2(1-1/d)})$

Table 2.1: Storage and work upper bounds for the multifrontal method applied to d-dimensional grid graphs with n degrees of freedom in each direction (i.e., $N = n^d$ total degrees of freedom)

factorization is

$$\begin{aligned}\mathrm{Mem}(\mathcal{F}) &\leq \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{d\ell}\left(\frac{1}{2}(dw_\ell^{d-1})^2 + 2(dw_\ell^{d-1})^2\right)\\ &= C(n+1)^{2(d-1)} \sum_{\ell=0}^{\lfloor \log_2(n) \rfloor} 2^{\ell(2-d)}.\end{aligned}$$

When we set $d = 1$, $\mathrm{Mem}(\mathcal{F})$ is $O(n)$, which agrees with the fact that the Cholesky factor should be bidiagonal, when $d = 2$, we see that the memory requirement is $O(n^2 \log n) = O(N \log N)$, for $d = 3$ it is $O(n^4) = O(N^{4/3})$, and, for arbitrary $d \geq 3$, it is $O(n^{2(d-1)})$, which is equal to the amount of space required for the storage for a $n^{d-1} \times n^{d-1}$ dense matrix. Because our multifrontal triangular solve algorithms perform $O(1)$ flops per unit of storage, the work required for the triangular solves has the same asymptotic complexity as the memory requirements of the factorization.

It should now be overwhelmingly clear that, for nested-dissection based multifrontal methods, *the* deciding factor for the cost of the multifrontal method is the minimum separator size required for bisecting the underlying graph. For one-dimensional grid graphs, these separators consist of a single vertex, for two-dimensional grid graphs, a separator can be chosen as a line of n vertices, for three dimensions the separator can be chosen as a plane of n^2 vertices, and, in d dimensions, the separator can be chosen as a hyperplane of n^{d-1} vertices. Interestingly, it was shown in [57] that, if a graph can be embedded in a surface of genus g (for example, the surface of a sphere with g *handles* attached), then a separator of size $O(\sqrt{gN} + \sqrt{N})$ may be found. This implies that d-dimensional grid graphs have separators comparable to those of graphs of genus $N^{\frac{d-2}{d}}$.

2.3.5 Graph partitioning

As was previously mentioned, graph partitioning is an NP-complete problem [53], and so practical solution techniques invariably make use of heuristics. The most successful approach to date is *multilevel graph partitioning* [69, 70], which resembles *multigrid methods* [25, 26] in that the basic workflow is to successively coarsen the original problem, directly solve the problem once it is sufficiently coarse, and then push the solution back up the hierarchy.

In the case of graphs, this coarsening process merges sets of vertices into a single vertex whose edge list is the union of the edge lists of its constituents; one might say that each step of the coarsening process constructs a new graph out of a set of supernodes for the previous graph. After the graph is sufficiently coarsened, a brute-force graph partitioning algorithm can be applied, and then, by construction, this partition implies a valid (albeit likely suboptimal) partition for the parent graph. It is then natural to attempt to *refine* the candidate partition at each step via an algorithm like *Kernighan-Lin refinement* [83], though practical implementations should make use of variants of Fiduccia and Mattheyses's improvement of KL-refinement [48], e.g., the one used by METIS [81]. How to effectively parallelize this refinement process is still an open question.

2.4 Parallel multifrontal methods

It is now time to discuss the parallelization of each of the major steps of the multifrontal solution of a sparse Hermitian positive-definite system of equations:

1. reordering the unknowns (especially via nested dissection),
2. symbolic factorization,
3. numeric factorization, and
4. triangular solves.

We will begin with the easiest step, symbolic factorization, move on to numeric factorization, spend a while discussing several options for the triangular solves, and then briefly touch on remaining challenges with respect to parallel reorderings.

Our approach to the parallelization of each of these stages (with the possible exception of the reordering phase) exploits the independence of siblings within the elimination tree in order to expose as much trivial parallelism as possible, and fine-grain parallelism is therefore only employed towards the top of the tree, usually via distributed-memory dense linear algebra algorithms. The fundamentals of this approach are due to George, Liu, and Ng, who proposed a so-called *subtree-to-subcube mapping* [56] which splits the set of processes into two disjoint teams at each junction of a binary elimination tree. The term *subcube* was used because this decomposition maps naturally

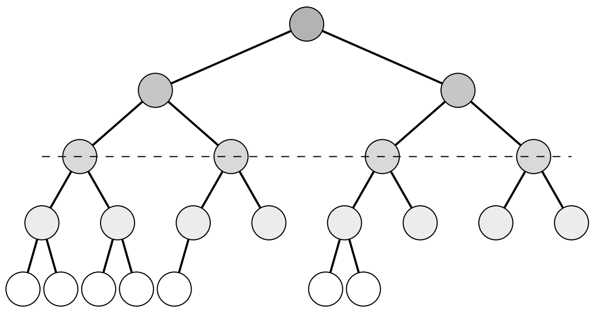


Figure 2.8: An example of a (slightly imbalanced) elimination tree distributed over four processes: the entire set of processes shares the root node, teams of two processes share each of the two children, and each process is assigned an entire subtree rooted at the dashed line.

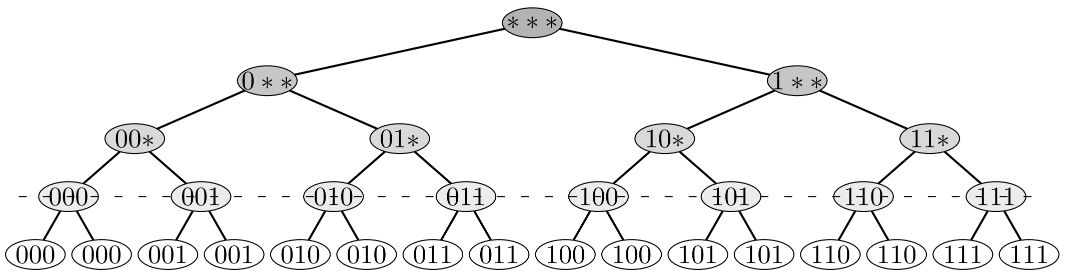


Figure 2.9: Overlay of the process ranks (in binary) of the owning subteams of each supernode from an elimination tree assigned to eight processes; a '*' is used to denote both 0 and 1, so that '$00*$' represents processes 0 and 1, '$01*$' represents processes 2 and 3, and '$* * *$' represents all eight processes.

to hypercube network topologies, but the idea trivially generalizes to general elimination forests and non-power-of-two numbers of processes, and so we will speak of *subtree-to-subteam mappings*. Figures 2.8 and 2.9 both demonstrate subtree-to-subteam mappings of an elimination tree. For the sake of simplicity, we will assume that the elimination tree is binary above the roots of the local subtrees.

2.4.1 Symbolic factorization

There is a natural way to combine subtree-to-subteam mappings with the symbolic factorization approach in Algorithm 2.9: every process begins the algorithm at the root of the tree, and, each time a set of recursions was to be launched over the children of that node, the set of processes is split into (roughly) even teams which each handle the subtree rooted at one of the children. For the sake of simplicity, *we require that the distributed portion of the tree is binary.* Thus, if p processes are available, $\lfloor \frac{p}{2} \rfloor$ processes can recurse on the first child while the remaining processes handle the second child, and, after both recursions finish, the two teams can share the factored structure of the two children. Any time the recursive routine is launched with a team consisting of a single process, we can simply revert to the sequential factorization algorithm. This approach is summarized in Algorithm 2.17, and an example of the part of the elimination tree touched by a single process is shown in Figure 2.10. From now on, when a subscript of loc is added to a tree data structure, e.g., $\mathcal{L}_{\text{loc}}$, the implication is that it is the restriction of the data structure to a particular process's portion of the tree.

2.4.2 Numeric factorization

The numeric factorization can be parallelized in a manner very similar to the symbolic factorization; the main difference is that, rather than each process performing a few local set operations, teams must work together to perform part of a distributed dense Cholesky factorization (this corresponds to *processing* the supernode, recall Algorithm 2.12). As was noted in [113] and, for the dense case, [38], two-dimensional (or higher) decompositions of the fronts are required in order to perform their partial factorizations in a scal-

Algorithm 2.17: Distributed symbolic factorization

```
Input: local original structure, $\hat{\mathcal{L}}_{\mathrm{loc}}$, local elimination tree, $\mathcal{E}_{\mathrm{loc}}$,
       root node, $s$, number of processes, $p$, and process rank, $r$
Output: local factored structure, $\mathcal{L}_{\mathrm{loc}}$
1  if $p > 1$ then
2      if $r < \lfloor \frac{p}{2} \rfloor$ then
3          Recurse($\hat{\mathcal{L}}_{\mathrm{loc}}$, $\mathcal{E}_{\mathrm{loc}}$, $c_0(s)$,$\lfloor \frac{p}{2} \rfloor$,$r$)
4          Exchange $\mathcal{L}_{c_0(s)}$ for $\mathcal{L}_{c_1(s)}$ with other team
5      else
6          Recurse($\hat{\mathcal{L}}_{\mathrm{loc}}$, $\mathcal{E}_{\mathrm{loc}}$, $c_1(s)$,$p-\lfloor \frac{p}{2} \rfloor$, $r-\lfloor \frac{p}{2} \rfloor$)
7          Exchange $\mathcal{L}_{c_1(s)}$ for $\mathcal{L}_{c_0(s)}$ with other team
8  else
9      foreach $c \in \mathcal{C}(s)$ do  Recurse($\hat{\mathcal{L}}_{\mathrm{loc}}$, $\mathcal{E}_{\mathrm{loc}}$,$c$,1,0)
10 $\mathcal{L}_s := \hat{\mathcal{L}}_s \cup \bigcup_{c \in \mathcal{C}(s)} \mathcal{L}_c \setminus \mathcal{D}_s$
```

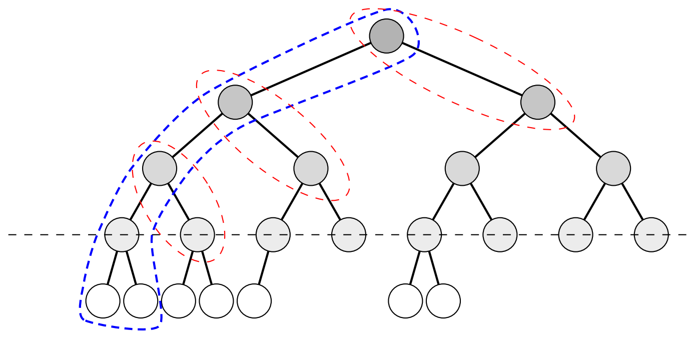


Figure 2.10: An example of the portion of a symbolic factorization to be computed by a single process (enclosed in the dashed blue line) for an elimination tree distributed over eight processes. Each step towards the top of the tree requires an interaction with the team which computed the factored structure of the other child (each of these interactions is surrounded with dashed red lines).

able manner. And, in the words of Schreiber, a scalable dense factorization of the root front is a *sine qua non* for the overall scalability of the multifrontal method. As was discussed in the previous chapter, when memory is not constrained, it is worthwhile to consider so-called *2.5D* [116] and *3D* [9, 76] dense factorizations, which replicate the distributed matrix across several teams of processes in order to lower the communication volume.

Algorithm 2.18: Distributed multifrontal factorization

```
Input: local part of 2D-distributed original frontal tree, $\hat{\mathcal{F}}^{2D}_{\text{loc}}$,
       root node, $s$, number of processes, $p$, and process rank, $r$
Output: local part of 2D-distributed factored frontal tree, $\mathcal{F}^{2D}_{\text{loc}}$
1 if $p > 1$ then
2     if $r < \lfloor \frac{p}{2} \rfloor$ then Recurse($\hat{\mathcal{F}}^{2D}_{\text{loc}}$, $c_0(s)$,$\lfloor \frac{p}{2} \rfloor$,$r$)
3     else Recurse($\hat{\mathcal{F}}^{2D}_{\text{loc}}$, $c_1(s)$,$p-\lfloor \frac{p}{2} \rfloor$, $r-\lfloor \frac{p}{2} \rfloor$)
4     $F_s := \hat{F}_s$
5     foreach $j \in \{0,1\}$ do AllToAll to add $U_{c_j}$ into $F_s$
6     Partial factorization of $F_s$ with 2D (or higher) distribution
7 else
8     Apply Algorithm 2.12 to $\hat{\mathcal{F}}^{2D}_{\text{loc}}$ beginning at node $s$
```

Although this chapter is focused on *multifrontal* approaches to solving sparse linear systems, it is important to briefly survey the wide variety of existing distributed-memory sparse-direct solvers [33]. The solvers PSPASES [80] and WSMP [62] are both descendants of the approach in [63], which is also the foundation for the approach followed in this dissertation. Another related solver is DSCPACK [103], which was the testing ground for *selective inversion* [102], an approach which we will devote one of the following subsections to. DSCPACK follows the same basic framework as PSPASES and WSMP, though it unfortunately does not make use of level 3 BLAS for processing its distributed fronts [103].

The packages SPOOLES [12] and MUMPS [3] both implement distributed-memory multifrontal methods, but the former makes use of one-dimensional distributions for all of its fronts in order to simplify partial pivoting, while the latter only makes use of two-dimensional distributions for the root front. Thus, neither approach can be expected to exhibit comparable scalability to the approach of [63]. In addition, MUMPS was originally designed to make use of a *master-slave* paradigm, and a number of artifacts still remain that obstruct its usage for truly large-scale problems (for example, MUMPS requires that right-hand sides be stored entirely on the root process at the beginning of the solution process [1]).

The remaining two commonly used packages are PaStiX [72] and `SuperLU_Dist` [87], both of which use alternatives to the multifrontal method. PaStiX couples a *column fan-in*, or *left-looking supernodal* approach [10] with an intelligent selection criteria for choosing between one-dimensional and two-dimensional dense distributions, though it appears to only be competitive with the approach of [63] for modest numbers of processors [72]. On the other hand, `SuperLU_Dist` focuses on structurally non-symmetric matrices and uses a row-pipelined approach [88] which resembles a sparse version of the High Performance Linpack algorithm [95, 126]. One potential pitfall to this approach is that the size of supernodes must be artificially restricted in order to preserve the efficiency of the parallelization: for instance, if the root separator arising from nested dissection of a 3D grid graph was treated as a single supernode, then the runtime and memory usage of the parallel scheme would at best be $O(N^2/\sqrt{p})$ and $O(N^{4/3}/\sqrt{p})$, respectively.

As was mentioned at the beginning of the chapter, the difference between the factorization approach used later in this dissertation versus that of

PSPASES and WSMP is *which* two-dimensional distribution is chosen for the fronts. While PSPASES and WSMP make use of *block-cyclic* distributions, Clique, the implementation which will be used within our parallel sweeping preconditioner, makes use of *element-wise* distributions [71, 93, 99] and does not alter the details of the data distribution so that parallel extend-add operations only require communication with a single process [63]. The major advantage Clique is its flexibility with respect to triangular solve algorithms (though a custom Kronecker-product compression scheme is also employed in a later chapter).

2.4.3 Standard triangular solves

We will now assume that a distributed factorization has already been performed and that $O(1)$ right-hand sides are ready to be solved in the form of the matrix X, which we will arrange into a right-hand side tree, $\mathcal{Y}$. Furthermore, we will also apply the subtree-to-subteam mapping to $\mathcal{Y}$ in order to determine which teams of processes share each matrix in the tree. However, we must still decide *how* each matrix should be shared between its team of processes. Let us recall that each member of $\mathcal{Y}$ has the form

$$Y_s = \begin{pmatrix} X(\mathcal{D}_s, :) \\ \text{zeros}(|\mathcal{L}_s|, k) \end{pmatrix},$$

when X has k columns.

Consider the multifrontal triangular solve described by Algorithm 2.13. It should be clear that essentially all of the work of the multifrontal triangular solve lies within Steps 6 and 7, which respectively perform dense triangular solves and a dense matrix-matrix multiplication. For now, we will consider the regime where only a small number of right-hand sides need to be solved, and so Steps 6 and 7 should temporarily be interpreted as a dense triangular solve with a single right-hand side and a dense matrix-vector multiply. *Thus, the distribution of each member of the right-hand side tree $\mathcal{Y}$ can be chosen to maximize the efficiency of the dense triangular solves and matrix-vector multiplication.*

Unfortunately, the critical path of forward/backward substitution is $O(n)$ for an $n \times n$ dense triangular matrix, and so the $O(n^2)$ work can only be effectively distributed amongst p processes, i.e., have a runtime of $O(n^2/p)$, when $p \leq n$. Thus, the per-process memory requirements for an efficient triangular solve increase proportionally with the size of the dense triangular matrix, n. This observation is consistent with the *isoefficiency* analysis of [65] and [79], which respectively analyzed pipelined distributed multifrontal triangular solves with 1D and 2D distributions of the frontal matrices (cf. [68]). For both distribution schemes, their conclusion was that, in order to maintain constant efficiency in the multifrontal triangular solves arising from 2D and 3D grid graphs, the number of processes, p, can respectively only grow as fast as $\Omega(N^{1/2})$ and $\Omega(N^{2/3})$ (where the former term holds within logarithmic factors of N). Thus, the $O(N \log N/p)$ and $O(N^{4/3}/p)$ per-process memory requirements must respectively grow at rates of $\Omega(N^{1/2})$ and $\Omega(N^{2/3})$ in order to ensure that the triangular solves remain efficient (the former term again only holds within logarithmic factors of N).

Under most circumstances, this scalability issue can be ignored, as a multifrontal triangular solve requires asymptotically less work than a multifrontal factorization (e.g., $O(N \log N)$ vs. $O(N^{3/2})$ in 2D and $O(N^{4/3})$ vs. $O(N^2)$ in 3D). Thus, as long as efficiency of the parallel triangular solve does not drop too quickly (e.g., at a rate faster than $O(N^{1/2}/\log N)$ in 2D and $O(N^{2/3})$ in

3D), the factorization can be expected to remain the dominant term. Examples of parallel multifrontal forward and backward eliminations based upon element-wise 2D frontal distributions and 1D right-hand side distributions are therefore given in Algorithms 2.19 and 2.20, though it is also reasonable to redistribute the fronts into one-dimensional distributions and to make use of multifrontal triangular solves based upon Algorithms C.7 and C.15.[4] One-dimensional distributions should be used for each right-hand side submatrix Y_s because, by assumption, there are only $O(1)$ right-hand sides, and so one-dimensional distributions must be used if the right-hand sides are to be distributed amongst $O(p)$ processes.

Algorithm 2.19: Distributed multifrontal forward substitution

```
Input: local part of 2D-distributed factored frontal tree, $\mathcal{F}^{2D}_{\rm loc}$,
       local part of 1D-distributed right-hand side tree, $\mathcal{Y}^{1D}_{\rm loc}$,
       root node, $s$, number of processes, $p$, and process rank, $r$
1 if $p > 1$ then
2     if $r < \lfloor \frac{p}{2} \rfloor$ then Recurse($\mathcal{F}^{2D}_{\rm loc}$, $\mathcal{Y}^{1D}_{\rm loc}$,$c_0(s)$, $\lfloor \frac{p}{2} \rfloor$,$r$)
3     else Recurse($\mathcal{F}^{2D}_{\rm loc}$, $\mathcal{Y}^{1D}_{\rm loc}$,$c_1(s)$, $p-\lfloor \frac{p}{2} \rfloor$, $r-\lfloor \frac{p}{2} \rfloor$)
4     foreach $j \in \{0,1\}$ do AllToAll to add $Z_{c_j}$ into $Y_s$
5     $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:)L(\mathcal{D}_s,\mathcal{D}_s)^{-1}$ via Algorithm C.8
6     $Z_s := Z_s - L(\mathcal{L}_s,\mathcal{D}_s)X(\mathcal{D}_s,:)$ via Algorithm C.1
7 else
8     Apply Algorithm 2.14 to $\mathcal{F}^{2D}_{\rm loc}$ and $\mathcal{Y}^{1D}_{\rm loc}$ beginning at node $s$
```

Note that, when more than $O(1)$ triangular solves must be performed with each factorization, efficient parallelizations *are* important. There are essentially two important situations:

[4]We will not discuss pipelined dense triangular solves since they are less practical with element-wise distributions. See [79] for a pipelined two-dimensional dense triangular solve.

Algorithm 2.20: Distributed multifrontal backward substitution

```
   Input: $\mathcal{F}^{2D}_{\text{loc}}$, $\mathcal{Y}^{1D}_{\text{loc}}$, $s$, $p$, $r$
1  if $p > 1$ then
2      if $\mathcal{A}(s) \neq \emptyset$ then AllToAll to update $Y_s$ with $Y_{\text{parent}(s)}$
3      $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:) - L(\mathcal{L}_s,\mathcal{D}_s)^H Z_s$ via Algorithm C.2
4      $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:)L(\mathcal{D}_s,\mathcal{D}_s)^{-H}$ via Algorithm C.12
5      if $r < \lfloor\frac{p}{2}\rfloor$ then Recurse($\mathcal{F}^{2D}_{\text{loc}}$, $\mathcal{Y}^{1D}_{\text{loc}}$,$c_0(s)$, $\lfloor\frac{p}{2}\rfloor$,$r$)
6      else Recurse($\mathcal{F}^{2D}_{\text{loc}}$, $\mathcal{Y}^{1D}_{\text{loc}}$,$c_1(s)$, $p-\lfloor\frac{p}{2}\rfloor$, $r-\lfloor\frac{p}{2}\rfloor$)
7  else
8      if $\mathcal{A}(s) \neq \emptyset$ then AllToAll to update $Y_s$ with $Y_{\text{parent}(s)}$
9      $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:) - L(\mathcal{L}_s,\mathcal{D}_s)^H Z_s$
10     $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:)L(\mathcal{D}_s,\mathcal{D}_s)^{-H}$
11     foreach $c \in \mathcal{C}(s)$ do Recurse($\mathcal{F}^{2D}_{\text{loc}}$, $\mathcal{Y}^{1D}_{\text{loc}}$, $c$,1,0)
```

1. many right-hand sides must be solved one at a time, and
2. many right-hand sides can be solved at once.

The following subsection addresses the first case, which is important for iterative methods which make use of sparse-direct triangular solves (usually on subproblems) as part of a preconditioner.

2.4.4 Selective inversion

An examination of the triangular solve algorithms from the previous subsection, Algorithms 2.19 and 2.20, reveals that all of the scalability problems result from the dense triangular solves against the diagonal blocks of the frontal matrices in Steps 8 and 5, respectively. The idea of *selective inversion* [102, 103] is to perform extra work between the multifrontal factorization and solution phases in order to invert the lower-triangular diagonal blocks in place, so that *the problematic distributed dense triangular solves become distributed dense matrix-vector multiplication* (see Algorithms 2.21 and 2.22).

Significantly less work is required for this inversion process than the original multifrontal factorization and it has been observed that, for moderate to large numbers of processes, the extra time spent within selective inversion is recouped after a small number of triangular solves are performed [98, 102]. Large performance improvements from making use of selective inversion are shown in Chapter 3. It is also important to note that a similar idea appeared in earlier work on out-of-core dense solvers [114], where the diagonal blocks of *dense* triangular matrices were inverted in place in order to speed up subsequent triangular solves.

Algorithm 2.21: Selective inversion

```
Input: $\mathcal{F}^{2D}_{\rm loc}$, $s$, $p$, $r$
Output: local part of 2D-distributed selectively-inverted frontal
        tree, $\mathcal{G}^{2D}_{\rm loc}$
1 $G_s := F_s$
2 if $p > 1$ then
3     Invert $L(\mathcal{D}_s, \mathcal{D}_s)$ in top-left corner of $G_s$ via Algorithm C.21
4     if $r < \lfloor \frac{p}{2} \rfloor$ then Recurse($\mathcal{F}^{2D}_{\rm loc}$, $c_0(s)$,$\lfloor \frac{p}{2} \rfloor$,$r$)
5     else Recurse($\mathcal{F}^{2D}_{\rm loc}$, $c_1(s)$,$p-\lfloor \frac{p}{2} \rfloor$, $r-\lfloor \frac{p}{2} \rfloor$)
6 else
7     foreach $d \in \mathcal{A}^{-1}(s)$ do $G_d := F_d$
```

The numerical stability of our replacement of triangular solves with an explicit calculation of the inverse followed by matrix-vector multiplication warrants at least a short discussion. Although avoiding direct inversion is perhaps the first lesson taught to budding numerical analysts, results dating back to Wilkinson [40, 132] show that, under reasonable assumptions, and with a careful calculation of the inverse, $x := \text{inv}(A)\, x$ can be as accurate as with a backwards-stable solver. Although these results hold for essentially arbitrary square matrices, we need only apply them to lower triangular matrices in

Algorithm 2.22: Distributed multifrontal forward substitution after selective-inversion

```
    Input: $\mathcal{G}^{2D}_{\rm loc}$, $\mathcal{Y}^{1D}_{\rm loc}$, $s$, $p$, $r$
1 if $p > 1$ then
2     if $r < \lfloor \frac{p}{2} \rfloor$ then Recurse($\mathcal{G}^{2D}_{\rm loc}$, $\mathcal{Y}^{1D}_{\rm loc}$,$c_0(s)$, $\lfloor \frac{p}{2} \rfloor$,$r$)
3     else Recurse($\mathcal{G}^{2D}_{\rm loc}$, $\mathcal{Y}^{1D}_{\rm loc}$,$c_1(s)$, $p - \lfloor \frac{p}{2} \rfloor$, $r - \lfloor \frac{p}{2} \rfloor$)
4     foreach $j \in \{0,1\}$ do AllToAll to add $Z_{c_j}$ into $Y_s$
5     $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:)\,\mathrm{inv}(L(\mathcal{D}_s,\mathcal{D}_s))$ via Algorithm C.1
6     $Z_s := Z_s - L(\mathcal{L}_s,\mathcal{D}_s)X(\mathcal{D}_s,:)$ via Algorithm C.1
7 else
8     Apply Algorithm 2.14 to $\mathcal{G}^{2D}_{\rm loc}$ and $\mathcal{Y}^{1D}_{\rm loc}$ beginning at node $s$
```

order to exploit selective inversion with confidence. A detailed analysis of the requirements for accurately forming both $\mathrm{inv}(L)\,b$ and $\mathrm{inv}(L)^H\,b$ will be the subject of future work.

2.4.5 Solving with many right-hand sides

When many right-hand sides are available at once, which is the case in many *seismic inversion* [121] algorithms, there are many more paths to scalable multifrontal triangular solves. In particular, let us recall that the dense operation `TRSM` [37], which performs triangular solves with multiple right-hand sides, can be made scalable when there are sufficiently many right-hand sides due to the independence of each solve. This is in stark contrast to triangular solves with $O(1)$ right-hand sides, which are difficult (if not impossible) to efficiently parallelize, and so it is natural to consider building a multifrontal triangular solve on top of a distributed dense `TRSM` algorithm, such as Algorithm C.9.

As in the single right-hand side case, the distribution of each submatrix of the right-hand side tree, $\mathcal{Y}$, should be chosen in order to maximize the efficiency of the most expensive operations which will be performed with it. In the case of *many* right-hand sides, which we will loosely define to be at least $O(\sqrt{p})$ right-hand sides when p processes are to be used, Steps 8-9 and 5-6 from Algorithms 2.19 and 2.20 should each be thought of as a pair of distributed `TRSM` and `GEMM` calls. If each matrix Y_s in the right-hand side tree is stored in a two-dimensional data distribution, then we may directly exploit Algorithms C.9 and C.3 in order to build a high-performance multifrontal triangular solve with many right-hand sides (and the adjoint dense kernels may be used to construct an efficient adjoint solve). The resulting forward elimination approach is listed in Algorithm 2.23.

Algorithm 2.23: Distributed multifrontal forward substitution with many right-hand sides

```
  Input: $\mathcal{F}^{2D}_{\text{loc}}$, $\mathcal{Y}^{2D}_{\text{loc}}$, $s$, $p$, $r$
1 if $p > 1$ then
2     if $r < \lfloor \frac{p}{2} \rfloor$ then Recurse($\mathcal{F}^{2D}_{\text{loc}}$, $\mathcal{Y}^{2D}_{\text{loc}}$,$c_0(s)$, $\lfloor \frac{p}{2} \rfloor$,$r$)
3     else Recurse($\mathcal{F}^{2D}_{\text{loc}}$, $\mathcal{Y}^{2D}_{\text{loc}}$,$c_1(s)$, $p - \lfloor \frac{p}{2} \rfloor$, $r - \lfloor \frac{p}{2} \rfloor$)
4     foreach $j \in \{0,1\}$ do AllToAll to add $Z_{c_j}$ into $Y_s$
5     $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:)L(\mathcal{D}_s,\mathcal{D}_s)^{-1}$ via Algorithm C.9
6     $Z_s := Z_s - L(\mathcal{L}_s,\mathcal{D}_s)X(\mathcal{D}_s,:)$ via Algorithm C.3
7 else
8     Apply Algorithm 2.14 to $\mathcal{F}^{2D}_{\text{loc}}$ and $\mathcal{Y}^{2D}_{\text{loc}}$ beginning at node $s$
```

2.4.6 Graph partitioning

The elephant in the room for parallel sparse direct solvers is that no scalable implementations of effective heuristics for graph partitioning yet exist. Although ParMETIS [82] is extremely widely used and useful for partitioning graphs which cannot be stored in memory on a single node, there are still difficulties in the parallelization of partition refinements, and, in practice [89],

Algorithm 2.24: Distributed multifrontal forward substitution with many right-hand sides after selective-inversion

```
   Input: $\mathcal{G}^{2D}_{\rm loc}$, $\mathcal{Y}^{2D}_{\rm loc}$, $s$, $p$, $r$
1  if $p > 1$ then
2      if $r < \lfloor \frac{p}{2} \rfloor$ then Recurse($\mathcal{G}^{2D}_{\rm loc}$, $\mathcal{Y}^{2D}_{\rm loc}$,$c_0(s)$, $\lfloor \frac{p}{2} \rfloor$,$r$)
3      else Recurse($\mathcal{G}^{2D}_{\rm loc}$, $\mathcal{Y}^{2D}_{\rm loc}$,$c_1(s)$, $p-\lfloor \frac{p}{2} \rfloor$, $r-\lfloor \frac{p}{2} \rfloor$)
4      foreach $j \in \{0,1\}$ do AllToAll to add $Z_{c_j}$ into $Y_s$
5      $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:)\,\mathrm{inv}(L(\mathcal{D}_s,\mathcal{D}_s))$ via Algorithm C.3
6      $Z_s := Z_s - L(\mathcal{L}_s,\mathcal{D}_s)X(\mathcal{D}_s,:)$ via Algorithm C.3
7  else
8      Apply Algorithm 2.14 to $\mathcal{G}^{2D}_{\rm loc}$ and $\mathcal{Y}^{2D}_{\rm loc}$ beginning at node $s$
```

distributed sparse-direct solvers often default to the sequential algorithms of METIS. It is therefore beneficial for large-scale applications to analytically perform nested dissection if their discretization is structured enough to allow it. This is the approach taken for the parallel sweeping preconditioner discussed in later chapters.

2.5 Summary

An introduction to nested-dissection based multifrontal Cholesky factorization and triangular solves was provided in the context of a reordered dense right-looking Cholesky factorization. Parallelizations of these operations were then discussed, Raghavan's *selective inversion* [102] was reviewed, and then two new parallel multifrontal triangular solve algorithms targeted towards simultaneously solving many right-hand sides were introduced. The author expects that these algorithms will be a significant contribution towards frequency-domain seismic inversion procedures, which typically require the solution of roughly $O(N^{2/3})$ linear systems. In the following chapter, an algorithm will be described which replaces a full 3D multifrontal factorization with a preconditioner which only requires $O(N^{4/3})$ work to set up, yet only requires $O(N \log N)$ work to solve usual seismic problems. It is easy to see that, if $O(N^{2/3})$ linear systems are to be solved, asymptotically more work is performed solving the linear systems and setting up the preconditioner.

Chapter 3

Sweeping preconditioners

The fundamental idea of both this and the following chapter is to organize our problem in such a manner that we may assign physical meanings to the temporary matrices generated during a factorization process, and then to exploit our knowledge of the physics in order to efficiently construct (or, in the case of the next chapter, *compress*) an approximation to these temporary products. In particular, we will order our degrees of freedom so that the *Schur complements* generated during a block tridiagonal LDL^T factorization of a discretized boundary value problem may be approximately generated by the factorization of a boundary value problem posed over a much smaller subdomain. More specifically, if the original problem is three-dimensional, multifrontal factorizations may be efficiently employed on quasi-2D subdomains in order to construct an effective preconditioner.

3.1 Radiation boundary conditions

It can easily be seen that, if we had instead transformed the acoustic wave equation, Equation (1.2), into time-harmonic form using a time-dependence of $e^{i\omega t}$ instead of $e^{-i\omega t}$, then we would arrive at the exact same Helmholtz equation. But this implies that the transformation $t \mapsto -t$, which is simply *time reversal*, does not have an effect on the resulting Helmholtz equation. It is therefore the responsibility of the *boundary conditions*[1] to break this symmetry so that we may describe cause and effect in a sensical manner. For instance, if we create a disturbance at some point in the domain, the desired response is for a wave front to radiate outwards from this point as time progresses, rather than coming in "from infinity" at the beginning of time in anticipation of our actions.

Sommerfeld introduced a *radiation condition* [111] for precisely this reason; in three dimensions, it is the requirement that the solution $u(x)$ satisfies

$$\lim_{|x|\to\infty} |x| \left(\frac{\partial u}{\partial |x|} - i\omega u \right) = 0, \tag{3.1}$$

which can be seen to hold for the *radiating* 3D Helmholtz Green's function,

$$G_r(x,y) = \frac{e^{i\omega|x-y|}}{4\pi|x-y|}, \tag{3.2}$$

but not the *absorbing* Green's function,

$$G_a(x,y) = \overline{G_r(x,y)} = \frac{e^{-i\omega|x-y|}}{4\pi|x-y|}. \tag{3.3}$$

It is instructive to consider the evolution of these functions in their time-harmonic forms when the point source is at the origin, for example, an animation of

$$G_r(x,0)e^{-i\omega t} = \frac{e^{i\omega(|x|-t)}}{4\pi|x|}$$

shows that waves are propagating outwards from the origin, while $G_a(x,0)e^{-i\omega t}$ can be seen to be absorbing waves into the origin.[2]

[1]some readers will perhaps insist that we speak of *remote conditions* instead of applying boundary conditions "at infinity"

[2]And thus, if a time-dependence of $e^{i\omega t}$ is desired, the radiation condition should be conjugated and the definitions of G_a and G_r should be reversed (i.e., conjugated).

Essentially the same argument holds for 3D time-harmonic Maxwell's equations, which are typically equipped with the *Silver-Müller radiation conditions* [124],

$$\lim_{|x|\to\infty} (\mathcal{H} \times x - |x|\mathcal{E}) = 0, \text{ and} \tag{3.4}$$

$$\lim_{|x|\to\infty} (\mathcal{E} \times x + |x|\mathcal{H}) = 0. \tag{3.5}$$

Likewise, the 3D time-harmonic linear elastic equations make use of the *Kupradze-Sommerfeld radiation conditions* [24, 123],

$$\lim_{|x|\to\infty} |x| \left(\frac{\partial u_s}{\partial |x|} - i\kappa_s u_s \right) = 0, \text{ and} \tag{3.6}$$

$$\lim_{|x|\to\infty} |x| \left(\frac{\partial u_p}{\partial |x|} - i\kappa_p u_p \right) = 0, \tag{3.7}$$

where u_s and u_p respectively refer to the *solenoidal* (divergence-free) and *irrotational* (curl-free) components of the solution, u, and κ_s and κ_p refer to the wave numbers associated with shear and pressure waves, respectively. Note that the decomposition of the solution into its solenoidal and irrotational parts is referred to as the *Helmholtz decomposition* [32].

3.2 ***Moving PML*** sweeping preconditioner

The focus of this dissertation is on parallelization of a class of sweeping preconditioners which makes use of auxiliary problems with artificial radiation boundary conditions in order to approximate the Schur complements that arise during a block LDL^T factorization. The approach is referred to as a *moving PML* preconditioner since the introductory paper represented the artificial radiation boundary conditions using Perfectly Matched Layers [19, 29, 44].

One interpretation of radiation conditions is that they allow for the analysis of a finite portion of an infinite domain, as their purpose is to enforce the condition that waves propagate outwards and not reflect at the boundary of the truncated domain. This concept is crucial to understanding the Schur complement approximations that take place within the moving PML sweeping preconditioner which is reintroduced in this document for the sake of completeness.

For the sake of simplicity, we will describe the preconditioner in the context of a second-order finite-difference discretization over the unit cube, with PML used to approximate a radiation boundary condition on the $x_3 = 0$ face and homogeneous Dirichlet boundary conditions applied on all other boundaries (an $x_1 x_3$ cross-section is shown in Fig. 3.1). If the domain is discretized into an $n \times n \times n$ grid, then ordering the vertices in the grid such that vertex (i_1, i_2, i_3) is assigned index $i_1 + i_2 n + i_3 n^2$ results in a block tridiagonal system of equations, say

$$\begin{pmatrix} A_{0,0} & A_{1,0}^T & & & \\ A_{1,0} & A_{1,1} & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & A_{n-1,n-2}^T \\ & & & A_{n-1,n-2} & A_{n-1,n-1} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-2} \\ f_{n-1} \end{pmatrix}, \tag{3.8}$$

where $A_{i,j}$ propagates sources from the i'th $x_1 x_2$ plane into the j'th $x_1 x_2$ plane, and the overall linear system is complex symmetric (*not* Hermitian) due to the imaginary terms introduced by the PML [44].

If we were to ignore the sparsity within each block, then the naïve factorization and solve algorithms shown in Algorithms 3.1 and 3.2 would be appropriate for a direct solver, where the application of S_i^{-1} makes use of the factorization of S_i. While the computational complexities of these approaches are significantly higher than the multifrontal algorithms discussed in

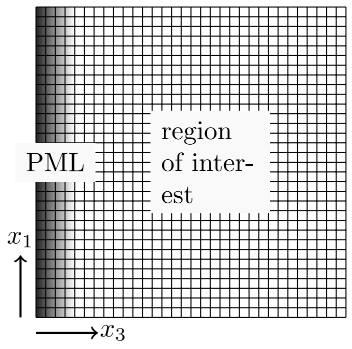


Figure 3.1: An x_1x_3 cross section of a cube with PML on its $x_3 = 0$ face. The domain is shaded in a manner which loosely corresponds to its extension into the complex plane.

the previous chapter, which have an $O(N^2)$ factorization cost and an $O(N^{4/3})$ solve complexity for regular three-dimensional meshes, they are the conceptual starting points of the sweeping preconditioner.[3]

Algorithm 3.1: Naïve block tridiagonal LDL^T factorization. $O(n^7) = O(N^{7/3})$ work is required.

```
1 $S_0 := A_{0,0}$
2 Factor $S_0$
3 for $i = 0, ..., n-2$ do
4     $S_{i+1} := A_{i+1,i+1} - A_{i+1,i} S_i^{-1} A_{i+1,i}^T$
5     Factor $S_{i+1}$
6 end
```

Suppose that we paused Algorithm 3.1 after computing the i'th Schur complement, S_i, where the i'th x_1x_2 plane is in the interior of the domain (i.e., it is not in a PML region). Due to the ordering imposed on the degrees of freedom of the discretization, the first i Schur complements are equivalent

[3]In fact, they are the starting points of *both* classes of sweeping preconditioners. The $\mathcal{H}$-matrix approach essentially executes these algorithms with $\mathcal{H}$-matrix arithmetic.

Algorithm 3.2: Naïve block LDL^T solve. $O(n^5) = O(N^{5/3})$ work is required.

```
   // Apply $L^{-1}$
1  for $i = 0, ..., n-2$ do
2      $u_{i+1} := u_{i+1} - A_{i+1,i}(S_i^{-1} u_i)$
3  end
   // Apply $D^{-1}$
4  for $i = 0, ..., n-1$ do
5      $u_i := S_i^{-1} u_i$
6  end
   // Apply $L^{-T}$
7  for $i = n-2, ..., 0$ do
8      $u_i := u_i - S_i^{-1}(A_{i+1,i}^T u_{i+1})$
9  end
```

to those that would have been produced from applying Algorithm 3.1 to an auxiliary problem formed by placing a homogeneous Dirichlet boundary condition on the $(i+1)$'th x_1x_2 plane and ignoring all of the subsequent planes (see the left half of Fig. 3.2). While this observation does not immediately yield any computational savings, it does allow for a qualitative description of the inverse of the i'th Schur complement, S_i^{-1}: it is the restriction of the half-space Green's function of the exact auxiliary problem onto the i'th x_1x_2 plane (recall that PML can be interpreted as approximating the effect of a domain extending to infinity).

The main approximation made in the sweeping preconditioner can now be succinctly described: since S_i^{-1} is a restricted half-space Green's function which incorporates the velocity field of the first i planes, it is natural to approximate it with another restricted half-space Green's function which only takes into account the *local* velocity field, i.e., by moving the PML next to the i'th plane (see the right half of Fig. 3.2).

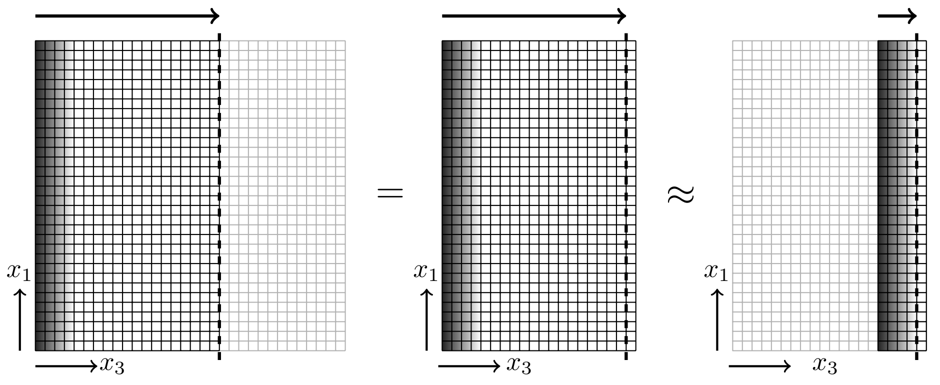


Figure 3.2: (Left) A depiction of the portion of the domain involved in the computation of the Schur complement of an x_1x_2 plane (marked with the dashed line) with respect to all of the planes to its left during execution of Alg. 3.1. (Middle) An equivalent auxiliary problem which generates the same Schur complement; the original domain is truncated just to the right of the marked plane and a homogeneous Dirichlet boundary condition is placed on the cut. (Right) A local auxiliary problem for generating an approximation to the relevant Schur complement; the radiation boundary condition of the exact auxiliary problem is moved next to the marked plane.

If we use $\gamma(\omega)$ to denote the number of grid points of PML as a function of the frequency, ω, then it is important to recognize that the depth of the approximate auxiliary problems in the x_3 direction is $\Omega(\gamma)$.[4] If the depth of the approximate auxiliary problems was $O(1)$, then combining nested dissection with the multifrontal method over the regular $n \times n \times O(1)$ mesh would only require $O(n^3)$ work and $O(n^2 \log n)$ storage [54]. However, if the PML size is a slowly growing function of frequency, then applying 2D nested dissection to the *quasi-2D* domain requires $O(\gamma^3 n^3)$ work and $O(\gamma^2 n^2 \log n)$ storage, as the number of degrees of freedom in each front increases by a factor of γ and dense factorizations have cubic complexity.

Let us denote the quasi-2D discretization of the local auxiliary problem for the i'th plane as H_i, and its corresponding approximation to the Schur complement S_i as $\tilde{S}_i$. Since $\tilde{S}_i$ is essentially a dense matrix, it is advantageous to come up with an abstract scheme for applying $\tilde{S}_i^{-1}$: Assuming that H_i was ordered in a manner consistent with the $(i_1, i_2, i_3) \mapsto i_1 + i_2 n + i_3 n^2$ global ordering, the degrees of freedom corresponding to the i'th plane come last and we find that

$$H_i^{-1} = \begin{pmatrix} \star & \star \\ \star & \tilde{S}_i^{-1} \end{pmatrix}, \tag{3.9}$$

where the irrelevant portions of the inverse have been marked with a $\star$. Then, trivially,

$$H_i^{-1} \begin{pmatrix} 0 \\ u_i \end{pmatrix} = \begin{pmatrix} \star & \star \\ \star & \tilde{S}_i^{-1} \end{pmatrix} \begin{pmatrix} 0 \\ u_i \end{pmatrix} = \begin{pmatrix} \star \\ \tilde{S}_i^{-1} u_i \end{pmatrix}, \tag{3.10}$$

which implies a method for quickly computing $\tilde{S}_i^{-1} u_i$ given a factorization of H_i:

[4]In all of the experiments in this chapter, $\gamma(\omega)$ was either 5 or 6, and the subdomain depth never exceeded 10.

Algorithm 3.3: Application of $\tilde{S}_i^{-1}$ to u_i given a multifrontal factorization of H_i. $O(\gamma^2 n^2 \log n)$ work is required.

```
1 Form $\hat{u}_i$ as the extension of $u_i$ by zero over the artificial PML
2 Form $\hat{v}_i := H_i^{-1}\hat{u}_i$
3 Extract $\tilde{S}_i^{-1}u_i$ from the relevant entries of $\hat{v}_i$
```

From now on, we write T_i to refer to the application of the (approximate) inverse of the Schur complement for the i'th plane.

There are two more points to discuss before defining the full serial algorithm. The first is that, rather than performing an approximate LDL^T factorization using a discretization of $\mathcal{A}$, the preconditioner is instead built from a discretization of an *artificially damped* version of the Helmholtz operator, say

$$\mathcal{J} \equiv \left[-\Delta - \frac{(\omega + i\alpha)^2}{c^2(x)}\right], \tag{3.11}$$

where $\alpha \approx 2\pi$ is responsible for the artificial damping. This is in contrast to shifted Laplacian preconditioners [18, 46], where α is typically $O(\omega)$ [47], and our motivation is to avoid introducing large long-range dispersion error by damping the long range interactions in the preconditioner. Just as A refers to the discretization of the original Helmholtz operator, $\mathcal{A}$, we will use J to refer to the discretization of the artificially damped operator, $\mathcal{J}$.

The second point is much easier to motivate: since the artificial PML in each approximate auxiliary problem is of depth $\gamma(\omega)$, processing a single plane at a time would require processing $O(n)$ subdomains with $O(\gamma^3 n^3)$ work each. Clearly, processing $O(\gamma)$ planes at once has the same asymptotic complexity as processing a single plane, and so combining $O(\gamma)$ planes reduces the setup cost from $O(\gamma^3 N^{4/3})$ to $O(\gamma^2 N^{4/3})$. Similarly, the memory usage becomes $O(\gamma N \log N)$ instead of $O(\gamma^2 N \log N)$.[5] If we refer to these sets of $O(\gamma)$ contiguous planes as *panels*, which we label from 0 to $m-1$, where $m = O(n/\gamma)$, and correspondingly redefine $\{u_i\}$, $\{f_i\}$, $\{S_i\}$, $\{T_i\}$, and $\{H_i\}$, we have the following algorithm for setting up an approximate block LDL^T factorization of the discrete artificially damped Helmholtz operator:

Algorithm 3.4: Setup phase of the sweeping preconditioner. $O(\gamma^2 N^{4/3})$ work is required.

```
1 $S_0 := J_{0,0}$
2 Factor $S_0$
3 parallel for $i = 1, ..., m-1$ do
4     Form $H_i$ by prefixing PML to $J_{i,i}$
5     Factor $H_i$
6 end
```

Once the preconditioner is set up, it can be applied using a straightforward modification of Algorithm 3.2 which avoids redundant solves by combining the application of L^{-1} and D^{-1} (see Algorithm 3.5). Given that the preconditioner can be set up with $O(\gamma^2 N^{4/3})$ work, and applied with $O(\gamma N \log N)$ work, it provides a near-linear scheme for solving Helmholtz equations if only $O(1)$ iterations are required for convergence. The experiments in this chapter seem to indicate that, as long as the velocity model is not completely surrounded by reflecting boundary conditions and/or high-velocity barriers, the sweeping preconditioner indeed only requires $O(1)$ iterations.

Although this chapter is focused on the parallel solution of Helmholtz equations, Tsuji et al. have shown that the moving PML sweeping preconditioner is equally effective for time-harmonic Maxwell's equations [124, 125],

[5]Note that increasing the number of planes per panel provides a mechanism for interpolating between the sweeping preconditioner and a full multifrontal factorization.

Algorithm 3.5: Application of the sweeping preconditioner. $O(\gamma N \log N)$ work is required.

```
   // Apply $L^{-1}$ and $D^{-1}$
1  for $i = 0, ..., m-2$ do
2      $u_i := T_i u_i$
3      $u_{i+1} := u_{i+1} - J_{i+1,i} u_i$
4  end
5  $u_{m-1} := T_{m-1} u_{m-1}$
   // Apply $L^{-T}$
6  for $i = m-2, ..., 0$ do
7      $u_i := u_i - T_i(J_{i+1,i}^T u_{i+1})$
8  end
```

and current work supports its effectiveness for time-harmonic linear elasticity. The rest of this document will be presented in the context of the Helmholtz equation, but we emphasize that all parallelizations should carry over to more general wave equations in a conceptually trivial way.

3.3 Related work

A domain decomposition variant of the sweeping preconditioner was recently introduced [120] which results in fast convergence rates, albeit at the expense of requiring PML padding on both sides of each subdomain. Recalling our previous analysis with respect to the PML size, γ, the memory usage of the preconditioner scales linearly with the PML size, while the setup cost scales quadratically. Thus, the domain decomposition approach should be expected to use twice as much memory and require four times as much work for the setup phase. On the other hand, doubling the subdomain sizes allows for more parallelism in both the setup and solve phases, and less sweeps seem to be required.

Another closely related method is the *Analytic ILU factorization* [52]. Like the sweeping preconditioner, it uses local approximations of the Schur complements of the block LDL^T factorization of the Helmholtz matrix represented in block tridiagonal form. There are two crucial differences between the two methods:

- Roughly speaking, AILU can be viewed as using Absorbing Boundary Conditions (ABC's) [42] instead of PML when forming approximate subdomain auxiliary problems. While ABC's result in strictly 2D local subproblems, versus the *quasi-2D* subdomain problems which result from using PML, they are well-known to be less effective approximations of the Sommerfeld radiation condition (and thus the local Schur complement approximations are less effective). The sweeping preconditioner handles its non-trivial subdomain factorizations via a multifrontal algorithm.

- Rather than preconditioning with an approximate LDL^T factorization of the original Helmholtz operator, the sweeping preconditioner sets up an approximate factorization of a *slightly damped* Helmholtz operator in order to mitigate the dispersion error which would result from the Schur complement approximations.

These two improvements are responsible for transitioning from the $O(\omega)$ iterations required with AILU to the $O(1)$ iterations needed with the sweeping preconditioner (for problems without internal resonance).

Two other iterative methods warrant mentioning: the two-grid shifted-Laplacian approach of [27] and the multilevel-ILU approach of [23]. Although both require $O(\omega)$ iterations for convergence, they have very modest memory requirements. In particular, [27] demonstrates that, with a properly tuned two-grid approach, large-scale heterogeneous 3D problems can be solved with impressive timings.

There has also been a recent effort to extend the fast-direct methods presented in [134] from definite elliptic problems into the realm of low-to-moderate frequency time-harmonic wave equations [130, 131]. While their work has resulted in a significant constant speedup versus applying a classical multifrontal algorithm to the full 3D domain [131], their results have so far still demonstrated the same $O(N^2)$ asymptotic complexity as standard sparse-direct methods.

3.4 Parallel sweeping preconditioner

The setup and application stages of the sweeping preconditioner (Algs. 3.4 and 3.5) essentially consist of $m = O(n/\gamma)$ multifrontal factorizations and solves, respectively. The most important detail is that *the subdomain factorizations can be performed in parallel, while the subdomain solves must happen sequentially.*[6] When we also consider that each subdomain factorization requires $O(\gamma^3 n^3)$ work, while subdomain solves only require $O(\gamma n^2 \log n)$ work, we see that, relative to the subdomain factorizations, subdomain solves must extract $m = O(n/\gamma)$ more parallelism from a factor of $O(\gamma^2 n/\log n)$ less operations. We thus have a strong hint that, unless the subdomain solves are carefully handled, they will be the limiting factor in the scalability of the sweeping preconditioner.

[6]While it is tempting to try to expose more parallelism with techniques like cyclic reduction (which is a special case of a multifrontal algorithm), their straightforward application destroys the Schur complement properties that we exploit for our fast algorithm.

3.4.1 The need for scalable triangular solves

Roughly speaking, the analysis in [79] shows that, if p_F processes are used in the multifrontal factorization of our quasi-2D subdomain problems, then we must have $\gamma n = \Omega(p_F^{1/2})$ in order to maintain constant efficiency as p_F is increased; similarly, if p_S processes are used in the multifrontal triangular solves for a subdomain, then we must have $\gamma n \approx \Omega(p_S)$ (where we use $\approx$ to denote that the equality holds within logarithmic factors). Since we can simultaneously factor the $m = O(n/\gamma)$ subdomain matrices, we denote the total number of processes as p and set $p_S = p$ and $p_F = O(p/m)$; then the subdomain factorizations only require that $n^3 = \Omega(p/\gamma)$, while the subdomain solves have the much stronger constraint that $n \approx \Omega(p/\gamma)$. This last constraint should be considered unacceptable, as we have the conflicting requirement that $n^3 \approx O(p/\gamma)$ in order to store the factorizations in memory. It is therefore advantageous to consider more scalable alternatives to standard multifrontal triangular solves, even if they require additional computation. In particular, for modest numbers of right-hand sides, we will make use of *selective inversion* [102], and for large numbers of right-hand sides, we may make use of the GEMM and TRSM-based algorithms discussed at the end of Chapter 2. Since each application of the sweeping preconditioner requires two multifrontal solves for each of the $m = O(n/\gamma)$ subdomains, which are relatively small and likely distributed over a large number of processes, we will later see that high-performance triangular solves are crucial for the competitiveness of the sweeping preconditioner on parallel architectures.

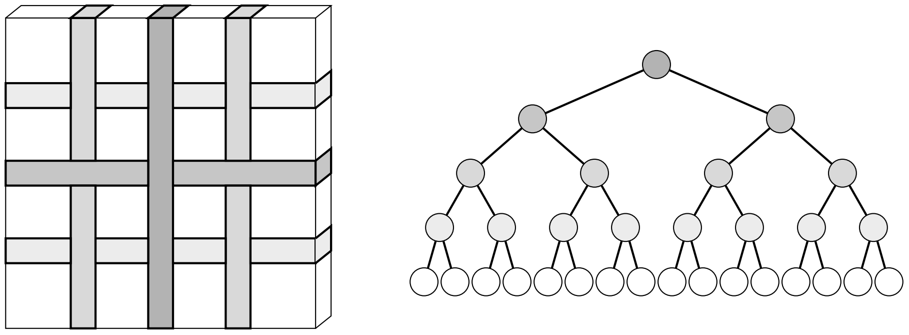


Figure 3.3: A separator-based elimination tree (right) over a quasi-2D subdomain (left)

3.4.2 Global vector distributions

The goal of this subsection is to describe an appropriate scheme for distributing vectors which are supported over the entire domain (as opposed to only over a panel auxiliary problem). And while the factorizations themselves may have occurred on subteams of $O(p/m)$ processes each, in order to make use of all available processes for every subdomain solve, at this point we assume that each auxiliary problem's frontal tree has been selectively inverted and is distributed using a subtree-to-subteam mapping (recall Fig. 2.9) over the entire set of p processes.[7]

Since a subtree-to-subteam mapping will assign each supernode of an auxiliary problem to a team of processes, and each panel of the original 3D domain is by construction a subset of the domain of an auxiliary problem, it is straightforward to extend the supernodal subteam assignments to the full domain. We should then be able to distribute global vectors so that no

[7]In cases where the available solve parallelism has been exhausted but the problem cannot be solved on less processes due to memory constraints, it would be preferable to solve with subdomains stored on subsets of processes.

$$\begin{pmatrix} 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \\ 1 & 3 & 5 & 1 & 3 & 5 & 1 \\ 0 & 2 & 4 & 0 & 2 & 4 & 0 \end{pmatrix} \qquad \begin{array}{ccccc} 0 & - & 2 & - & 4 \\ | & & | & & | \\ 1 & - & 3 & - & 5 \end{array}$$

Figure 3.4: Overlay of the owning process ranks of an 7×7 matrix distributed over a 2×3 process grid in the $[M_C, M_R]$ distribution, where M_C assigns row i to process row i mod 2, and M_R assigns column j to process column i mod 3 (left). The process grid is shown on the right.

communication is required for readying panel subvectors for subdomain solves (via extension by zero for interior panels, and trivially for the first panel). Since our nested dissection process does not partition in the shallow dimension of quasi-2D subdomains (see Fig. 3.3), extending a vector defined over a panel of the original domain onto the PML-padded auxiliary domain simply requires individually extending each supernodal subvector by zero in the x_3 direction.

Consider an element-wise two-dimensional cyclic distribution [99] of a frontal matrix F over q processes using an $r \times c$ process grid, where r and c are $O(\sqrt{q})$. Then the (i, j) entry will be stored by the process in the $(i \bmod r, j \bmod c)$ position in the process grid (see Figure 3.4). Using the notation from [99], this distributed front would be denoted as $F[M_C, M_R]$, while its top-left quadrant would be referred to as $F_{TL}[M_C, M_R]$.

Loosely speaking, $F[X, Y]$ means that each column of F is distributed using the scheme denoted by X, and each row is distributed using the scheme denoted by Y. For the element-wise two-dimensional distribution used for F, $[M_C, M_R]$, we have that the columns of F are distributed like Matrix Columns (M_C), and the rows of F are distributed like Matrix Rows (M_R). While this notation may seem vapid when only working with a single distributed matrix, it is useful when working with products of distributed matrices. For instance, if a '$\star$' is used to represent rows/columns being redundantly stored (i.e., not distributed), then the result of every process multiplying its local submatrix of $A[X,\star]$ with its local submatrix of $B[\star,Y]$ forms a distributed matrix $C[X,Y] = (AB)[X,Y] = A[X,\star]\,B[\star,Y]$, where the last expression refers to a per-process local multiplication.

We can now decide on a distribution for each supernodal subvector, say x_s, based on the criteria that it should be fast to form $F_{TL}x_s$ and $F_{TL}^T x_s$ in the same distribution as x_s, given that F_{TL} is distributed as $F_{TL}[M_C,M_R]$. Suppose that we define a Column-major Vector distribution (V_C) of x_s, say $x_s[V_C,\star]$, to mean that entry i is owned by process $i \bmod q$, which corresponds to position ($i \bmod r$, $\lfloor i/r \rfloor \bmod c$) in the process grid (if the grid is constructed with a column-major ordering of the process ranks; see the left side of Fig. 3.5). Then a call to `MPI_Allgather` [39] within each row of the process grid would allow for each process to collect all of the data necessary to form $x_s[M_C,\star]$, as for any process row index $s \in \{0,1,...,r-1\}$,

$$\{i \in \mathbb{N}_0 : i \bmod r = s\} = \bigcup_{t=0}^{c-1} \{i \in \mathbb{N}_0 : i \bmod q = s + tr\}. \tag{3.12}$$

See the left side of Fig. 3.6 for an example of an $[M_C,\star]$ distribution of a 7×3 matrix.

Similarly, if x_s was distributed with a Row-major Vector distribution (V_R), as shown on the right side of Fig. 3.5, say $x_s[V_R,\star]$, so that entry i is owned by the process in position $(\lfloor i/c \rfloor \bmod r, i \bmod c)$ of the process grid, then a call to `MPI_Allgather` within each column of the process grid would

$$\begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} 0 \\ 2 \\ 4 \\ 1 \\ 3 \\ 5 \\ 0 \end{pmatrix}$$

Figure 3.5: Overlay of the owning process ranks of a vector of height 7 distributed over a 2×3 process grid in the $[V_C, \star]$ vector distribution (left) and the $[V_R, \star]$ vector distribution (right).

$$\begin{pmatrix} \{0,2,4\} \\ \{1,3,5\} \\ \{0,2,4\} \\ \{1,3,5\} \\ \{0,2,4\} \\ \{1,3,5\} \\ \{0,2,4\} \end{pmatrix}, \qquad \begin{pmatrix} \{0,1\} \\ \{2,3\} \\ \{4,5\} \\ \{0,1\} \\ \{2,3\} \\ \{4,5\} \\ \{0,1\} \end{pmatrix}$$

Figure 3.6: Overlay of the owning process ranks of a vector of height 7 distributed over a 2×3 process grid in the $[M_C, \star]$ distribution (left) and the $[M_R, \star]$ distribution (right).

provide each process with the data necessary to form $x_s[M_R, \star]$. Under reasonable assumptions, both of these redistributions can be shown to have per-process communication volume lower bounds of $\Omega(n/\sqrt{p})$ (if F_{TL} is $n \times n$) and latency lower bounds of $\Omega(\log_2(\sqrt{p}))$ [28]. We also note that translating between $x_s[V_C, \star]$ and $x_s[V_R, \star]$ simply requires permuting which process owns each local subvector, so the communication volume would be $O(n/p)$, while the latency cost is $O(1)$.

We have thus described efficient techniques for redistributing $x_s[V_C, \star]$ to both the $x_s[M_R, \star]$ and $x_s[M_C, \star]$ distributions, which are the first steps for our parallel algorithms for forming $F_{TL}x_s$ and $F_{TL}^T x_s$, respectively: Denoting the distributed result of each process in process column $t \in \{0, 1, ..., c-1\}$ multiplying its local submatrix of $F_{TL}[M_C, M_R]$ by its local subvector of $x_s[M_R, \star]$ as $z^{(t)}[M_C, \star]$, it holds that $(F_{TL}x_s[M_C, \star] = \sum_{t=0}^{c-1} z^{(t)}[M_C, \star]$. Since Equation (3.12) also implies that each process's local data from a $[V_C, \star]$ distribution is a subset of its local data from a $[M_C, \star]$ distribution, a simultaneous summation and scattering of $\{z^{(t)}[M_C, \star]\}_{t=0}^{c-1}$ within process rows, perhaps via `MPI_Reduce_scatter` or `MPI_Reduce_scatter_block`, yields the desired result, $(F_{TL}x_s)[V_C, \star]$. An analogous process with $(F_{TL}[M_C, M_R])^T = F_{TL}^T[M_R, M_C]$ and $x_s[M_C, \star]$ yields $(F_{TL}^T x_s)[V_R, \star]$, which can then be cheaply permuted to form $(F_{TL}^T x_s)[V_C, \star]$. Both calls to `MPI_Reduce_scatter_block` can be shown to have the same communication lower bounds as the previously discussed `MPI_Allgather` calls [28].

As discussed at the beginning of this section, defining the distribution of each supernodal subvector specifies a distribution for a global vector, say $[\mathcal{G}, \star]$. While the $[V_C, \star]$ distribution shown in the left half of Fig. 3.5 simply assigns entry i of a supernodal subvector x_s, distributed over q processes, to process $i \bmod q$, we can instead choose an alignment parameter, σ, where $0 \leq \sigma < q$, and assign entry i to process $(i + \sigma) \bmod q$. If we simply set $\sigma = 0$ for every supernode, as the discussion at the beginning of this subsection implied, then at most $O(\gamma n)$ processes will store data for the root separator supernodes of a global vector, as each root separator only has $O(\gamma n)$ degrees of freedom by construction. However, there are $m = O(n/\gamma)$ root separators, so we can easily allow for up to $O(n^2)$ processes to share the storage of a global vector if the alignments are carefully chosen. It is important to notice that the top-left quadrants of the frontal matrices for the root separators can each be distributed over $O(\gamma^2 n^2)$ processes, so $O(\gamma^2 n^2)$ processes can actively participate in the corresponding triangular matrix-vector multiplications.

3.4.3 Parallel preconditioned GMRES(k)

Since, by hypothesis, only $O(1)$ iterations of GMRES(k) will be required for convergence with the sweeping preconditioner, a cursory inspection of Algorithm 3.5 reveals that the vast majority of the work required for GMRES(k) will be in the multifrontal solves during each preconditioner application, but that a modest amount of time will also be spent in sparse matrix-vector multiplication with the discrete Helmholtz operator, A, and the off-diagonal blocks of the discrete artificially damped Helmholtz operator, J. It is thus important to parallelize the sparse matrix-vector multiplies, but it is not crucial that the scheme be optimal, and so we simply distribute A and J in the same manner as vectors, i.e., with the $[\mathcal{G}, \star]$ distribution derived from the auxiliary problems' frontal distributions.

We also note that, if we are to apply the preconditioner to several vectors at once, then we must make use of a version of GMRES(k) which can handle several right-hand sides at once. The approach taken in the following experiments is a straight-forward modification of Algorithm B.1 which simply maintains a separate Krylov subspace and Rayleigh quotient for each right-hand side and iterates until all relative residuals are sufficiently small.

3.5 Experimental results

Most of the techniques discussed thus far were instantiated as part of a distributed-memory multifrontal solver, *Clique* [100], and a distributed-memory moving-PML sweeping preconditioner, *Parallel Sweeping Preconditioner (PSP)* [101]. Our experiments were performed on the Texas Advanced Computing Center (TACC) machine, Lonestar, which is comprised of 1,888 compute nodes, each equipped with two hex-core 3.33 GHz processors and 24 GB of memory, which are connected with QDR InfiniBand using a fat-tree topology. Our tests launched eight MPI processes per node in order to provide each MPI process with 3 GB of memory.

Our experiments took place over five different 3D velocity models:

1. A uniform background with a high-contrast barrier. The domain is the unit cube and the wave speed is 1 except in $[0,1] \times [0.25, 0.3] \times [0, 0.75]$, where it is 10^{10}.

2. A wedge problem over the unit cube, where the wave speed is set to 2 if $Z \leq 0.4 + 0.1x_2$, 1.5 if otherwise $Z \leq 0.8 - 0.2x_2$, and 3 in all other cases.

3. A two-layer model defined over the unit cube, where $c = 4$ if $x_2 < 0.5$, and $c = 1$ otherwise.

4. A waveguide over the unit cube: $c(\mathbf{x}) = 1.25(1-0.4e^{-32(|x_1-0.5|^2+|x_2-0.5|^2)})$.
5. The SEG/EAGE Overthrust model [4], see Fig. 3.8.

In all of the following experiments, the shortest wavelength of each model is resolved with roughly ten grid points and the performance of the preconditioner is tested using the following four forcing functions:

1. a single *shot* centered at $\mathbf{x}_0$, $f_0(\mathbf{x}) = ne^{-10n\|\mathbf{x}-\mathbf{x}_0\|^2}$,
2. three shots, $f_1(\mathbf{x}) = \sum_{i=0}^{2} ne^{-10n\|\mathbf{x}-\mathbf{x}_i\|^2}$,
3. a Gaussian beam centered at $\mathbf{x}_2$ in direction $\mathbf{d}$, $f_2(\mathbf{x}) = e^{i\omega\mathbf{x}\cdot\mathbf{d}}e^{-4\omega\|\mathbf{x}-\mathbf{x}_2\|^2}$, and
4. a plane wave in direction $\mathbf{d}$, $f_3(\mathbf{x}) = e^{i\omega\mathbf{x}\cdot\mathbf{d}}$,

where $\mathbf{x}_0 = (0.5, 0.5, 0.1)$, $\mathbf{x}_1 = (0.25, 0.25, 0.1)$, $\mathbf{x}_2 = (0.75, 0.75, 0.5)$, and $\mathbf{d} = (1, 1, -1)/\sqrt{3}$. Note that, in the case of the Overthrust model, these source locations should be interpreted proportionally (e.g., an x_3 value of 0.1 means a depth which is 10% of the total depth of the model). Due to the oscillatory nature of the solution, we simply choose the zero vector as our initial guess in all experiments.

The first experiment was meant to test the convergence rate of the sweeping preconditioner over domains spanning 50 wavelengths in each direction (resolved to ten points per wavelength) and each test made use of 256 nodes of Lonestar. During the course of the tests, it was noticed that a significant amount of care must be taken when setting the amplitude of the derivative of the PML takeoff function (i.e., the "C" variable in Equation (2.1) from [44]).

	velocity model			
	barrier	wedge	two-layers	waveguide
Hz	50	75	50	37.5
PML amplitude	3.0	4.0	4.0	2.0
# of its.	28	49	48	52

Table 3.1: The number of iterations required for convergence for four model problems (with four forcing functions per model). The grid sizes were 500^3 and roughly 50 wavelengths were spanned in each direction. Five grid points were used for all PML discretizations, four planes were processed per panel, and the damping factors were all set to 7.

For the sake of brevity, hereafter we refer to this variable as the *PML amplitude*. A modest search was performed in order to find a near-optimal value for the PML amplitude for each problem. These values are reported in Table 3.1 along with the number of iterations required for the relative residuals for all four forcing functions to reduce to less than 10^{-5}.

It was also observed that, at least for the waveguide problem, the convergence rate would be significantly improved if 6 grid points of PML were used instead of 5. In particular, the 52 iterations reported in Table 3.1 reduce to 27 if the PML size is increased by one. Interestingly, the same number of iterations are required for convergence of the waveguide problem at half the frequency (and half the resolution) with five grid points of PML. Thus, it appears that the optimal PML size is a slowly growing function of the frequency.[8] We also note that, though it was not intentional, each of the these first four velocity models is invariant in one or more direction, and so it would be straightforward to sweep in a direction such that only $O(1)$ panel factorizations would need to be performed, effectively reducing the complexity of the

[8]A similar observation is also made in [120].

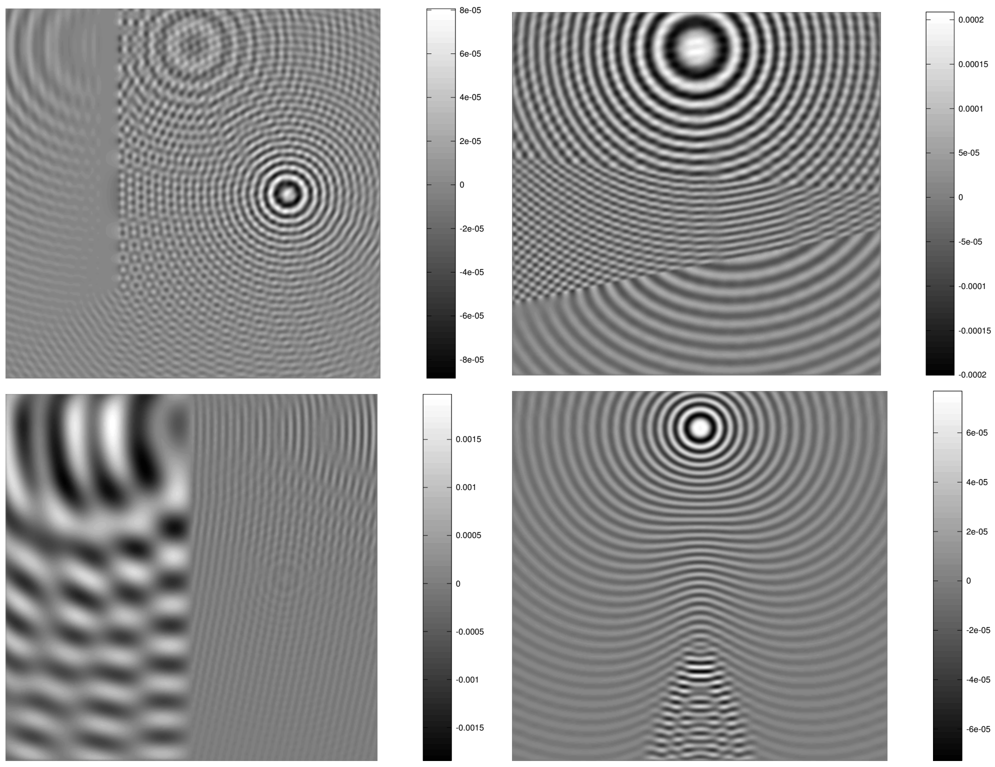


Figure 3.7: A single x_2x_3 plane from each of the four analytical velocity models over a 500^3 grid with the smallest wavelength resolved with ten grid points. (Top-left) the three-shot solution for the barrier model near $x_1 = 0.7$, (bottom-left) the three-shot solution for the two-layer model near $x_1 = 0.7$, (top-right) the single-shot solution for the wedge model near $x_1 = 0.7$, and (bottom-right) the single-shot solution for the waveguide model near $x_1 = 0.55$.

setup phase to $O(\gamma^2 N)$.

The last experiment was meant to simultaneously test the convergence rates and scalability of the new sweeping preconditioner using the Overthrust velocity model (see Fig. 3.8) at various frequencies, and the results are reported in Table 3.2. It is important to notice that this is not a typical weak scaling test, as the number of grid points per process was fixed, *not* the local memory usage or computational load, which both grow superlinearly with respect to the total number of degrees of freedom. Nevertheless, including the setup phase, it took less than three minutes to solve the 3.175 Hz problem with four right-hand sides with 128 cores, and just under seven and a half minutes to solve the corresponding 8 Hz problem using 2048 cores. Also, while it is by no means the main message of this section, the timings without making use of selective inversion are also reported in parentheses, as the technique is not widely implemented.[9]

3.6 Summary

A parallelization of the moving-PML sweeping preconditioner was presented and successfully applied to five challenging heterogeneous velocity models, including the SEG/EAGE Overthrust model. The primary challenges were:

- ensuring that the subdomain multifrontal solves were scalable (e.g., through selective inversion),
- ensuring that the PML profile was appropriately matched to the velocity

[9]Other than Clique, the only other implementation appears to be in DSCPACK [103].

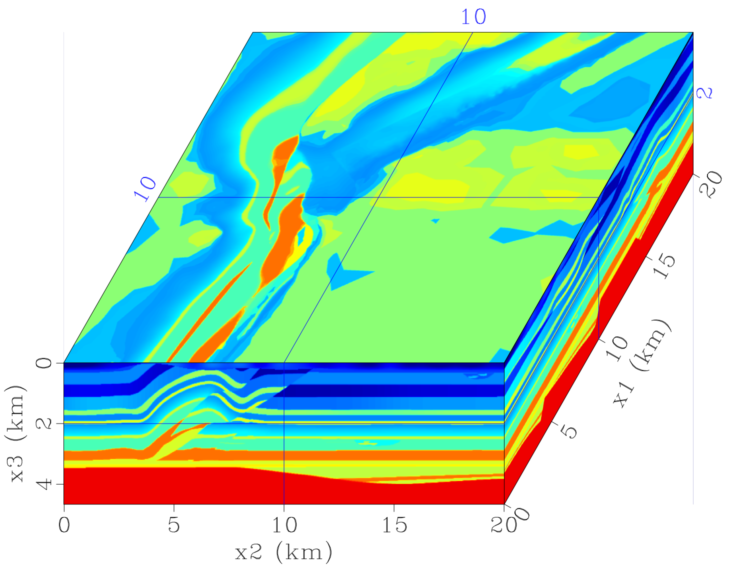


Figure 3.8: Three cross-sections of the SEG/EAGE Overthrust velocity model, which represents an artificial 20 km × 20 km × 4.65 km domain, containing an *overthrust* fault, using samples every 25 m. The result is an 801 × 801 × 187 grid of wave speeds varying discontinuously between 2.179 km/sec (blue) and 6.000 km/sec (red).

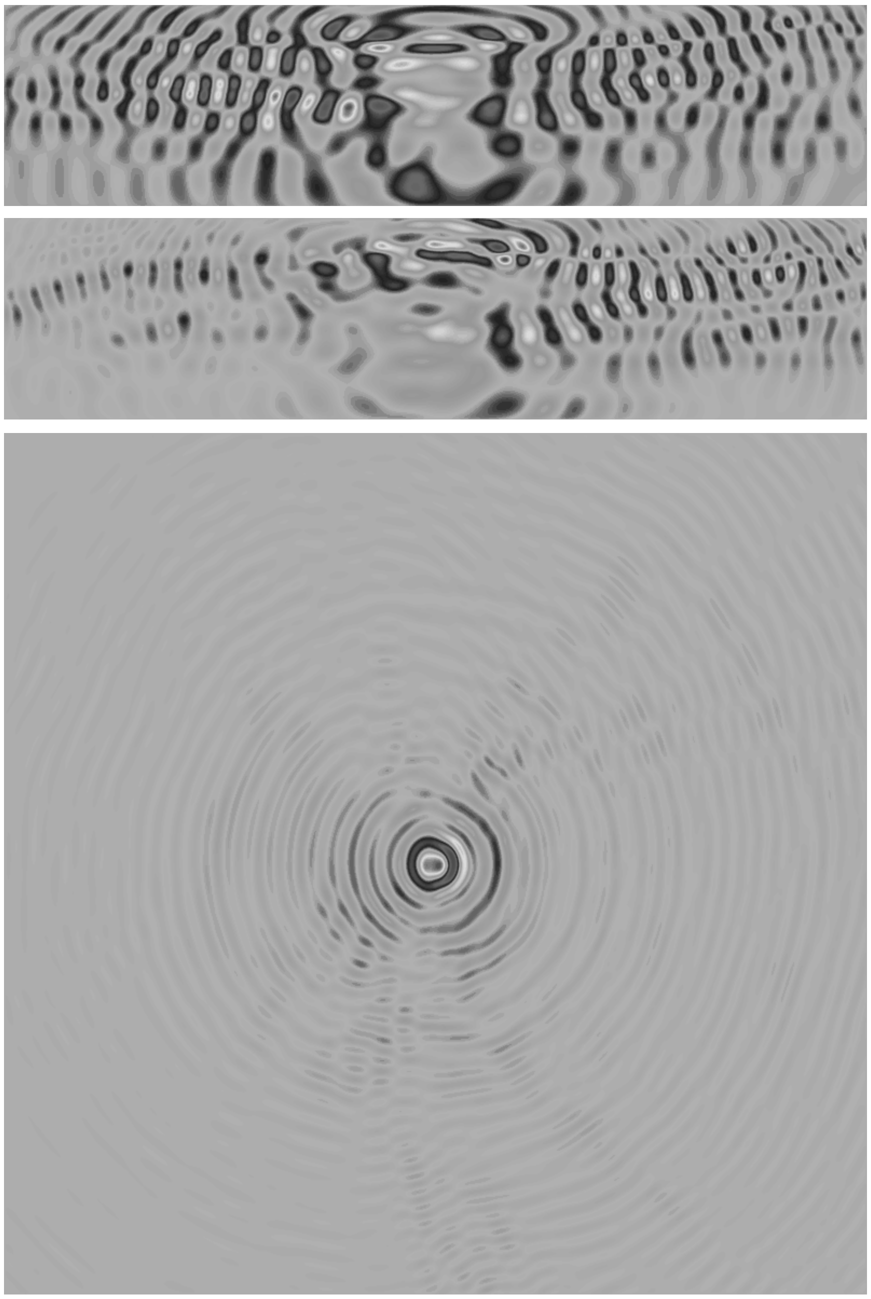


Figure 3.9: Three planes from an 8 Hz solution with the Overthrust model at its native resolution, $801 \times 801 \times 187$, with a single localized shot at the center of the x_1x_2 plane at a depth of 456 m: (top) a x_2x_3 plane near $x_1 = 14$ km, (middle) an x_1x_3 plane near $x_2 = 14$ km, and (bottom) an x_1x_2 plane near $x_3 = 0.9$ km.

	number of processes				
	128	256	512	1024	2048
Hz	3.175	4	5.04	6.35	8
grid	$319\times319\times75$	$401\times401\times94$	$505\times505\times118$	$635\times635\times145$	$801\times801\times187$
setup	48.40 (46.23)	66.33 (63.41)	92.95 (86.90)	130.4 (120.6)	193.0 (175.2)
apply	1.87 (4.26)	2.20 (5.11)	2.58 (9.61)	2.80 (13.3)	3.52 (24.5)
3 digits	42	44	42	39	40
4 digits	54	57	57	58	58
5 digits	63	69	70	68	72

Table 3.2: Convergence rates and timings (in seconds) on TACC's Lonestar for the SEG/EAGE Overthrust model, where timings in parentheses do not make use of selective inversion. All cases used a complex double-precision second-order finite-difference stencil with five grid spacings for all PML (with a magnitude of 7.5), and a damping parameter of 2.25π. The preconditioner was configured with four planes per panel and eight processes per node. The 'apply' timings refer to a single application of the preconditioner to four right-hand sides.

field, and

- avoiding unnecessary communication by distributing right-hand sides in a manner which conforms to the distributions of the multifrontal factorizations of the subdomain auxiliary problems.

The results presented in this chapter are admittedly quite conservative in their usage of ten grid points per shortest wavelength, and, in fact, several experiments were withheld which aggressively solved the Overthrust model at a frequency of 30 Hz over a $1201 \times 1201 \times 287$ grid using a second-order finite-difference stencil. Although the pollution error [15] was almost certainly quite high, we note that the number of iterations required for convergence was essentially the same as for the 8 Hz example which we described in detail, and that the pollution effects can be greatly mitigated by switching to spectral elements [123, 124] without a significant increase in computational cost.

Chapter 4

A compressed sweeping preconditioner

We have shown that the sweeping preconditioner requires $O(\gamma^2 N^{4/3})$ work in its setup phase and $O(\gamma N \log N)$ storage space, where γ is the number of grid points used for artificial PML. Since γ appears to need to grow roughly logarithmically with the number of wavelengths spanned by the domain [98, 120], it would be worthwhile to find a means of lessening its impact on the memory requirements of the sweeping preconditioner. We will shortly see that the *translation invariance* of the Green's functions for homogeneous free-space time-harmonic wave equations provides theoretical support for such a compression scheme.

Rather than listing the free-space Green's functions for the most important classes of time-harmonic wave equations and showing that their translation invariance is self-evident, we will instead consider the translation invariance as a direct result of the lack of a frame of reference for a problem posed over an infinite homogeneous domain. We will use the 3D Helmholtz Green's function as an example, but it is important to keep in mind that the techniques generalize to other time-harmonic wave equations.

4.1 Theory

Consider evaluating the Green's function for the free-space 3D Helmholtz equation,

$$G(x,y) = \begin{cases} \frac{\exp(i\omega|x-y|)}{4\pi|x-y|}, & x \neq y, \\ +\infty, & x = y, \end{cases}$$

with x and y restricted to a set of points evenly spaced over a line segment, say $x, y \in \{jh\hat{\mathbf{e}}\}_{j=0}^{q-1} \subset \mathbb{R}^3$, where $h > 0$ and $\hat{\mathbf{e}} \in \mathbb{R}^3$ is an arbitrary unit vector. Despite the fact that there are q^2 different choices for the pair (x, y), $|x - y|$ can only take on q different values, namely, $\{0, h, \ldots, (q-1)h\}$ (and thus G can also only take on at most q unique values). This redundancy is primarily due to the *translation invariance* of G, i.e.,

$$G(x,y) = G(x+t, y+t) \;\; \forall\, t \in \mathbb{R}^3.$$

Our next step will be to discuss situations where G is only translation invariant in a particular direction.

Lemma 4.1.1. *Let $G : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C} \cup \{\infty\}$ be invariant under translation in some direction $\hat{\mathbf{e}} \in \mathbb{R}^d$. That is to say,*

$$G(x,y) = G(x+t, y+t) \;\; \forall\, x, y \in \mathbb{R}^3 \text{ and } t \propto \hat{\mathbf{e}}.$$

If we extrude two points $u, v \in \mathbb{R}^d$ with spacing $h \in \mathbb{R}$ to form the sets $U = \{u + kh\hat{\mathbf{e}}\}_{k=0}^{q-1}$ and $V = \{v + \ell h\hat{\mathbf{e}}\}_{\ell=0}^{q-1}$, then G restricted to $U \times V$ can take on at most $2q - 1$ unique values.

If we further assume that G satisfies the mirror symmetry condition

$$G(u + kh\hat{\mathbf{e}}, v) = G(u - kh\hat{\mathbf{e}}, v), \quad k = 0, \ldots, q-1,$$

then the image of $U \times V$ through G consists of at most q unique values.

Proof. Due to the translation invariance of G in the direction $\hat{\mathbf{e}}$,

$$G(u + kh\hat{\mathbf{e}}, v + \ell h\hat{\mathbf{e}}) = G(u + (k - \ell)h\hat{\mathbf{e}}, v),$$

and so $-q < k - \ell < q$ implies the first result.

Now suppose that $G(u + hk\hat{\mathbf{e}}, v) = G(u - hk\hat{\mathbf{e}}, v)$ for $k = 0, \ldots, q - 1$. Then

$$G(u + (k - \ell)h\hat{\mathbf{e}}, v) = G(u + |k - \ell|h\hat{\mathbf{e}}, v),$$

where $0 \leq |k - \ell| < q$. $\square$

Remark 4.1.1. The *mirror symmetry condition*,

$$G(u + kh\hat{\mathbf{e}}, v) = G(u - kh\hat{\mathbf{e}}, v),$$

is useful in many cases. Consider the function $G(x, y) = e^{i\omega|x-y|}$, where $x, y \in \mathbb{R}^3$, and suppose that $u - v \perp \hat{\mathbf{e}}$.

Lemma 4.1.2. *Let $G : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C} \cup \{\infty\}$ be invariant under translation in some direction $\hat{\mathbf{e}} \in \mathbb{R}^d$. Then, for any $\{(u_i, v_i)\}_{i=0}^{m-1} \subset \mathbb{R}^d \times \mathbb{R}^d$, the $m \times q^2$ matrix*

$$\mathbf{K}(i, j_1 + qj_2) = G(u_i + j_1 h\hat{\mathbf{e}}, v_i + j_2 h\hat{\mathbf{e}}), \quad 0 \leq i < m,\ 0 \leq j_1, j_2 < q,$$

has at most $2q - 1$ unique columns.

If we also assume that G satisfies the mirror symmetry condition

$$G(u_i + hk\hat{\mathbf{e}}, v_i) = G(u_i - hk\hat{\mathbf{e}}, v_i), \ \ \forall\, 0 \leq i < m, 0 \leq k < q,$$

then $\mathbf{K}$ has at most q unique columns.

Proof. Using translation invariance,

$$\mathbf{K}(i, j_1 + qj_2) = G(u_i + (j_1 - j_2)h\hat{\mathbf{e}}, v_i),$$

and thus, since $-q < j_1 - j_2 < q$, there are at most $2q - 1$ unique columns.

If we then assume that $G(u_i + hk\hat{\mathbf{e}}, v_i) = G(u_i - hk\hat{\mathbf{e}}, v_i)$ for each $i = 0, \dots, m-1$ and $k = 0, \dots, q-1$, then

$$\mathbf{K}(i, j_1 + qj_2) = G(u_i + |j_1 - j_2|h\hat{\mathbf{e}}, v_i),$$

where $0 \leq |j_1 - j_2| < q$. $\square$

Remark 4.1.2. Although Lemma 4.1.2 may seem overly specific, translation invariance in a particular direction arises naturally for Green's functions of constant-coefficient problems posed over unbounded domains which are invariant in a particular direction, such as the infinitely tall rectangle $[0, 1] \times (-\infty, +\infty)$. We will now show how the *method of mirror images* [77] can be used to extend these ideas to semi-infinite domains, e.g., $[0, 1] \times (-\infty, 0]$ with a zero Dirichlet boundary condition imposed over $[0, 1] \times \{0\}$. Once we have done so, we will have a strong argument for the compressibility of the frontal matrices arising from the semi-infinite subdomain auxiliary problems posed by the sweeping preconditioner.

Example 4.1.1. Suppose that $G : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{C} \cup \{\infty\}$ is a Green's function representing the potential generated at some point $x \in \mathbb{R}^3$ due to a point source at $y \in \mathbb{R}^3$, and that G is both translation invariant and mirror symmetric (even) with respect to its last coordinate, i.e., if $u, v \in \mathbb{R}^2 \times \{0\}$, then

$$G(u + \gamma\hat{\mathbf{e}}_3, v) = G(u - \gamma\hat{\mathbf{e}}_3, v).$$

Then we can use the original Green's function, G, to construct a half-space Green's function, say $\tilde{G}$, with domain $(\mathbb{R}^2 \times (-\infty, 0])^2$ and boundary condition $\tilde{G}(x, y) = 0$ when $x \in \mathbb{R}^2 \times \{0\}$. In particular, we may set

$$\tilde{G}(u, v) = G(u, v) - G(u, R(v)),$$

where $R(v)$ is the reflection of v over the plane $\mathbb{R}^2 \times \{0\}$ and represents the location of an artificial charge which counteracts the charge located at v over the plane $\mathbb{R}^2 \times \{0\}$.

The contribution of the conceptual *mirror image charge*, $-G(u, R(v))$, is often referred to as a *reflection* of the charge located at v off of the boundary located at $\mathbb{R}^2 \times \{0\}$ (see Figure 4.1). In general, the introduction of the Dirichlet boundary condition destroys translation invariance in the direction normal to the boundary condition, but we will now show that an analogue of Lemma 4.1.2 still holds.

Lemma 4.1.3. *Let a function* $G : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C} \cup \{\infty\}$ *be invariant under translation in the direction* $\hat{\mathbf{e}} \in \mathbb{R}^d$ *and let* $\{(u_i, v_i)\}_{i=0}^{m-1} \subset \mathbb{R}^d \times \mathbb{R}^d$, *where each* v_i *is orthogonal to* $\hat{\mathbf{e}}$. *If we then define*

$$\tilde{G}(x, y) = G(x, y) - G(x, R(y)),$$

where $R(y) = y - 2\hat{\mathbf{e}}(\hat{\mathbf{e}}, y)$, *then the* $m \times q^2$ *matrix*

$$\mathbf{K}(i, j_1 + qj_2) = \tilde{G}(u_i + j_1 h\hat{\mathbf{e}}, v_i + j_2 h\hat{\mathbf{e}}), \quad 0 \le i < m,\ 0 \le j_1, j_2 < q,$$

has a rank of at most $3q - 2$.

If G *also satisfies the mirror symmetry condition*

$$G(u_i + hj\hat{\mathbf{e}}, v_i) = G(u_i - hj\hat{\mathbf{e}}, v_i), \quad 0 \le i < m, 0 \le j < q,$$

then $\mathbf{K}$ *has a rank of at most* $2q - 1$.

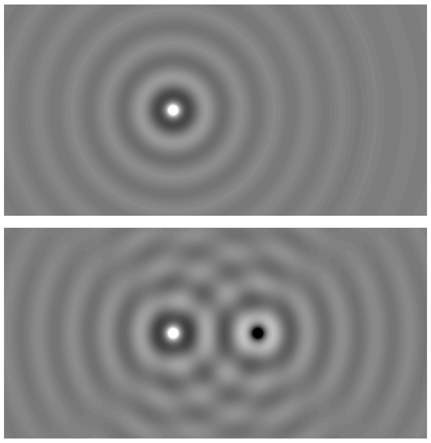


Figure 4.1: (Top) The real part of a 2D slice of the potential generated by a source in the obvious location and (bottom) the same potential superimposed with that of a mirror-image charge. The potential is identically zero along the vertical line directly between the two charges.

Proof. By definition,

$$\mathbf{K}(i, j_1 + qj_2) = G(u_i + j_1 h\hat{\mathbf{e}}, v_i + j_2 h\hat{\mathbf{e}}) - G(u_i + j_1 h\hat{\mathbf{e}}, R(v_i + j_2 h\hat{\mathbf{e}})),$$

and, since $v_i \perp \hat{\mathbf{e}}$, $R(v_i) = v_i$, and we find that

$$\mathbf{K}(i, j_1 + qj_2) = G(u_i + (j_1 - j_2)h\hat{\mathbf{e}}, v_i) - G(u_i + (j_1 + j_2)h\hat{\mathbf{e}}, v_i).$$

The columns of $\mathbf{K}$ are therefore linear combinations of the vectors

$$p_j(i) = G(u_i + jh\hat{\mathbf{e}}, v_i),$$

where $-q < j < 2q - 1$, and so we have established the first result.

If we additionally assume that

$$G(u_i + hj\hat{\mathbf{e}}, v_i) = G(u_i - hj\hat{\mathbf{e}}, v_i), \;\; \forall\, 0 \le i < m, 0 \le j < q,$$

then

$$\mathbf{K}(i, j_1 + qj_2) = G(u_i + |j_1 - j_2|h\hat{\mathbf{e}}, v_i) - G(u_i + (j_1 + j_2)h\hat{\mathbf{e}}, v_i).$$

It follows that the columns of $\mathbf{K}$ are again linear combinations of the vectors

$$p_j(i) = G(u_i + jh\hat{\mathbf{e}}, v_i),$$

but now $0 \leq j < 2q - 1$, which yields the second result. $\square$

Example 4.1.2. Consider the Green's function $G : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{C} \cup \{\infty\}$ for the free-space 3D Helmholtz equation,

$$G(x, y) = \begin{cases} \frac{\exp(i\omega|x-y|)}{4\pi|x-y|}, & x \neq y, \\ +\infty, & x = y, \end{cases}$$

and set $\tilde{G}(x, y) = G(x, y) - G(x, R(y))$, where $R(y) = y - 2\hat{\mathbf{e}}_3(\hat{\mathbf{e}}_3, y)$. Then $\tilde{G}$ restricted to $(\mathbb{R}^2 \times (-\infty, 0])^2$ is a Green's function for the constant-coefficient half-space Helmholtz problem,

$$\left[-\Delta - \omega^2\right] u(x) = f(x),$$

where $x \in \mathbb{R}^2 \times (-\infty, 0]$ and $u(x) = 0$ when x lies on the half-space boundary, $\mathbb{R}^2 \times \{0\}$.

Now suppose that we are interested in representing $\tilde{G}$ over a portion of a plane lying against the half-space boundary, say $\{0\} \times [0, 1] \times [0, \lambda]$ (see Figure 4.2). If we let $\{u_i\}_{i=0}^{s-1}$ be an arbitrary collection of points from $\{0\} \times [0, 1] \times \{0\}$, and set $\hat{\mathbf{e}} = \hat{\mathbf{e}}_3$, then for any integer $q > 1$, Lemma 4.1.3 implies that the rank of the $s^2 \times q^2$ matrix

$$\mathbf{K}(i_1 + si_2, j_1 + qj_2) = \tilde{G}(u_{i_1} + \frac{j_1\lambda}{q-1}\hat{\mathbf{e}}_3, u_{i_2} + \frac{j_2\lambda}{q-1}\hat{\mathbf{e}}_3),$$

where $0 \leq i_1, i_2 < s$ and $0 \leq j_1, j_2 < q$, is at most $3q - 2$.

Furthermore, since every vector u_i is orthogonal to $\hat{\mathbf{e}}_3$, so is every linear combination of the u_i's. Thus,

$$\|(u_{i_1} + \gamma\hat{\mathbf{e}}_3) - u_{i_2}\|_2 = \sqrt{|u_{i_1} - u_{i_2}|^2 + |\gamma|^2},$$

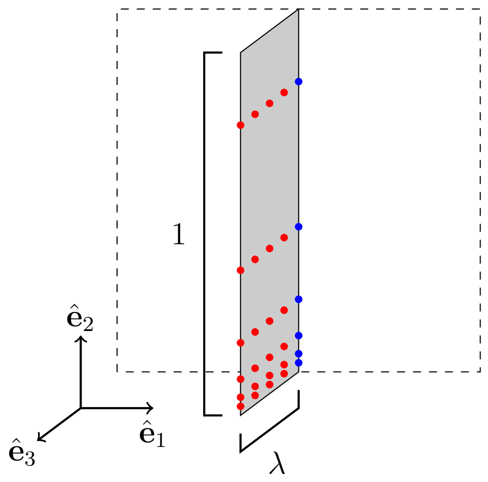


Figure 4.2: A set of six unevenly spaced points (in blue) along the line segment $\{0\} \times [0,1] \times \{0\}$ and their four evenly spaced translations (in red) in the direction $\hat{\mathbf{e}}_3$.

and so the sign of γ is irrelevant when evaluating $\tilde{G}(u_{i_1}+\gamma\hat{\mathbf{e}}_3, u_{i_2})$. We therefore satisfy the mirror symmetry conditions for Lemma 4.1.3 and find that the rank of $\mathbf{K}$ is at most $2q-1$.

Remark 4.1.3. Although our previous example used identical source and target sets, notice that this is not a requirement of Lemma 4.1.3.

Remark 4.1.4. The $s^2 \times q^2$ matrix $\mathbf{K}$ formed in Example 4.1.2 is somewhat of an unconventional way to represent a discrete Green's function: the $(j_1 + qj_2)$'th column of $\mathbf{K}$ stores the potentials generated over s points on the line $\{0\} \times [0,1] \times \{j_1 h\}$ for each of s different point sources lying on the line $\{0\}\times[0,1]\times\{j_2 h\}$, resulting in s^2 entries per column. It is much more common to instead work with the square $qs \times qs$ matrix

$$\tilde{\mathbf{G}}(i,j) = \tilde{G}(x_i, x_j),$$

where $\{x_i\}_{i=0}^{qs-1}$ is an enumeration of the entire set of samples of the plane segment $\{0\} \times [0, 1] \times [0, \lambda]$. This representation is satisfying because $\tilde{\mathbf{G}}$ can be interpreted as mapping a charge distribution over the plane segment to its resulting potential field over the same region.

It is natural to now ask how a low-rank representation of $\mathbf{K}$ can be translated into a data-sparse representation of $\tilde{\mathbf{G}}$. The following theorem makes use of the fact that, with a particular ordering of $\tilde{\mathbf{G}}$, $\tilde{\mathbf{G}}$ is a $q \times q$ matrix of blocks of size $s \times s$, where each block is a column of $\mathbf{K}$. The result is that $\tilde{\mathbf{G}}$ may be represented as the sum of $O(q)$ Kronecker-product matrices [128, 129].

Definition 4.1.1 (Kronecker-Zehfuss product)**.** Given an $m_1 \times m_2$ matrix $\mathbf{Y}$ and an $n_1 \times n_2$ matrix $\mathbf{Z}$, the *Kronecker product* $\mathbf{Y} \otimes \mathbf{Z}$, sometimes also called the *Zehfuss product*, is defined as the $m_1 n_1 \times m_2 n_2$ matrix

$$\mathbf{Y} \otimes \mathbf{Z} = \begin{bmatrix} \psi_{0,0}\mathbf{Z} & \cdots & \psi_{0,m_2-1}\mathbf{Z} \\ \vdots & \ddots & \vdots \\ \psi_{m_1-1,0}\mathbf{Z} & \cdots & \psi_{m_1-1,m_2-1}\mathbf{Z} \end{bmatrix},$$

where $\psi_{i,j}$ is the (i, j) entry of $\mathbf{Y}$.

Definition 4.1.2. The "vec" operator takes an $m \times n$ array X and produces a one-dimensional array of length mn, where x is formed by successively stacking the columns of X. That is, the (i, j) entry of X becomes the $i + jm$ entry of x, and we write $x = \text{vec}(X)$.

Remark 4.1.5. The previous definition was purposefully written to avoid specifying that the result of the vec operator must be a vector, as it is often useful to reshape two-dimensional arrays whose entries are more general than scalar values. In particular, the following theorem uses the vec operator on a two-dimensional array of points from $\mathbb{R}^d$.

Theorem 4.1.4. *Let $G : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{C} \cup \{\infty\}$ be invariant under translation in the direction $\hat{\mathbf{e}} \in \mathbb{R}^d$ and define*

$$\tilde{G}(x, y) = G(x, y) - G(x, R(y)),$$

where $R(y) = y - 2\hat{\mathbf{e}}(\hat{\mathbf{e}}, y)$. Furthermore, let $\{a_i\}_{i=0}^{s_a-1}, \{b_i\}_{i=0}^{s_b-1} \subset \mathbb{R}^d$ where each b_i is orthogonal to $\hat{\mathbf{e}}$, and define an array of qs_a target points, $x = vec(X)$, and an array of qs_b source points, $y = vec(Y)$, where $X(i, j) = a_i + jh\hat{\mathbf{e}}$ and $Y(i, j) = b_i + jh\hat{\mathbf{e}}$. Then the $qs_a \times qs_b$ matrix

$$\tilde{\mathbf{G}}(i, j) = \tilde{G}(x_i, y_j)$$

can be represented as a sum of Kronecker products,

$$\tilde{\mathbf{G}} = \sum_{t=0}^{3q-3} \mathbf{Y}_t \otimes \mathbf{Z}_t,$$

where each $\mathbf{Y}_t$ is $q \times q$ and each $\mathbf{Z}_t$ is $s_a \times s_b$.

If we further assume the mirror symmetry conditions

$$G(a_i + hk\hat{\mathbf{e}}, b_j) = G(a_i - hk\hat{\mathbf{e}}, b_j), \;\; \forall\, 0 \le i < s_a, 0 \le j < s_b, 0 \le k < q,$$

then $\tilde{\mathbf{G}}$ can be represented as the sum of only $2q-1$ Kronecker product matrices.

Proof. Partition $\tilde{G}$ as

$$\tilde{\mathbf{G}} = \begin{bmatrix} \tilde{\mathbf{G}}_{0,0} & \cdots & \tilde{\mathbf{G}}_{0,q-1} \\ \vdots & \ddots & \vdots \\ \tilde{\mathbf{G}}_{q-1,0} & \cdots & \tilde{\mathbf{G}}_{q-1,q-1} \end{bmatrix},$$

where each submatrix $\tilde{\mathbf{G}}_{i,j}$ is $s_a \times s_b$. Due to the ordering imposed upon $\tilde{\mathbf{G}}$, the matrix $\mathbf{K}$ formed such that its $i + qj$'th column is equal to $\text{vec}(\tilde{\mathbf{G}}_{i,j})$ satisfies the conditions for Lemma 4.1.3, which shows that the rank of $\mathbf{K}$ is at most $3q-2$. That is, we may decompose $\mathbf{K}$ as

$$\mathbf{K} = \sum_{t=0}^{3q-3} \mathbf{z}_t \mathbf{y}_t^T.$$

Since each column of $\mathbf{K}$ corresponds to some $s_a \times s_b$ block of $\tilde{\mathbf{G}}$, each rank-one contribution to $\mathbf{K}$ corresponds to a Kronecker product contribution to $\tilde{\mathbf{G}}$, say $\mathbf{Y}_t \otimes \mathbf{Z}_t$, where $\mathbf{Y}_t$ is $q \times q$, $\mathbf{Z}_t$ is $s_a \times s_b$, $\text{vec}(\mathbf{Y}_t) = \mathbf{y}_t$, and $\text{vec}(\mathbf{Z}_t) = \mathbf{z}_t$. This shows the first result.

Now suppose that

$$G(a_i + hk\hat{\mathbf{e}}, b_j) = G(a_i - hk\hat{\mathbf{e}}, b_j), \ \ \forall\, 0 \le i < s_a, 0 \le j < s_b, 0 \le k < q.$$

Then $\mathbf{K}$ satisfies the second condition of Lemma 4.1.3 and has a rank of at most $2q-1$. $\square$

Lemma 4.1.5. *For any matrices $\mathbf{Y}$ and $\mathbf{Z}$ and conforming vector $\mathbf{x}$,*

$$(\mathbf{Y} \otimes \mathbf{Z})\mathbf{x} = vec(\mathbf{Z}\mathbf{X}\mathbf{Y}^T),$$

where $\mathbf{X}$ is defined such that $vec(\mathbf{X}) = \mathbf{x}$.

Proof. Let $\mathbf{Y}$ be $m \times n$. Then, by definition of the Kronecker product,

$$(\mathbf{Y} \otimes \mathbf{Z})\mathbf{x} = \sum_{k=0}^{n-1} \begin{bmatrix} \mathbf{Z}\mathbf{x}_k \psi_{0,k} \\ \mathbf{Z}\mathbf{x}_k \psi_{1,k} \\ \vdots \\ \mathbf{Z}\mathbf{x}_k \psi_{m-1,k} \end{bmatrix},$$

where $\psi_{i,j}$ is the (i,j) entry of $\mathbf{Y}$. If we then denote the $i'th$ row of $\mathbf{Y}$ by $\hat{\mathbf{y}}_i$ and define $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{n-1}]$, so that $\text{vec}(\mathbf{X}) = \mathbf{x}$, we see that

$$(\mathbf{Y} \otimes \mathbf{Z})\mathbf{x} = \begin{bmatrix} \mathbf{Z}\mathbf{X}\hat{\mathbf{y}}_0^T \\ \mathbf{Z}\mathbf{X}\hat{\mathbf{y}}_1^T \\ \vdots \\ \mathbf{Z}\mathbf{X}\hat{\mathbf{y}}_{m-1}^T \end{bmatrix} = \text{vec}(\mathbf{Z}\mathbf{X}\mathbf{Y}^T).$$

□

Corollary 4.1.6. *Given a $q \times q$ matrix $\mathbf{Y}$, an $s_a \times s_b$ matrix $\mathbf{Z}$, and a vector $\mathbf{x}$ of length qs_b, the product $(\mathbf{Y} \otimes \mathbf{Z})\mathbf{x}$ can be formed with $2q^2 \min\{s_a, s_b\} + 2qs_as_b$ total additions and multiplications.*

Proof. The product of an $m \times k$ matrix and a $k \times n$ matrix can be computed with $2mnk$ operations and we are free to form $\mathbf{Z}\mathbf{X}\mathbf{Y}^T$ as either $\mathbf{Z}(\mathbf{X}\mathbf{Y}^T)$ or $(\mathbf{Z}\mathbf{X})\mathbf{Y}^T$. □

Theorem 4.1.7. *Suppose a $qs_a \times qs_b$ matrix $\tilde{\mathbf{G}}$ satisfies the first set of conditions for Theorem 4.1.4. Then the Kronecker product representation of $\tilde{\mathbf{G}}$ only requires the storage of $(3q-2)(q^2 + s_as_b)$ scalars, and this representation can be be used to linearly transform a vector with $(3q-2)(2q^2 \min\{s_a, s_b\} + 2qs_as_b)$ floating-point operations.*

If $\tilde{\mathbf{G}}$ satisfies the second set of conditions for Theorem 4.1.4, then only $(2q-1)(q^2 + s_as_b)$ scalars of storage are required, and the operator may be applied in this form with $(2q-1)(2q^2 \min\{s_a, s_b\} + 2qs_as_b)$ total additions and multiplications.

Proof. This is essentially an immediate consequence of Theorem 4.1.4 and Corollary 4.1.6; we need only recognize that the each Kronecker product $\mathbf{Y}_t \otimes \mathbf{Z}_t$ only requires storage of the $q \times q$ matrix $\mathbf{Y}_t$ and the $s_a \times s_b$ matrix $\mathbf{Z}_t$. □

Remark 4.1.6. It is now important to compare the results from Theorem 4.1.7 to the case where $\tilde{\mathbf{G}}$ is simply stored as a dense matrix, which would require $q^2 s_a s_b$ unit of storage and $2q^2 s_a s_b$ floating-point operations for naïvely mapping a vector. In particular, let us consider the case where $q = s_a = s_b$. Then the memory requirement for the Kronecker product scheme, assuming that the second set of conditions for 4.1.4 are satisfied, is essentially $4q^3$ entries of storage versus the q^4 entries required for the standard dense storage scheme, which is clearly a factor of $q/4$ compression. On the other hand, if $q \ll s_a, s_b$, then we could expect a factor of $q/2$ compression.

This lossless compression scheme comes at somewhat of a price. A direct comparison of the work required for performing the matrix-vector multiplications shows that, as long as $q \leq \min\{s_a, s_b\}$, the Kronecker product scheme may require as much as four times the work as the dense storage scheme. On the other hand, the Kronecker product approach makes use of matrix-matrix multiplication instead of matrix-vector multiplications and can therefore take better advantage of cache (i.e., it can use level 3 BLAS operations instead of level 2). A proper sequential implementation of the Kronecker product scheme should therefore not be significantly slower than the standard scheme.

Example 4.1.3. Suppose that we are interested in representing the Green's function for the constant-coefficient 3D Helmholtz equation,

$$\left[-\Delta - \omega^2\right]_x G_D(x,y) = \delta(x-y), \text{ in } [0,1] \times [0,1] \times (-\infty, \infty),$$

with zero Dirichlet boundary conditions on the finite boundaries and radiation conditions posed in the remaining two directions, where the subscript x implies that the operator $-\Delta - \omega^2$ acts on the x variable of the Green's function $G_D(x,y)$. Due to Dirichlet boundary conditions being posed on both sides of the domain in the first two dimensions, mirror imaging techniques for constructing the Green's function for this problem from the free-space Green's function would require an infinite summation, as the reflections from the left wall would reflect off the right wall, which would then reflect off of the left wall, ad infinitum.

We will avoid discussion of the convergence of such summations [21] and simply recognize that the resulting Green's function will remain translation invariant in the third coordinate. Then we may use the method of mirror imaging in the third dimension to construct another Green's function, say $\tilde{G}_D(x,y) = G_D(x,y) - G_D(x, R(y))$, for the problem

$$\left[-\Delta - \omega^2\right]_x \tilde{G}_D(x,y) = \delta(x-y), \text{ in } [0,1] \times [0,1] \times (-\infty, 0],$$

where $R(y) = y - 2\hat{\mathbf{e}}_3(\hat{\mathbf{e}}_3, y)$ is responsible for reflecting y over the newly imposed zero Dirichlet boundary condition on $[0,1] \times [0,1] \times \{0\}$. If x and y are both perpendicular to $\hat{\mathbf{e}}_3$, then G_D retains the free-space Helmholtz property that $G_D(x + \gamma\hat{\mathbf{e}}_3, y) = G_D(x - \gamma\hat{\mathbf{e}}_3, y)$ for any $\gamma \in \mathbb{R}$, and we see that Theorem 4.1.7 can be invoked to show that samples of $\tilde{G}_D$ over a grid which is uniform in the $\hat{\mathbf{e}}_3$ direction are compressible.

Theorem 4.1.8. *The Green's function, say $\tilde{G}_D$, for the Helmholtz problem*

$$\left[-\Delta - \omega^2\right]_x \tilde{G}_D(x,y) = \delta(x-y), \text{ in } [0,1] \times [0,1] \times (-\infty, 0],$$

with zero Dirichlet boundary conditions on the finite boundaries and the Sommerfeld radiation condition in the remaining direction, is compressible in the following sense. For any points $\{a_i\}_{i=0}^{s-1} \subset \hat{\mathbf{e}}_3^{\perp}$, if we define the array of points $x = vec(X)$, where X is the $s \times q$ matrix defined entrywise as $X(i,j) = a_i + jh\hat{\mathbf{e}}_3$, the matrix

$$\tilde{\mathbf{G}}_D(i,j) = \tilde{G}_D(x_i, x_j)$$

has the Kronecker product decomposition

$$\tilde{\mathbf{G}} = \sum_{t=0}^{2q-2} \mathbf{Y}_t \otimes \mathbf{Z}_t,$$

where $\mathbf{Y}_t$ is $q \times q$ and $\mathbf{Z}_t$ is $s \times s$. Furthermore, this representation requires only $(2q-1)(q^2+s^2)$ units of storage and can be used to linearly transform a vector with $(2q-1)(2q^2s+2qs^2)$ units of work.

4.2 Interpretation, application, and parallelization

We have just provided a significant amount of theoretical support for a compression scheme for the frontal matrices formed during the multifrontal factorization of each auxiliary problem posed by the sweeping preconditioner. In particular, the inverse of the Schur complement of each frontal matrix, say

$$S_s^{-1} = [L(\mathcal{D}_s, \mathcal{D}_s)]^{-T}[D(\mathcal{D}_s, \mathcal{D}_s)]^{-1}[L(\mathcal{D}_s, \mathcal{D}_s)]^{-1},$$

is closely related to the Green's function resulting from the subproblem posed over the subdomain covered by supernode s and its descendants, but with zero Dirichlet boundary conditions posed over the artificial boundaries introduced by the ancestor separators (consider Figure 3.3); Theorem 4.1.8 was specifically designed to handle these artificial Dirichlet boundary conditions.

There are several points worth discussing:

- The theory from the previous section demonstrates precise cases where the discrete Green's function can be losslessly compressed as a sum of Kronecker products, but we will make use of a thresholded singular value decomposition in order to find an approximate compression in more general scenarios (e.g., heterogeneous media and PML boundary conditions).

- We must modify our multifrontal scheme to directly compute each front's inverse Schur complement, and then compress this inverse. Unfortunately, much less can be said about the compression of the bottom-left quadrant of the fronts.

Our Cholesky-like multifrontal LDL^T factorization will thus need to be modified in order to perform a *block* LDL^T factorization (see Algorithm 4.1), and the top-left and bottom-left quadrants of each front will then need to be shuffled into a different form and then compressed via a singular value decomposition (see Algorithm 4.2), where each rank-one contribution to the permuted matrix is a Kronecker-product contribution to the original matrix. We can then alter triangular solves with an uncompressed block LDL^T factorization (shown in Algorithm 4.3) so that each operation which originally involved a large dense matrix multiplication is replaced with the application of a sum of Kronecker product matrices based upon Lemma 4.1.5.

Algorithm 4.1: Block LDL^T multifrontal factorization

```
Input: Symmetric matrix, $A$, elimination tree, $\mathcal{E}$, and root node, $s$
Output: a block $LDL^T$ factorization of $A$
1 parallel foreach $c \in \mathcal{C}(s)$ do  Recurse($A$,$\mathcal{E}$,$c$)
2 $U_s :=$ zeros($|\mathcal{L}_s|, |\mathcal{L}_s|$)
3 foreach $c \in \mathcal{C}(s)$ do
      $\begin{pmatrix} A(\mathcal{D}_s,\mathcal{D}_s) & A(\mathcal{D}_s,\mathcal{L}_s) \\ A(\mathcal{L}_s,\mathcal{D}_s) & U_s \end{pmatrix} := \begin{pmatrix} A(\mathcal{D}_s,\mathcal{D}_s) & A(\mathcal{D}_s,\mathcal{L}_s) \\ A(\mathcal{L}_s,\mathcal{D}_s) & U_s \end{pmatrix} \Leftrightarrow U_c$
4 $S_s^{-1} := A(\mathcal{D}_s,\mathcal{D}_s)^{-1}$
5 $L(\mathcal{L}_s,\mathcal{D}_s) := A(\mathcal{L}_s,\mathcal{D}_s)$
6 $U_s := U_s - L(\mathcal{L}_s,\mathcal{D}_s) S_s^{-1} L(\mathcal{L}_s,\mathcal{D}_s)^H$
```

For example, Step 4 of Algorithm 4.3 and Step 2 of Algorithm 4.4 must each be replaced with an operation of the form

$$X(\mathcal{D}_s,:) := \sum_{t=0}^{r-1} (Y_t \otimes Z_t) X(\mathcal{D}_s,:),$$

where $\|S_s^{-1} - \sum_t Y_t \otimes Z_t\|_2 \leq \epsilon \|S_s^{-1}\|_2$.

Algorithm 4.2: Structured Kronecker product compression

```
   Input: $rs_a \times rs_b$ matrix $\mathbf{G}$ and tolerance, $\epsilon$
   Output: $r \times r$ matrices $\{Y_t\}_{t=0}^{r-1}$ and $s_a \times s_b$ matrices $\{Z_t\}_{t=0}^{r-1}$ such
           that $\|\mathbf{G} - \sum_t Y_t \otimes Z_t\|_2 \le \epsilon\|\mathbf{G}\|_2$
1  Form the $s_a s_b \times r^2$ matrix $\mathbf{K}$ such that
   $K(:, i{+}jr) = \mathrm{vec}(\mathbf{G}(is_a\!:\!(i{+}1)s_a{-}1, js_b\!:\!(j{+}1)s_b{-}1))$
2  Compute SVD of $K$, $K = U\Sigma V^H$
3  Remove every triplet $(\sigma, u, v)$ such that $\sigma < \epsilon\|A\|_2$ to form the
   truncated SVD, $K \approx \hat{U}\hat{\Sigma}\hat{V}^H = \sum_{t=0}^{r-1}(\sigma_t u_t)v_t^H$
4  parallel foreach $t = 0 : r-1$ do
5      Form $Y_t = \mathrm{vec}^{-1}(\overline{v_t})$
6      Form $Z_t = \mathrm{vec}^{-1}(\sigma u_t)$
7  end
```

Algorithm 4.3: Block LDL^T multifrontal forward solve

```
   Input: block $LDL^T$ factorization, $J$, right-hand side matrix of
          width $k$, $X$, elimination tree, $\mathcal{E}$, and root node, $s$
   Output: $X := D^{-1}L^{-1}X$
1  parallel foreach $c \in \mathcal{C}(s)$ do  Recurse($J$,$X$,$\mathcal{E}$,$c$)
2  $Z_s :=$ zeros$(|\mathcal{L}_s|, k)$
3  foreach $c \in \mathcal{C}(s)$ do  $\begin{pmatrix} X(\mathcal{D}_s,:) \\ Z_s \end{pmatrix} := \begin{pmatrix} X(\mathcal{D}_s,:) \\ Z_s \end{pmatrix} \Leftrightarrow Z_c$
4  $X(\mathcal{D}_s,:) := S_s^{-1}X(\mathcal{D}_s,:)$
5  $Z_s := Z_s - L(\mathcal{L}_s,\mathcal{D}_s)X(\mathcal{D}_s,:)$
```

Algorithm 4.4: Block LDL^T multifrontal backward solve

```
   Input: block $LDL^T$ factorization, $J$, right-hand side matrix of
          width $k$, $X$, elimination tree, $\mathcal{E}$, and root node, $s$
   Output: $X := L^{-T}X$
1  if $\mathcal{A}(s) \neq \emptyset$ then  $X(\mathcal{D}_s,:) := X(\mathcal{D}_s,:) - L(\mathcal{L}_s,\mathcal{D}_s)^H X(\mathcal{L}_s,:)$
2  $X(\mathcal{D}_s,:) := S_s^{-1}X(\mathcal{D}_s,:)$
3  parallel foreach $c \in \mathcal{C}(s)$ do  Recurse($J$,$X$,$\mathcal{E}$,$c$)
```

4.2.1 Mapping a single vector

For the moment, let us suppose that the submatrix $X(\mathcal{D}_s, :)$ has only a single column, which we will simply refer to as the vector b. Then we must perform the update

$$b := \sum_{t=0}^{r-1} (Y_t \otimes Z_t) b = \sum_{t=0}^{r-1} \text{vec}(Z_t W Y_t^T),$$

where $\text{vec}(W) = b$. But this is clearly equivalent to

$$\sum_{t=0}^{r-1} (Y_t \otimes Z_t) b = \text{vec}\left(\begin{pmatrix} Z_0, & \cdots & Z_{r-1} \end{pmatrix} \begin{pmatrix} WY_0^T \\ \vdots \\ WY_{r-1}^T \end{pmatrix} \right) \equiv \text{vec}(ZE), \qquad (4.1)$$

which we might form in the five steps shown in Algorithm 4.5. Notice that both the second and fourth steps can be performed with standard algorithms for parallel matrix-matrix multiplication, such as Scalable Universal Matrix Multiplication (SUMMA) [109, 127].

Algorithm 4.5: Mapping a vector with a sum of Kronecker products, $b := \sum_{t=0}^{r-1} (Y_t \otimes Z_t) b$

```
1 $W := \text{vec}^{-1}(b)$
2 $\tilde{E} := W[Y_0^T, \ldots, Y_{r-1}^T]$
3 Shuffle row-panel $\tilde{E}$ into column-panel $E$
4 $M := [Z_0, \ldots, Z_{r-1}]E$
5 $b := \text{vec}(M)$
```

4.2.2 Mapping several vectors

Let us now investigate an alternative to Equation (4.1) designed for simultaneously mapping several vectors via matrix-matrix multiplication. We will begin by generalizing the vec operator in order to keep our notation compact.

Definition 4.2.1. Given an $m \times qk$ matrix W, which we may partition as

$$W = \begin{pmatrix} W_0, & W_1, & \cdots & W_{k-1} \end{pmatrix},$$

where each W_j is $m \times q$, $\text{vec}(W, k)$ denotes the $mq \times k$ matrix whose column with index j is given by $\text{vec}(W_j)$. Furthermore, given an $mq \times k$ matrix B, when the dimensions of the result are clear, we may denote the unique $m \times qk$ matrix W such that $\text{vec}(W, k) = B$ as $\text{vec}^{-1}(B, k)$.

We will now let $W = \text{vec}^{-1}(B, k)$ be the unique $m \times qk$ matrix such that $\text{vec}(W, k) = B$. Then, if we define E_j based upon column j of B in the same manner as in Equation (4.1) and recall that $Z = [Z_0, Z_1, \ldots, Z_{r-1}]$, we may write

$$\sum_{t=0}^{r-1} (Y_t \otimes Z_t)B = [\text{vec}(ZE_0), \ldots, \text{vec}(ZE_{k-1})] = \text{vec}(ZE, k), \tag{4.2}$$

where

$$E = \begin{pmatrix} W_0 Y_0^T & \cdots & W_{k-1} Y_0^T \\ W_0 Y_1^T & \cdots & W_{k-1} Y_1^T \\ \vdots & \ddots & \vdots \\ W_0 Y_{r-1}^T & \cdots & W_{k-1} Y_{r-1}^T \end{pmatrix}, \tag{4.3}$$

which is a permuted version of the matrix

$$\tilde{E} = \begin{pmatrix} W_0 \\ W_1 \\ \vdots \\ W_{k-1} \end{pmatrix} \begin{pmatrix} Y_0^T & Y_1^T & \cdots & Y_{r-1}^T \end{pmatrix} \equiv \tilde{W} \tilde{Y}^T. \tag{4.4}$$

We may thus form the update $B := \sum_{t=0}^{r-1} (Y_t \otimes Z_t)B$ in an analogous manner as in the previous subsection (see Algorithm 4.6). As before, the entire process involves two dense matrix-matrix multiplications (Steps 2 and 4) and several permutations.

Algorithm 4.6: Mapping a matrix with a sum of Kronecker products, $B := \sum_{t=0}^{r-1}(Y_t \otimes Z_t)B$

```
1 $\tilde{W} := \mathrm{vec}^{-1}(B, k)$
2 $\tilde{E} := \tilde{W}[Y_0^T, \ldots, Y_{r-1}^T]$
3 Shuffle $\tilde{E}$ into $E$
4 $M := ZE$
5 $B := \mathrm{vec}(M, k)$
```

4.3 Results

We now reconsider the waveguide model discussed in the previous chapter using $250 \times 250 \times 250$ and $500 \times 500 \times 500$ second-order finite-difference stencils with frequencies of 18.75 Hz and 37.5 Hz, respectively. Table 4.1 demonstrates that our compression scheme is quite successful for the discrete Green's functions (the diagonal blocks) produced within our modified multifrontal factorization. In particular, for the 250^3 grid, a factor of 16.41 memory compression was measured for the top-most panel of the waveguide (which consists of 14 planes), whereas the compression ratio was 12.51 for the 500^3 grid. It is perhaps unsurprising that the compression algorithm is less successful when applied in an ad-hoc manner to the connectivity between the supernodes, as these matrices do not correspond to discrete Green's functions and therefore do not enjoy any obvious benefits from the (approximate) translation invariance of the free-space Green's function. It was also observed (see Figure 4.3) that the compressed preconditioner behaved essentially the same as the original preconditioner for the 250^3 waveguide example.

	original	compressed	ratio
diagonal blocks	2544 MB (11295 MB)	155.0 MB (903.2 MB)	16.41 (12.51)
connectivity	6462 MB (31044 MB)	1336 MB (7611 MB)	4.836 (4.079)
total	9007 MB (42339 MB)	1491 MB (8513 MB)	6.039 (4.973)

Table 4.1: Compression of the top-panel, which consists of 14 planes, of a 250^3 (and 500^3) waveguide with ten points per wavelength using relative tolerances of 1e-2 and 5e-2 for the diagonal blocks and connectivity, respectively

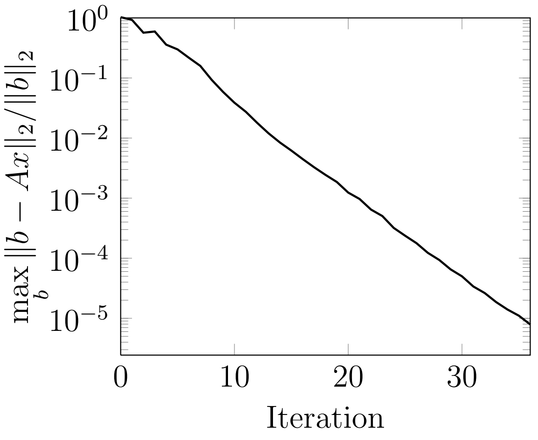


Figure 4.3: Convergence of compressed moving PML sweeping preconditioner in GMRES(20) for a 250^3 waveguide problem, with ten points per wavelength and relative tolerances of 0.01 and 0.05 for the Krocker product approximations of the top-left and bottom-left quadrants of each frontal matrix, respectively

4.4 Summary and future work

We have introduced a compression scheme for the sweeping preconditioner motivated by the lossless compression of certain constant-coefficient discrete Green's functions in terms of sums of Kronecker products and explained how to express each of the significant operations in a multifrontal triangular solve with the compressed frontal trees in terms of dense matrix-matrix multiplication. A detailed study of how to properly adapt SUMMA to applying sums of Kronecker products for the typical matrix sizes of our application will be left as future work. The author would also like to emphasize that, while several other researchers have investigated hierarchical low-rank compression schemes for the dense frontal matrices [110, 134], the proposed scheme is based upon *Kronecker-product* approximations designed to exploit the translation invariance of the free-space Green's function. Another difference is that the primary goal is to *lower the memory requirements* rather than to reduce the computational cost.

While the observed factor of five memory compression is useful, it is clear from Table 4.1 that most of the benefits come from the compression of the diagonal blocks of the subdomain multifrontal factorizations (where the developed theory directly applies). If a similar compression scheme was found for the off-diagonal blocks, then it would perhaps allow us to solve problems at twice the frequency as with the standard sweeping preconditioner. It is therefore worthwhile to develop theory targeted directly towards the interactions between supernodes.

Chapter 5

Contributions and future work

5.1 Contributions

The major contributions detailed throughout the previous chapters essentially boil down to:

- high-performance multifrontal triangular solves,
- an efficient parallel moving-PML sweeping preconditioner,
- a compressed moving-PML sweeping preconditioner, and
- a high-performance algorithm for applying compressed fronts.

We will now briefly summarize each of these items and then provide pointers to the source code and documentation for the supporting software.

5.1.1 High-performance multifrontal triangular solves

Two new high-performance multifrontal triangular solve schemes were proposed in Chapter 2, one of which is an extension of selective inversion to large amounts of right-hand sides, and the other avoids selective inversion and exploits the fact that, while dense triangular solves with a few right-hand sides are not scalable, dense triangular solves with *many* right-hand sides are. The fundamental idea behind each of these approaches is to eschew traditional one-dimensional distributions for each supernode's portion of the right-hand side vectors in favor of two-dimensional data distributions which can be used within scalable dense matrix-matrix multiplication and triangular solve algorithms. Both of these approaches will be of significant interest for applications which require the solution of large numbers of linear systems with the same sparse matrix (especially frequency-domain seismic inversion).

5.1.2 Parallel moving-PML sweeping preconditioner

A parallelization of the moving-PML sweeping preconditioner was presented in Chapter 3 and efficiently applied to several large-scale analytical models, as well as a realistic seismic model (Overthrust [4]) using several thousand cores. The main ideas were:

- ensuring that the subdomain multifrontal solves were scalable, for instance, through the usage of selective inversion,
- distributing right-hand sides in a manner which conforms to the distributed subdomain multifrontal factorizations in order to avoid unnecessary communication before and after each subdomain triangular solve, and
- making use of a scalable scheme for indefinite multifrontal factorization which does not needlessly sacrifice performance in the name of numerical stability.

5.1.3 Compressed moving-PML sweeping preconditioner

A new Kronecker-product compression scheme for the fronts of the subdomain auxiliary problems was theoretically motivated through the translation invariance of the underlying free-space Green's function and a significant compression ratio was demonstrated for a large waveguide model. In particular, it was shown that translation invariance is only required in the direction normal to the half-space boundary in order to apply mirror-imaging techniques to yield a compression scheme for the half-space Green's function. This compression scheme was shown to allow for the replacement of the diagonal blocks of the sweeping preconditioners subdomain multifrontal factorizations with the sum of a small number of *Kronecker product* matrices.

5.1.4 High-performance compressed front applications

It was shown that, through the usage of the identity

$$(Y \otimes Z)x = \mathrm{vec}(ZXY^T),$$

where the vec operator concatenates the columns of its input matrix and X is defined such that $\mathrm{vec}(X) = x$, that the frontal matrices compressed as a sum of a small number of Kronecker products may be applied to (sets of) vectors using scalable algorithms for dense matrix-matrix multiplication. The result is that, after the Kronecker product compression, standard multifrontal triangular solves need only be slightly modified in order to perform the necessary linear transformations directly with the compressed data.

5.1.5 Reproducibility

Each of the major software efforts involved in the research behind this dissertation is available under an open-source license along with a significant amount of documentation:

- Parallel Sweeping Preconditioner (PSP) is available at `http://bitbucket.org/poulson/psp`,
- Clique is available at `http://bitbucket.org/poulson/clique`,
- Elemental is available at `http://code.google.com/p/elemental`, and
- Madagascar is available at `http://ahay.org`.

Each of these packages is actively used by a number of researchers scattered throughout the world, and significant efforts have been expended in order to make all published claims easily repeatable.

5.2 Future work

While many significant contributions were made in this thesis, there are invariably a number of lines of future work. Several of the areas which the author believes to be the most important are listed here:

- Performing large-scale experiments to measure the benefits of the new high-performance triangular solve algorithms.
- Performing a careful stability analysis of selective inversion.

- Extending the current implementation of the parallel sweeping preconditioner to more sophisticated discretizations for both time-harmonic Maxwell's and linear elastic equations.

- Making use of an appropriate scheme for choosing *a priori* profiles for the Perfectly Matched Layers based upon the material coefficients.

- Performing a detailed study of distributed Kronecker-product based matrix-matrix multiplication.

- And, perhaps most importantly, fundamental research should be conducted towards effectively solving time-harmonic wave equations which are close to resonance (e.g., cavity problems).

Finally, as was mentioned in the introductory chapter, one of the main applications of this work will be seismic inversion. As these techniques mature, the author will begin investigating their incorporation into effective inversion procedures.

Appendices

Appendix A

Finite-dimensional spectral theory

The purpose of this appendix is to provide a brief introduction to finite-dimensional spectral theory, especially in preparation for discussions of Krylov subspace methods in Appendix B. Arguably the most useful tool in understanding in and analyzing matrices (and, more generally, linear operators) is the notion of a *spectrum*, which, informally, is the set of scalars ξ associated with a linear operator A such that $A-\xi I$ does not have a well-defined inverse. Of course, it is important to specify precisely what we mean by *scalars*, and so, for the remainder of this document, all of our analysis will take place within the complex field, $\mathbb{C}$. The reason for this generality is two-fold:

- *the fundamental theorem of algebra* yields the existence of n roots to polynomials of degree n in $\mathbb{C}$, but there is no analogue in the set of real numbers, $\mathbb{R}$, and
- the sweeping preconditioner applies to matrices which are complex by construction.

A proof of the fundamental theorem of algebra is beyond the scope of this dissertation, and so we will simply state it without proof and point the interested reader to [117] for a concise proof based upon Liouville's theorem, or to [30] for a more elementary argument based upon the minimum-modulus principle which dates back to the amateur mathematician, Jean-Robert Argand [7].

The motivating ideas of this appendix are essentially a blend of [14], [74], [58], and [67]. Like [14], we prove the existence of eigenpairs without resorting to determinants,[1] whereas our proof of the existence of a shared eigenvector among two commuting matrices is due to [74], which leads to a beautifully simple (and seemingly nonstandard) proof of the spectral theorem for normal matrices. Our proof of the singular value decomposition is essentially identical to that of [58], and our debt to [67] and primarily stylistic.

The basic outline of this appendix is as follows:

1. an introduction of vector spaces and matrices,
2. a proof of the existence of eigenpairs via the introduction of Krylov subspaces,
3. a proof of the existence of the Schur decomposition (as well as for pairs of commuting matrices) and of the singular value decomposition,
4. the specialization of Schur decompositions to normal matrices (the spectral decomposition) and its relation to the SVD, and, finally,
5. a discussion of generalized eigenspaces, the Cayley-Hamilton theorem, and the class of diagonalizable matrices.

A.1 Vector spaces and matrices

In order to avoid getting sidetracked during the introduction of a spectrum, we will first define the basic terminology which is used for describing

[1]via an argument based upon *Krylov subspaces*, despite the fact that the term *Krylov* never appears in [14]

vector spaces. Ideally the reader is already intimately familiar with the concept of *linear independence* and the notion of a *basis* of a vector space, but their definitions will be provided for the sake of completeness. However, we will not list all of the properties required for a proper definition of a vector space or provide a proof of the finite-dimensional Riesz representation theorem. Please see [67] for a detailed discussion of this background material.

Definition A.1.1 (complex vector space)**.** The symbol $\mathbb{C}^n$ will be used to denote the set of all complex vectors of length n. For each $n \geq 1$, $\mathbb{C}^n$ is called a *vector space*, as, in addition to various other properties inherited from $\mathbb{C}$, for any *scalars* $\alpha, \beta \in \mathbb{C}$ and *vectors* $x, y \in \mathbb{C}^n$, the linear combination $\alpha x + \beta y$ is also a member of $\mathbb{C}^n$.

Definition A.1.2 (linear independence)**.** A set of k vectors, say $\{x_j\}_{j=0}^{k-1} \subset \mathbb{C}^n$, is called *linearly independent* if no nontrivial combination of its members is equal to the zero vector. More specifically, $\{x_j\}_{j=0}^{k-1}$ is linearly independent if

$$\sum_{j=0}^{k-1} \alpha_j x_j \neq 0$$

for every set of coefficients $\{\alpha_j\}_{j=0}^{k-1} \subset \mathbb{C}$ which has at least one nonzero entry. Notice that this is equivalent to the statement that no member of the set of vectors can be written as a linear combination of the others.

Definition A.1.3 (subspace)**.** A (linear) *subspace* of the vector space $\mathbb{C}^n$ is any set $\mathcal{W} \subset \mathbb{C}^n$ such that, for any $w_1, w_2 \in \mathcal{W}$ and $\alpha, \beta \in \mathbb{C}$, $\alpha w_1 + \beta w_2 \in \mathcal{W}$. Notice that $\{0\}$ is a subspace of $\mathbb{C}^n$.

Definition A.1.4 (basis)**.** Any set of vectors $\{x_j\}_{j=0}^{k-1} \subset \mathbb{C}^n$ which can be combined to form every member of some subspace $\mathcal{W} \subset \mathbb{C}^n$ is called a *basis* for that subspace. More specifically, $\{x_j\}_{j=0}^{k-1}$ is a basis for $\mathcal{W}$ if, for every $w \in \mathcal{W}$, there exist coefficients $\{\alpha_j\}_{j=0}^{k-1} \subset \mathbb{C}$ such that

$$w = \sum_{j=0}^{k-1} \alpha_j x_j.$$

Definition A.1.5 (span)**.** We say that a basis for a subspace *spans* that subspace, and, for any arbitrary set of vectors $\{x_j\}_{j=0}^{k-1} \subset \mathbb{C}^n$, we may define the set

$$\operatorname{span}\{x_j\}_{j=0}^{k-1} = \{x \in \mathbb{C}^n : x = \sum_{j=0}^{k-1} \alpha_j x_j \text{ for some } \{\alpha_j\}_{j=0}^{k-1} \subset \mathbb{C}\}. \tag{A.1.1}$$

It is easy to see that this set is in fact a linear subspace of $\mathbb{C}^n$. We will also occasionally use the shorthand $\operatorname{span} A$ for denoting the span of the columns of a matrix A.

Definition A.1.6 (standard basis vectors)**.** The j'th *standard basis vector*, $\mathbf{e}_j \in \mathbb{C}^n$, is defined component-wise as being zero everywhere except for entry j, which is equal to one. Clearly $\{\mathbf{e}_j\}_{j=0}^{n-1}$ spans $\mathbb{C}^n$.

Definition A.1.7 (dimension)**.** The *dimension* of a subspace $\mathcal{W} \subset \mathbb{C}^n$ is the minimum number of members of $\mathcal{W}$ which may span $\mathcal{W}$, and it is denoted by $\dim \mathcal{W}$. We also call such a set of vectors a *minimal basis.*

Proposition A.1. *For every subspace $\mathcal{W} \subset \mathbb{C}^n$, at most $\dim \mathcal{W}$ linearly independent vectors may be chosen from $\mathcal{W}$.*

Proof. Please see [67] for an inductive proof. □

Corollary A.2. *If $\dim \mathcal{W}$ members of $\mathcal{W}$ are linearly independent, then they span $\mathcal{W}$.*

Proof. Suppose not. Then there exists a member of $\mathcal{W}$ which is independent of the original d vectors, which implies a set of $d+1$ linearly independent vectors, which contradicts the previous theorem. $\square$

Definition A.1.8 (linear transformation,linear functional)**.** Any function $A : \mathbb{C}^n \to \mathbb{C}^m$ is called *linear* if, for any scalars $\alpha, \beta \in \mathbb{C}$ and vectors $x, y \in \mathbb{C}^n$,

$$A(\alpha x + \beta y) = \alpha A x + \beta A y.$$

When $m = 1$ (i.e., the result lies in $\mathbb{C}$), we will call such a transformation a *linear functional.*

Definition A.1.9 (inner product)**.** For any $x, y \in \mathbb{C}^n$, we define their (Euclidean) *inner product* as

$$(y, x) = \sum_{j=0}^{n-1} \overline{\eta_j} \xi_j, \tag{A.1.2}$$

where ξ_j and η_j are respectively the j'th entries of x and y. The inner product (y, x) will also often be denoted by $y^H x$. It is easy to see that the Euclidean inner product satisfies the following properties for any $\alpha, \beta \in \mathbb{C}$ and $x, y \in \mathbb{C}^n$:

1. (*conjugate symmetry*), $(y, x) = \overline{(x, y)}$
2. (*sesquilinearity*), $(\beta y, \alpha x) = \overline{\beta}(y, \alpha x) = \alpha \overline{\beta}(y, x)$, and
3. (*positive-definiteness*), $(x, x) \geq 0$, with equality if and only if $x = 0$.

Theorem A.3 (Riesz representation theorem [67])**.** *Every linear functional $f : \mathbb{C}^n \to \mathbb{C}$ can be represented as the inner-product of its argument with a particular vector, say v_f:*

$$f(x) = (y_f, x) = y_f^H x. \tag{A.1.3}$$

We say that the Riesz representation theorem identifies each linear functional on $\mathbb{C}^n$ with a vector in $\mathbb{C}^n$.

Proof. Please see [67] for details. $\square$

Remark A.1.1. Every linear transformation $A : \mathbb{C}^n \to \mathbb{C}^m$ can be decomposed into m linear functionals, say $\{f_i\}_{i=0}^{m-1}$, where, for any $x \in \mathbb{C}^n$,

$$Ax = \begin{pmatrix} f_0(x) \\ f_1(x) \\ \vdots \\ f_{m-1}(x) \end{pmatrix}.$$

Definition A.1.10 (matrices)**.** Every linear transformation $A : \mathbb{C}^n \to \mathbb{C}^m$ can be identified with an $m \times n$ array of numbers called a *matrix*, where each row of the matrix is the conjugate of the Riesz representation of the linear functional f_i. Then, $y = Ax$ implies that

$$\eta_i = \sum_j \alpha_{i,j} \xi_j,$$

where $\alpha_{i,j}$ refers to the (i, j) entry of the matrix associated with A, ξ_j refers to the j'th entry of ξ, and η_i refers to the i'th entry of y. From now on, we will identify all finite-dimensional linear transformations with their corresponding matrix.

Definition A.1.11. We will follow the convention of [74] and denote the set of $m \times n$ matrices with complex coefficients as $M_{m,n}$, and the set of $n \times n$ (*square*) matrices as M_n.

Definition A.1.12 (transpose and Hermitian-transpose)**.** We define the *transpose* of $A \in M_{m,n}$ entry-wise as the $n \times m$ matrix

$$A^T(i, j) = A(j, i). \tag{A.1.4}$$

Likewise, we define the *Hermitian-transpose* of A, also known as the *conjugate-transpose* and *adjoint* of A, as the $n \times m$ matrix

$$A^H(i, j) = \overline{A(j, i)}.$$

Proposition A.4. *For any square matrix $A \in M_n$,*

$$(y, Ax) = (A^H y, x)$$

for all $x, y \in \mathbb{C}^n$. This is the usual setting in mind when referring to A^H as the adjoint of A.

Proof. Once we recognize that the (i, j) entry of a matrix A is given by (e_i, Ae_j), the entry-wise definition of the adjoint implies that $(y, Ax) = (A^H y, x)$ at least holds when x and y are both standard basis vectors. We can then extend this result to arbitrary x and y in $\mathbb{C}^n$ by representing each as a sum of standard basis vectors (and making use of the sesqui-linearity of the inner product). $\square$

Proposition A.5. *For any square matrices $A, B \in M_n$,*

$$(AB)^H = B^H A^H, \text{ and } \quad (AB)^T = B^T A^T. \tag{A.1.5}$$

Proof. We may apply the previous proposition with standard basis vectors in order to show that both equations hold entrywise. For example,

$$(e_i, (AB)^H e_j) = ((AB)e_i, e_j) = (Be_i, A^H e_j) = (e_i, B^H A^H e_j),$$

which shows the first result. A similar argument holds for $(AB)^T$. $\square$

A.2 Existence of the spectrum

Definition A.2.1 (singular matrix)**.** A matrix $A \in M_n$ is called *singular* if there exists some nonzero vector $x \in \mathbb{C}^n$ such that $Ax = 0$.

Definition A.2.2 (kernel)**.** Given a matrix $A \in M_n$, the set of all vectors $x \in \mathbb{C}^n$ such that $Ax = 0$ is called the *kernel* or *null space* of A. That is,

$$\ker A = \{x \in \mathbb{C}^n : Ax = 0\}.$$

Proposition A.1. *The kernel of any matrix $A \in M_n$ is in fact a subspace of $\mathbb{C}^n$, and, if it is nontrivial, then A is not injective, and therefore not invertible.*

Proof. Suppose that $\alpha, \beta \in \mathbb{C}$ and $x, y \in \ker A$. Then $A(\alpha x + \beta y) = \alpha Ax + \beta Ay = 0$. For the second claim, we recognize that, if $Ax = 0$, then $A(\alpha x + z) = Az$ for any $\alpha \in \mathbb{C}$ and $z \in \mathbb{C}^n$. □

Definition A.2.3 (spectrum)**.** The spectrum of a matrix $A \in M_n$, denoted by $\sigma(A)$, is the set of all scalars $\lambda \in \mathbb{C}$ such that $A - \lambda I$ is singular.[2]

Definition A.2.4 (eigenvalue,eigenvector,eigenpair)**.** Suppose that λ is a member of the spectrum of some $A \in M_n$. Then, since $(A - \lambda I)x = 0$ for some $x \in \mathbb{C}^n$, we may trivially rearrange this equation into the form

$$Ax = \lambda x,$$

which shows that, in some sense, A behaves like the scalar λ when acting on the vector x. For this reason, each $\lambda \in \sigma(A)$ is called an *eigenvalue* of A, and its corresponding vector, x, is called an *eigenvector*. It is also common to refer to the pair (λ, x) as an *eigenpair* of a matrix.

[2]This definition must become significantly more technical when considering infinite-dimensional linear operators. Roughly speaking, the spectrum of a linear operator A is the set of all $\xi \in \mathbb{C}$ such that the *resolvent* operator, $(\xi I - A)^{-1}$, is not well-defined and bounded. See [84] for details.

Remark A.2.1. We will now follow the approach of [14] in showing the crucial fact that *every* complex finite-dimensional square matrix, $A \in M_n$, where $n \geq 1$, has at least one eigenpair. Along the way, we will introduce the extremely-important notion of a *Krylov subspace*, which plays a central role in the body of this document.

Lemma A.2. *For any two scalars $\xi_1, \xi_2 \in \mathbb{C}$, and any matrix $A \in M_n$,*

$$(A - \xi_1 I)(A - \xi_2 I) = (A - \xi_2 I)(A - \xi_1 I).$$

More generally, for any finite set of shifts $\{\xi_j\}_{j=0}^{k-1} \subset \mathbb{C}$, the ordering of the terms in the product

$$\prod_{j=0}^{k-1} (A - \xi_j I)$$

is irrelevant.

Definition A.2.5 (commuting matrices [74])**.** Any two matrices $A, B \in M_n$ such that $AB = BA$ are said to *commute.*

Definition A.2.6. The *polynomials of degree n*, $\mathbb{P}_n$, are the functions $p : \mathbb{C} \to \mathbb{C}$ which can be expressed as

$$p(z) = \sum_{j=0}^{n} \alpha_j z^j,$$

where each $\alpha_j \in \mathbb{C}$ is called a *coefficient* of the polynomial p.

Definition A.2.7 (Krylov subspace)**.** The *Krylov subspace* of matrix $A \in M_n$ and vector $r \in \mathbb{C}^n$ of order k is defined as

$$\mathcal{K}_k(A, r) = \operatorname{span}\{r, Ar, \ldots, A^{k-1} r\}, \tag{A.2.6}$$

which is precisely the space of all polynomials of degree less than or equal to $k - 1$ of A acting on r, which we denote by $\mathbb{P}_{k-1}(A) r$.

Lemma A.3 (Krylov lemma)**.** *For any matrix $A \in M_n$ and vector $x \in \mathbb{C}^n$, there exists a polynomial $p \in \mathbb{P}_n$ such that $p(A)x = 0$.*

Proof. Consider the Krylov subspace

$$\mathcal{K}_{n+1}(A, x) = \operatorname{span}\{x, Ax, \ldots, A^n x\} = \mathbb{P}_n(A)x$$

for some arbitrary vector $x \in \mathbb{C}^n$. Since we know that at most n linearly independent vectors may be chosen from $\mathbb{C}^n$ (Lemma A.1), there exists some linear combination of the vectors $\{x, Ax, \ldots, A^n x\}$ which is equal to zero, i.e., there exists some nonzero $p \in \mathbb{P}_n$ such that $p(A)x = 0$. □

Theorem A.4 (fundamental theorem of algebra [7])**.** *Every polynomial p of degree $n \geq 1$ can be factored as*

$$p(z) = \gamma_n (z - \xi_0)(z - \xi_1) \cdots (z - \xi_n),$$

where, for each $\xi_j \in \mathbb{C}$, $p(\xi_j) = 0$. Each ξ_j is called a root *of the polynomial p.*

Proof. See [117] for details. □

Theorem A.5 (existence of an eigenpair [14])**.** *Every matrix $A \in M_n$, where $n \geq 1$, has at least one eigenpair.*

Proof. Let $r \in \mathbb{C}^n$ and let $p \in \mathbb{P}_n$ be the polynomial guaranteed by Lemma A.3 such that $p(A)r = 0$. We can now employ the fundamental theorem of algebra (Theorem A.4) in order to assert the existence of the factorization

$$p(z) = \gamma_n \prod_{j=0}^{n} (z - \xi_j),$$

where $\gamma_n \neq 0$, and, when combined with Lemma A.2, we see that

$$p(A)r = \gamma_n \left(\prod_{j=0}^{n} (A - \xi_j I) \right) r = 0,$$

where the ordering of the factors $A - \xi_j I$ is irrelevant. Let us therefore simply write

$$p(A)r = \gamma_n (A - \xi_n I) \cdots (A - \xi_1 I)(A - \xi_0 I) r = 0,$$

and consider evaluating the middle expression from right to left, e.g., by considering the sequence $\{r_j\}_{j=0}^{n+1}$, where $r_0 \equiv r$ and

$$r_j \equiv (A - \xi_{j-1} I) \cdots (A - \xi_1 I)(A - \xi_0 I) r = (A - \xi_{j-1} I) r_{j-1},$$

when $j \geq 1$. Since we know that r_0 is nonzero, and $r_{n+1} = p(A)r = 0$, let r_q be the first term which is zero (as all subsequent terms must also be zero). Then $(A - \xi_{q-1} I) r_{q-1} = 0$ even though $r_{q-1} \neq 0$, and so, by definition, (ξ_{q-1}, r_{q-1}) is an eigenpair of A. □

Remark A.2.2. With just a few more lemmas at our disposal, we can actually prove that, if two matrices commute, they share a common eigenvector. In fact, this is true for any *family* of commuting matrices [74], but we will simply stick to pairs of commuting matrices.

Definition A.2.8 (invariant subspace)**.** Any subspace $\mathcal{W} \subset \mathbb{C}^n$ such that $A(\mathcal{W}) \subset \mathcal{W}$ is called an *invariant subspace* of $A \in M_n$.

Remark A.2.3. Note that the one-dimensional space spanned by an eigenvector of A is an invariant subspace, as is the trivial subspace, $\{0\}$. In fact, we will now show that *every* non-trivial invariant subspace of A contains an eigenvector of A.

Lemma A.6. *If a subspace $\mathcal{W} \subset \mathbb{C}^n$ is invariant under $A \in M_n$, then there exists an eigenvector of A within $\mathcal{W}$.*

Proof. Let W be a matrix whose columns form a minimal basis for the invariant subspace $\mathcal{W}$. Then, for each $w \in \mathcal{W}$, we may express w as a linear combination of the columns of W, i.e., $w = Wy$. Since $\mathcal{W}$ is invariant, each column of AW therefore lies within $\mathcal{W}$, and so we may find a matrix Z such that

$$AW = WZ.$$

Now, if we apply Theorem A.5 to the matrix Z, we know that $Zy = \lambda y$ for some y of appropriate dimension. Then, we may multiply both sides of $AW = WZ$ by y to find that

$$WZy = W(\lambda y) = AWy,$$

which shows that Wy is an eigenvector of A with eigenvalue λ. $\square$

Lemma A.7. *If a matrices $A, B \in M_n$ commute, then, for any finite-degree polynomial p, A and $p(B)$ also commute.*

Proof. Since, for any $k \geq 1$,

$$AB^k = (AB)B^{k-1} = (BA)B^{k-1} = B(AB^{k-1}),$$

we may move a single power of B from the right to the left side of A as many times as we like, which shows that A commutes with every monomial of B, say B^k. Since polynomials are merely linear combinations of monomials, the result follows. $\square$

Lemma A.8. *If, given some matrix $A \in M_n$ and vector $x \in \mathbb{C}^n$, the vectors $\{x, Ax, \ldots, A^{k-1}x\}$ are linearly dependent, then $\mathcal{K}_k(A, x)$ is invariant under A.*

Proof. Since the result is trivial for $k = 1, 2$, suppose that we have shown the result for $k < j$. Then, if $\{x, Ax, \ldots, A^{j-1}x\}$ is linearly dependent, then there are two possibilities:

- $\{x, Ax, \ldots, A^{j-2}x\}$ is linearly dependent, which implies that $\mathcal{K}_{j-1}(A, x)$ is invariant, which shows that $\mathcal{K}_k(A, x)$ is invariant for all $k \geq j$, or
- $A^{j-1}x$ can be expressed as a linear combination of $\{x, Ax, \ldots, A^{j-2}x\}$, which implies that $A(A^{j-1}x) \in \mathcal{K}_j(A, x)$ so that $A(\mathcal{K}_j(A, x)) \subset \mathcal{K}_j(A, x)$.

In both cases, if $\{x, Ax, \ldots, A^{j-1}x\}$ is linearly dependent, $\mathcal{K}_j(A, x)$ is invariant under A, and so the proof holds by induction. $\square$

Theorem A.9 (existence of a shared eigenpair [74])**.** *If the matrices $A, B \in M_n$ commute, then they share a common eigenvector.*

Proof. We first apply Theorem A.5 to yield an eigenvector x of A. Since $\{x, Bx, \ldots, B^{n+1}x\}$ is linearly dependent, Lemma A.8 shows that $\mathcal{K}_{n+2}(B, x)$ is invariant under B, and thus Lemma A.6 provides an eigenvector $y \in \mathcal{K}_{n+2}(B, x)$ of B. Since every member of $\mathcal{K}_{n+2}(B, x)$ may be written as $p(B)x$ for some $p \in \mathbb{P}_{n+1}$, Lemma A.7 shows that $p(B)x$ is also an eigenvector of A. $\square$

A.3 Schur decompositions

We will now introduce the classes of *triangular* and *unitary* matrices, and then show that every matrix $A \in M_n$ can be decomposed into a product of such matrices. We will then show that this decomposition provides a significant amount of insight into the spectrum of A.

Definition A.3.1 (triangular matrices)**.** A matrix $L \in M_n$ is called *lower-triangular* if it is zero below its diagonal, i.e., if $i > j$ implies that $L(i,j) = 0$. Likewise, a matrix $U \in M_n$ is called *upper-triangular* if it is zero *above* its diagonal, i.e., $i < j$ implies that $U(i,j) = 0$.

Lemma A.1. *For any triangular matrix $T \in M_n$, $\lambda \in \sigma(T)$ if and only if there is a diagonal entry of T equal to λ.*

Proof. Since $T - \lambda I$ has a zero diagonal entry if and only if T has a diagonal value equal to λ, we need only show that a triangular matrix is singular if and only if it has a zero diagonal entry.

Suppose that a triangular matrix T has no nonzero diagonal entries, and that $Tx = 0$ for some vector $x \in \mathbb{C}^n$. Then we may reverse this multiplication process, through a process known as *backward elimination*, in order to find that x is identically zero. Notice that backward elimination is only well-defined when the diagonal of T does not contain a zero. In particular, let us partition the equation $Tx = 0$ into

$$\begin{pmatrix} T_{1,1} & t_{1,2} \\ 0 & \tau_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

in order to see that $\chi_2 = 0$. But we then recursive application of this procedure to $T_{1,1}x_1 = 0$ shows that $x = 0$.

Now suppose that the j'th diagonal entry of T is zero. Then the first j columns of T lie within the $j-1$-dimensional subspace span $\{\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{j-2}\}$, and thus, by Lemma A.1, they cannot be linearly independent. We may therefore define a vector x, which is only nonzero in its first j components such that $Tx = 0$. □

Definition A.3.2 (two-norm)**.** The *two-norm* of the vector $x \in \mathbb{C}^n$, also known as the *Euclidean norm*, is defined as

$$\|x\|_2 = \sqrt{(x, x)}, \tag{A.3.7}$$

where the square-root can be interpreted in the usual sense because $(x, x) \geq 0$, with equality if and only if $x = 0$. The two-norm is perhaps the most natural way to define the length of a vector.

Definition A.3.3 (unitary matrix)**.** A matrix $Q \in M_n$ is called *unitary* if, for every $x, y \in \mathbb{C}^n$, $(x, y) = (Qx, Qy)$; as a special case, we also have that $\|Qx\|_2 = \|x\|_2$. The interpretation of the matrix Q as having unit-magnitude (hence the name *unitary*) can be made rigorous using the following definition.

Proposition A.2. *The product of two unitary matrices is also unitary.*

Proof. Let $U, V \in M_n$ both be unitary. Then, for any $x, y \in \mathbb{C}^n$,

$$(x, y) = (Vx, Vy) = (U(Vx), U(Vy)).$$

□

Definition A.3.4 (operator two-norm)**.** The *operator*, or *induced* two-norm of the matrix $A \in M_n$ is defined as

$$\|A\|_2 = \max_{\substack{x \in \mathbb{C}^n, \\ \|x\|_2 = 1}} \|Ax\|_2, \tag{A.3.8}$$

where the maximum can be shown to exist due to the Heine-Borel theorem [106], but we will avoid diving into real analysis in order to explain this technical detail. This operator norm of A can be geometrically interpreted as the maximum length of any transformed vector which originated on the surface of the unit-ball in $\mathbb{C}^n$.

Proposition A.3. *For any unitary matrix Q, $\|Q\|_2 = 1$.*

Proof. By definition of the operator two-norm,

$$\|Q\|_2 = \max_{\substack{x \in \mathbb{C}^n \\ \|x\|_2 = 1}} \|Qx\|_2,$$

and, by the definition of unitary matrices, $\|Qx\|_2 = \|x\|_2$. Since the maximum is only taken over the unit ball, where $\|x\|_2 = 1$, the result follows. $\square$

Proposition A.4. *A matrix is unitary if and only if its adjoint is its inverse.*

Proof. We will first show that, if Q is unitary, $Q^H Q = I$. The (i, j) entry of $Q^H Q$ is given by $(e_i, Q^H Q e_j)$, and, by Proposition A.4, it is also equivalent to (Qe_i, Qe_j). By the definition of a unitary transformation, $(Qe_i, Qe_j) = (e_i, e_j)$, which satisfies the desired properties. A similar argument shows that $QQ^H = I$.

Now suppose that $A^H A = I$ for some matrix $A \in M_n$. Since $(e_i, e_j) = (e_i, A^H A e_j) = (Ae_i, Ae_j)$ for every pair of standard basis vectors, e_i and e_j, by the sesquilinearity of the inner product we find that $(x, y) = (Ax, Ay)$ for every $x, y \in \mathbb{C}^n$, which shows that A is unitary by definition. $\square$

Definition A.3.5. A set of vectors $\{x_j\}_{j=0}^{k-1} \subset \mathbb{C}^n$ is called *orthonormal* if $(x_i, x_j) = 0$ when $i \neq j$, and $(x_i, x_j) = 1$ when $i = j$. Notice that we have just shown that the rows and columns of a unitary matrix are both orthonormal.

Proposition A.5. *Every subspace $\mathcal{W} \subset \mathbb{C}^n$ has an orthonormal basis of size $\dim \mathcal{W}$.*

Proof. Let $\{w_j\}_{j=0}^{d-1} \subset \mathcal{W}$ be a linearly independent basis of $\mathcal{W}$, where $d = \dim \mathcal{W}$. Then, we will construct an orthonormal basis through a process known as *Gram-Schmidt orthogonalization.* We begin by putting $q_0 = w_0/\|w_0\|_2$, which must have unit norm. Then we may define $\tilde{q}_1 = w_1 - q_0(q_0, w_1)$, which is easily verified to be a nonzero vector (the original basis is linearly independent) which is orthogonal to w_0. Since $\tilde{q}_1$ is nonzero, we may normalize it to construct $q_1 = \tilde{q}_1/\|\tilde{q}_1\|_2$ so that $\{q_0, q_1\}_{j=0}^{1}$ is orthonormal.

Now, suppose that we have constructed an orthonormal set $\{q_j\}_{j=0}^{k-1}$, where $k < d$. Then we may define

$$\tilde{q}_k = w_k - \sum_{j=0}^{k-1} q_j(q_j, w_k),$$

which effectively removes all of the components of $\tilde{q}_k$ in the directions of the previous orthonormal set. Since the linear independence of the $\{w_j\}_{j=0}^{d-1}$ basis proves that $\tilde{q}_k$ is nonzero, we may normalize it in order to construct a new orthonormal set, $\{q_j\}_{j=0}^{k}$, which completes the proof by induction. $\square$

Definition A.3.6 (orthogonal complement)**.** The orthogonal complement for a subspace $\mathcal{W} \subset \mathbb{C}^n$ is defined as

$$\mathcal{W}^{\perp} = \{x \in \mathbb{C}^n : \forall\, w \in \mathcal{W}, (w, x) = 0\}.$$

Proposition A.6. *For any subspace $\mathcal{W} \subset \mathbb{C}^n$, $\mathcal{W}^{\perp}$ is also a subspace of $\mathbb{C}^n$.*

Proof. Let $x, y \in \mathcal{W}^{\perp}$ and $\alpha, \beta \in \mathbb{C}$. Then $(w, \alpha x + \beta y) = \alpha(w, x) + \beta(w, y)$ by the sesquilinearity of the inner product, and, because x and y lie in $\mathcal{W}^{\perp}$, each of these terms is identically zero, and thus $\alpha x + \beta y$ also lies within $\mathcal{W}^{\perp}$. $\square$

Proposition A.7. *For any subspace $\mathcal{W} \subset \mathbb{C}^n$,*

$$dim\,\mathcal{W} + dim\,\mathcal{W}^{\perp} = n,$$

and each $x \in \mathbb{C}^n$ may be uniquely decomposed as $x = w + z$, where $w \in \mathcal{W}$ and $z \in \mathcal{W}^{\perp}$. We may say that $\mathbb{C}^n$ is a direct sum *of $\mathcal{W}$ and its orthogonal complement, which we denote by*

$$\mathcal{W} \oplus \mathcal{W}^{\perp} = \mathbb{C}^n.$$

Proof. Let $W = \{w_j\}_{j=0}^{d-1} \subset \mathcal{W}$ and $Z = \{z_j\}_{j=0}^{k-1} \subset \mathcal{W}^{\perp}$ respectively be orthonormal bases for $\mathcal{W}$ and $\mathcal{W}^{\perp}$, where $d = \dim \mathcal{W}$ and k is an as-of-yet undetermined number.

Since each $z_i \in \mathcal{W}^{\perp}$, $(z_i, w_j) = (w_j, z_i) = 0$ for every pair (i, j). Because each of these sets is orthonormal, the combined set is orthonormal as well (and, of course, also linearly independent). This implies that $\dim \mathcal{W}^{\perp} \leq n - d$.

Now suppose that $\dim \mathcal{W}^{\perp} < n - d$. Then, by Proposition A.1, there exists some vector $x \in \mathbb{C}^n$ which is orthogonal to both $\mathcal{W}$ and $\mathcal{W}^{\perp}$. But this is a contradiction, as, by definition, any vector which is orthogonal to $\mathcal{W}$ is a member of $\mathcal{W}^{\perp}$. We have thus shown that $\dim \mathcal{W}^{\perp} = n - d$.

In order to show the second property, for $x \in \mathbb{C}^n$, set

$$y = \sum_{j=0}^{d-1} w_j (w_j, x),$$

and

$$z = \sum_{j=0}^{n-d-1} z_j (z_j, x),$$

so that $y \in \mathcal{W}$, $z \in \mathcal{W}^{\perp}$. But then we can see that $x = y + z$, as $y + z = QQ^H x$, where Q is the unitary matrix with columns equal to $(w_0, \ldots, w_{d-1}, z_0, \ldots, z_{n-d-1})$.

In order to show uniqueness, let $x = \hat{y} + \hat{z}$ be another such decomposition, so that $y - \hat{y} = \hat{z} - z$. Since the left side of the equation lies in $\mathcal{W}$, and the right side lies in $\mathcal{W}^{\perp}$, it can only be satisfied when both $y - \hat{y}$ and $z - \hat{z}$ are zero. $\square$

Corollary A.8. *Every vector $x \in \mathbb{C}^n$ of unit length can be extended into an orthonormal basis for $\mathbb{C}^n$.*

Proof. If we define $\mathcal{W} = \operatorname{span} x$, then Lemma A.7 shows that there exists an orthonormal basis of dimension $n-1$ for $x^{\perp}$, which implies that, when combined with x, is an orthonormal basis for all of $\mathbb{C}^n$. $\square$

Theorem A.9 (QR decomposition)**.** *Every matrix $A \in M_{m,n}$ has a* QR *decomposition,*

$$A = QR,$$

where $Q \in M_m$ is unitary and $R \in M_{m,n}$ is zero in every entry (i,j) such that $i > j$ (and is therefore referred to as quasi upper-triangular*).*

Proof. We will construct Q from a Gram-Schmidt orthogonalization process similar to the one used in Proposition A.5, but when we reach the j'th column of A, if it is linearly dependent on the previous columns, we will choose its corresponding column of Q from the orthogonal complement of the span of the previous columns. If we continue this process until the first $\min\{m,n\}$ columns have been processed, forming some matrix $\hat{Q} \in M_{m,\min\{m,n\}}$ with orthonormal columns with same span as the columns of A, then we may apply Proposition A.7 in order to extend it into a unitary matrix $Q \in M_m$. But then, for any $j < \min\{m,n\}$, column i of Q is orthogonal to column j of A if $i > j$. This is precisely equivalent to the statement that $Q^H A = R$ for some quasi upper-triangular matrix R, which shows the result. $\square$

Remark A.3.1. The following theorem is being presented in somewhat of a different context than normal. In particular, please keep in mind that we have only proven that the spectrum of $A \in M_n$ is non-empty. However, once we establish the following theorem (via an admittedly tedious inductive argument), we will immediately be able to say much more about the spectrum of A.

Theorem A.10 (Schur decomposition [74]). *Every square matrix $A \in M_n$ can be decomposed as*

$$A = QTQ^H, \tag{A.3.9}$$

where Q is unitary and T is upper triangular.

Proof. Due to Theorem A.5, there exists an eigenpair (λ_0, x_0) of A, so that $Ax_0 = \lambda_0 x_0$, and, without loss of generality, we may choose x to have unit norm. Then Corollary A.8 implies that we may extend x_0 into an orthonormal basis for $\mathbb{C}^n$, and so we may define the unitary matrix $Q_0 = (x_0, X_0)$ via this extension. Then

$$Q_0^H(AQ_0) = Q_0^H(\lambda x_0, AX_0) = \begin{pmatrix} \lambda & r_0 \\ 0 & B_0 \end{pmatrix},$$

where the zero in the bottom-left of the last term is due to the orthonormality of the columns of Q_0. We can of course rewrite this equation as

$$A = Q_0 \begin{pmatrix} \lambda & r_0 \\ 0 & B_0 \end{pmatrix} Q_0^H,$$

and it is natural to attempt to recursively apply our approach to B_0, as Proposition A.2 shows that the product of unitary matrices is also unitary. The reader need only recognize that

$$\begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix} \begin{pmatrix} R_{TL} & R_{TR}U \\ 0 & R_{BR} \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix}^H = \begin{pmatrix} R_{TL} & R_{TR} \\ 0 & UR_{BR}U^H \end{pmatrix},$$

and that U unitary implies that $\begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix}$ is also unitary in order to complete the proof. $\square$

Corollary A.11. *Given a Schur decomposition QTQ^H of a matrix $A \in M_n$, the diagonal values of the upper-triangular matrix T are the eigenvalues of A. More precisely, $\lambda \in \sigma(A)$ if an only if there is some index j such that $T(j,j) = \lambda$.*

Proof. Since $A - \lambda I = QTQ^H - \lambda I = Q(T - \lambda I)Q^H$, $\lambda \in \sigma(A)$ if and only if $\lambda \in \sigma(T)$. But Lemma A.1 shows that $\lambda \in \sigma(T)$ if and only if λ lies on the diagonal of T. $\square$

Remark A.3.2. We conclude this section with a slight generalization of Theorem A.10 with important consequences which will be discussed in the next section.

Theorem A.12 (simultaneous Schur decomposition [74])**.** *Every pair of commuting matrices $A, B \in M_n$ has a* simultaneous Schur decomposition

$$A = QSQ^H, \quad B = QTQ^H,$$

where Q is unitary and both S and T are upper-triangular.

Proof. We may essentially repeat the proof of Theorem A.10, but, at each step, we may apply Theorem A.9 instead of Theorem A.5 in order to simultaneously reduce both matrices to triangular form. $\square$

Theorem A.13 (singular value decomposition [58])**.** *Any matrix $A \in M_{m,n}$ can be decomposed as*

$$A = U\Sigma V^H,$$

where $U \in M_m$ and $V \in M_n$ are unitary, and $\Sigma \in M_{m,n}$ is at most nonzero in the first $\min\{m, n\}$ *entries of its main diagonal, and these non-negative entries are called the* singular values. *The columns of U and V are respectively referred to as the* left *and* right singular vectors. *We will denote the j'th diagonal value of Σ by σ_j, where $0 \le j < \min\{m, n\}$.*

Proof. We may prove this important decomposition in a manner very similar to the Schur decomposition, but rather than choosing an eigenpair at each step of the recursive process, we instead choose a unit-vector which, when linearly transformed by A, produces a vector of maximal norm. In particular, we begin by choosing some unit-vector $x_0 \in \mathbb{C}^n$ such that $\|Ax_0\|_2 = \|A\|_2$ (and such a vector exists due to the Heine-Borel theorem). Then we may choose another unit-vector y_0 such that $Ax_0 = \sigma_0 y_0$, where $\sigma_0 = \|Ax_0\|_2 = \|A\|_2$, and then respectively extend x_0 and y_0 into unitary matrices $U_0 = (x_0, X_0) \in M_m$ and $V_0 = (y_0, Y_0) \in M_n$ via Corollary A.8.

We then find that

$$U_0^H A V_0 = U_0^H(\sigma_0 y_0, AX_0) = \begin{pmatrix} \sigma_0 & r_0^H \\ 0 & B_0 \end{pmatrix},$$

for some vector $r_0 \in \mathbb{C}^{n-1}$ and matrix $B_0 \in M_{m-1,n-1}$. The key observation [58] is that

$$(U_0^H A V_0) \begin{pmatrix} \sigma_0 \\ r_0 \end{pmatrix} = \begin{pmatrix} \sigma_0^2 + r_0^H r_0 \\ B_0 r_0 \end{pmatrix},$$

which implies that

$$\|A\|_2 = \|U_0^H A V_0\|_2 \ge \frac{\sqrt{(\sigma_0^2 + r_0^H r_0)^2 + \|B_0 r_0\|_2^2}}{\sqrt{\sigma_0^2 + r_0^H r_0}} \ge \sqrt{\sigma_0^2 + r_0^H r_0},$$

which contradicts $\|A\|_2 = \sigma_0$ whenever $r_0 \ne 0$. We have therefore shown that $r_0 = 0$ so that we may write

$$A = U_0 \begin{pmatrix} \sigma_0 & 0 \\ 0 & B_0 \end{pmatrix} V_0^H,$$

and an inductive technique essentially identical to that of Theorem A.10 completes the proof. □

Definition A.3.7 (range,rank,nullity)**.** The *range* of a matrix $A \in M_{m,n}$ is the set of all vectors $y \in \mathcal{C}^m$ such that $Ax = y$ for some $x \in \mathbb{C}^n$. We denote it as

$$\mathrm{r}an\, A = \{y \in \mathbb{C}^n : Ax = y \text{ for some } x \in \mathbb{C}^n\}.$$

The dimension of the range is referred to as the *rank* of the matrix, while the dimension of the kernel is called its *nullity.*

Corollary A.14 (rank-nullity)**.** *The dimensions of the range and null space of a matrix $A \in M_n$ add up to n, and, in particular, given a singular value decomposition $A = U\Sigma V^H$,*

$$ran\, A = ran\, U\Sigma, \tag{A.3.10}$$

and

$$\ker\, A = \ker\, \Sigma V^H. \tag{A.3.11}$$

That is to say, the rank is equal to the number of nonzero singular values, and the nullity is equal to n minus this number.

Proof. The result is essentially immediate now that we have the existence of the SVD: the number of linearly independent columns in $U\Sigma$ is the number of nonzero singular values, and a similar argument holds for ΣV^H. □

Definition A.3.8 (consistent norm)**.** A matrix norm is said to be *consistent* if, for any $A, B \in M_n$,

$$\|AB\| \leq \|A\|\|B\|.$$

Proposition A.15 (consistency of two-norm)**.** *The operator two-norm is consistent.*

Proof. Let $A, B \in M_n$ have singular value decompositions $A = U_A \Sigma_A V_A^H$ and $B = U_B \Sigma_B V_B^H$, and recognize that $\|A\|_2$ is the largest singular value of A. Then

$$\begin{aligned} \|AB\|_2 = \|U_A \Sigma_A V_A^H U_B \Sigma_B V_B^H\|_2 &= \|\Sigma_A V_A^H U_B \Sigma_B\|_2 \\ \leq \|A\|_2 \|B\|_2 \|V_A^H U_B\|_2 &= \|A\|_2 \|B\|_2. \end{aligned}$$

□

A.4 The spectral theorem

Definition A.4.1 (normal matrix)**.** A matrix A is said to be *normal* if it commutes with its adjoint, which is to say,

$$AA^H = A^H A.$$

Theorem A.1 (spectral theorem)**.** *A matrix $A \in M_n$ has a* spectral decomposition*,*

$$A = Q \Lambda Q^H, \tag{A.4.12}$$

where Q is unitary and Λ is diagonal, if and only if A is normal. Since we may rewrite this equation as

$$AQ = Q\Lambda,$$

we see that

1. *the j'th column of q and diagonal value of Λ are an eigenpair of A, and*

2. *the eigenvectors of A span $\mathbb{C}^n$.*

These observations justify the name spectral theorem*.*

Proof. Suppose that A is normal. Then A commutes with its adjoint, and Theorem A.12 shows that we may find a simultaneous Schur decomposition of A and A^H, say $A = QSQ^H$ and $A^H = QTQ^H$, where Q is unitary and S and T are both upper-triangular. Then, we may equate the adjoint of the Schur decomposition of A with the Schur decomposition of A^H in order to find that $S^H = T$, where S^H is lower-triangular and T is upper-triangular. But, this is only possible when both S and T are diagonal, so that $\overline{S} = T$ and $A = QSQ^H$ is in fact a spectral decomposition of A.

Now suppose that A has a spectral decomposition $A = Q\Lambda Q^H$. Then $AA^H = (Q\Lambda Q^H)(Q\Lambda^H Q) = Q|\Lambda|^2 Q^H = (Q\Lambda^H Q^H)(Q\Lambda Q^H) = A^H A.$ □

Definition A.4.2 (Hermitian matrix)**.** A matrix $A \in M_n$ is said to be *Hermitian* when it equals its adjoint, i.e.,

$$A = A^H.$$

Since every matrix commutes with itself, every Hermitian matrix is normal.

Corollary A.2 (Hermitian spectral decomposition)**.** *A matrix $A \in M_n$ can be decomposed as*

$$A = Q\Lambda Q^H,$$

where Q is unitary and Λ is a real, diagonal matrix consisting of the eigenvalues of A.

Proof. Since A is normal, we know that $A = Q\Lambda Q^H$ for some unitary Q and diagonal Λ (consisting of the eigenvalues of A), but we do not yet know that Λ is real. But this is easily shown since $A = A^H$ implies that $\Lambda = \Lambda^H$, and since Λ is diagonal, its values must also be real. □

Remark A.4.1. Given any matrix $A \in M_{m,n}$, its singular value decomposition, $A = U\Sigma V^H$, can be interpreted in terms of the Hermitian eigenvalue decompositions of AA^H and A^HA. In particular,

$$A^HA = (U\Sigma V^H)^H(U\Sigma V^H) = V(\Sigma^H\Sigma)V^H,$$

which shows that the right singular vectors, V, are the eigenvectors of A^HA, and the associated eigenvalues lie along the diagonal of $\Sigma^H\Sigma$. Due to the structure of Σ, the first $\min\{m,n\}$ eigenvalues are equal to σ_j^2, while the remaining $n - \min\{m,n\}$ eigenvalues are equal to zero. Likewise,

$$AA^H = U(\Sigma\Sigma^H)U^H,$$

which also has σ_j^2 as its j'th eigenvalue, for $0 \leq j < \min\{m,n\}$, and the remaining $m - \min\{m,n\}$ eigenvalues are zero.

A.5 Generalized eigenspaces and matrix polynomials

We still have much left to prove about the spectrum of general matrices. For instance, though we have shown that a scalar λ is an eigenvalue of a general matrix $A \in M_n$ if and only if it lies on the diagonal of the triangular matrix produced by its Schur decomposition, we have said nothing about *how many times it appears*. We will now provide definitions of the *multiplicity* of an eigenvalue in the spirit of [14].

Definition A.5.1 (eigenspace)**.** The eigenspace of an eigenvalue λ of a matrix $A \in M_n$ is the set of all eigenvectors corresponding to the eigenvalue λ,

$$\mathcal{W}_\lambda = \{x \in \mathbb{C}^n : (A - \lambda I)x = 0\}. \tag{A.5.13}$$

Notice that $\mathcal{W}_\lambda = \ker(A - \lambda I)$.

Example A.5.1. Unfortunately not all square matrices have eigenspaces which span their vector space. Consider the matrix

$$T = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix},$$

which, because it is upper triangular, has the spectrum $\sigma(T) = \{0\}$. It is easy to see that its only eigenspace is

$$\mathcal{W}_0 = \operatorname{span} \begin{pmatrix} 1 \\ 0 \end{pmatrix},$$

which clearly does not span $\mathbb{C}^2$. Thankfully we may loosen our definition of an eigenspace in order to span the entire vector space.

Definition A.5.2 (generalized eigenvector [14])**.** A nonzero vector $x \in \mathbb{C}^n$ is called a *generalized eigenvector* for a matrix $A \in M_n$ if there exists some integer $k \geq 1$ and some scalar $\lambda \in \mathbb{C}$ such that

$$(A - \lambda I)^k x = 0.$$

Proposition A.1. *Every $\lambda \in \mathbb{C}$ which admits a generalized eigenvector for a matrix $A \in M_n$ must be a member of the spectrum of A.*

Proof. Since the product of nonsingular matrices is again nonsingular, $A - \lambda I$ must be singular in order for $(A - \lambda I)^k$ to map a nonzero vector x to the zero vector. □

Definition A.5.3 (generalized eigenspace)**.** The *generalized eigenspace* for an eigenvalue $\lambda \in \sigma(A)$ is defined as

$$\mathcal{G}_\lambda = \{x \in \mathbb{C}^n : (A - \lambda I)^k x = 0 \text{ for some } k \geq 1\}.$$

Notice that, $\mathcal{W}_\lambda \subset \mathcal{G}_\lambda$ by definition.

Proposition A.2 ([14])**.** *For any $A \in M_n$ and $\lambda \in \sigma(A)$,*

$$\mathcal{G}_\lambda = \ker (A - \lambda I)^n.$$

Proof. Since $(A-\lambda I)^k x = 0$ implies that $(A-\lambda I)^{k+1} x = 0$, we need only show that, if $(A-\lambda I)^k x = 0$, where $k > n$, then we also have that $(A-\lambda I)^n x = 0$. In order to do so, we will show that, if k is the first index such that $(A-\lambda I)^k x = 0$, then the vectors $\{x, (A-\lambda I)x, \ldots, (A-\lambda I)^{k-1}x\}$ are linearly independent, and the result will follow from Lemma A.1.

Suppose that

$$\sum_{j=0}^{k-1} \alpha_j (A-\lambda I)^j x = 0.$$

Since each of the terms $(A-\lambda I)^j$, $j < k$, is nonzero by assumption, showing that each α_j is zero will show that $\{x, (A-\lambda I)x, \ldots, (A-\lambda I)^{k-1}\}$ are linearly independent. If we multiply each term in this equation by $(A-\lambda I)^{k-1}$, then $(A-\lambda I)^{j+k-1}x = 0$ for each $j \geq 1$, which shows that $\alpha_0 = 0$. Similarly, if we multiply each term by $(A-\lambda I)^{k-2}$, we discover that $\alpha_1 = 0$, and we may repeat the process to find that each α_j is 0. Since the vectors $\{(A-\lambda I)^j x\}_{j=0}^{k-1}\}$ are linearly independent, we must have that $k \leq n$. $\square$

Lemma A.3. *Suppose that the subspace $\mathcal{W} \subset \mathbb{C}^n$ is invariant under $A \in M_n$, and Q is a matrix whose columns form an orthonormal basis for $\mathcal{W}$. Then, if y is a generalized eigenvector of $Q^H A Q$, Qy is a generalized eigenvector of A.*

Proof. If y is a generalized eigenvector of $Q^H A Q$, then $(Q^H A Q - \lambda I)^k y = 0$ for some $\lambda \in \mathbb{C}$ and $k \leq n$. Then, we may express the last equation term-by-term (using the binomial formula) as

$$\sum_{j=0}^{k} \binom{k}{j} (Q^H A Q)^{k-j} (-\lambda)^j y = 0,$$

and, due to the invariance of $\mathcal{W}$ under A, $QQ^HAQ = AQ$, and $Q^HAQz = 0$ if and only if $AQz = 0$, so we may write

$$\left(\sum_{j=0}^{k} \binom{k}{j} (-\lambda)^j A^{k-j}\right) Qy = 0.$$

But this is precisely equal to the statement

$$(A - \lambda I)^k (Qy) = 0,$$

which shows that Qy is a generalized eigenvector of A. □

Definition A.5.4 (Rayleigh quotient)**.** For any matrix $A \in M_n$ and matrix $Q \in M_{n,m}$ with orthonormal columns, the matrix

$$R_A(Q) = Q^HAQ$$

is called the *Rayleigh quotient* of A with respect to Q. In some sense, it is the projection of A onto the space spanned by the columns of Q. It will be fundamental to our discussion of Krylov subspace methods in Appendix B.

Lemma A.4 ([14])**.** *For any matrix $A \in M_n$ and eigenvalue $\lambda \in \sigma(A)$, we may strengthen the rank-nullity theorem for $(A - \lambda I)^n$ to say that*

$$ran\,(A - \lambda I)^n \oplus \ker\,(A - \lambda I)^n = \mathbb{C}^n.$$

Proof. We need only show that $\operatorname{ran}(A-\lambda I)^n \cap \ker(A-\lambda I)^n = \{0\}$ in order to have the result follow from the rank-nullity theorem. So, suppose $x \in \mathbb{C}^n$ is in both the range and kernel of $(A-\lambda I)^n$. Because x is in the range of $(A-\lambda I)^n$, there is some y such that $(A-\lambda I)^n y = x$. But then $(A-\lambda I)^{2n} y = (A-\lambda I)^n x = 0$, where the last equality follows since x is in the kernel of $(A - \lambda I)^n$. But we have then shown that $y \in \mathcal{G}_\lambda(A)$, which implies that $(A - \lambda I)^n y = 0$ by Proposition A.2. But then $x = 0$, which shows the result. □

Theorem A.5 ([14]). *The generalized eigenvectors of a matrix $A \in M_n$ span $\mathbb{C}^n$.*

Proof. The result follows from an inductive argument which decomposes $\mathbb{C}^n$ into the range and kernel of $(A-\lambda I)^n$, via Lemma A.4, and then uses Lemma A.3 in order to show that the generalized eigenvectors of the projection of $(A-\lambda I)^n$ onto its range are also generalized eigenvectors of $(A-\lambda I)^n$. Please see [14] for a detailed proof. $\square$

Theorem A.6. *Given a matrix $A \in M_n$, every vector $x \in \mathbb{C}^n$ can be written as a unique linear combination of members of the generalized eigenspaces of A, i.e.,*

$$\bigoplus_{\lambda \in \sigma(A)} \mathcal{G}_\lambda = \mathbb{C}^n.$$

Furthermore, if $dim\,\mathcal{G}_\lambda = k$ and $y \in \mathcal{G}_\lambda$, then $(A-\lambda I)^k y = 0$.

Proof. Please see [14] for details. $\square$

Corollary A.7. *The eigenvectors of a matrix $A \in M_n$ span $\mathbb{C}^n$ if and only if $\mathcal{W}_\lambda = \mathcal{G}_\lambda$ for each $\lambda \in \sigma(A)$.*

Definition A.5.5 (characteristic polynomial)**.** For any $A \in M_n$, define its *characteristic polynomial*, $p_A \in \mathbb{P}_n$, as

$$p_A(z) = \prod_{\lambda \in \sigma(A)} (z-\lambda)^{\dim \mathcal{G}_\lambda}.$$

Theorem A.8 (Cayley-Hamilton)**.** *Every matrix satisfies its characteristic polynomial, i.e., $p_A(A) = 0$.*

Proof. Let x be an arbitrary member of $\mathbb{C}^n$, which, by Theorem A.6, we may decompose as

$$x = \sum_{\lambda \in \sigma(A)} g_\lambda,$$

where $g_\lambda \in \mathcal{G}_\lambda$. Then

$$p_A(A)x = \left(\prod_{\lambda \in \sigma(A)} (A - \lambda I)^{\dim \mathcal{G}_\lambda} \right) x,$$

and each of the matrices in the product commutes. Thus, for each component g_λ of x,

$$p_A(A)g_\lambda = \left(\prod_{\xi \in \sigma(A)} (A - \xi I)^{\dim \mathcal{G}_\xi} \right) g_\lambda,$$

and, if we evaluate $(A - \lambda I)^{\dim \mathcal{G}_\lambda} g_\lambda$ first, Theorem A.6 implies that it is zero, so we see that

$$p_A(A)x = \sum_{\lambda \in \sigma(A)} p(A)g_\lambda = 0.$$

□

Definition A.5.6 (eigenvalue multiplicity [14])**.** We say that the *algebraic multiplicity* of an eigenvalue $\lambda \in \sigma(A)$ is the dimension of its corresponding generalized eigenspace, $\mathcal{G}_\lambda$, while its *geometric multiplicity* is the dimension of its standard eigenspace, $\mathcal{W}_\lambda$. This is consistent with the usual definitions based upon the determinant.

Proposition A.9 (diagonalizability)**.** *Every matrix $A \in M_n$ which can be decomposed as*

$$A = X \Lambda X^{-1},$$

where Λ is diagonal, is called diagonalizable*. Since $AX = X\Lambda$ is an expression of the eigenpairs of A, and X can be inverted if and only if the eigenvectors* *span $\mathbb{C}^n$, Corollary A.7 shows that A is diagonalizable if and only if the geometric multiplicity of every eigenvalue is equal to its algebraic multiplicity.*

Remark A.5.1. We have already shown that all normal matrices are diagonalizable (in fact, with a unitary eigenvector matrix).

Proposition A.10. *For any polynomial $p \in \mathbb{P}_k$ and diagonalizable matrix $A \in M_n$, where $A = X\Lambda X^{-1}$,*

$$p(A) = Xp(\Lambda)X^{-1}.$$

Proof. Since each polynomial is a sum of monomials, we need only consider $(X\Lambda X^{-1})^k$, where the result immediately follows. $\square$

Remark A.5.2. It can be shown that the eigenvalues of a matrix $A \in M_n$ depend continuously on the entries of A [74], which implies that an arbitrarily small perturbation of any non-diagonalizable matrix may be used to separate every cluster of eigenvalues with geometric multiplicity less than its algebraic multiplicity. But then every matrix is arbitrarily close to a diagonalizable matrix (that is to say, the class of diagonalizable matrices is *dense* in M_n), so it is tempting to think that one could always find a suitably accurate eigenvalue decomposition for a nondiagonalizable matrix. The problem with this idea is that the resulting eigenvector matrix would almost certainly be so close to singular as to be numerically useless. For detailed discussions of the subject of *perturbation theory*, please see one of the many excellent texts on the subject, such as [133], [73], and [34].

A.6 Summary

A compact introduction to the classical linear algebra theorems needed for discussing Krylov subspace methods has been given, with an emphasis placed upon using basic dimensional arguments (in the spirit of [14]) as a replacement for cumbersome arguments based upon determinants. The following appendix makes use of these theorems in order to provide a rigorous introduction to Krylov subspace methods and the Generalized Minimum Residual method (GMRES), which are both fundamental to prerequisites for Chapters 3 and 4.

Appendix B

Krylov subspace methods

Krylov subspace methods, which are intimately related to the *power method* [119], provide an economical means of finding approximate eigenpairs and, as a sideproduct, also yield algorithms for the approximate solution of linear systems. Let us recall that power iteration proceeds by computing the sequence of vectors r, Ar, A^2r, $\ldots$, $A^{k-1}r$ in the hope that, for a randomly chosen starting vector r, $A^{k-1}r$ quickly becomes an accurate approximation to the dominant eigenvector of A.

If we also recall the definition of a Krylov subspace (Definition A.2.7), then we see that power iteration produces precisely the vectors which generate the Krylov subspace,

$$\mathcal{K}_k(A, r) = \operatorname{span}\{r, Ar, \ldots, A^{k-1}r\} = \mathbb{P}_{k-1}(A)r, \tag{B.1}$$

where $\mathbb{P}_{k-1}$ is the space of polynomials of degree $k-1$. The difference between a Krylov subspace eigensolver and the power method is qualitatively simple: rather than choosing the eigenvector estimate to be the most recent iterate, $A^{k-1}r$, Krylov eigensolvers find the "best" estimate available from linear combinations of the entire set of iterates.

Arguably the most important feature of Krylov subspaces is that, with a typically modest amount of computation, one arrives at a subspace which is approximately invariant under A. More specifically, given a matrix V whose columns span a Krylov subspace, it is often the case that the columns of AV are approximately contained in the same Krylov subspace. It will be helpful to introduce a few more concepts before attempting to clarify this statement and its important implications.

It is worth noting that this appendix draws heavily from the content and style of [119], [107], and [122]. In particular, the focus on Rayleigh quotients and Krylov decompositions is due to [119], and much of the Chebyshev polynomial approximation discussion is drawn from [107].

B.1 Rayleigh quotients

For an eigenvector estimate v, the corresponding eigenvalue estimate can be cheaply computed through the *Rayleigh quotient*,

$$R_A(v) = \frac{v^H A v}{v^H v},$$

which simplifies to $R_A(v) = v^H A v$ when v is a unit vector. In particular, because $R_A(v)$ is continuous, and the Rayleigh quotient of an exact eigenvector is its exact eigenvalue, we can expect approximate eigenvectors to be mapped to approximate eigenvalues.

We now generalize the notion of a Rayleigh quotient from *translating an approximate eigenvector into an approximate eigenvalue* into a mapping which *translates an approximate invariant subspace into a matrix whose eigenvalues approximate those of A.* As in Definition A.5.4, for any matrix V with orthonormal columns, the Rayleigh quotient is equal to

$$R_A(V) = V^H A V.$$

Definition B.1.1 (projection matrix)**.** The *projection* of a vector $x \in \mathbb{C}^n$ onto the space spanned by the (orthonormal) columns of a matrix V is given by

$$\mathcal{P}_V x = VV^H x,$$

while the projection of x onto the *orthogonal complement* (Definition A.3.6) of the space spanned by the columns of V is equal to

$$\mathcal{P}_{V^\perp} x = (I - VV^H)x.$$

Notice that, if $x \in \operatorname{span} V$, then $\mathcal{P}_V x = x$, and $\mathcal{P}_{V^\perp} x = 0$.

Proposition B.1. *If V is a matrix with orthonormal columns such that span V is an invariant subspace of a matrix A, then*

$$A(Vy) = \lambda(Vy) \Leftrightarrow R_A(V)y = \lambda y.$$

Proof. We first assume that $A(Vy) = \lambda(Vy)$. Then $V^H AVy = \lambda V^H Vy = \lambda y$ since V has orthonormal columns.

Now assume that $V^H AVy = \lambda y$. Then $VV^H A(Vy) = \lambda(Vy)$, which is identically equal to $\mathcal{P}_V(AVy) = \lambda(Vy)$. Since Vy is a member of span V, which is invariant under A, $A(Vy) = \mathcal{P}_V(AVy)$. $\square$

Due to the continuity of the Rayleigh quotient, we can expect any V whose range is nearly an invariant subspace of A to map to a matrix whose eigenpairs yields approximate eigenpairs of A: For any matrix V with orthonormal columns, we can set $Z = R_A(V) = V^H AV$ and decompose

$$AV = \mathcal{P}_V AV + \mathcal{P}_{V^\perp} AV = VZ + \mathcal{P}_{V^\perp} AV. \tag{B.1.2}$$

We can thus characterize how close V is to being an invariant subspace through the size of the residual matrix,

$$E \equiv \mathcal{P}_{V^\perp} AV = (I - VV^H)AV = AV - VR_A(V). \tag{B.1.3}$$

Given any eigenpair of the Rayleigh quotient, say $Zy = \lambda y$, we see that

$$AVy = \lambda Vy + Ey, \tag{B.1.4}$$

which shows that (λ, Vy) can be interpreted as an approximate eigenpair of A, known as a *Ritz pair*. The norm of the vector Ey then provides a means of characterizing the error in the approximation.

Another useful interpretation is that a Ritz pair (λ, Vy) is an exact eigenpair of a perturbation of A, i.e., if we set $\hat{A} = A - EV^H$, then

$$\hat{A}Vy = \lambda Vy, \tag{B.1.5}$$

where we have simply made use of the fact that $V^H V = I$ in order to rearrange Equation (B.1.4). Since $\|A - \hat{A}\|_2 = \|EV^H\|_2 = \|E\|_2$, we see that, when V is close to an invariant subspace of A, Ritz pairs are eigenpairs of a matrix which is close to A.

B.2 Krylov decompositions

It is easy to see that the residual matrix resulting from a basis for a Krylov subspace is at most rank-one.

Lemma B.1. *Using the shorthand notation that* $\mathcal{K}_k \equiv \mathcal{K}_k(A, r)$,

$$\mathcal{K}_{k+1} \cap \mathcal{K}_k^\perp = \mathcal{P}_{\mathcal{K}_k^\perp}(\mathcal{K}_{k+1}) = span\, \mathcal{P}_{\mathcal{K}_k^\perp}(A^k r).$$

Proof. By definition,

$$\mathcal{K}_{k+1} = \operatorname{span}\{\mathcal{K}_k, A^k r\},$$

and thus

$$\mathcal{P}_{\mathcal{K}_k^\perp}(\mathcal{K}_{k+1}) = \operatorname{span}\mathcal{P}_{\mathcal{K}_k^\perp}(A^k r).$$

□

Lemma B.2. *If a matrix V has orthonormal columns whose span is equal to the Krylov subspace $\mathcal{K}_k(A, r)$, then we may find vectors v and w such that*

$$\mathcal{P}_{V^\perp} A V = v w^H,$$

where $v \in span\,\mathcal{P}_{V^\perp}(A^k r)$ and $\|vw^H\|_2 = \|w\|_2$.

Proof. Due to Lemma B.1, each column of $\mathcal{P}_{V^\perp}AV$ is a scalar multiple of $u = \mathcal{P}_{V^\perp}(A^k r)$. If $u = 0$, we may set both v and w equal to zero. Otherwise, we may set $v = u/\|u\|_2$ and the j'th entry of w equal to $\mathcal{P}_{V^\perp}(Av_j)^H v$. □

We can now state the main result for this section.

Theorem B.3 (Orthonormal Krylov decomposition)**.** *If a matrix V has orthonormal columns whose span is equal to the Krylov subspace $\mathcal{K}_k(A, r)$, then we may decompose its image AV as*

$$AV = VZ + vw^H,$$

where $Z = R_A(V)$, $v \in span\,\mathcal{P}_{V^\perp}(A^k r)$, and $\|vw^H\|_2 = \|w\|_2$.

Proof. Equation (B.1.2) shows that, for *any* matrix V with orthonormal columns, we may decompose AV as $VR_A(V) + \mathcal{P}_{V^\perp}AV$. Lemma B.2 exploits the fact that $\operatorname{span} V$ is a Krylov subspace in order to yield the necessary rank-one decomposition of the second term. □

The "orthonormal" qualifier hints at the fact that, with a proper generalization of the Rayleigh quotient, we may lift the requirement that V must have orthonormal columns [119]. However, for the purposes of this dissertation, orthonormal Krylov decompositions are more than sufficient.

B.3 Lanczos and Arnoldi decompositions

Despite their generality, Krylov decompositions are a relatively modern means of working with Krylov subspaces [118]. Traditional approaches place further restrictions on the basis vectors in order to expose additional structure. For the remainder of this section, we will assume that the Krylov subspace $\mathcal{K}_k(A, r)$ is full-rank and make use of the conventions that $\mathcal{K}_0(A, r) = \{0\}$ and $V_j = [v_0, \ldots, v_{j-1}]$.

Lemma B.1. *If a matrix V has orthonormal columns chosen such that*

$$v_j \in \mathit{span}\, \mathcal{P}_{V_j^\perp}(A^j r) = \mathcal{K}_{j+1} \cap \mathcal{K}_j^\perp, \tag{B.3.6}$$

then the Rayleigh quotient $R_A(V)$ is upper Hessenberg.

Proof. The (i, j) entry of $R_A(V) = V^H AV$ is given by $v_i^H Av_j$, and $v_i \in \mathcal{K}_i^\perp$ is orthogonal to $Av_j \in \mathcal{K}_{j+2}$ as long as $i \geq j + 2$. □

Lemma B.2. *If a matrix V consists of k orthonormal columns satisfying Equation* (B.3.6)*, then the residual matrix $E = \mathcal{P}_{V^\perp} AV$ is can only contain nonzero entries in its last column.*

Proof. $Av_j \in \mathcal{K}_{j+2} \subset \mathcal{K}_k = \operatorname{span} V$ for all $j \leq k - 2$, and so $\mathcal{P}_{V^\perp}(Av_j) = 0$ for the first $k - 1$ columns of V. □

Theorem B.3 (Arnoldi decomposition [8]). *If a matrix V consists of k orthonormal columns satisfying Equation* (B.3.6)*, then its image AV can be decomposed as*

$$AV = VH + v(\beta e_k)^H,$$

where $H = R_A(V)$ is upper Hessenberg, $v \in \operatorname{span} \mathcal{P}_{V^\perp}(A^k r)$, and $\|v(\beta e_k)^H\|_2 = \beta$.

Proof. We can again apply Equation (B.1.2) to show that, for *any* matrix V with orthonormal columns, we may decompose AV as $VR_A(V) + \mathcal{P}_{V^\perp}AV$. Lemma B.1 then shows that $R_A(V)$ is upper Hessenberg, and Lemma B.2 shows that only the last column of $\mathcal{P}_{V^\perp}AV$ can be nonzero. If we label this last column as u, then, when $u = 0$, we may set $v = 0$ and $\beta = 0$, otherwise, we may set $v = u/\|u\|_2$ and $\beta = \|u\|_2$. □

Corollary B.4 (Lanczos decomposition [85]). *If a matrix V consists of k orthonormal columns satisfying Equation* (B.3.6)*, and the underlying operator A is Hermitian, then the image AV can be decomposed as*

$$AV = VT + v(\beta e_k)^H,$$

where $T = R_A(V)$ is Hermitian and tridiagonal, $v \in \operatorname{span} \mathcal{P}_{V^\perp}(A^k r)$, and $\|v(\beta e_k)^H\|_2 = \beta$.

Proof. We may simply combine Theorem B.3 with the fact that, if A is Hermitian, $R_A(V) = V^H AV$ must also be Hermitian. Since Hermitian Hessenberg matrices are tridiagonal, we have finished the proof. □

B.4 Introduction to FOM and GMRES

Suppose that we are interested in the solution of the nonsingular system of equations

$$Ax = b,$$

and that we have constructed a matrix V whose columns are orthonormal and whose span both contains b and is invariant under A. It is natural to expect that we can project the system of equations onto $\operatorname{span} V$ in order to quickly solve the linear system. The following theorem shows that this is in fact possible.

Proposition B.1. *If a matrix V has orthonormal columns whose span is invariant under the nonsingular matrix A, then:*

(i) The Rayleigh quotient $Z = R_A(V)$ is invertible,

(ii) $A^{-1}b = VZ^{-1}V^Hb$ for all $b \in \operatorname{span} V$, and

(iii) $A : \operatorname{span} V \to \operatorname{span} V$ is a bijection.

Proof. Since $\operatorname{span} V$ is invariant under A, we may invoke Proposition B.1 to conclude that every eigenvalue of Z is also an eigenvalue of A. Because A is nonsingular, it cannot have a zero eigenvalue, and therefore Z must also be nonsingular.

Due to the invariance of $\operatorname{span} V$, $A(\operatorname{span} V) \subset \operatorname{span} V$, and because A is invertible, $A : \operatorname{span} V \to \operatorname{span} V$ is injective.

Given that both A and Z are now known to be invertible, we may rewrite the invariance relation $AV = VZ$ as $A^{-1}(VV^H) = VZ^{-1}V^H$, which shows that, for any $b \in \operatorname{span} V$, $A^{-1}b = VZ^{-1}V^H b \in \operatorname{span} V$. $A : \operatorname{span} V \to \operatorname{span} V$ is therefore surjective. $\square$

Corollary B.2. *If a matrix V has orthonormal columns whose span is the Krylov subspace $\mathcal{K}_k(A,b)$, then $\mathcal{K}_k(A,b) = \mathcal{K}_{k+1}(A,b)$ implies that $\mathcal{K}_k(A,b)$ is invariant under A, and that, if A is nonsingular, the unique solution to the linear system $Ax = b$ is given by $x = VZ^{-1}V^H b$.*

Proof. We have only to show that $\mathcal{K}_k(A,b) = \mathcal{K}_{k+1}(A,b)$ implies that $\mathcal{K}_k(A,b)$ is invariant under A. This fact can easily be seen since $A\mathcal{K}_k(A,b) \subset \mathcal{K}_{k+1}(A,b) = \mathcal{K}_k(A,b)$. $\square$

Definition B.4.1 (Full Orthogonalization Method (FOM))**.** Given a nonsingular linear system, $Ax = b$, and a matrix V composed of orthonormal columns whose span is the Krylov subspace $\mathcal{K}_k(A,b)$, any method which attempts to approximate the solution x with $VZ^{-1}V^H b$, even when $\operatorname{span} V$ is not necessarily invariant under A, is referred to as a *Full Orthogonalization Method* (FOM). The approximate solution, if it exists, will be denoted by x_k.

Remark B.4.1. Given any vector x_0, we can instead apply FOM to the linear system

$$A(x - x_0) = b - Ax_0,$$

where x_0 is referred to as the *initial guess*, and $r_0 \equiv b - Ax_0$ is called the *initial residual.* FOM could then be applied by attempting to construct the approximate solution $x_k = x_0 + VZ^{-1}V^H r_0$, where the columns of V span $\mathcal{K}_k(A, r_0)$ instead of $\mathcal{K}_k(A,b)$.

Just as in the previous sections, we have extended an insight which is precise for invariant subspaces to subspaces which are ideally close to being invariant. We will unfortunately now see that the FOM can fail catastrophically when $\operatorname{span} V$ is not invariant.

Example B.4.1. Let $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ and $b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, and set $V = b$ so that V has a single orthonormal column which spans the Krylov subspace $\mathcal{K}_1(A, b)$. Even though A is both nonsingular and Hermitian, the Rayleigh quotient $R_A(V) = b^H Ab$ is identically zero.

Despite the fact that FOM can completely fail on trivial problems, it turns out that a relatively minor modification results in an extremely robust algorithm. We will now show that, even though the inverse of the Rayleigh quotient can fail to exist, a unique solution can always be found in the least-squares sense.

Lemma B.3. *If* $\dim \mathcal{K}_{k+1}(A, b) = k + 1$ *and* $\operatorname{span} V = \mathcal{K}_k(A, b)$*, then the matrix* AV *has full column-rank.*

Proof. Because $\mathcal{K}_{k+1}(A, b) = \operatorname{span}\{b, AV\}$, the dimension of the column space of AV is at least $\dim \mathcal{K}_{k+1}(A, b) - 1 = k$, and thus $\operatorname{rank}(AV) = k$. □

Theorem B.4. *Given an Arnoldi decomposition* $AV = VH + v(\beta e_k)^H$*, where* $\operatorname{span} V = \mathcal{K}_k(A, b)$*, there exists a unique least-squares solution to the equivalent problems*

$$\arg\min_y \|AVy - b\|_2 = \arg\min_y \|\tilde{H}y - \beta e_1\|_2,$$

where $\tilde{H} = \begin{pmatrix} H \\ \|b\|_2 e_k^H \end{pmatrix}$.

Proof. If $\mathcal{K}_k(A, b)$ is invariant, then Corollary B.2 implies that $y = Z^{-1}V^H b$ is the unique (exact) solution. Otherwise, we must have that $\dim \mathcal{K}_{k+1}(A, b) =$

$k+1$, and so Lemma B.3 shows that the matrix AV has full column rank, which implies that a unique least-squares solution can be found, e.g., through a QR decomposition of AV.

If we set $\beta = \|b\|_2$, then $b = V(\beta e_1)$, and we can make use of the Arnoldi decomposition to find

$$AVy - b = (VH + v(\beta e_k)^H)y - V(\beta e_1).$$

Since the matrix $[Vv]$ has orthonormal columns, if we define $\tilde{H}$ as above, then

$$\arg\min_z \|AVz - b\|_2 = \arg\min_z \|\tilde{H}z - \beta e_1\|_2,$$

where the vector e_1 is now of length $k+1$. □

Definition B.4.2 (Generalized minimum residual method (GMRES) [108])**.** Given a nonsingular linear system, $Ax = b$, the label GMRES applies to any method which exploits an Arnoldi decomposition, say $AV = VH + v(\beta e_k)^H$, to find the minimizer

$$y = \arg\min_z \|AVz - b\|_2 = \arg\min_z \|\tilde{H}z - \beta e_1\|_2,$$

which implies that the minimum-residual solution is $x_k = Vy$.

Remark B.4.2. Just like FOM, GMRES can easily incorporate an initial guess x_0 by first finding

$$y = \arg\min_z \|AVz - r_0\|_2,$$

where $r_0 = b - Ax_0$ and the columns of V span $\mathcal{K}_k(A, r_0)$. The approximate solution to $Ax = b$ is then given by $x_k = x_0 + Vy$.

Definition B.4.3 (Minimum residual method (MINRES) [94])**.** Given a Hermitian nonsingular linear system, $Ax = b$, the label *MINRES* applies to any method which exploits a Lanczos decomposition, say $AV = VT + v(\beta e_k)^H$, to find the minimizer

$$y = \arg\min_z \|AVz - b\|_2 = \arg\min_z \|\tilde{T}z - \beta e_1\|_2,$$

which implies that the minimum-residual solution is $x_k = Vy$. That is to say, MINRES is the specialization of GMRES to Hermitian systems of equations.

Proposition B.5. *If GMRES is applied to the nonsingular linear system $Ax = b$, yielding an approximate solution x_k, then*

$$\|b - Ax_k\|_2 = \min_{p\in\mathbb{P}_{k-1}} \|b - Ap(A)b\|_2 = \min_{m\in\mathbb{M}_k} \|m(A)b\|,$$

where $\mathbb{P}_{k-1}$ is the space of $k-1$ degree polynomials and

$$\mathbb{M}_k = \{p \in \mathbb{P}_k : p(0) = 1\}$$

is the space of monic *polynomials of degree k.*

Proof. The left term immediately follows from the definition of GMRES and the fact that the search space is $\mathcal{K}_k(A, b) = \mathbb{P}_{k-1}(A)b$. The right term follows from the fact that $p \in \mathbb{P}_{k-1}$ if and only if $1 + xp(x) \in \mathbb{M}_k$. $\square$

Corollary B.6. *If GMRES is applied to the nonsingular system $Ax = b$, where A is diagonalizable, say, $A = W\Lambda W^{-1}$, then*

$$\|b - Ax_k\|_2 = \min_{m\in\mathbb{M}_k} \|Wm(\Lambda)W^{-1}b\|_2 \le \kappa_2(W) \cdot \min_{m\in\mathbb{M}_k} \|m\|_{\sigma(A)} \cdot \|b\|_2.$$

Proof. The first equality only requires that, for any polynomial p, $p(A) = p(W\Lambda W^{-1}) = Wp(\Lambda)W^{-1}$, which was shown in Proposition A.10. The inequality relies on the consistency of the matrix two-norm (Proposition A.15), i.e.,

$$\|Wm(\Lambda)W^{-1}\|_2 \le \|W\|_2\|W^{-1}\|_2\|m(\Lambda)\|_2 = \kappa_2(W) \cdot \max_{\lambda \in \sigma(A)} |m(\lambda)|.$$

$\square$

Corollary B.7. *If GMRES is applied to the nonsingular linear system $Ax = b$, where A is normal, and the spectral decomposition of A is $Q\Lambda Q^H$, then*

$$\|b - Ax_k\|_2 = \min_{m \in \mathbb{M}_k} \|m(\Lambda)Q^H b\|_2 \le \min_{m \in \mathbb{M}_k} \|m\|_{\sigma(A)} \cdot \|b\|_2.$$

Proof. We have made use of two elementary facts about unitary matrices in order to improve upon Corollary B.6: $\kappa_2(Q) = 1$, and $\|Qx\|_2 = \|x\|_2$ for any vector x. $\square$

Remark B.4.3. The previous bounds show that the residual norms produced by GMRES are at least weakly controlled by the conditioning of the eigenvectors and the distribution of the eigenvalues. Before we begin to discuss the important subject of *preconditioning*, which, in the context of GMRES, loosely corresponds to mitigating the effects of these two terms, it is important that we touch on how to replace the $\|m\|_{\sigma(A)}$ term with something more concrete. The main idea is that, if the spectrum is known to be contained within some well-defined region, then we may propose a monic polynomial, evaluate its maximum magnitude over the region of interest, and then use this value as an upper-bound for the minimum value over all polynomials.

Example B.4.2. Suppose that $A = I - \gamma U$, where $|\gamma| < 1$ and U is an arbitrary unitary matrix with spectral decomposition $Q\Lambda Q^H$. Then each eigenvalue of U lies on the unit circle, and, since $A = Q(I - \gamma\Lambda)Q^H$, the eigenvalues of A lie on the circle of radius $|\gamma|$ centered at 1. It is then natural to investigate the candidate polynomial

$$m(z) = (1 - z)^k \in \mathbb{M}_k,$$

which evaluates to 1 at the origin and γ^k over the spectrum of A. Then, by Corollary B.7, we have that

$$\|b - Ax_k\|_2 \le |\gamma|^k \|b\|_2.$$

Remark B.4.4. It turns out that, as the spectrum becomes dense on the boundary of the circle, the candidate polynomial from the last example becomes optimal. This can easily be shown using the following theorem due to S. Bernstein, which we state without proof.

Theorem B.8 (S. Bernstein [6, 20])**.** *If $p \in \mathbb{P}_k$ and $\|p\|_{C_1(0)} = 1$, then $\|p\|_{C_R(0)} \le R^k$ for $R \ge 1$, with equality only if $p(z) = \gamma z^k$, with $|\gamma| = 1$.*

Corollary B.9. *For any radius $r < |z_0|$,*

$$\min_{m \in \mathbb{M}_k} \|m\|_{C_r(z_0)} = \frac{r^k}{|z_0|^k},$$

and the minimizing monic polynomial is given by $(z_0 - z)^k / z_0^k$.

Proof. Showing that this value is attainable simply requires a rescaling of the previous example. In particular, we can define the monic polynomial

$$m(z) = \frac{1}{z_0^k}(z_0 - z)^k \in \mathbb{M}_k,$$

which, for any $z \in C_r(z_0)$, evaluates to $r^k/|z_0|^k$.

Now suppose that there exists $\tilde{m}(z) \in \mathbb{M}_k$ such that

$$\|\tilde{m}\|_{C_r(z_0)} = \frac{\eta^k}{|z_0|^k} < \frac{r^k}{|z_0|^k}.$$

We may then define the polynomial

$$p(z) = \frac{|z_0|^k}{\eta^k}\tilde{m}(zr + z_0),$$

then we see that $\|p\|_{C_1(0)} = 1$, and, setting $R = |z_0|/r$,

$$p(-z_0/r) = \frac{|z_0|^k}{\eta^k}\tilde{m}(0) = \frac{|z_0|^k}{\eta^k} > R^k,$$

which contradicts Theorem B.8. $\square$

Corollary B.10. *For any radius $r < |z_0|$,*

$$\min_{m \in \mathbb{M}_k} \|m\|_{\bar{D}_r(z_0)} = \frac{r^k}{|z_0|^k},$$

and the minimizing monic polynomial is given by $(z_0 - z)^k/z_0^k$.

Proof. Since $C_r(z_0) \subset \bar{D}_r(z_0)$ and $(z_0 - z)^k/z_0^k$ reaches its extrema on the boundary of $D_r(z_0)$, $(z_0 - z)^k/z_0^k$ is also optimal over the entire closed disc $\bar{D}_r(z_0)$. $\square$

Remark B.4.5. The optimal polynomial only depends upon the center, and not the radius, of the disc.

While the optimal monic polynomial is easily found for the case where the spectrum is only known to be contained within a particular circle in the complex plane which does not contain the origin, there are several other cases of significant interest. In particular, the following well-known theorem [51], due to Flanders and Shortley, but apparently inspired by Tukey and Grosch, can be immediately used to show that monic Chebyshev polynomials are the optimally small monic polynomials over intervals of the real line which are separated from the origin.

Theorem B.11. *Among all polynomials $p \in \mathbb{P}_k$ satisfying $p(\gamma) = 1$, where $|\gamma| > 1$, the unique minimizer of $\|p\|_{[-1,1]}$ is given by $T_k(z)/T_k(\gamma)$, where T_k is the k'th-order Chebyshev polynomial of the first kind. In addition, the maximum absolute value of this polynomial over $[-1,1]$ is $1/|T_k(\gamma)|$.*

Corollary B.12. *Let $[\alpha, \beta]$ be an interval of the real line which does not contain the origin. Then*

$$\min_{m \in \mathbb{M}_k} \|m\|_{[\alpha,\beta]} = \frac{1}{\left|T_k\left(\frac{\alpha+\beta}{\beta-\alpha}\right)\right|},$$

and the unique minimizing monic polynomial is $T_k\left(\frac{\alpha+\beta+2z}{\beta-\alpha}\right)/T_k\left(\frac{\alpha+\beta}{\beta-\alpha}\right)$.

Proof. We may perform a change of variables on each $m \in \mathbb{M}_k$, say

$$m(z) = p\left(\frac{\mu - z}{w}\right),$$

where $\mu = (\alpha+\beta)/2$, and $w = (\beta - \alpha)/2$, where $z \mapsto (\mu - z)/w$ provides a bijection between $[\alpha, \beta]$ and $[-1, 1]$. Since $m(0) = 1$ implies that $p(\mu/w) = 1$, p is a candidate for Theorem B.11 if we choose $\gamma = \mu/w$. $\square$

Corollary B.13. *Let $i[\alpha, \beta]$ be an interval of the imaginary axis which does not contain the origin. Then*

$$\min_{m \in \mathbb{M}_k} \|m\|_{[\alpha,\beta]} = \frac{1}{\left|T_k\left(\frac{\alpha+\beta}{\beta-\alpha}\right)\right|},$$

and the unique minimizing polynomial is $T_k\left(\frac{\alpha+\beta+2iz}{\beta-\alpha}\right)/T_k\left(\frac{\alpha+\beta}{\beta-\alpha}\right)$.

Proof. The proof is essentially identical to that of Corollary B.12, but we instead set $\mu = i(\alpha + \beta)/2$ and $w = i(\beta - \alpha)/2$. $\square$

The last classical case concerns the minimization of a monic polynomial over an ellipse which is separated from the origin. Interestingly enough, though the monic Chebyshev polynomials cannot provide an optimal solution [50], they are *asymptotically optimal* in the sense that, as the polynomial order increases to infinity, the supremum norm of the best candidate converges to the minimal value. In order to avoid diving into the connections between Chebyshev polynomials, the Joukowski map, and Bernstein ellipses [96], we simply define the *near-best* Chebyshev polynomial. Please see [107] or [49] for details.

Definition B.4.4 (Near-best Chebyshev polynomial [107])**.** Given an ellipse ε, with center z_0 and focal distance d, we say that the monic Chebyshev polynomial

$$\tilde{m}(z) = \frac{T_k\left(\frac{z_0 - z}{d}\right)}{T_k\left(\frac{z_0}{d}\right)}$$

is the *near-best* polynomial for the minimization problem

$$\arg\min_{m \in \mathbb{M}_k} \|m\|_{\varepsilon},$$

and it satisfies

$$\|\tilde{m}\|_{\varepsilon} = \frac{T_k\left(\frac{a}{d}\right)}{\left|T_k\left(\frac{z_0}{d}\right)\right|},$$

where a is the (possibly complex) semi-major axis of the ellipse ε.

On the other hand, it is important to recognize that upper bounds based upon the supremum norm of monic polynomials over intervals containing the spectrum can be extremely far from optimal, even for normal matrices.

The reason is that the right-hand side, b, may only have components in the directions of a handful of eigenvectors of A, and a monic polynomial can be constructed with its roots at the appropriate eigenvalues such that $m(A)b$ is exactly zero.

Example B.4.3. Suppose that the right-hand side, b, is an eigenvector of the nonsingular matrix A with eigenvalue λ. Then GMRES applied to the system $Ax = b$ will converge in a single iteration, as $b - A(1/\lambda)b = 0$, and $1/\lambda \in \mathbb{P}_0$.

Proposition B.14. *If a vector b lies in a subspace spanned by a set of eigenvectors of a nonsingular matrix A with r distinct eigenvalues, then GMRES applied to the system $Ax = b$ will converge in at most r iterations.*

Proof. Let $\{\lambda_j\}_{j=1}^r$ be the eigenvalues corresponding to the eigenvectors which comprise b and set

$$m(z) = \prod_{j=1}^r \frac{\lambda_j - z}{\lambda_j} \in \mathbb{M}_r.$$

Then

$$m(A)b = \left(\prod_{j=1}^r \frac{-1}{\lambda_j} \right) (A - \lambda_1 I) \cdots (A - \lambda_r I)b.$$

If we express b as a linear combination of the relevant eigenvectors, say

$$b = \sum_{j=1}^r \gamma_j y_j,$$

then the fact that

$$(A - \lambda_i I) \sum_{j=1}^r \gamma_j y_j = \sum_{j=1}^r (\lambda_j - \lambda_i)\gamma_j y_j,$$

shows that

$$m(A)b = \left(\sum_{j=1}^r \frac{-1}{\lambda_j} \right) \sum_{j=1}^r \prod_{i=1}^r (\lambda_j - \lambda_i)\gamma_j y_j = 0,$$

since each term $\prod_{i=1}^r (\lambda_j - \lambda_i)\gamma_j y_j$ is identically zero. □

B.5 Implementing GMRES

A completely naïve implementation of GMRES would essentially execute the following steps:

1. sequentially construct each vector b, Ab, $A(Ab)$, $A(A(Ab))$, etc., to form a matrix B whose columns span the Krylov subspace $\mathcal{K}_k(A, b)$,
2. apply Gram-Schmidt orthogonalization to the columns of the matrix B in order to find an orthonormal basis for $\mathcal{K}_k(A, b)$, say V,
3. form the Rayleigh quotient $H = V^H(AV)$ in the obvious manner,
4. solve the least-squares problem $y = \arg\min_z \|\bar{H} - \beta e_1\|_2$,
5. set the GMRES estimate to be $x_k = Vy$,
6. compute the residual $r_k = \|b - Ax_k\|_2$, and, finally,
7. stop if the residual is small enough, or, otherwise, repeat the entire process with a larger value for k.

There are unfortunately numerous practical problems with this approach, such as:

- B is likely to be extremely ill-conditioned, perhaps to the point of artificially losing linear independence of its columns due to round-off error,
- the classical Gram-Schmidt algorithm is well-known to be numerically unstable, and should be replaced with the modified Gram-Schmidt algorithm [58],

- if we choose to increase k in order to use a large Krylov subspace then the previous work need not be thrown away,
- for large $N \times N$ sparse systems, the $O(Nk)$ memory requirements for storing the basis for $\mathcal{K}_k(A, b)$ can quickly get out of hand, and so it is important to come up with a so-called *restart* technique, and, finally,
- for problems which do not converge quickly, it is crucial to apply GMRES to a *preconditioned* operator, say $M^{-1}A$, where the *preconditioner* M must be carefully designed in order to balance its construction and application costs with its effectiveness at accelerating GMRES convergence.

In practice, a *restarted* version of GMRES called GMRES(k) is typically used. We may describe its fundamental details in one sentence: after k iterations of GMRES, if the residual is not sufficiently small, the current estimate x_k is used to restart GMRES. We will conclude the appendix by noting that, because k is typically quite small, e.g., 20, the only steps of GMRES(k) which need to be parallelized are the application of the sparse matrix and the preconditioner, and the computation of inner products and norms [16, 17]. Please see Algorithm B.1 for a straight-forward, but reasonably practical, implementation of GMRES(k).

B.6 Summary

A brief introduction to the theory and implementation of the Generalized Minimum Residual (GMRES) method was provided in order to serve as a reference to those without extensive experience implementing iterative

Algorithm B.1: Pseudocode for a simple implementation of preconditioned GMRES(k)

```
Input: Nonsingular matrix, $A$, preconditioner, $M$, right-hand
       side, $b$, and initial guess, $x_0$
Output: Estimate, $x$, for the solution, $A^{-1}b$
1  $x := x_0$
2  $r := b - Ax, \;\; \rho := \|r\|_2$
3  while $\rho/\|b\|_2 <$ tolerance do
       // Run GMRES for $k$ steps with $x$ as initial guess
4      $x_0 := x, \;\; H := \text{zeros}(k, k)$
5      $w := M^{-1}r$
6      $\beta := \|w\|_2, \;\; v_0 := w/\beta$
7      for $j = 0 : k - 1$ do
8          $d := Av_j, \;\; \delta := \|d\|_2$
9          $d := M^{-1}d$
           // Run $j$'th step of Arnoldi
10         for $i = 0 : j$ do
11             $H(i, j) := v_i^H d$
12             $d := d - H(i, j)v_i$
13         end
14         $H(j + 1, j) = \delta := \|d\|_2$
15         if $j + 1 \neq k$ then $v_{j+1} := v_{j+1}/\delta$
           // Solve for residual minimizer
16         $y := \arg\min_z \|H(0{:}j{+}1, 0{:}j)z - \beta \mathbf{e}_0\|_2$
           // Form next iterate and residual
17         $x := x_0 + V_j y$
18         $r := b - Ax, \;\; \rho := \|r\|_2$
19         if $\rho/\|b\|_2 <$ tolerance then break
20     end
21 end
```

methods. The author hopes that this appendix will serve to dispel some of the mystique surrounding GMRES(k), as it is surprisingly straight-forward to implement, both sequentially and in parallel.

Appendix C

Dense linear algebra algorithms

Several algorithms are listed in this appendix for the sole purpose of providing an anchor for the (compressed) multifrontal techniques discussed in Chapters 2 and 4. The following fundamental dense linear algebra operations are covered:

- matrix-matrix multiplication,
- triangular solves,
- Cholesky factorization, and
- triangular inversion.

Matrix-matrix multiplication is perhaps the most important operation to understand, as it is at the heart of most high-performance dense linear algebra operations (by design), as well as the high-performance Kronecker-product application scheme discussed in Chapter 4. Please see [127] and [109] for detailed discussions of the ideas behind the matrix-matrix multiplication algorithms which follow. The triangular solve algorithms form the foundation for the multifrontal triangular solve algorithms of Chapter 2, dense Cholesky factorization is, of course, at the heart of our multifrontal Cholesky factorization scheme, and, finally, dense triangular inversion is used as part of selective inversion.

The notation used within the following algorithms is in the style of FLAME [60] and Elemental [99]. In addition, the operations described here are essentially all implemented as part of the Elemental library, and thorough benchmarks are provided for several of these operations within [99] and [109], as well as thorough discussions of the underlying notation (which is closely followed by the actual implementations). In most cases, the progression follows that of LAPACK [5], ScaLAPACK [31], and PLAPACK [2], which is to build an "unblocked" implementation for small sequential problems, use this at the core of a "blocked" algorithm for larger sequential problems, and then to embed this blocked algorithm within a distributed-memory algorithm.

Algorithm C.1: Distributed dense matrix-matrix multiplication with few right-hand sides

```
Input: Scalar, $\alpha$, $m \times k$ matrix, $A$, $k \times n$ matrix, $B$, scalar, $\beta$,
       and $m \times n$ matrix, $C$. $A$ is in a 2D distribution, but $B$
       and $C$ are in 1D distributions.
Output: $C := \alpha AB + \beta C$
1 $B[M_R,\star] \leftarrow B[V_C,\star]$
2 $\hat{Z}[M_C,\star] := \alpha A[M_C,M_R]B[M_R,\star]$
3 $C[V_C,\star] := \beta C[V_C,\star] + \text{SumScatter}(\hat{Z}[M_C,\star])$
```

Algorithm C.2: Distributed dense adjoint matrix-matrix multiplication with few right-hand sides

```
Input: Scalar, $\alpha$, $k \times m$ matrix, $A$, $k \times n$ matrix, $B$, scalar, $\beta$,
       and $m \times n$ matrix, $C$. $A$ is in a 2D distribution, but $B$
       and $C$ are in 1D distributions.
Output: $C := \alpha A^H B + \beta C$
1 $B[M_C,\star] \leftarrow B[V_C,\star]$
2 $\hat{Z}[M_R,\star] := \alpha A^H[M_R,M_C]B[M_C,\star]$
3 $C[V_C,\star] := \beta C[V_C,\star] + \text{SumScatter}(\hat{Z}[M_R,\star])$
```

Algorithm C.3: Distributed dense matrix-matrix multiplication with many right-hand sides

```
Input: Scalar, $\alpha$, $m \times k$ matrix, $A$, $k \times n$ matrix, $B$, scalar, $\beta$,
       $m \times n$ matrix, $C$, and a blocksize, $n_b$. $A$, $B$, and $C$ are
       each in 2D distributions.
Output: $C := \alpha AB + \beta C$
1 $C := \beta C$
2 $b := \min\{k, n_b\}$
3 $A \rightarrow \begin{pmatrix} A_1 & A_2 \end{pmatrix}$ and $B \rightarrow \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$, where $A_1$ is $m \times b$ and $B_1$ is $b \times n$
  // $\mathbf{C := C + \alpha A_1 B_1}$
4 $A_1[M_C, \star] \leftarrow A_1[M_C, M_R]$
5 $B_1[\star, M_R] \leftarrow B_1[M_C, M_R]$
6 $C[M_C, M_R] := C + \alpha A_1[M_C, \star] B_1[\star, M_R]$
7 if $k > n_b$ then Recurse($\alpha$,$A_2$,$B_2$,1,$C$,$n_b$)
```

Algorithm C.4: Distributed dense adjoint matrix-matrix multiplication with many right-hand sides

```
Input: Scalar, $\alpha$, $k \times m$ matrix, $A$, $k \times n$ matrix, $B$, scalar, $\beta$,
       $m \times n$ matrix, $C$, and a blocksize, $n_b$. $A$, $B$, and $C$ are
       each in 2D distributions.
Output: $C := \alpha A^H B + \beta C$
1 $C := \beta C$
2 $b := \min\{k, n_b\}$
3 $A \rightarrow \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ and $B \rightarrow \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}$, where $A_1$ is $b \times m$ and $B_1$ is $b \times n$
  // $\mathbf{C := C + \alpha A_1^H B_1}$
4 $A_1[\star, M_C] \leftarrow A_1[M_C, M_R]$
5 $B_1[\star, M_R] \leftarrow B_1[M_C, M_R]$
6 $C[M_C, M_R] := C + \alpha A_1^H[M_C, \star] B_1[\star, M_R]$
7 if $k > n_b$ then Recurse($\alpha$,$A_2$,$B_2$,1,$C$,$n_b$)
```

Algorithm C.5: Unblocked dense triangular solve

```
    Input: $n \times n$ lower-triangular matrix $L$ and an $n \times k$ matrix $X$
    Output: $X := L^{-1}X$
1   $L \rightarrow \begin{pmatrix} \lambda_{1,1} & 0 \\ \ell_{2,1} & L_{2,2} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} x_1 \\ X_2 \end{pmatrix}$, where $\lambda_{1,1}$ is $1 \times 1$ and $x_1$ is $1 \times k$
2   $x_1 := x_1/\lambda_{1,1}$
3   if $n > 1$ then
4       $X_2 := X_2 - \ell_{2,1}x_1$
5       Recurse($L_{2,2}$,$X_2$)
```

Algorithm C.6: Dense triangular solve

```
    Input: $n \times n$ lower-triangular matrix, $L$, an $n \times k$ matrix, $X$, and
           a blocksize, $n_b$
    Output: $X := L^{-1}X$
1   $b := \min\{n, n_b\}$
2   $L \rightarrow \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, where $L_{1,1}$ is $b \times b$ and $X_1$ is $b \times k$
3   $X_1 := L_{1,1}^{-1}X_1$ via Algorithm C.5
4   if $n > n_b$ then
5       $X_2 := X_2 - L_{2,1}X_1$
6       Recurse($L_{2,2}$,$X_2$)
```

Algorithm C.7: Dense triangular solve with few right-hand sides (1D distribution)

```
    Input: $n \times n$ lower-triangular matrix, $L$, an $n \times k$ matrix, $X$, and
           a blocksize, $n_b$. $L$ and $X$ are both in 1D distributions.
    Output: $X := L^{-1}X$
1   $b := \min\{n, n_b\}$
2   $L \rightarrow \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, where $L_{1,1}$ is $b \times b$ and $X_1$ is $b \times k$
    // $\mathbf{X_1 := L_{1,1}^{-1} X_1}$
3   $X_1[*,*] \leftarrow X_1[V_C,*]$
4   $L_{1,1}[*,*] \leftarrow L_{1,1}[V_C,*]$
5   $X_1[*,*] := L_{1,1}^{-1}[*,*]X_1[*,*]$ via Algorithm C.6
6   $X_1[V_C,*] \leftarrow X_1[*,*]$
7   if $n > n_b$ then
        // $\mathbf{X_2 := X_2 - L_{2,1} X_1}$
8       $X_2[V_C,*] := X_2[V_C,*] - L_{2,1}[V_C,*]X_1[*,*]$
9       Recurse($L_{2,2}$,$X_2$)
```

Algorithm C.8: Dense triangular solve with few right-hand sides (2D distribution)

```
    Input: $n \times n$ lower-triangular matrix, $L$, an $n \times k$ matrix, $X$, and
           a blocksize, $n_b$. $L$ is in a 2D distribution, but $X$ is in a 1D
           distribution. An $n \times k$ matrix, $\hat{Z}$, initially filled with zeros
           and in an $[M_C, *]$ distribution, should also be passed in.
    Output: $X := L^{-1}X$
 1  $b := \min\{n, n_b\}$
 2  $L \rightarrow \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}$, $X \rightarrow \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, and $\hat{Z} \rightarrow \begin{pmatrix} \hat{Z}_1 \\ \hat{Z}_2 \end{pmatrix}$, where $L_{1,1}$ is
    $b \times b$ and $X_1$ and $\hat{Z}_1$ are $b \times k$
    // Combine and apply all previous partial updates to $\mathbf{X_1}$
 3  $X_1[V_C, *] := X_1[V_C, *] + \text{SumScatter}(\hat{Z}_1[M_C, *])$
    // $\mathbf{X_1 := L_{1,1}^{-1} X_1}$
 4  $X_1[*, *] \leftarrow X_1[V_C, *]$
 5  $L_{1,1}[*, *] \leftarrow L_{1,1}[M_C, M_R]$
 6  $X_1[*, *] := L_{1,1}^{-1}[*, *] X_1[*, *]$ via Algorithm C.6
 7  $X_1[V_C, *] \leftarrow X_1[*, *]$
 8  if $n > n_b$ then
        // Add portion of $-\mathbf{L_{2,1} X_1}$ onto past partial updates
 9      $X_1[M_R, *] \leftarrow X_1[*, *]$
10      $\hat{Z}_2[M_C, *] := \hat{Z}_2[M_C, *] - L_{2,1}[M_C, M_R] X_1[M_R, *]$
11      Recurse($L_{2,2}$,$X_2$,$\hat{Z}_2$)
```

Algorithm C.9: Distributed dense triangular solve with many right-hand sides

```
Input: $n \times n$ lower-triangular matrix, $L$, an $n \times k$ matrix, $X$, and
       a blocksize, $n_b$. $L$ and $X$ are both in 2D distributions.
Output: $X := L^{-1}X$
1  $b := \min\{n, n_b\}$
2  $L \rightarrow \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$, where $L_{1,1}$ is $b \times b$ and $X_1$ is
   $b \times k$
   // $\mathbf{X_1 := L_{1,1}^{-1} X_1}$
3  $X_1[*, V_R] \leftarrow X_1[M_C, M_R]$
4  $L_{1,1}[*, *] \leftarrow L_{1,1}[M_C, M_R]$
5  $X_1[*, V_R] := L_{1,1}^{-1}[*, *] X_1[*, V_R]$ via Algorithm C.6
6  if $n > n_b$ then
       // $\mathbf{X_2 := X_2 - L_{2,1} X_1}$ and delayed write of $\mathbf{X_1}$
7      $L_{2,1}[M_C, *] \leftarrow L_{2,1}[M_C, M_R]$
8      $X_1[*, M_R] \leftarrow X_1[*, V_R]$
9      $X_2[M_C, M_R] := X_2[M_C, M_R] - L_{2,1}[M_C, *] X_1[*, M_R]$
10     $X_1[M_C, M_R] \leftarrow X_1[*, M_R]$
11     Recurse($L_{2,2}$,$X_2$)
```

Algorithm C.10: Unblocked dense adjoint triangular solve

```
Input: $n \times n$ lower-triangular matrix $L$ and an $n \times k$ matrix $X$
Output: $X := L^{-H}X$
1  $L \rightarrow \begin{pmatrix} L_{0,0} & 0 \\ \ell_{1,0} & \lambda_{1,1} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_0 \\ x_1 \end{pmatrix}$, where $\lambda_{1,1}$ is $1 \times 1$ and $x_1$ is
   $1 \times k$
2  $x_1 := x_1 / \overline{\lambda_{1,1}}$
3  if $n > 1$ then
4      $X_0 := X_0 - \ell_{1,0}^H x_1$
5      Recurse($L_{0,0}$,$X_0$)
```

Algorithm C.11: Dense adjoint triangular solve

```
Input: $n \times n$ lower-triangular matrix $L$, an $n \times k$ matrix $X$, and a
       blocksize, $n_b$
Output: $X := L^{-H} X$
1 $b := \min\{n, n_b\}$
2 $L \rightarrow \begin{pmatrix} L_{0,0} & 0 \\ L_{1,0} & L_{1,1} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_0 \\ X_1 \end{pmatrix}$, where $L_{1,1}$ is $b \times b$ and $X_1$ is
  $b \times k$
3 $X_1 := L_{1,1}^{-H} X_1$ via Algorithm C.10
4 if $n > n_b$ then
5     $X_0 := X_0 - L_{1,0}^H X_1$
6     Recurse($L_{0,0}$,$X_0$)
```

Algorithm C.12: Dense adjoint triangular solve with few right-hand sides (2D distribution)

```
Input: $n \times n$ lower-triangular matrix $L$, an $n \times k$ matrix $X$, and a
       blocksize, $n_b$. $L$ is in a 2D distribution and $X$ is in a 1D
       distribution. An $n \times k$ matrix, $\hat{Z}$, initially filled with zeros
       and in an $[M_C, *]$ distribution, should also be passed in.
Output: $X := L^{-H} X$
1  $b := \min\{n, n_b\}$
2  $L \rightarrow \begin{pmatrix} L_{0,0} & 0 \\ L_{1,0} & L_{1,1} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_0 \\ X_1 \end{pmatrix}$, where $L_{1,1}$ is $b \times b$ and $X_1$ is
   $b \times k$
   // Combine and apply all previous partial updates to $\mathbf{X_1}$
3  $X_1[V_C, *] := X_1[V_C, *] + \text{SumScatter}(\hat{Z}_1[M_R, *])$
   // $\mathbf{X_1 := L_{1,1}^{-H} X_1}$
4  $X_1[*, *] \leftarrow X_1[V_C, *]$
5  $L_{1,1}[*, *] \leftarrow L_{1,1}[M_C, M_R]$
6  $X_1[*, *] := L_{1,1}^{-H}[*, *] X_1[*, *]$ via Algorithm C.10
7  $X_1[V_C, *] \leftarrow X_1[*, *]$
8  if $n > n_b$ then
       // Add portion of $-\mathbf{L_{1,0}^H X_1}$ onto past partial updates
9      $\hat{Z}_0[M_R, *] := \hat{Z}_0[M_R, *] - L_{1,0}^H[M_R, M_C] X_1[M_C, *]$
10     Recurse($L_{0,0}$,$X_0$,$\hat{Z}_0$)
```

Algorithm C.13: Distributed dense adjoint triangular solve with many right-hand sides

```
    Input: $n \times n$ lower-triangular matrix $L$, an $n \times k$ matrix $X$, and a
           blocksize, $n_b$. $L$ and $X$ are both in a 2D distribution.
    Output: $X := L^{-H} X$
 1  $b := \min\{n, n_b\}$
 2  $L \rightarrow \begin{pmatrix} L_{0,0} & 0 \\ L_{1,0} & L_{1,1} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_0 \\ X_1 \end{pmatrix}$, where $L_{1,1}$ is $b \times b$ and $X_1$ is
    $b \times k$
    // $\mathbf{X_1 := L_{1,1}^{-H} X_1}$
 3  $X_1[*, V_R] \leftarrow X_1[M_C, M_R]$
 4  $L_{1,1}[*, *] \leftarrow L_{1,1}[M_C, M_R]$
 5  $X_1[*, V_R] := L_{1,1}^{-H}[*, *] X_1[*, V_R]$ via Algorithm C.10
 6  if $n > n_b$ then
        // $\mathbf{X_0 := X_0 - L_{1,0}^H X_1}$ and delayed write of $\mathbf{X_1}$
 7      $L_{1,0}[*, M_C] \leftarrow L_{1,0}[M_C, M_R]$
 8      $X_1[*, M_R] \leftarrow X_1[*, V_R]$
 9      $X_0[M_C, M_R] := X_0[M_C, M_R] - L_{1,0}^H[M_C, *] X_1[*, M_R]$
10      $X_1[M_C, M_R] \leftarrow X_1[*, M_R]$
11      Recurse($L_{0,0}$,$X_0$)
```

Algorithm C.14: Dense adjoint triangular solve (lazy variant)

```
    Input: $n \times n$ lower-triangular matrix $L$, an $n \times k$ matrix $X$, and a
           blocksize, $n_b$
    Output: $X := L^{-H} X$
  1 $L \rightarrow \begin{pmatrix} L_{TL} & 0 \\ L_{BL} & L_{BR} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_T \\ X_B \end{pmatrix}$, where $L_{BR}$ is $0 \times 0$ and $X_B$
    is $0 \times k$
  2 while $size(L_{BR}) \neq size(L)$ do
  3     $b := \min\{\text{height}(L_{TL}), n_b\}$
  4     $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ \hline L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$ and
        $\left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \rightarrow \left(\begin{array}{c} X_0 \\ X_1 \\ \hline X_2 \end{array}\right)$, where $L_{1,1}$ is $b \times b$ and $X_1$ is $b \times k$
  5     $X_1 := X_1 - L_{2,1}^H X_2$
  6     $X_1 := L_{1,1}^{-H} X_1$ via Algorithm C.10
  7     $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|cc} L_{0,0} & 0 & 0 \\ \hline L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$, and
        $\left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \leftarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ X_2 \end{array}\right)$
  8 end
```

Algorithm C.15: Dense adjoint triangular solve with few right-hand sides (lazy variant, 1D distribution)

```
   Input: $n \times n$ lower-triangular matrix $L$, an $n \times k$ matrix $X$, and a
          blocksize, $n_b$
   Output: $X := L^{-H} X$
1  $L \rightarrow \begin{pmatrix} L_{TL} & 0 \\ L_{BL} & L_{BR} \end{pmatrix}$ and $X \rightarrow \begin{pmatrix} X_T \\ X_B \end{pmatrix}$, where $L_{BR}$ is $0 \times 0$ and $X_B$
   is $0 \times k$
2  while $size(L_{BR}) \neq size(L)$ do
3      $b := \min\{\text{height}(L_{TL}), n_b\}$
4      $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ \hline L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$ and
       $\left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \rightarrow \left(\begin{array}{c} X_0 \\ X_1 \\ \hline X_2 \end{array}\right)$, where $L_{1,1}$ is $b \times b$ and $X_1$ is $b \times k$
       // $\mathbf{X_1 := X_1 - L_{2,1}^H X_2}$
5      $\hat{Z}_1[*,*] := -L_{2,1}^H[*,V_C] X_2[V_C,*]$
6      $X_1[V_C,*] := X_1[V_C,*] + \text{SumScatter}(\hat{Z}_1[*,*])$
       // $\mathbf{X_1 := L_{1,1}^{-H} X_1}$
7      $X_1[*,*] \leftarrow X_1[V_C,*]$
8      $L_{1,1}[*,*] \leftarrow L_{1,1}[V_C,*]$
9      $X_1[*,*] := L_{1,1}^{-H}[*,*] X_1[*,*]$ via Algorithm C.10
10     $X_1[V_C,*] \leftarrow X_1[*,*]$
11     $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|cc} L_{0,0} & 0 & 0 \\ \hline L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$, and
       $\left(\begin{array}{c} X_T \\ \hline X_B \end{array}\right) \leftarrow \left(\begin{array}{c} X_0 \\ \hline X_1 \\ X_2 \end{array}\right)$
12 end
```

Algorithm C.16: Unblocked dense right-looking Cholesky

```
    Input: $n \times n$ HPD matrix, $A$
    Output: $A$ is overwritten with its lower Cholesky factor, $L$
1   $A \rightarrow \begin{pmatrix} \alpha_{1,1} & \star \\ a_{2,1} & A_{2,2} \end{pmatrix}$, where $\alpha_{1,1}$ is a scalar
2   $\lambda_{1,1} := \sqrt{\alpha_{1,1}}$
3   if $n > 1$ then
4       $\ell_{2,1} := a_{2,1}/\lambda_{1,1}$
5       $A_{2,2} := A_{2,2} - \ell_{2,1}\ell_{2,1}^H$
6       Recurse($A_{2,2}$)
```

Algorithm C.17: Dense right-looking Cholesky

```
    Input: $n \times n$ HPD matrix, $A$, and blocksize, $n_b$
    Output: $A$ is overwritten with its lower Cholesky factor, $L$
1   $b := \min\{n, n_b\}$
2   $A \rightarrow \begin{pmatrix} A_{1,1} & \star \\ A_{2,1} & A_{2,2} \end{pmatrix}$, where $A_{1,1}$ is $b \times b$
3   $L_{1,1} :=$ Cholesky($A_{1,1}$)
4   via Algorithm C.16
5   if $n > n_b$ then
6       $L_{2,1} := A_{2,1}L_{1,1}^{-H}$
7       $A_{2,2} := A_{2,2} - L_{2,1}L_{2,1}^H$
8       Recurse($A_{2,2}$,$n_b$)
```

Algorithm C.18: Distributed dense right-looking Cholesky

```
    Input: $n \times n$ HPD matrix, $A$, in a 2D distribution, and a
           blocksize $n_b$
    Output: $A$ is overwritten with its lower Cholesky factor, $L$
 1  $b := \min\{n, n_b\}$
 2  $A \rightarrow \begin{pmatrix} A_{1,1} & \star \\ A_{2,1} & A_{2,2} \end{pmatrix}$, where $A_{1,1}$ is $b \times b$
    // $\mathbf{L_{1,1}} :=$ Cholesky($\mathbf{A_{1,1}}$)
 3  $A_{1,1}[*,*] \leftarrow A_{1,1}[M_C, M_R]$
 4  $L_{1,1}[*,*] :=$ Cholesky($A_{1,1}[*,*]$) via Algorithm C.17
 5  $L_{1,1}[M_C, M_R] \leftarrow L_{1,1}[*,*]$
 6  if $n > n_b$ then
        // $\mathbf{L_{2,1} := A_{2,1} L_{1,1}^{-H}}$
 7      $A_{2,1}[V_C, *] \leftarrow A_{2,1}[M_C, M_R]$
 8      $L_{2,1}[V_C, *] := A_{2,1}[V_C, *] L_{1,1}^{-H}[*,*]$
        // $\mathbf{A_{2,2} := A_{2,2} - L_{2,1} L_{2,1}^H}$ and delayed write of $\mathbf{L_{2,1}}$
 9      $L_{2,1}[M_C, *] \leftarrow L_{2,1}[V_C, *]$
10      $L_{2,1}[M_R, *] \leftarrow L_{2,1}[V_C, *]$
11      $A_{2,2}[M_C, M_R] := A_{2,2}[M_C, M_R] - L_{2,1}[M_C, *] L_{2,1}^H[*, M_R]$
12      $L_{2,1}[M_C, M_R] \leftarrow L_{2,1}[M_C, *]$
13      Recurse($A_{2,2}$,$n_b$)
```

Algorithm C.19: Unblocked dense triangular inversion

```
    Input: $n \times n$ lower-triangular matrix, $L$
    Output: $L$ is overwritten with its inverse
1   $L \rightarrow \begin{pmatrix} L_{TL} & 0 \\ L_{BL} & L_{BR} \end{pmatrix}$, where $L_{TL}$ is $0 \times 0$
2   while $size(L_{TL}) \neq size(L)$ do
3       $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} L_{0,0} & 0 & 0 \\ \hline \ell_{1,0} & \lambda_{1,1} & 0 \\ L_{2,0} & \ell_{2,1} & L_{2,2} \end{array}\right)$, where $\lambda_{1,1}$ is a scalar
4       $\ell_{1,0} := -\ell_{1,0}/\lambda_{1,1}$
5       $L_{2,0} := L_{2,0} + \ell_{2,1}\ell_{1,0}$
6       $\ell_{2,1} := \ell_{2,1}/\lambda_{1,1}$
7       $\lambda_{1,1} := 1/\lambda_{1,1}$
8       $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} L_{0,0} & 0 & 0 \\ \ell_{1,0} & \lambda_{1,1} & 0 \\ \hline L_{2,0} & \ell_{2,1} & L_{2,2} \end{array}\right)$
9   end
```

Algorithm C.20: Dense triangular inversion

```
    Input: $n \times n$ lower-triangular matrix, $L$, and blocksize, $n_b$
    Output: $L$ is overwritten with its inverse
 1  $L \rightarrow \begin{pmatrix} L_{TL} & 0 \\ L_{BL} & L_{BR} \end{pmatrix}$, where $L_{TL}$ is $0 \times 0$
 2  while $size(L_{TL}) \neq size(L)$ do
 3      $b := \min\{\text{height}(L_{BR}), n_b\}$
 4      $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} L_{0,0} & 0 & 0 \\ \hline L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$, where $L_{1,1}$ is $b \times b$
 5      $L_{1,0} := -L_{1,1}^{-1} L_{1,0}$
 6      $L_{2,0} := L_{2,0} + L_{2,1} L_{1,0}$
 7      $L_{2,1} := L_{2,1} L_{1,1}^{-1}$
 8      $L_{1,1} := L_{1,1}^{-1}$ via Algorithm C.19
 9      $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ \hline L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$
10  end
```

Algorithm C.21: Distributed dense triangular inversion

```
Input: $n \times n$ lower-triangular matrix, $L$, in a 2D distribution, and
       a blocksize, $n_b$
Output: $L$ is overwritten with its inverse
1  $L \rightarrow \begin{pmatrix} L_{TL} & 0 \\ L_{BL} & L_{BR} \end{pmatrix}$, where $L_{TL}$ is $0 \times 0$
2  while $\mathit{size}(L_{TL}) \neq \mathit{size}(L)$ do
3      $b := \min\{\text{height}(L_{BR}), n_b\}$
4      $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|cc} L_{0,0} & 0 & 0 \\ \hline L_{1,0} & L_{1,1} & 0 \\ L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$, where $L_{1,1}$ is $b \times b$
       // $\mathbf{L_{1,0} := -L_{1,1}^{-1} L_{1,0}}$
5      $L_{1,0}[*, V_R] \leftarrow L_{1,0}[M_C, M_R]$
6      $L_{1,1}[*, *] \leftarrow L_{1,1}[M_C, M_R]$
7      $L_{1,0}[*, V_R] := -L_{1,1}^{-1}[*, *] L_{1,0}[*, V_R]$
       // $\mathbf{L_{2,0} := L_{2,0} + L_{2,1} L_{1,0}}$ and delayed write of $\mathbf{L_{1,0}}$
8      $L_{2,1}[M_C, *] \leftarrow L_{2,1}[M_C, M_R]$
9      $L_{1,0}[*, M_R] \leftarrow L_{1,0}[*, V_R]$
10     $L_{2,0}[M_C, M_R] := L_{2,0}[M_C, M_R] + L_{2,1}[M_C, *] L_{1,0}[*, M_R]$
11     $L_{1,0}[M_C, M_R] \leftarrow L_{1,0}[*, M_R]$
       // $\mathbf{L_{2,1} := L_{2,1} L_{1,1}^{-1}}$
12     $L_{2,1}[V_C, *] \leftarrow L_{2,1}[M_C, *]$
13     $L_{2,1}[V_C, *] := L_{2,1}[V_C, *] L_{1,1}^{-1}[*, *]$
14     $L_{2,1}[M_C, M_R] \leftarrow L_{2,1}[V_C, *]$
       // $\mathbf{L_{1,1} := L_{1,1}^{-1}}$
15     $L_{1,1}[*, *] := L_{1,1}^{-1}[*, *]$ via Algorithm C.20
16     $L_{1,1}[M_C, M_R] \leftarrow L_{1,1}[*, *]$
17     $\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{cc|c} L_{0,0} & 0 & 0 \\ L_{1,0} & L_{1,1} & 0 \\ \hline L_{2,0} & L_{2,1} & L_{2,2} \end{array}\right)$
18 end
```

Bibliography

[1] MUltifrontal Massively Parallel Solver Users' guide: version 4.10.0. `http://graal.ens-lyon.fr/MUMPS/doc/userguide_4.10.0.pdf`, May 10, 2011.

[2] Philip Alpatov, Greg Baker, Carter Edwards, John A. Gunnels, Greg Morrow, James Overfelt, Robert A. van de Geijn, and Yuan-Jye J. Wu. PLAPACK: Parallel Linear Algebra Package – design overview. In *Proceedings of Supercomputing*, 1997.

[3] Patrick R. Amestoy, Iain S. Duff, Jacko Koster, and Jean-Yves L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[4] Fred Aminzadeh, Jean Brac, and Tim Kunz. 3-D Salt and Overthrust models. In *SEG/EAGE 3-D Modeling Series 1*, Tulsa, OK, 1997. Society for Exploration Geophysicists.

[5] Edward J. Anderson, Zhaojun Bai, Christian H. Bischof, James W. Demmel, Jack J. Dongarra, J. Du Croz, Anne Greenbaum, Sven Hammarling, A. McKenney, and Danny C. Sorensen. LAPACK: A portable linear algebra library for high-performance computers. In *Proceedings Supercomputing '90*, pages 2–11. IEEE Computer Society Press, Los Alamitos, California, 1990.

[6] Nesmith C. Ankeny and Theodore J. Rivlin. On a theorem of S. Bernstein. *Pacific Journal of Mathematics*, 5(2):849–852, 1955.

[7] Jean-Robert Argand. Essai sur une manière de représenter des quantités imaginaires dans les constructions géométriques. *les Annales de mathématiques pures et appliquées*, 4:133–147, 1813.

[8] Walter E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9:17–29, 1951.

[9] Cleve Ashcraft. A taxonomy of distributed dense LU factorization methods. Boeing Computer Services Technical Report ECA-TR-161, March 1991.

[10] Cleve Ashcraft, Stanley C. Eisenstat, and Joseph W.-H. Liu. A fan-in algorithm for distributed sparse numerical factorization. *SIAM Journal on Scientific and Statistical Computing*, 11(3):593–599, 1990.

[11] Cleve Ashcraft and Roger G. Grimes. The influence of relaxed supernode partitions on the multifrontal method. *ACM Transactions on Mathematical Software*, 15(4):291–309, 1989.

[12] Cleve Ashcraft and Roger G. Grimes. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, page 10, March 22-27, 1999.

[13] Cleve Ashcraft, Roger G. Grimes, John G. Lewis, Barry W. Peyton, and Horst D. Simon. Progress in sparse matrix methods for large sparse linear systems on vector supercomputers. *International Journal of Supercomputer Applications*, 1:10–30, 1987.

[14] Sheldon Axler. Down with determinants! *American Mathematical Monthly*, 102(2):139–154.

[15] Ivo M. Babuška and S. A. Sauter. Is the pollution effect of the FEM avoidable for the Helmholtz equation considering high wave numbers? *SIAM Review*, 42(3):451–484, 2000.

[16] Satish Balay, Jed Brown, , Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Uusers manual. Technical Report ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012.

[17] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2012.

[18] Alvin Bayliss, Charles I. Goldstein, and E. Turkel. An iterative method for the Helmholtz equation. *Journal of Computational Physics*, 49:443–457, 1983.

[19] Jean-Pierre Bérenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114:185–200, 1994.

[20] S. Bernstein. Sur l'ordre de la meilleure approximation des fonctions continues par des polynomes de degré donné. *Mém. Acad. Roy. Belg.*, 1912.

[21] Gregory Beylkin, Christopher Kurcz, and Lucas Monzón. Fast algorithms for Helmholtz Green's functions. *Proceedings of the Royal Society A: Mathematical, Physical, and Engineering Sciences*, 464(2100):3301–3326, 2008.

[22] Matthias Bollhöfer and Yousef Saad. Multilevel preconditioners constructed from inverse-based ILUs. *SIAM Journal on Scientific Computing*, 27:1627–1650, 2006.

[23] Matthis Bollhöfer, Marcus Grote, and Olaf Schenk. Algebraic multilevel preconditioner for the Helmholtz equation in heterogeneous media. *SIAM Journal on Scientific Computing*, 31:3781–3805, 2009.

[24] James H. Bramble and Joseph E. Pasciak. A note on the existence and uniqueness of solutions of frequency domain elastic wave problems: *a priori* estimates in H^1. *Mathematical Analysis and Applications*, 345:396–404, 2008.

[25] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[26] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial*. SIAM, 2000.

[27] Henri Calandra, Serge Gratton, Xavier Pinel, and Xavier Vasseur. An improved two-grid preconditioner for the solution of three-dimensional Helmholtz problems in heterogeneous media. Technical Report TR/PA/12/2, CERFACS, Toulouse, France, 2012.

[28] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[29] Weng Cho Chew and William H. Weedon. A 3D perfectly matched medium from modified Maxwell's equations with stretched coordinates. *Microwave and Optical Technology Letters*, 7(13):599–604, 1994.

[30] Lindsay N. Childs. *A concrete introduction to higher algebra.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, NY, 2008.

[31] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 120–127. IEEE Comput. Soc. Press, 1992.

[32] Robert Dautray and Jacques-Louis Lions. *Spectral theory and applications*, volume 3 of *Mathematical analysis and numerical methods for science and technology.* Springer-Verlag, New York, NY, 1985.

[33] Tim Davis. Summary of available software for sparse direct methods. `http://www.cise.ufl.edu/research/sparse/codes/`, April 2012.

[34] James W. Demmel. *Applied numerical linear algebra.* SIAM, Philadelphia, PA, 1997.

[35] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.

[36] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S. Duff. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[37] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, 1988.

[38] Jack J. Dongarra, Robert A. van de Geijn, and David W. Walker. A look at scalable dense linear algebra libraries. In *Proceedings of Scalable High Performance Computer Conference*, pages 372–379, Williamsburg, VA, 1992.

[39] Jack J. Dongarra and David W. Walker. MPI: a standard message passing interface. *Supercomputer*, 12(1):56–68, 1996.

[40] Alex Druinsky and Sivan Toledo. How accurate is inv$(a) * b$? *ArXiv e-prints*, January 2012.

[41] Iain S. Duff and John K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[42] Björn Engquist and Andrew Majda. Absorbing Boundary Conditions for the numerical simulation of waves. *Mathematics of Computation*, 31:629–651, 1977.

[43] Björn Engquist and Lexing Ying. Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation. *Communications on Pure and Applied Mathematics*, 64:697–735, 2011.

[44] Björn Engquist and Lexing Ying. Sweeping preconditioner for the Helmholtz equation: moving perfectly matched layers. *SIAM Journal on Multiscale Modeling and Simulation*, 9:686–710, 2011.

[45] Yogi A. Erlangga. Advances in iterative methods and preconditioners for the Helmholtz equation. *Archives of Computational Methods in Engineering*, 15:37–66, 2008.

[46] Yogi A. Erlangga, Cornelis Vuik, and Cornelis W. Oosterlee. On a class of preconditioners for solving the Helmholtz equation. *Applied Numerical Mathematics*, 50:409–425, 2004.

[47] Oliver G. Ernst and Martin J. Gander. Why it is difficult to solve Helmholtz problems with classical iterative methods. In I. Graham, T. Hou, O. Lakkis, and R. Scheichl, editors, *Numerical Analysis of Multiscale Problems*, pages 325–363, New York, NY, 2011. Springer-Verlag.

[48] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of 19th IEEE Design Automation Conference*, pages 175–181. IEEE, 1982.

[49] Bernd Fischer and Roland Freund. On the constrained Chebyshev approximation problem on ellipses. *Journal of Approximation Theory*, 62(3):297–315, 1990.

[50] Bernd Fischer and Roland Fruend. Chebyshev polynomials are not always optimal. *Journal of Approximation Theory*, 65:261–272, 1991.

[51] Donald A. Flanders and George Shortley. Numerical determination of fundamental modes. *Journal of Applied Physics*, 21(1326):1326–1332, 1950.

[52] Martin J. Gander and Frédéric Nataf. AILU for Helmholtz problems: a new preconditioner based on the analytic parabolic factorization. *Journal of Computational Acoustics*, 9:1499–1506, 2001.

[53] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63, New York, NY, 1974. ACM.

[54] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[55] Alan George and Joseph W. H. Liu. An optimal algorithm for symbolic factorization of symmetric matrices. *SIAM Journal on Computing*, 9:583–593, 1980.

[56] Alan George, Joseph W. H. Liu, and Esmond G. Ng. Communication reduction in parallel sparse Cholesky factorization on a hypercube. In Michael T. Heath, editor, *Hypercube Multiprocessors*, pages 576–586, Philadelphia, PA, 1987. SIAM.

[57] John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms*, 5(3):391–407, 1984.

[58] Gene H. Golub and Charles F. van Loan. *Matrix computations.* Johns Hopkins University Press, Baltimore, MD, 1996.

[59] Lars Grasedyck and Wolfgang Hackbusch. Construction and arithmetics of $\mathcal{H}$-matrices. *Computing*, 70(4):295–334, 2003.

[60] John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Transactions on Mathematical Software*, 27(4):422–455, 2001.

[61] Anshul Gupta. Analysis and design of scalable parallel algorithms for scientific computing. Ph.D. Dissertation, University of Minnesota, Minneapolis, Minnesota, 1995.

[62] Anshul Gupta and Mahesh V. Joshi. WSMP: A high-performance shared- and distributed-memory parallel sparse linear equations solver. IBM Research Report RC 22038 (98932), 2001.

[63] Anshul Gupta, George Karypis, and Vipin Kumar. A highly scalable parallel algorithm for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, 1997.

[64] Anshul Gupta, Seid Koric, and Thomas George. Sparse matrix factorization on massively parallel computers. In *Proceedings of Conference on High Performance Computing Networking, Storage, and Analysis*, number 1, New York, NY, 2009. ACM.

[65] Anshul Gupta and Vipin Kumar. Parallel algorithms for forward elimination and backward substitution in direct solution of sparse linear systems. In *Proceedings of Supercomputing*, San Diego, CA, 1995. ACM.

[66] Wolfgang Hackbusch. A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2):89–108, 1999.

[67] Paul R. Halmos. *Finite-dimensional vector spaces.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, NY, 1987.

[68] Michael T. Heath and Padma Raghavan. LAPACK Working Note 62: distributed solution of sparse linear systems. Technical Report UT-CS-93-201, University of Tennessee, Knoxville, TN, 1993.

[69] Bruce A. Hendrickson and Robert W. Leland. The Chaco Users Guide: Version 2.0. Sandia Technical Report SAND94–2692, 1994.

[70] Bruce A. Hendrickson and Robert W. Leland. A multi-level algorithm for partitioning graphs. In *Proceedings of Supercomputing*, San Diego, CA, 1995. ACM.

[71] Bruce A. Hendrickson and David E. Womble. The torus-wrap mapping for dense matrix calculations on massively parallel computers. *SIAM Journal on Scientific and Statistical Computing*, 15:1201–1226, 1994.

[72] Pascal Henon, Pierre Ramet, and Jean Roman. PaStiX: A parallel sparse direct solver based on static scheduling for mixed 1D/2D block distributions. In *Proceedings of Irregular'2000 workshop of IPDPS*, volume 1800 of *Lecture Notes in Computer Science*, pages 519–525, Cancun, Mexico, 2000. Springer Verlag.

[73] Nicholas J. Higham. *Accuracy and stability of numerical algorithms.* SIAM, Philadelphia, PA, 1996.

[74] Roger A. Horn and Charles R. Johnson. *Matrix analysis.* Cambridge University Press, Cambridge, NY, 1985.

[75] Bruce M. Irons. A frontal solution program for finite element analysis. *International Journal for Numerical Methods in Engineering*, 2:5–32, 1970.

[76] Dror Irony and Sivan Toledo. Trading replication for communication in parallel distributed-memory dense solvers. *Parallel Processing Letters*, pages 79–94, 2002.

[77] James Jeans. *The mathematical theory of electricity and magnetism.* Cambridge University Press, 1908.

[78] Steven G. Johnson. Notes on Perfectly Matched Layers (PMLs). Technical report, Massachusetts Institute of Technology, Cambridge, MA, 2010.

[79] Mahesh V. Joshi, Anshul Gupta, George Karypis, and Vipin Kumar. A high performance two dimensional scalable parallel algorithm for solving sparse triangular systems. In *Proceedings of the Fourth International Conference on High-Performance Computing*, HIPC '97, pages 137–, Washington, DC, 1997. IEEE Computer Society.

[80] Mahesh V. Joshi, George Karypis, Vipin Kumar, Anshul Gupta, and Fred Gustavson. PSPASES: Building a high performance scalable parallel direct solver for sparse linear systems. In Tianruo Yang, editor, *Parallel Numerical Computations with Applications*, pages 3–18, Norwell, MA, 1999. IEEE.

[81] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[82] George Karypis and Vipin Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Parallel and Distributed Computing*, 48:71–85, 1998.

[83] Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

[84] Erwin Kreyszig. *Introductory functional analysis with applications.* Wiley, 1978.

[85] Cornelius 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*, 45:255–282, 1950.

[86] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, 1979.

[87] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, 2005.

[88] Xiaoye S. Li and James W. Demmel. A scalable sparse direct solver using static pivoting. In *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, page 10, March 22-24, 1999.

[89] Xiaoye S. Li, James W. Demmel, John R. Gilbert, Laura Grigori, Meiyue Shao, and Ichitaro Yamazaki. SuperLU Users' Guide. Technical Report LBNL-44289, October 2011.

[90] Robert J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2), 1979.

[91] Joseph W. H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Review*, 34(1):82–109, 1992.

[92] Per-Gunnar Martinsson and Vladimir Rokhlin. A fast direct solver for scattering problems involving elongated structures. *Journal of Computational Physics*, 221(1):288–302, 2007.

[93] Dianne P. O'Leary and G. W. Stewart. Data-flow algorithms for parallel matrix computations. *Communications of the ACM*, 28:840–853, 1985.

[94] Chris C. Paige and Michael A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal of Numerical Analysis*, 12(4):617–629, 1975.

[95] Antoine Petitet, R. Clint Whaley, Jack J. Dongarra, and Andy Cleary. HPL - a portable implementation of the High-Performance Linpack benchmark for distributed-memory computers. `http://www.netlib.org/benchmark/hpl`, 2012.

[96] Rodrigo B. Platte, Lloyd N. Trefethen, and Arno B. J. Kuijlaars. Impossibility of fast stable approximation of analytic functions from equispaced samples. *SIAM Review*, 53(2):308–318, 2011.

[97] Alex Pothen and Sivan Toledo. Elimination structures in scientific computing. In Dinesh Mehta and Sartaj Sahni, editors, *Handbook on Data Structures and Applications*, pages 1–59.29. CRC Press, 2004.

[98] Jack Poulson, Björn Engquist, Siwei Li, and Lexing Ying. A parallel sweeping preconditioner for heterogeneous 3D Helmholtz equations. *ArXiv e-prints*, March 2012.

[99] Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, and Nichols A. Romero. Elemental: A new framework for distributed

memory dense matrix computations. *ACM Transactions on Mathematical Software*, 39(2).

[100] Jack Poulson and Lexing Ying. Clique. `http://bitbucket.org/poulson/clique/`, November 2012.

[101] Jack Poulson and Lexing Ying. Parallel Sweeping Preconditioner. `http://bitbucket.org/poulson/psp/`, November 2012.

[102] Padma Raghavan. Efficient parallel sparse triangular solution using selective inversion. *Parallel Processing Letters*, 8(1):29–40, 1998.

[103] Padma Raghavan. Domain-Separator Codes for the parallel solution of sparse linear systems. Technical Report CSE-02-004, The Pennsylvania State University, University Park, PA, 2002.

[104] Junuthula N. Reddy. *An introduction to continuum mechanics with applications*. Cambridge University Press, New York, NY, 2008.

[105] Theodore J. Rivlin. *Chebyshev polynomials: From approximation theory to algebra and number theory*. Wiley-Interscience, second edition, 1990.

[106] Walter Rudin. *Principles of mathematical analysis*. McGraw-Hill, New York, NY, third edition, 1976.

[107] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, Philadelphia, PA, 2003.

[108] Yousef Saad and Martin H. Schultz. A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[109] Martin D. Schatz, Jack Poulson, and Robert A. van de Geijn. Scalable Universal Matrix Multiplication: 2D and 3D variations on a theme. Technical Report, University of Texas at Austin, Austin, TX, 2012.

[110] Phillip G. Schmitz and Lexing Ying. A fast direct solver for elliptic problems on general meshes in 2D. *Journal of Computational Physics*, 231:1314–1338, 2012.

[111] Steven H. Schot. Eighty years of Sommerfeld's radiation condition. *Historia Mathematica*, 19:385–401, 1992.

[112] Robert Schreiber. A new implementation of sparse Gaussian elimination. *ACM Transactions on Mathematical Software*, 8(3):256–276, 1982.

[113] Robert Schreiber. Scalability of sparse direct solvers. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 191–209, New York, NY, 1993. Springer-Verlag.

[114] David S. Scott. Out of core dense solvers on Intel parallel supercomputers. In *Proceedings of Frontiers of Massively Parallel Computation*, pages 484–487, 1992.

[115] Laurent Sirgue and R. Gerhard Pratt. Efficient waveform inversion and imaging: A strategy for selecting temporal frequencies. *Geophysics*, 69(1):231–248, 2004.

[116] Edgar Solomonik and James W. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par 2011 Parallel Processing*, volume 6853 of *Lecture Notes in Computer Science*, pages 90–109. Springer Berlin Heidelberg, 2011.

[117] Elias M. Stein and Rami Shakarchi. *Complex analysis.* Princeton Lectures in Analysis. Princeton University Press, Princeton, NJ, 2003.

[118] G. W. Stewart. A Krylov–Schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 23(3):601–614, March 2001.

[119] G. W. Stewart. *Matrix algorithms Volume II: Eigensystems.* SIAM, Philadelphia, PA, 2001.

[120] Christiaan C. Stolk. A rapidly converging domain decomposition method for the Helmholtz equation. *ArXiv e-prints*, August 2012.

[121] Albert Tarantola. Inversion of seismic reflection data in the acoustic approximation. *Geophysics*, 49:1259–1266, 1984.

[122] Lloyd N. Trefethen and David Bau. *Numerical linear algebra.* SIAM, Philadelphia, PA, 1997.

[123] Paul Tsuji. Fast algorithms for frequency-domain wave propagation. Ph.D. Dissertation, University of Texas at Austin, Austin, TX, 2012.

[124] Paul Tsuji, Björn Engquist, and Lexing Ying. A sweeping preconditioner for time-harmonic Maxwell's equations with finite elements. *Journal of Computational Physics*, 231(9):3770–3783, 2012.

[125] Paul Tsuji, Björn Engquist, and Lexing Ying. A sweeping preconditioner for Yee's finite difference approximation of time-harmonic Maxwell's equations. *Frontiers of Mathematics in China*, 7(2):347–363, 2012.

[126] Robert A. van de Geijn. Massively parallel LINPACK benchmark on the Intel Touchstone DELTA and iPSC/860 systems. In *Proceedings of Intel Supercomputer Users Group*, Dallas, TX, 1991.

[127] Robert A. van de Geijn and Jerrell Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. *Concurrency: Practice and Experience*, 9:255–274, 1997.

[128] Charles F. van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1-2):85–100, 2000.

[129] Charles F. van Loan and Nikos Pitsianis. Approximation with Kronecker products. In *Linear Algebra for Large Scale and Real Time Applications*, pages 293–314. Kluwer Publications, 1993.

[130] Shen Wang, Maarten V. de Hoop, and Jianlin Xia. On 3D modeling of seismic wave propagation via a structured parallel multifrontal direct Helmholtz solver. *Geophysical Prospecting*, 59:857–873, 2011.

[131] Shen Wang, Xiaoye S. Li, Jianlin Xia, Yingchong Situ, and Maarten V. de Hoop. Efficient scalable algorithms for hierarchically semiseparable matrices. Technical report, Purdue University, West Lafayette, IN, 2011.

[132] James H. Wilkinson. *Rounding errors in algebraic processes.* Prentice-Hall, Englewood Cliffs, NJ, 1963.

[133] James H. Wilkinson. *The algebraic eigenvalue problem.* Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, England, 1965.

[134] Jianlin Xia, Shiv Chandrasekaran, Ming Gu, and Xiaoye S. Li. Superfast multifrontal method for large structured linear systems of equations. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1382–1411, 2009.