An Adaptive Fast Multipole Method-Based PDE Solver in Three Dimensions

by

Matthew Harper Langston

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

January 2012

Denis Zorin

To Victoria, the love of my life and the most important person in it.

Acknowledgments

I would like to thank all of the professors, colleagues, and friends, without whom, this would not have been possible. First, my advisor, Denis Zorin for his constant help and advice over these many years, Leslie Greengard, Mike Shelley, and George Biros for their invaluable advice and encouragement. I also would like to acknowledge the Department of Energy CPES grant, which helped support this work, including C.S. Chang, the researchers at the Princeton Plasma Physics Laboratory and Oak Ridge National Laboratory. Additionally, I appreciate the support of the New York University High Performance Computing group, on whose machines much of this work was tested, debugged and run.

Many thanks to my colleagues, Adrian Secord, Elif Tosun, Evgeniy Parilov, Robb Bifano, Will Casey, Yotam Gingold, Matt Grimes, Denis Kovacs, Jason Reisman, Chris Wu, Chris Poultney, Ilya Rosenberg, Lexing Ying, Shravan Veerapaneni, Ilya Lashuk, Rahul Sampath, Abtin Rahimian, and many others for their constant support.

To my family, especially my sister Marisa Simon Langston, my brother-in-law Mitchell Simon, and my mother Judith Langston for being there when I needed. To my in-laws, Arthur Lichtendorf and his wife Judith Lichtendorf for inviting me up to their apartment for late-night sessions of good food and a relaxing environment; I only wish Arthur were still with us to help celebrate the completion. To those who saw me enter this journey but are no longer with me to see me finish, I wish you could be here for me to show my appreciation in everything you did for me, especially Arthur, his first wife Susan, my grandparents Dorothy Bandy Melton, Eleanor Floyd, and John Melton. I could have never done any of this without their support while they were with me, and I definitely could not have done it without the strength of their memories.

I thank my son Max for helping me see what is most important in life when I was in the greatest states of despair. Most importantly, I thank my wife, Victoria Lichtendorf, for putting up with these many years of late nights, strange schedules, and stress-filled sleepless nights. Without your constant support, none of this would have ever been possible.

ABSTRACT

Many problems in scientific computing require the accurate and fast solution to a variety of elliptic partial differential equations (PDEs). These problems become increasingly difficult in three dimensions when forces become non-homogeneously distributed and geometries are complex.

We present an adaptive fast volume solver using a new version of the Fast Multipole Method (FMM), incorporated with a pre-existing boundary integral formulation for the development of an adaptive embedded boundary solver.

For the fast volume solver portion of the algorithm, we present a kernel-independent, adaptive fast multipole method of arbitrary order accuracy for solving elliptic PDEs in three dimensions with radiation boundary conditions. The algorithm requires only a Green's function evaluation routine for the governing equation and a representation of the source distribution (the right-hand side) that can be evaluated at arbitrary points. The performance of the method is accelerated in two ways. First, we construct a piecewise polynomial approximation of the right-hand side and compute far-field expansions in the FMM from the coefficients of this approximation. Second, we precompute tables of quadratures to handle the near-field interactions on adaptive octree data structures, keeping the total storage requirements low through the exploitation of symmetries. We additionally show how we extend the free-space volume solver to solvers with periodic and homogeneous Dirichlet boundary conditions.

For adaptively solving PDEs with Dirichlet boundary conditions in complex geometries, we incorporate the fast free-space volume solver into an exisiting embedded boundary approach, developing interpolation methods to maintain the accuracy of the volume solver. These methods use the existing FMM-based octree structure to locate appropriate interpolation points, building polynomial approximations to this larger set of forces and evaluating these polynomials to the locally under-refined grid in the area of interest.

We present numerical examples for the Poisson, modified Helmholtz and Stokes equations for a variety of boundary conditions and geometries as well as studies of the interpolation procedures and stability of far-field and polynomial constructions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Many problems in scientific computing call for the efficient solution to linear partial differential equations (PDEs) with constant coefficients. On regular grids with separable Dirichlet, Neumann or periodic boundary conditions, such equations can be solved using fast, direct methods. For free-space boundary conditions and highly nonuniform source distributions defined on adaptive and/or unstructured grids, alternative approaches are necessary. We describe a *direct* high-order adaptive solver for inhomogeneous linear constant-coefficient PDEs in three dimensions with decay conditions at infinity:

$$\mathcal{L}(u)(\mathbf{x}) = f(\mathbf{x})\ supp(f) \subset \Omega, \tag{1}$$

where Ω is a bounded domain in $\mathbf{R}^3$, and $u(\mathbf{x}) = O(1/|\mathbf{x}|)$ as $|\mathbf{x}|$ goes to infinity. Our solver uses a kernel-independent fast multipole method (FMM) (Ying et al., 2003; Ying et al., 2004b), which can be applied to any PDE, for which a free-space Green's function evaluation routine is provided. It handles highly nonuniform sources in an efficient manner, using an adaptive approximation of the right-hand side. The structure of the solver allows for natural integration with FMM-based boundary integral equation techniques, leading to the construction of an adaptive kernel-independent embedded boundary solver for inhomogeneous PDEs in complex geometries with Dirichlet boundary conditions. In particular, the solution is often desired inside or outside of some complex domain ω with boundary γ as in figure 1

In regular geometries, this has been well-studied, and in two dimensions there are a variety of solvers which work with complex smooth geometries or highly non-uniform force distributions. In three dimensions, in particular, there are current fast methods for smooth complex geometries and uniform force distributions, based in potential theory. We will discuss several of these

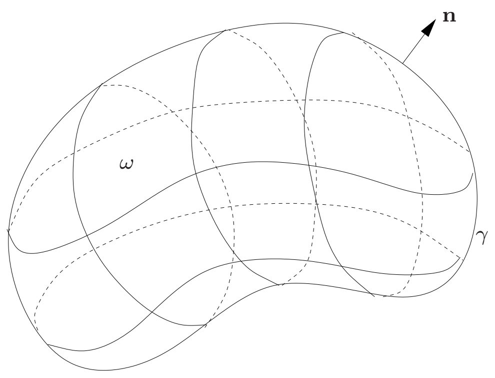


Figure 1: General 3D shape with domain ω and boundary γ.

approaches.

Potential theory has laid the groundwork for the analysis and computation of elliptic boundary value PDEs. Applications of this theory have been seen in fluid mechanics, electrodynamics, acoustics and planetary simulations, among others. Potential theory uses integral equations for the Poisson equation in unbounded space, where the solution in two and three dimensions is the convolution of the force with a fundamental Green's solution. In particular, if we consider (1) to be the Poisson equation,

$$-\Delta u(\mathbf{x}) = f(\mathbf{x}),$$

the solution to in the absence of physical boundaries can be written as the convolution of the force with the Green's function $G(\mathbf{x}, \mathbf{y}) = (1/4\pi(|\mathbf{x} - \mathbf{y}|)^{-1})$ in three dimensions:

$$u(\mathbf{x}) = \frac{1}{4\pi} \int \frac{f(\mathbf{y})}{|\mathbf{x} - \mathbf{y}|} d\mathbf{y}$$

For smooth and piecewise-smooth boundaries, boundary integral equations can be developed using integral equations There are several advantages to using potential theory based approaches, including

- No need for complex mesh generation for calculating volume potentials,
- Far-field boundary conditions can be satisfied, and
- Higher degrees of accuracy.

Unfortunately, numerical approaches based in potential theory can be computationally expensive performed naïvely. In order to obtain asymptotic optimality, several issues need to be addressed. Specifically,

- Fast solvers should be incorporated. The operators resulting from a boundary integral approach can result in dense systems of linear equations, which are difficult to solve with traditional methods. Solvers exist for decreasing computational complexity while maintaining high levels of accuracy.
- Discretization rules for the integral operators must be chosen wisely. The kernels based on the fundamental solutions can include singularities, so quadrature methods or projection methods in the development of such rules must be developed wisely.
- Boundaries must be represented carefully. For smooth boundaries, rules have been developed in two and three dimensions to insure high levels of accuracy.
- Evaluating solutions near boundaries is often needed. For potential theory based methods, difficulties arise with nearly singular integration.
- Distributed body forces may be nonuniform or highly so. For many problems, body forces are not available everywhere; evaluations of potentials need to incorporate the ability to take this into account.

Currently, the ability to incorporate a fast integral equation solver for elliptic PDEs, allowing for non-uniform distributions of forces in three dimensions with high accuracy is needed. The main added benefits of the proposed method will be:

- Ability to handle highly non-uniform force distributions. Using adaptive methods based on integral equations without grids, current methods, which rely on uniform force distributions for solving the free-space Poisson equation, can be improved.

- Ability to handle complex geometries. Current methods support fast methods for arbitrarily smooth geometries, and we incorporate these into our method.

- High accuracy. Using methods based in potential theory allows for direct evaluation of the solution at a non-uniform distribution of target points at high levels of accuracy. quickly.

- In the use of the kernel-independent FMM, an added benefit will be the straightforward extension to a large class of elliptic PDEs

Our approach is based on the Embedded Boundary Integral Approach from the work of (Ying et al., 2004a; Mayo, 1984). For distributed forces available everywhere, (Ying et al., 2006) discretizes the regular domain for an FFT-solver. For a non-uniform force distribution, this approach is inadequate. We propose to use the volume integral approach of (Ethridge and Greengard, 2001; Ethridge, 2000) along with the kernel-independent Fast Multipole Method (Ying et al., 2004b; Ying et al., 2003). For smooth boundaries, Nyström methods are available along with nearly singular integration for evaluation of potentials near the boundary (Ying et al., 2006; Bruno and Kunyansky, 2001).

We begin by discussing many of the methods used for solving elliptic PDEs in free-space and in more complex geometries and then discuss our methods.

1 RELATED WORK

Several approaches have been developed for solving second-order constant-coefficient PDEs. The majority of these approaches fall under several categories; specifically, most involve fast Fourier transforms or some variation, grid methods, or fast direct solvers. We discuss each of these approaches below, focusing on fast direct methods last as this is the major focus of our approach, specifically the Fast Multipole Method.

1.1 Fast-Fourier Based Solvers

For regular grids in separable coordinate systems (rectangles, disks, spheres, etc) fast methods for constant-coefficient second order PDEs are well-established (Buzbee et al., 1970; Canuto et al., 1987). These methods generally rely on cyclic reduction and/or fast Fourier transforms (FFTs) to achieve nearly linear scaling. FFT-based methods are based on regular discretizations of a domain. As an example of a regular grid, consider figure 1.1

If we were solving the Poisson equation for example and assume we are able to evaluate the discrete Laplacian on this regular grid, it is possible to build a system of linear equations, $Au = b$ where A is a block tridiagonal symmetric Toeplitz (TST) matrix. For example, let our domain in figure 1.1 be $\Omega = (0, L_x) \times (0, L_y)$, and let N_x and N_y be integers defining how many sections we will discretize Ω into in the x and y directions, respectively. Let the step-size in the x-direction be $h_x = L_x/N_x$, and in the y-direction be $h_y = L_y/N_y$. Therefore, our grid-points are defined as $\mathbf{x}_{i,j} = (i * h_x, j * h_y)$, where $0 \leq i \leq N_x$, $0 \leq j \leq N_y$. Grid-points on the boundary, $\partial\Omega$ are explicitly defined as:

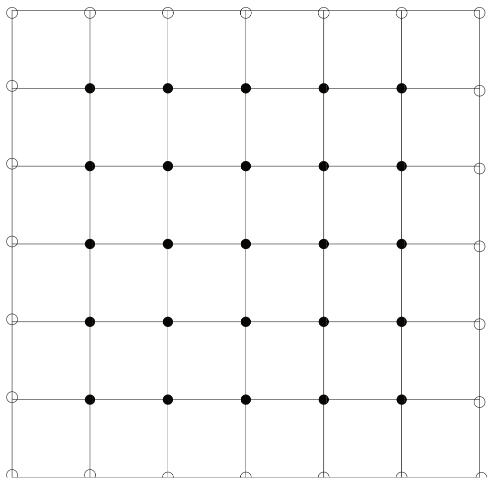


Figure 1.1: The 2-D computational grid for a regular rectangle. Dark circles refer to interior points and clear circles refer to boundary points. In this example, $h_x = h_y$

$$\partial\Omega = \left\{ \begin{array}{lll} \mathbf{x}_{0,j} & | \quad 0 \le j \le N_y & \cup \\ \mathbf{x}_{N_x,j} & | \quad 0 \le j \le N_y & \cup \\ \mathbf{x}_{i,0} & | \quad 0 \le i \le N_x & \cup \\ \mathbf{x}_{i,N_y} & | \quad 0 \le i \le N_x & \end{array} \right\}.$$

Using the notation that $u(\mathbf{x}_{i,j}) = u_{i,j}$, we can reference the 1D $O(h^2)$ approximation of the second-derivative to find approximations in both the x and y directions for the second derivatives at interior points:

$$\frac{\partial^2 u_{ij}}{\partial x^2} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2} + O(h^2) \tag{1.1}$$

$$\frac{\partial^2 u_{ij}}{\partial y^2} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2} + O(h^2) \tag{1.2}$$

Assuming that $h_x = h_y = h$ and combining these equations, we arrive at the following approximation for the Laplacian, requiring five interior points:

$$\Delta u_{ij} \approx \frac{u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}}{h^2}$$

Another way of representing this approximation to the Laplacian at interior points is through the use of a graphical *stencil*. For example, in Figure 1.2, we show how we can approximate the Poisson Equation and easily set up our matrix of coefficients at interior points via a stencil (boundary points are handled differently for different boundary condition types).

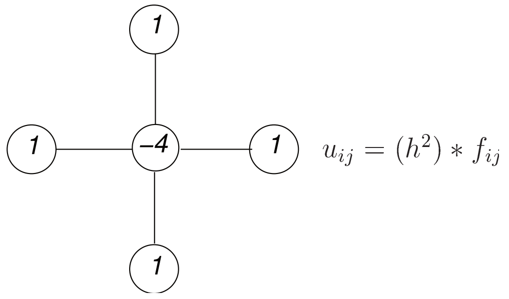


Figure 1.2: Stencil for the 5-point $O(h^2)$ approximation to the Laplacian

For 3D, consider figure 1.3. In this figure, assume that the black point is the only point in the interior of the domain Ω, and we are evaluating Δu there. Again using a Taylor expansion, we can quickly establish a scheme that is an $O(h^2)$ approximation. Similar to equations 1.1 and 1.2, define the operator, $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ when applied to $u(\mathbf{x})$. Since we are in 3D, let $\mathbf{x} = (x, y, z)$. Define Ω in 3D as $(0, L_x) \times (0, L_y) \times (0, L_y z)$ and let N_x, N_y, and N_z be integers defining how many sections we will discretize Ω into in the x, y, and z directions, respectively. Let the step-size in the x-direction be $h_x = L_x/N_x$, in the y-direction be $h_y = L_y/N_y$, and in the z-direction be $h_z = L_y/N_z$ such that the grid-points are defined as $\mathbf{x}_{i,j,k} = (i * h_x, j * h_y, k * h_z)$ where $0 \leq i \leq N_x$, $0 \leq j \leq N_y$, and $0 \leq k \leq N_z$. This is a direct evolution from 2D, and the definition of the boundary follows directly. For $u_{i,j,k}$, we can define all three parts of the operator, Δ:

$$\frac{\partial^2 u_{i,j,k}}{\partial x^2} = \frac{u_{i-1,j,k} - 2u_{i,j,k} + u_{i+1,j,k}}{h_x^2} + O(h^2)$$

$$\frac{\partial^2 u_{i,j,k}}{\partial y^2} = \frac{u_{i,j-1,k} - 2u_{i,j,k} + u_{i,j+1,k}}{h_y^2} + O(h^2)$$

$$\frac{\partial^2 u_{i,j,k}}{\partial z^2} = \frac{u_{i,j,k-1} - 2u_{i,j,k} + u_{i,j,k+1}}{h_z^2} + O(h^2)$$

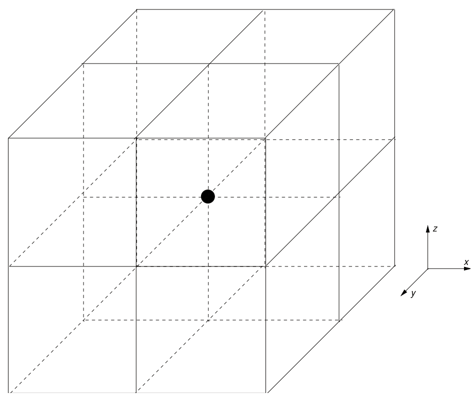


Figure 1.3: Regular 3d grid, showing a single point in black with its 27 (including itself) surrounding points, from which we can build an $O(h^2)$ or $O(h^4)$ finite-difference scheme, assuming $h = h_x = h_y = h_z$.

Letting $h_x = h_y = h_z$, we can define an $O(h^2)$ convergent 7-point scheme for interior points of our domain, Ω, and hence numerically approximate our Poisson equation, $-\Delta u(\mathbf{x}) = f(\mathbf{x})$ as

$$\frac{6u_{i,j,k} - u_{i-1,j,k} - u_{i+1,j,k} - u_{i,j-1,k} - u_{i,j+1,k} - u_{i,j,k-1} - u_{i,j,k+1}}{h^2} = f_{i,j,k}.$$

Higher-order finite-difference schemes can be taken into account by using additional points. For example, in two dimensions, nine-point $O(h^4)$ schemes can be built, as described in (Iserles, 1996). Similarly, 27-point schemes can be derived in three dimensions.

Regardless of the form of A, there is a long history of solving such systems of linear equations. This field is too dense to review here, but we discuss several of the more popular approaches in section A.5. For this problem, specifically with A built using one of these regular stencils as above, however, knowing that A is a TST matrix, we can rewrite A as

$$Tu_{j-1} + Su_j + Tu_{j+1} = b_j, j = 1, ..., n.$$

The form of T and S are based on the chosen finite-difference schemes for the Laplacian on rectangular grids. The matrices, S and T are also TST, for which we can write S and T as (Iserles, 1996) $S = Q_S \Lambda_S Q_S^T$ and $T = Q_T \Lambda_T Q_T^T$, respectively, where the matrices of eigenvectors, eigenvectors have the form $\mathbf{q}_j$, $q_{j,k} = \left(\sqrt{\frac{2}{n+1}}\right) \sin\left(\frac{\pi jk}{n+1}\right)$. Since $Q_S = Q_T = Q$ where $Q = Q^T$, rewrite each row of the system of equations as

$$\Lambda_T \hat{u}_{j-1} + \Lambda_S \hat{u}_j + \Lambda_T \hat{u}_{j+1} = \hat{b}_j \tag{1.3}$$

$$\hat{u}_j = Qu_j,\ \hat{b}_j = Qb_j,\ j = 1, ...n.$$

Let $\hat{U}$ be the matrix whose jth column is $\hat{u}_j$ and let $\hat{u}_j^t$ be jth column of $\hat{U}^T$, which is also the jth row of $\hat{U}$. Indicate λ_j^S as the jth eigenvalue of S, or the jth entry along the diagonal of Λ_S. Similarly, λ_j^T is the jth eigenvalue of T. Let Γ_j be a TST matrix whose main diagonal is filled by λ_j^S and the first diagonals in both directions are filled with λ_j^T. Rewrite equation 1.3 as

$$\Gamma_j \hat{u}_j^t = \hat{b}_j^t.$$

The main benefit of using row-wise operations is that we were able to construct a series of systems of linear equations for each column of U, where this new system involves another TST matrix (each Γ_j is TST), so we already know the eigenvalues and eigenvectors; however, as suggested by (Iserles, 1996), the system is a tridiagonal banded system of linear equations. So, if U is an $n \times n$ system, we can solve $\Gamma_j \hat{u}_j^t = \hat{b}_j^t$ using band Cholesky (see section A.5.1) with $O(n)$ operations since the bandwidth is 1. This process is commonly referred to as the Hockney method.

As Q is just a matrix of discrete sine transforms, matrix-vector multiplication with Q is sped this up via FFTs, reducing an FFT solver to $O(NlogN)$ operations, compared to a dense $O(N^3)$ direct solver. We discuss FFTs and cyclic reduction further in section A.5.4.

Further examples of similar approaches to solving the Poisson equation using FFTs can be seen in (Swartztrauber, 1984), and the Fortran algorithm (Swartztrauber and Sweet, 1979). The latter is also available as part of the popular FISHPACK package, a publicly available collection of Fortran codes. Along with FFTPACK, FISHPACK has been developed at NCAR (the National Center for Atmospheric Research). FFTW (Frigo and Johnson, 1997) is currently the fastest implementation (Frigo and Johnson, 1998; Frigo and Johnson, 2005) and has been incorporated into a variety of packages, and is used in current research such as (Ying, 2004) for acceleration of FFT-based aspects of elliptic PDE solvers.

Cyclic reduction (Buzbee et al., 1969; Buzbee et al., 1970; Swartztrauber, 1974; Swartztrauber and Sweet, 1996) is a generalization of the Hockney method for the FFT decomposition. In regular geometries, variations of this method have been applied to the Poisson equation in (Buneman, 1969; Buzbee et al., 1970; Buzbee et al., 1971). FFT-based methods, however, are not as directly useful for irregular discretizations of a domain or when boundaries are complex.

1.2 Structured and Unstructured Grid Methods

For many problems, adaptive meshes resulting from adaptive mesh refinement (AMR) strategies are essential (Berger et al., 1996; Aftosmis et al., 1998; Minion, 1999), and existing solvers typically rely on domain decomposition strategies (Chesshire and Henshaw, 1991) or multigrid acceleration (Chan and Smith, 1994; Hackbusch and Trottenberg, 1982; Johansen and Colella, 1998; Martin and Cartwright, 1996b). We discuss the general Multigrid Method approach in more detail in section A.5.3.

For complex geometries, unstructured grid generation techniques are often used (e.g., (Mavripilis, 1997)), providing a way to increase grid resolution of a solution at points of interest. These areas could include locations near high vorticity in vortex-based fluid solvers. Additionally, near boundaries, it is often desirable for the grid to be more highly refined. In such cases, both the grid generation process and the solution of the resulting linear systems can be computationally expensive. The lack of regularity in the data structures adds complexities in parallelization as well (Adams and Demmel, 1999; Chan and Smith, 1994).

One popular technique for local mesh refinement in areas of interest is Adaptive Mesh Refinement (AMR) (Aftosmis et al., 1998). Originally designed for inviscid compressible fluids (Berger and Colella, 1989; Berger and Oliger, 1984), AMR has been extended to a variety of situations for a variety of PDEs (Berger et al., 1996). In particular for our interest, AMR has been utilized for solving the Poisson equation (Martin and Cartwright, 1996b; Martin, 1998) using multigrid acceleration (Hackbusch and Trottenberg, 1982; Hackbusch, 1985; Adams, 1998). Specifically, (Martin and Cartwright, 1996b) has been built as an AMR Poisson code (Martin and Cartwright, 1996a) as well as incorporated into the FLASH software package (Rosner et al., 2000). Additional grid-methods for solving the Poisson equation include (Chan and Smith, 1994; Johansen and Colella, 1998).

(Martin and Cartwright, 1996b) begin with the Poisson equation $\Delta u = f$ on some domain Ω. The domain is discretized on a regular coarse grid, denoted by Ω^0, which includes all of Ω. The Poisson equation is solved on Ω^0 and then areas of high error are located and resolved locally there. We denote Ω^l as the union of all areas of refinement at level l, where Γ^l is the boundary of that level's domain. Grids at different levels are not allowed to overlap here, such that they are *properly nested*. Modifications can be made to remove this constraint. The result is $\Omega^{l_{max}} \subset ... \subset \Omega^2 \subset \Omega^1 \subset \Omega^0$. An example can be seen in figure 1.4

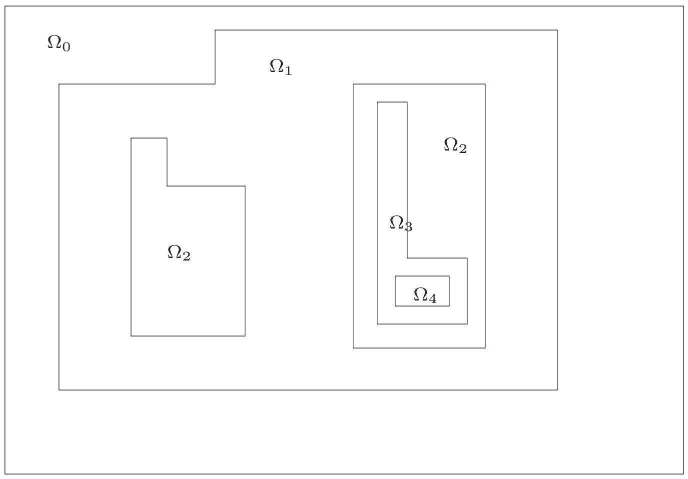


Figure 1.4: Nested domains used in AMR-based computations

While maintaining a hierarchy of nested grids, it is also often necessary to pass information between different grid levels. For this, a restriction (or projection) operator, R, restricts information from a gridpoint in refinement level l to one in $l-1$. Similarly, a prolongation (or refinement) operator, P, refines information at points in refinement level l to a point in $l+1$. This is analogous to operators seen in the multigrid methods (section A.5.3).

For the interior of a refinement level, Ω^l, it is straightforward to discretize the Laplacian there if the grid structure is regular (for unstructured grids, use finite elements for building a system of equations (Adams, 1998; Adams, 2004; Adams et al., 2004)).

At points which cross the boundary Γ^l, computing Δ_h is more complicated. Figure 1.5, provides two examples.

The full solver for this method involves a multigrid solver for refining the residual. The process begins at the finest level, and then using this information to move along a *V-cycle* as required to the coarsest levels and back; we introduce and discuss V-cycles in section A.5.3. In (Martin and Cartwright, 1996b), great attention has to be paid to maintain consistencies between the different refinement levels as the operators only apply to specific subdomains of the full domain Ω. Once a solution is acquired, decisions are made as to whether more refinement is required in a specific area.

When grids are unstructured, simple approximations to the Laplacian are not available, and finite element methods need to be used (Adams, 1998; Adams and Demmel, 1999; Adams et al., 2004); we review finite elements and how they can be used for discretization the operators, specifically Laplace, in section A.4. This increases difficulties in generating hierarchies of grids for using multigrid. As discussed in section A.5.3, coarse grids for multigrid are determined from fine grids. When grids no longer have regular structure, building these grids becomes complex. Geometric multigrid methods build coarse grids based on geometry information such as spacing between grid points. Algebraic multigrid methods (Ruge and Stüben, 1987; Brezina et al., 2004a) use coefficient-weighting techniques. Multigrid methods are based on using solvers, which reduce residuals based on the directions of the *strongest* connections in the matrices, i.e., the entries of each matrix with the largest possible value at a specific location. Coarsening and weight calculations for interpolation can be done during initialization often, leading to good complexity for algebraic multigrid. (Adams et al., 2003; Brezina et al., 2004b) discuss parallelization issues.

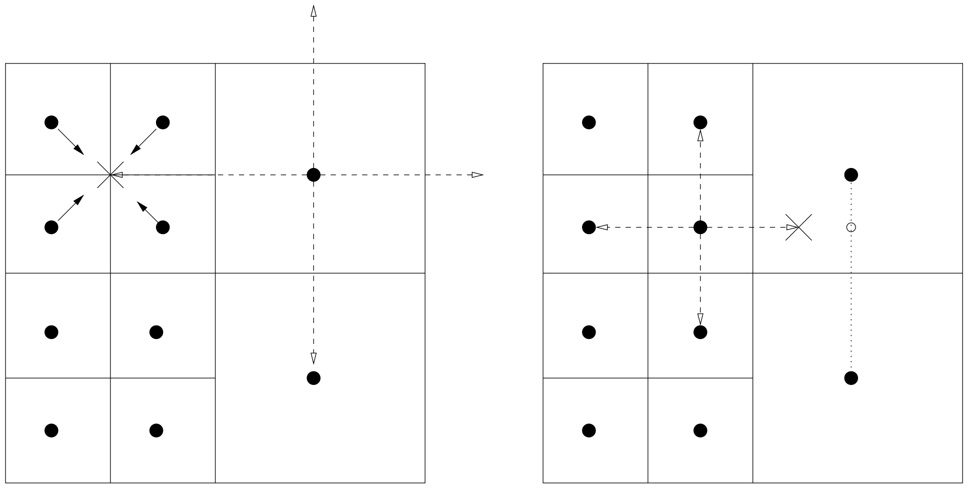


Figure 1.5: For approximating the Laplacian near boundaries, we need to use interpolation techniques. On the left, approximation of the discrete Laplacian is at a point in Ω^f, and on the right at a point in Ω^c. In both cases, approximation relies on points on the other side of Γ^f. On the left, one is discretizing the Laplacian at a coarse grid, Ω^c, which borders a finer grid, Ω^f, with boundary Γ^f. Computing the Laplacian at a point where a necessary neighbor crosses Γ^f is a straightforward matter of interpolating fine grid values to the needed point or using the restriction operator. For the image on the right, the Laplacian at the finer grid needs to be approximated, and a point is necessary in Ω^c for which we do not have values. In (Martin and Cartwright, 1996b), quadratic interpolation is used to find values at the solid white point using known information at the black points. The, another step of quadratic interpolation is used to obtain the information at the cross-mark from the solid white point, now known, and the point inside of Ω^f. Fine grid corners involve additional steps as outlined in (Martin and Cartwright, 1996b).

A more recent class of methods combines ideas from potential theory with finite difference methods. In (Greengard and Huang, 1999), fast direct solvers were used on a sequence of refined grids with boundary conditions inherited from the coarser levels. This results in discontinuities at coarse-fine interfaces which are corrected using a second pass through the grid hierarchy. In (Balls and Colella, 2002), the method of local corrections (MLC) (Anderson, 1986) was combined with multigrid methods to solve the Poisson equation on a hierarchy of nested grids. The authors also showed how to impose free-space boundary conditions on the computational domain. The fastest free-space Poisson solver for three-dimensional problems of which we are aware is described in (McCorquodale et al., 2007). It solves local Poisson problems on fine grids using FFT-based techniques and couples together the solutions on coarser grids using the MLC. This approach was shown to be very effective in parallel, with good scaling up to 1024 processors. (A similar two-dimensional scheme is described in (Greengard and Lee, 1996)). For unstructured meshes, the preceding methods don't apply without significant modification and most fast solvers are based on iterative methods using multigrid or domain decomposition acceleration (Brandt, 1977; Chan et al., 1989; Briggs et al., 2000).

We mention additional grid methods. The Immersed Boundary Method (Peskin, 1977; Peskin and McQueen, 1989; Peskin and Prinz, 1993) is a successful grid-based method, but the accuracy is low for our needs, due to the way in which the interface is modeled with delta functions. Another option is regular grids which use stencils to compensate for necessary jumps in potential across boundaries (Berger and Colella, 1989; Liu et al., 2000; Cheng et al., 2001). This avoids the necessity for keeping a hierarchy of grids. Unfortunately, for highly nonuniform force distributions, we cannot rely on such grids where distributed forces are expected to be known everywhere. In three dimensions, grid-based methods become increasingly difficult even if forces are uniformly distributed, and these methods can have difficulties for edges and corners in terms of necessary refinement.

1.3 Fast Direct Solvers

Another approach is to incorporate fast direct solvers, which are efficient, scale well in parallel and do not require maintaining a hierarchy of grids. In Section A.3, we describe how to formulate fundamental solutions for certain problems such as the Poisson and Stokes equations. In general, potential theory leads to a way of representing the potential, u, as a convolution of a density distribution function with a kernel, G, representing the fundamental Green's solution to a specific PDE. That is, for an unbounded elliptic PDE,

$$\mathcal{L}(u)(\mathbf{x}) = f(\mathbf{x}) \; supp(f) \subset \Omega, \tag{1.4}$$

where Ω is a bounded domain in $\mathbf{R}^d$, and $u(\mathbf{x})$ decays as $1/|\mathbf{x}|$ at infinity, the integral equation becomes

$$u(\mathbf{x}) = \int\limits_{\Omega} G(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y}$$

If the force is represented solely as singular sources, we represent this as a summation at n discrete points

$$u_i = \sum_{j=1}^{n} G_{i,j} f_j.$$

Here, f_j is a source density, given or interpolated at the n points (i.e., charge distribution or vorticity field value for vortex methods (Majda and Bertozzi, 2002)). Rather than (1.4), for example, the integral equation approach for the Poisson equation in three dimensions utilizes a Green's function of the form $G(\mathbf{x}, \mathbf{y}) = 1/4\pi \, ||\mathbf{x} - \mathbf{y}||$ and computes

$$u(\mathbf{x}) = \frac{1}{4\pi} \int_{\mathbf{R}^d} \frac{1}{|\mathbf{x} - \mathbf{y}|} g(\mathbf{y}) d\mathbf{y}. \tag{1.5}$$

Among the advantages of this integral equation approach (or, more precisely, the integral transform) is the increase in precision in computing derivatives. In PDE-based methods, if first or second derivatives of the solution are needed, accuracy tends to degrade due to the need for numerical differentiation. Instead, we can differentiate a kernel and compute derivatives from their integral representation as well. Other advantages are that free-space radiation conditions are automatically satisfied, we can obtain simple *a priori* error estimates, and high order accuracy is straightforward to achieve. However, the computational complexity of a naïve implementation is high: computing the solution u at N points $\mathbf{x}$ given N discretization points $\mathbf{y}$ requires $O(N^2)$ work. There have been a number of methods proposed to overcome this barrier. These include panel-clustering techniques (Börm and Hackbusch, 2005; Hackbusch and Nowak, 1989), hierarchical matrices ($\mathcal{H}$,$\mathcal{H}^2$-matrices) (Hackbusch, 1999; Hackbusch and Börm, 2002; Börm, 2006), the Barnes-Hut method (Barnes and Hut, 1986), and the Fast Multipole Method (FMM) (Rokhlin, 1985; Greengard, 1988; Greengard and Rokhlin, 1997; Cheng et al., 1999), originally designed for gravitational/Coulomb interactions. These schemes all achieve linear $O(N)$ or nearly linear $O(N \log N)$ scaling. Most of these methods fall into the class of what are often called *tree codes* because they separate near and far-field interactions on a hierarchy of spatial scales using quadtree (2D) or octree (3D) data structures.

We begin by discussing the Barnes-Hut method for the purposes of introducing tree structures, and then we concentrate on the FMM. We also briefly discuss Particle-In-Cell methods.

1.3.1 Barnes-Hut Method

The Barnes-Hut Method (Barnes and Hut, 1986) is based on a simple idea. Imagine we are trying to calculate the gravitational force exerted on Earth by a group of planets and stars; this force is based on the relative masses of other planets and their distances from Earth. Now, imagine that a group of planets in some galaxy are close together but very far away from Earth.

Due to the fact that we are using numerical techniques to make our calculations, the difference between calculating the distance between all of these planets individually versus treating them as one mass and using a single distance calculation to calculate the relative force may be minimal. However, this can result in a significant savings computationally.

We can formalize this idea using figure 1.6. Imagine Earth is at a distance r from the center of mass of Galaxy Y, which can be grouped into a box with width D, and all of the planets, except for a single one in Galaxy Y can be separated into a group of planets, designated as Galaxy Z of size D_1 and the single planet. Consider the ratio $\frac{D}{r}$. If this ratio is below some threshold, treat the planets as a single mass there in order to calculate r. If $\frac{D_1}{r_1}$ is small, do the same. So, imagine these two ratios are small enough such that the center of mass of Galaxy Y is calculated as the average of center of mass of Galaxy Z, influenced by its total mass plus the center of mass of the other isolated planet. Then, r is the distance between Earth and Y's center of mass.

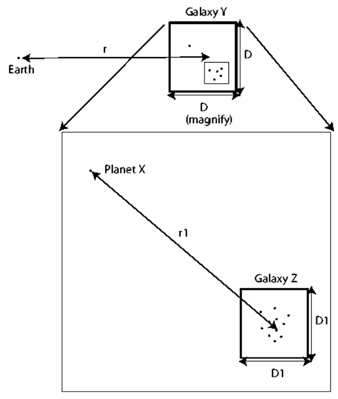


Figure 1.6: Using the relationships between galaxy sizes and relative distances, we can determine whether to use direct summations or far-field techniques.

We can continue recursively subdividing the space as necessary in order to separate and cluster planets and galaxies. To better understand how to represent the separations and clusterings, we turn to quadtree and octree data structures for two and three dimensions, respectively.

Quadtrees and Octrees

A *quadtree* is a search tree data structure for two dimensions where each node has at most four children. An *octree* is a tree data structure in three dimensions, where each node in the tree has at most eight children. For simplicity, quadtrees are mainly considered. The root of a quadtree points to an entire domain, Ω. For example, as in figure 1.7, reproduced from (Demmel, 1996), consider Ω is a unit square.

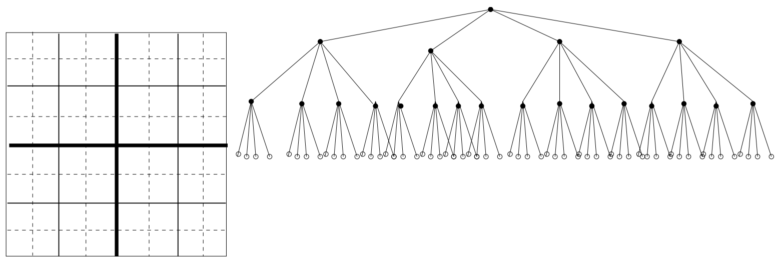


Figure 1.7: An example full, non-adaptive quadtree. The root of the corresponding quadtree stores information about the entire domain. Then subdivide Ω into four smaller domains (squares of equal size for our example), such that the four nodes in the second level of the quadtree store information about these four subdomains. Continue this recursively as shown such that the leaves of the quadtree correspond to the levels at which subdivision ceases. We can use this quadtree data structure to subdivide Ω until each of the smallest domains store the location of one or more particles.

Imagine we wish to subdivide some domain such that each leaf of a quadtree points to a subdomain which contains a single particle. In figure 1.8 we show a unit square domain with a small number of non-uniformly distributed particles. Instead of a uniform subdivision, the domain is subdivided such that the result is a nonuniform *adaptive* quadtree. The first child node of any subtree in the quadtree points to the upper-right subdomain and then counter-clockwise, as in the Cartesian coordinate system.

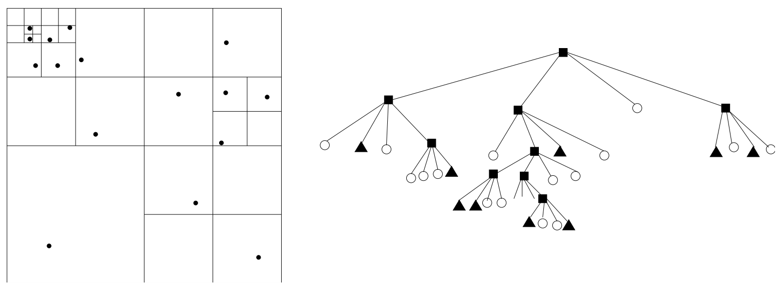


Figure 1.8: A unit square domain with a small number of randomly non-uniformly distributed particles. The resulting unit square with non-uniformly distributed particles is subdivided down to the level of a single particle enclosed by a single subdivided cell. The adaptive quadtree abstract data structure is shown to the right. Black squares represent interior nodes which point to their children. White circles represent leaves which point to the location of a single particle. Black triangles represent subdivided cells which point to an empty cell with no particles or children (a leaf with no particles). Typically, we would not store these nodes, instead representing them as *null*.

An optimal quadtree has height of $O(\log_4 n)$. So, the cost of building the quadtree is $O(n \log_4 n)$ for a uniformly-distributed set of particles. We now need to be able to use the tree to calculate the forces. For the gravitational force example, we describe how to calculate the center of mass for each cell in the quadtree and then use this to calculate the forces. First, however, we explain how the above ideas are straightforwardly extended to three-dimensions using octrees. For an octree data structure, each interior node will have eight child nodes. A simple explanatory picture can be seen in figure 1.9, showing a simple 3D unit box and the resulting two-level non-adaptive octree.

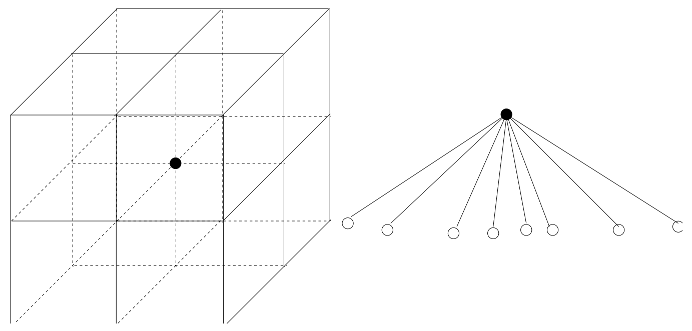


Figure 1.9: A simple two-level uniform octree.

The basic format for the Barnes-Hut Algorithm is as follows. Given a domain and n particles,

- Build the quadtree or octree as shown in the last section;
- Traverse the tree to compute the center of mass and total mass at each node in the tree;
- For all n particles, traverse the tree, computing the force particle-particle or particle-box as necessary.

We have already shown how to do the first step, and we now show how to do the second step. Assume mass is carried by each particle. We could also use charge distributions for problems in electrostatics or vorticity field values. The following approach allows us to compute the center of mass and total mass at each node in the tree. We show how this is done for a quadtree; extension to octrees is straightforward.

Computing the center of mass for each node in the tree involves a straightforward postorder traversal, computing the total mass and center of mass at every tree node along the way. To compute the total force for every particle in the tree, traverse the tree again in postorder fashion and calculate the distance, r from the particle to the center of mass of some node. Then, the force between the particle i and the total mass at at box Q's center of mass could be calculated by denoting i's mass as m_i and Q's total mass as m_Q and using the formula in 3D (G is a constant):

$$f_i = Gm_i \left(\frac{x_{cm} - x_i}{r^3}, \frac{y_{cm} - y_i}{r^3}, \frac{z_{cm} - z_i}{r^3} \right)$$

We have the basic components for computing the total force at a particle. Two types of interactions are necessary: a particle-particle interaction (*near-field*) when particles are close together and a particle-node interaction (*far-field*) when a particle is far from a group of planets. For this, we say if $\frac{D}{r} < \theta$, where θ is a preset value, use a node's center of mass and total mass instead of continuing to traverse that subtree for the force calculation.

Instead of gravitational forces, one can obviously use the fundamental solutions to the Laplace and Poisson equations to compute electrostatic potentials.

It can be proved that for appropriate distribution of particles and θ not too small, the running time to compute the centers of mass and then compute the forces is $O(n \log_4 n)$. Barnes-Hut is easy and fast to code, and we have done so for 2D and 3D for the purpose of vortex methods. Unfortunately, controlling the parameters for guaranteeing a certain level of accuracy in Barnes-Hut is not always straightforward (Salmon and Warren, 1994). For more control in error bounds in solutions to n-body problems, the Fast Multipole Method provides a better option.

1.3.2 Fast Multipole Methods

Because it can achieve arbitrary precision at modest cost with straightforward error estimates, we concentrate on the Fast Multipole Method (FMM) in the present setting. The classical FMM is kernel-specific and relies on detailed separation of variables solutions of the governing PDE (Greengard and Rokhlin, 1987), and we review this original approach in more detail in section 2.1.

While the original FMM considered the Laplace equation, the Helmholtz equation was subsequently treated in (Rokhlin, 1990). A three-dimensional version effective for all frequencies (and additional references) can be found in (Cheng et al., 2006a). The modified Helmholtz equation was discussed in (Boschitsch et al., 1999; Greengard and Huang, 2002), and the biharmonic equation in (Greengard et al., 1996; Gumerov and Duraiswami, 2006; Wang et al., 2007). The Stokes equations are somewhat exceptional, since they can be handled by a sequence of calls to the original (Coulomb) FMM (Y. Fu, 2000; Tornberg and Greengard, 2008). An attractive alternative that avoids much of the detailed analytic work of these methods is the kernel-independent approach of (Ying et al., 2003; Ying et al., 2004b; Lashuk et al., 2009). In this approach, expansions in special functions are replaced with equivalent source densities. The result is that the same numerical apparatus can be used for a variety of PDEs, and the user need only supply a subroutine for the evaluation of the relevant Green's function. We review this kernel-independent approach in more detail in section 2.2.

FMM-based solvers have been used for a variety of purposes in two dimensions. In (Greengard and Lee, 1996), FMM methods are developed for high-order accuracy in unbounded space. The authors of (Russo et al., 1994; Strain, 1997) use adaptive FMM grids and triangulation for solving vortex-based fluid problems.

While the bulk of the work on FMMs over the last two decades has concentrated on particle interactions or the acceleration of boundary integral equation methods, there has been some work on solving inhomogeneous PDEs. One option is to couple the FMM with finite difference methods to allow for fast solvers in complex geometry (Mayo, 1985; McKenney et al., 1995; Ying et al., 2004a). While this is a significant improvement in terms of range of applicability over classical fast solvers, these methods require a regular volume mesh on which is superimposed an irregular boundary. Adaptive FMMs for volume source distributions in two dimensions were described in (Cheng et al., 2006b; Ethridge and Greengard, 2001; Greengard and Lee, 1996). The present work extends these two-dimensional schemes to three dimensions, incorporates them into kernel-independent FMMs, and introduces several new performance optimizations. The result is an efficient, adaptive method that is capable of computing volume integrals in three dimensions for a broad variety of PDE kernels. We provide more background on these approaches in section 2.3.

It is also worth noting that there has been a significant body of work in the quantum chemistry community on accelerating volume integral calculations using the FMM, where collections of Gaussians are typically used to describe the charge distribution (White et al., 1994; Strain et al., 1996). These are Poisson problems in free-space but with a different approach to defining the right-hand side.

1.3.3 Particle-In-Cell Methods

We briefly discuss a class of methods known as Particle-In-Cell (PIC), or Particle-Mesh (PM) methods. PIC methods are often seen in simulations of particles that are advected through some domain. A common example includes vortex methods for simulating fluids (Cottet and Koumoutsakos, 2000). Another example is simulating electrostatic fields with particles and Maxwell's equations, using the Biot-Savart Law for the summation of potentials from charge distributions (Gibbon and G.Sutmann, 2002). These methods incorporate an Eulerian method for solving the necessary equations and Lagrangian techniques to advect particles through the domain. The specific differences for different PIC methods are in how mesh values are transferred to particles and how particle values are mapped back to the mesh. Specifically, given density values, q_j for some particle, j, compute the density ρ at some location $\mathbf{x}$ as follows

$$\rho(\mathbf{x}) = \sum_{j=1}^{N_{cell}} q_j W(\mathbf{x} - \mathbf{x}_j),$$

where N_{cell} is the number of cells, and W is a weighting function with compact support. Typically, there are many particles per mesh-grid cell so that the density on the mesh is interpolated smoothly. If the Poisson equation is being solved, a fast FFT-based solver could be employed. Then, use the same function, W, to interpolate the potentials back to the particles. A common weighting function is the Cloud-In-Cell (CIC) function, which in 1D looks like

$$W_{cic}(x) = 1 - |x|, |x| < 1.$$

The effect of this function is that it acts like the hat-function seen in finite elements. In 3D, the CIC function takes the form of $W_{cic}(\mathbf{x}) = W_{cic}(x)W_{cic}(y)W_{cic}(z)$.

One large problem with the PM method is that since they interpolate all particles in a cell to the grid and then back, near-field interactions are all but lost. This technique is good for long-range simulations, but we care about close interactions of particles. One idea is to combine the fact that PM performs well for far-field interactions and employ a different approach for evaluating near-field interactions between particles. One possible solution is the Particle-Particle, Particle-Mesh Method (P^3M) (Gibbon and G.Sutmann, 2002).

The idea behind P^3M is to break the potential for each particle into two parts, just as in the FMM method. That is, for the jth particle, let its total potential be u_j where

$$u_j = u_j^{PP} + u_j^{PM}.$$

The PM-part, u_j^{PM} can be computed as before, and u_j^{PP} is computed as a particle-to-particle summation, where we choose some radius r around u_j to locate particles. There are several points which must be addressed, however. First, if the area of the PP interactions is too large, too many direct summations are computed, which is computationally expensive. On the other hand, if the area is too small, some of the near-field effect is lost, which defeats the purpose of using P^3M. Another problem is that where the PM and PP calculations meet, they have to agree. The matching of the forces can be done during the regular PM calculation (Hockney and Eastwood, 1981; Gibbon and G.Sutmann, 2002).

PIC methods provide an alternative to FMM; however, the P^3M method or some variant is necessary for high-accuracy, and even then, accuracy can be difficult to gauge as choosing the near-field radius for the PP calculations takes some finesse (Hockney and Eastwood, 1981) while FMM guarantees desired accuracy. Further, PIC methods only work for bounded space, working best for periodic domains (Gibbon and G.Sutmann, 2002); whereas, we are interested in solving problems in unbounded space, for which PIC methods are not useful. Further, as stated, PIC can be used for vortex methods, but others have found that fast methods such as adaptive FMM are better for the calculation of force for each particle (Russo et al., 1994; Strain, 1997).

Finally, we state that one method that often sees consideration is Smoothed Particle Hydrodynamics (SPH) (Lucy, 1977; Gingold and Monaghan, 1977; Monaghan, 1992). SPH is a purely Lagrangian formulation, where particles carry all physical properties of a material; this is sometimes used in fluid simulations. Smoothing kernels are used to interpolate potentials or forces and particles are advected as in PIC methods. Each particle is given its own smoothing length, h, in lieu of using a grid, and they are allowed to vary over time. SPH drawbacks include the fact that energy can often be allowed to dissipate, so care needs to be taken in varying the particle lengths, and error estimates are often difficult to calculate. Additionally, the smoothing kernel is designed to calculate the influence only for nearby particles, so in effect, far-field interactions are not taken into account. This can be compensated for by increasing the radius of influence; however, this increases computation time such that SPH methods fail to remain competitive, especially in the face of their numerical deficiencies.

1.4 Embedded Boundary Solvers

We have looked at FFT techniques for problems in simple domains as well as fast summation techniques for problems in unbounded domains. However, neither of these are particularly useful on their own since we want to be able to solve more complicated problems such as those in figure 1. One approach is an Embedded Boundary Integral approach as in (Ying et al., 2004a).

For example, for an interior Dirichlet Poisson problem of the form

$$\begin{aligned} -\Delta u &= f \ \text{ in } \omega \qquad (1.6) \\ u &= g \ \text{ on } \gamma, \end{aligned}$$

such as in figure 1, the idea is to embed ω in a simpler domain and solve two PDEs: one with body forces and a simple domain and one absent body forces (homogeneous) with a complex domain whose boundary values are altered by the first PDE. This approach can be seen in (Mayo, 1984; Mayo, 1985; Mayo and Greenbaum, 1992; McKenney et al., 1995). This *Embedded Boundary Method* approach is discussed in greater detail in Chapter 4.

For the regular-grid solver, (Ying, 2004) uses FFT methods, assuming that the body force is available everywhere. The same is true for (Mayo, 1985; McKenney et al., 1995). This incorporates the homogeneous boundary conditions; however, one could also use volume integrals methods as discussed in Chapters 2 and 4, allowing for adaptivity and non-uniform source distributions in the volume solver.

1.5 Overview of Thesis

For simple geometries and well-known body forces, we have discussed how FFT-based methods perform well. For more complicated geometries, adaptive grid-based techniques such as AMR provide an alternative approach, and is a promising area of research, especially with algebraic multigrid for unstructured grids; these methods, however, require a hierarchy of coarse to fine grids, and these grids can be computationally intense to compute and parallelization is necessary to obtain good time complexity. The embedded boundary integral approach, on the other hand, holds an opportunity to incorporate existing fast summation methods, of which FMM provides the best accuracy and flexibility.

We have modified the kernel-independent FMM to incorporate for non-uniform force distributions in complex domains. Using the Embedded Boundary Integral approach, this will allow for the solving of the Poisson equation in complex smooth geometries in 2D and 3D. For the fast volume solver, we introduce in Chapter 2 a kernel-independent free-space volume solver, using precomputed interaction weights and symmetry techniques. We further discuss how to extend this solver to periodic or Dirichlet boundary conditions in the box. In Chapter 3 we look at numerical results for these volume solvers for a variety of PDEs.

In Chapter 4, we discuss how we incorporate our fast volume solver into an embedded boundary solver technique, including a pre-existing boundary integral method such that we can solve arbitrary elliptic PDEs in complex domains with volume forces giving only in the interior of our domain in a possibly non-homogeneous distribution. Chapter 5 looks at numerical results for several complex geometries and force distributions.

2 FMM-BASED KERNEL-INDEPENDENT 3D VOLUME SOLVER

In free space, linear constant-coefficient PDE solutions can be expressed directly in an integral form; that is, for a given location, the solution of the equation can be evaluated as a convolution of the volume force with a specific Green's function. By taking advantage of smoothness of the kernel functions, Fast Multipole Method (FMM) algorithms (Greengard and Rokhlin, 1987; Beatson and Greengard, 1997; Ethridge and Greengard, 2001) are able to significantly decrease complexity costs. In particular, the kernel-independent FMM (Ying et al., 2003; Ying et al., 2004b) makes it possible to efficiently solve any free-space PDE for which a smooth kernel evaluation function is provided. For example, given a linear, constant-coefficient PDE

$$\mathcal{L}(u)(\mathbf{x}) = g(\mathbf{x}), \tag{2.1}$$

classical mathematical methods can be used to compute the corresponding Green's function $K(\mathbf{x}, \mathbf{y})$ in free space such that

$$u(\mathbf{x}) = \int_{\Omega} K(\mathbf{x}, \mathbf{y}) g(\mathbf{y}) d\mathbf{y}\,, \tag{2.2}$$

where Ω is the support of g. $K(\mathbf{x}, \mathbf{y})$ is in general weakly singular; assuming $g(\mathbf{x})$ is given at N points and $u(\mathbf{x})$ is desired at N points, the non-local character of the integral representation, as indicated above, would lead to an $O(N^2)$ solution procedure. Thus, we need both a suitable quadrature approach and a fast algorithm for (2.2) to yield a useful numerical technique. Assuming this is achieved, a number of advantages follow. First, no linear system needs to be solved; adaptivity is achieved by using adaptive quadrature. Second, as mentioned earlier, first derivatives can be computed without loss of precision (higher-order derivatives are possible but it may be harder to achieve the same degree of accuracy due to increase in the order of singularity that needs to be integrated); the gradient of (2.2) becomes

$$\nabla u(\mathbf{x}) = \int_\Omega \nabla K(\mathbf{x},\mathbf{y}) g(\mathbf{y}) d\mathbf{y}. \tag{2.3}$$

Third, we have simple *a priori* error estimates. Let $\hat{g}(\mathbf{x})$ be the approximation to $g(\mathbf{x})$, and $\hat{\mathbf{Q}}[g](\mathbf{x})$ denote the quadrature approximation of $\int_\Omega K(\mathbf{x},\mathbf{y})g(\mathbf{y})dy$, with an error estimate of the form

$$\left|\hat{\mathbf{Q}}[g](\mathbf{x}) - \int_\Omega K(\mathbf{x},\mathbf{y})g(\mathbf{y})dy\right| \le \epsilon \|g\|_1, \tag{2.4}$$

where $\|g\|_1 = \sum_i |g_i|$ from sampled source locations, and the parameter ϵ in our case is the FMM approximation error estimated as in (Ying et al., 2004b). Let

$$\hat{u}(\mathbf{x}) = \hat{\mathbf{Q}}[\hat{g}](\mathbf{x}). \tag{2.5}$$

Then

$$\begin{aligned} e(\mathbf{x}) = u(\mathbf{x}) - \hat{u}(\mathbf{x}) &= \int_\Omega K(\mathbf{x},\mathbf{y})g(\mathbf{y})d\mathbf{y} - \int_\Omega \hat{\mathbf{Q}}[\hat{g}](\mathbf{x}) \\ &\le \int_\Omega K(\mathbf{x},\mathbf{y})[g(\mathbf{y}) - \hat{g}(\mathbf{y})]d\mathbf{y} + \left|\int_\Omega K(\mathbf{x},\mathbf{y})\hat{g}(\mathbf{y})d\mathbf{y} - \hat{\mathbf{Q}}[\hat{g}](\mathbf{x})\right| \\ &\le C_1 \|g(\mathbf{y}) - \hat{g}(\mathbf{y})\|_\infty + \|\hat{g}(\mathbf{y})\|_1\, \epsilon, \\ &\quad \text{where } C_1 = \max_x \int_\Omega |K(\mathbf{x},\mathbf{y})| d\mathbf{y}. \end{aligned} \tag{2.6}$$

The estimate above is much sharper than one typically obtains when discretizing the PDE itself, where the order of accuracy is determined by high derivatives of the solution. Here, it depends only on the quality of the approximation of the right-hand side. In particular, a k^{th}-order polynomial approximation leads to a k^{th}-order accurate scheme with a very small constant. (C_1 is a bounded quantity determined by the volume of Ω with no dependence on the data.)

Again, the principal drawback is that, when implemented naïvely, the complexity of the approach is quadratic in the number of sample points. FMM algorithms overcome this computational barrier by making systematic use of the smoothness of distant interactions on a hierarchy of spatial scales. The kernel-independent versions of the FMM are particularly useful in their generality; they make it possible to compute solutions of the form (2.2) for any (non-oscillatory) elliptic PDE, provided only a module which evaluates the kernel exists.

After describing the details of the approach, we demonstrate its performance for the Poisson equation (2.7), the modified Helmholtz equation (2.8), and the Stokes equations (2.9):

$$-\Delta u(\mathbf{x}) = g(\mathbf{x}), \tag{2.7}$$

$$\alpha u(\mathbf{x}) - \Delta u(\mathbf{x}) = g(\mathbf{x}),\ \alpha > 0, \text{ and} \tag{2.8}$$

$$\nabla p(\mathbf{x}) - \mu\Delta \mathbf{u}(\mathbf{x}) = \mathbf{g}(\mathbf{x}),\ \nabla\cdot\mathbf{u}(\mathbf{x}) = 0. \tag{2.9}$$

with corresponding kernels in three dimensions given by

$$K(\mathbf{x},\mathbf{y}) = \frac{1}{4\pi r}, \tag{2.10}$$

$$K(\mathbf{x},\mathbf{y}) = \frac{1}{4\pi r}e^{-\sqrt{\alpha}r}, \text{ and} \tag{2.11}$$

$$K(\mathbf{x},\mathbf{y}) = \frac{1}{8\pi\mu}\left(\frac{1}{r}I + \frac{\mathbf{r}\otimes\mathbf{r}}{r^3}\right), \text{ respectively.} \tag{2.12}$$

We use an adaptive piecewise polynomial force approximation, discussed in detail in section 2.4. As the input for our algorithm, we assume a smooth force is given as discrete point sources, which can in turn be approximated locally as polynomial coefficients. This approximation can be constructed from other types of force representations as long as the forces can be evaluated at grid enough locations with sufficient accuracy (Langston et al., 2011).

We begin by reviewing the analytic FMM in section 2.1, the kernel-independent FMM in section 2.2, the motivation for using FMM for a volume solver in section 2.3 for one and two dimensions, and discussion of the kernel-independent volume solver in section 2.4.

2.1 Analytic Fast Multipole Method

To establish terminology and notation, we summarize the structure of the original two-dimensional FMM for the Poisson equation (Greengard and Rokhlin, 1987). Given a force distribution g at N_{src} source locations, we wish to compute the induced potentials u_j at N_{trg} target locations, $\mathbf{x}_j$:

$$u_j = u(\mathbf{x}_j) = \int_{\mathbf{R}^2} K(\mathbf{x}_j, \mathbf{y}) g(\mathbf{y}) d\mathbf{y} \approx \sum_{i=1}^{N_{src}} K(\mathbf{x}_j, \mathbf{y}_i) g(\mathbf{y}_i) w_i \ , \ j = 1, \dots, N_{trg} \, , \qquad (2.13)$$

where $K(\mathbf{x}, \mathbf{y}) = -\log|\{\mathbf{x} - \mathbf{y}\}|/2\pi$ and w_i is a quadrature weight associated with source location $\mathbf{y}_i$. For $N_{src} \approx N_{trg} \approx N$, the FMM decreases the computational cost from $O(N^2)$ to $O(N)$ for a fixed user-prescribed level of accuracy. It does so by introducing a hierarchical quadtree partition of a bounding square D, enclosing all target and source points, and two series expansions for each box at each level of the hierarchy. More precisely, the root of the tree is associated with the square D and referred to as level $\ell = 0$. The boxes (squares) at level $l + 1$ are obtained recursively, subdividing each box at level l into four squares, referred to as its children. For a box B of diameter H, its *near field* $\mathcal{N}^B$ is defined to be the set of all boxes in D contained inside a box centered at B of width $3H$. The *neighbor list* L_N^B of a box B is defined to be the set of boxes sharing a vertex with B that are elements of $\mathcal{N}^B$; in the nonadaptive case, $L_N^B = \mathcal{N}^B$. The *far field* of B, denoted $\mathcal{F}^B$, is the complement of near field: $\mathcal{F}^B = D \setminus \mathcal{N}^B$. Finally, the *interaction list* of box B, denoted by L_I^B, is defined to be the children of B's parent's neighbors that are not neighbors themselves. Thus, $L_I^B \subseteq \mathcal{F}^B$. An example of a uniformly refined 2D domain and quadtree structure is shown in Figure 2.1. The depth of the tree is chosen so that the smallest boxes (leaf nodes in the tree structure) contain no more than some fixed number of points, say s. For simplicity, we first consider uniformly refined trees, where all leaves in the tree structure are at the same level. Let us note that the total number of boxes in the quadtree is bounded by $4N/3s$ ($8N/3s$ in three dimensions). Thus, if the workload per box is constant, then the net algorithm has $O(N)$ complexity.

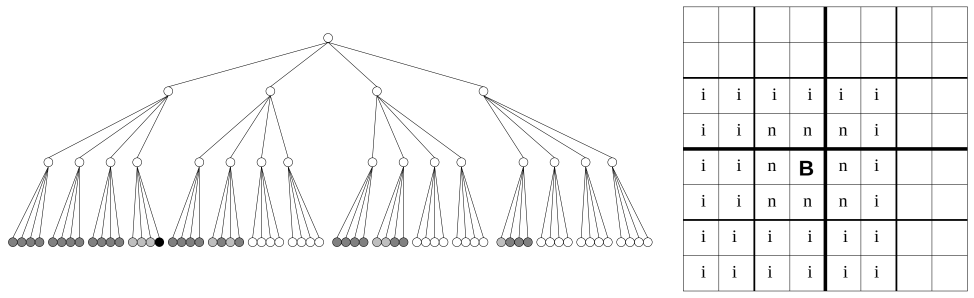


Figure 2.1: On the *right* is shown a domain which has been fully subdivided three times, resulting in a quadtree on the *left*. The root of the tree represents the entire domain at level 0. The non-leaf nodes at level 1 of the quadtree represent the four boxes on the right enclosed by the thickest black lines. The right image shows the neighbor list L_N^B for the box B (boxes marked with *n* on the grid, light gray in the tree), and L_I^B (boxes marked with *i* on the grid, dark gray in the tree). Here, $\mathcal{N}^B = L_N^B$ and $\mathcal{F}^B = (D \cup L_I^B) \setminus \mathcal{N}^B$.

Two types of series are associated with each box B in the hierarchy:

- The *multipole expansion* is defined as follows. Suppose m source densities ϕ_j at z_j are located inside a circle of radius R with center z_0 (i.e., $|z_j - z_0| < R$). For all z outside of the circle with radius r (i.e., $|z - z_0| > r$, $r > R$), approximate the potential $u(z)$ as:

$$u(z) = a_0 \log(z - z_0) + \sum_{k=1}^{p} \frac{a_k}{(z - z_0)^k} + O\left(\frac{R^p}{r^p}\right)$$

where the coefficients, a_k for $0 \leq k \leq p$, satisfy

$$a_0 = \sum_{j=1}^{m} \phi_j \text{ and } a_k = \sum_{j=1}^{p} \frac{-\phi_j (z_j - z_0)^k}{k}.$$

The coefficients, $\{a_k\}$ represent the multipole expansion.

- The *local expansion* is defined as follows. Suppose m source densities ϕ_j at z_j are located outside of the circle of radius r with center z_0 (i.e., $|z_j - z_0| < R$). For all z inside of the circle with radius R ($R < r$), approximate the potential $u(z)$ as:

$$u(z) = \sum_{k=0}^{p} c_k (z - z_0)^k + O\left(\frac{R^p}{r^p}\right)$$

where the coefficients, c_k for $0 \leq k \leq p$, satisfy

$$c_0 = \sum_{j=1}^{m} \phi_j \log{(z_0 - z_j)} \text{ and } c_k = \sum_{j=1}^{m} \frac{-\phi_j}{k(z_j - z_0)^k}.$$

The coefficients, $\{c_k\}$ represent the local expansion. For both expansions, p is chosen for the desired accuracy. As in Barnes-Hut, the domain is subdivided recursively: Imagine a unit square in two dimensions, and we subdivide our space hierarchically using a quadtree structure where leaves of the quadtree contain a maximum number of points, prespecified as some value s (this was set to 1 in the Barnes-Hut examples discussed earlier). For each cell in the quadtree, represent the potential in the box as a result of the sources in that cell via a multipole expansion. The potential in the box from sources in non-adjacent cells is represented as a local expansion.

The FMM computes the total field at a target point in leaf box B as the sum of (a) the field due to the source points contained in the boxes of the neighbor list L_N^B and (b) the contribution from sources in the far field $\mathcal{F}^B$. The contributions from source points inside the boxes of L_N^B are computed directly using (2.13), while the contributions from $\mathcal{F}^B$ are obtained by evaluating the local expansion of box B at the target.

All of the tools exist for an $O(n \log n)$ method, but one can do better using *translation* methods (Beatson and Greengard, 1997) defined as follows.

M2M or Multipole to Multipole Translation turns the multipole expansions of a box's children into its own multipole expansion. Suppose z_S is the center of a box whose parent has center z_M, and that the sequence $\{a_k\}$ represents the multipole expansion at z_S. Write the multipole expansion at z_M as

$$u(z) = b_0 \log(z - z_M) + \sum_{l=1}^{p} \frac{b_l}{(z - z_M)^l} + O(\epsilon),$$

where b_l for $0 \leq l \leq p$ is defined using z_S's multipole expansion,

$$b_0 = a_0 \text{ and } b_l = -\frac{a_0(z_S - z_M)^l}{l} + \sum_{k=1}^{l} a_k (z_S - z_M)^{l-k} \binom{l-1}{k-1}.$$

M2L or Multipole to Local Translation turns the multipole expansions of a box into the local expansion of non-adjacent box. Suppose z_M and z_L are the centers of two boxes which are non-adjacent, and they are at the same refinement level in the hierarchy (i.e, same level in the quadtree). If $\{b_k\}$ is the multipole expansion at z_M, write the local expansion at z_L as

$$u(z) = \sum_{l=0}^{p} c_l (z - z_L)^l + O(\epsilon),$$

where c_l for $0 \leq l \leq p$ is defined using z_M's multipole expansion

$$c_0 = b_0 \log(z_L - z_M) + \sum_{k=1}^{p} \frac{(-1)^k b_k}{(z_M - z_L)^k} \quad \text{, and}$$

$$c_l = \frac{b_0}{l(z_M - z_L)^l} + \frac{1}{(z_M - z_L)^l} \sum_{k=1}^{p} \frac{(-1)^k b_k}{z_M - z_L}^k \binom{l+k-1}{k-1}.$$

L2L or Local to Local Translation turns the expansion expansions of a box's parent into its own local expansion. Suppose that z_T is the center of a box and z_L is the center of its parent and that $\{c_l\}$ represents the local expansion at z_L. Write the local expansion at z_T as

$$u(z) = \sum_{l=0}^{p} d_l (z - z^T)^l + O(\epsilon),$$

where d_l for $0 \leq l \leq p$ is defined using z_T's local expansion

$$d_l = \sum_{k=l}^{p} c_k \binom{k}{l} (z_T - z_L)^{(k-l)}.$$

Figure 2.2 provides an example of moving between expansions for a series of points.

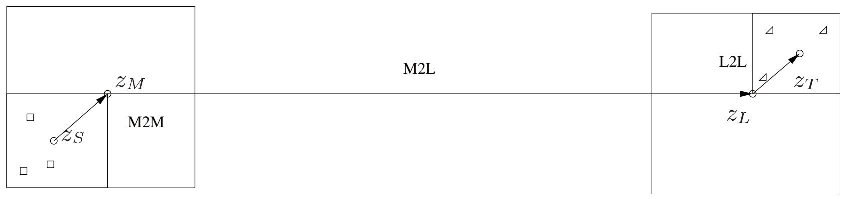


Figure 2.2: Example of expansions and translations. z_S indicates an encoding of the source points in its cell (small squares), and z_T indicates an encoding of the far-field influence to the target points (small triangles). M2M, M2L and L2L indicate the appropriate translations as defined.

In Barnes-Hut, forces were computed by starting at the coarsest point in the quadtree hierarchy and moving towards the coarsest level. The same is true for an $O(nlogn)$ multipole algorithm (Beatson and Greengard, 1997). For the $O(n)$ FMM, start at the finest level and move upwards then downwards. The essential task of the FMM is the *construction* of the local expansions in a hierarchical manner. This takes place in two steps.

The upward pass. This pass begins at the finest level of the tree data structure, converting force values at source points to multipole expansion coefficients for each leaf box; this computation is carried out by the *source-to-multipole* (S2M) operator, a $p \times s_B$ matrix, where s_B is the number of source points in B. The multipole coefficients for coarser level boxes are obtained recursively, mapping coefficients of multipole expansions with respect to children's centers to the multipole expansion with respect to B's center. This map, the *multipole-to-multipole* (M2M) operator, is linear and given by a $p \times p$ matrix for each child.

The downward pass. This pass starts at the coarsest level of the tree. For each box B, the local expansion of the far field is obtained by first shifting the local expansion of B's parent to the center of B. The mapping which carries this out is a $p \times p$ matrix referred to as the *local-to-local* (L2L) translation operator. We then need to add the contributions from the multipole expansions centered at each of the boxes in B's interaction list L_I^B. It is straightforward to check that these contributions are exactly the difference between the far field of B and the far field of B's parent. For each box in L_I^B, one converts its multipole expansion to a local expansion centered in B. This mapping from the vector of multipole coefficients a_k to the vector of local expansion coefficients c_k is referred to as the *multipole-to-local* (M2L) translation operator. It is also linear and given by a $p \times p$ matrix. It is easy to see that the work per box in both upward and downward passes is constant.

At the end of the downward pass, local expansions are available in each leaf node. These can then be evaluated at each target point. We refer to the evaluation of the local potential as the *local to target* (L2T) translation operator; if the number of target points in a box is t_B, then the $L2T$ operator is given by a $p \times t_B$ matrix.

To summarize, the FMM uses $S2M$, $M2M$, $M2L$, $L2L$ and $L2T$ linear operators, each of which is represented by a matrix. For the $M2M$ and $L2L$ operators, each matrix is determined uniquely by the relative position of a box and its parent - there are 4 such matrices for quadtrees and 8 for octrees. For $M2L$ operators, each matrix corresponds to the relative position of a box in the interaction list - there are 27 such matrices for quadtrees and 189 for octrees. These numbers can be considerably reduced by taking advantage of symmetries, a topic we will return to later. The $S2M$ and $L2T$ operators depend on source and target point locations, and can be different for each box.

Algorithm 1 outlines the basic FMM, omitting the technical details. For fixed s and p, the computation is constant-time per box, leading to an $O(N)$ method overall. Figure 2.3 illustrates the data flow involved in the $M2M$, $M2L$ and $L2L$ operators.

Algorithm 1 Non-Adaptive Analytic FMM

```
STEP 1 - CONSTRUCT TREE $T$ AND LISTS
build $T$ such that each leaf $B$ contains at most $s$ points
for each box $B$ in preorder traversal of $T$ do
    build list of nearest neighbors, $L_N^B$ and interaction list, $L_I^B$
end for
STEP 2 - UPWARD PASS
for each box $B$ in postorder traversal of $T$ do
    if $B$ is a leaf box then
        Construct multipole expansion from all sources using $S2M$ operator
    else
        Construct multipole expansion from children using the $M2M$ operator
    end if
end for
STEP 3 - DOWNWARD PASS
for each box $B$ in preorder traversal of $T$ do
    Compute the contribution to $B$'s local expansion from its parent's local expansion using the $L2L$ operator and
    from $L_I^B$ list using the $M2L$ translation operator
end for
for each leaf box $B$ in $T$ do
    Compute the potential at each target location from $B$'s local expansion using the $L2T$ operation and from $L_N^B$
    using direct calculations
end for
```

For the Laplace kernel, $1/r$, in three dimensions, far-field expansions are represented using spherical harmonics (Greengard and Rokhlin, 1988) rather than Laurent series in the analytic FMM. Significant speedups can be obtained by using plane-wave representations as well (Greengard and Rokhlin, 1997).

We next discuss how the work of (Ying et al., 2003; Ying et al., 2004b) adapts the above

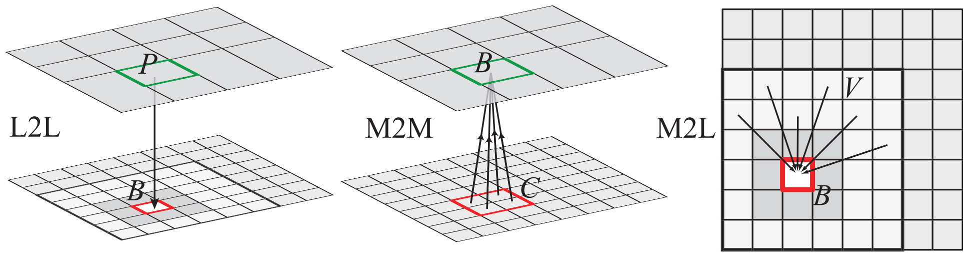


Figure 2.3: Boxes used by M2M, L2L and M2L operators. For box B at level ℓ, P in the $L2L$ operator represents the parent of B at level $\ell - 1$, and in the $M2M$ operator, C represents the children of B at level $\ell + 1$. Boxes labeled V in the $M2L$ operator reside in L_I^B.

algorithm into a kernel-independent FMM, one for which the kernel routine is assumed to be in black-box form only, allowing for multiple kernels to be used without changes to the basic design of the algorithm.

2.2 Kernel-Independent FMM

We provide some background for the original kernel-independent FMM, largely for two dimensions, reserving the majority of the discussion for section 2.4. The kernel-independent FMM (Ying et al., 2004b) is similar to the original FMM, where the main difference is in the representation of the densities, and how the translations (M2M, M2L, L2L) are computed. Instead of using the analytic expansions for potentials from the densities in a box, B, in the quadtree or octree, an equivalent density is computed at a surface enclosing B. This surface is a circle in 2D or a sphere or cube in 3D. For example, using the notation of (Ying et al., 2004b), $\mathbf{y}^{B,u}$ is the upward equivalent surface of B, and $\mathbf{y}^{B,d}$ is the downward equivalent surface of B. $\mathbf{y}^{B,u}$ must be chosen such that it encloses B but does not overlap with B's far-field. B's source densities are represented on the equivalent upward surface by $\phi^{B,u}$. The authors show that the potential generated by the original source densities is equivalent to the potential generated by the upward equivalent density, $\phi^{B,u}$, by using an upward check surface, $\mathbf{x}^{B,u}$ whose potential is called the upward check potential, $q^{B,u}$. The equality of the potentials for the original source densities versus the equivalent densities, assuming the correct construction of these equivalent densities, is shown by the following equation (N_s is the total number of source points in B):

$$\int_{\mathbf{y}}^{B,u} G(\mathbf{x},\mathbf{y})\phi^{B,u} d\mathbf{y} = \sum_{i}^{N=N_s} G(\mathbf{x},\mathbf{y}_i)\phi_i = q^{B,u},\ \mathbf{x} \in \mathbf{x}^{B,u}$$

Equivalent constructions are made for the potential generated from source densities in B's far-field using downward equivalent surfaces and check surfaces. The basic idea is shown in figure 2.4, reproduced from (Ying et al., 2004b).

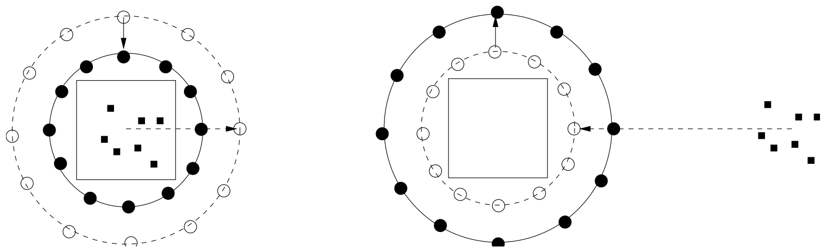


Figure 2.4: The left picture represents the calculation of the potential generated from B's local source densities. First, the potential at the upward check surface, the dashed line, is computed, and then this is used to compute the equivalent densities on the upward equivalent density surface, the solid line. The potential in B from its far-field is generated by using the equivalent densities on the downward equivalent surface, the solid line. First, the potential induced by the far field source densities is computed at the downward check surface, the dashed line, and this potential is used to compute the equivalent downward density.

Using the equivalent densities, the authors are now able to alter the multipole translations to be more efficient, and kernel-independent. The M2M translation from a child box A to its parent B by solving for the equivalent density $\phi^{B,u}$ is

$$\int_{\mathbf{y}^{B,u}} G(\mathbf{x},\mathbf{y})\phi^{B,u}d\mathbf{y} = \int_{\mathbf{y}^{A,u}} G(\mathbf{x},\mathbf{y})\phi^{A,u}d\mathbf{y},\ \mathbf{x} \in \mathbf{x}^{B,u}.$$

For the M2L translation, if A is now a box in the far field of B, we solve for $\phi^{B,d}$,

$$\int_{\mathbf{y}^{B,d}} G(\mathbf{x},\mathbf{y})\phi^{B,d}d\mathbf{y} = \int_{\mathbf{y}^{A,u}} G(\mathbf{x},\mathbf{y})\phi^{A,u}d\mathbf{y},\ \mathbf{x} \in \mathbf{x}^{B,d}.$$

For the L2L translation, if A is now the parent of B, we solve for $\phi^{B,d}$,

$$\int_{\mathbf{y}^{B,d}} G(\mathbf{x},\mathbf{y})\phi^{B,d}d\mathbf{y} = \int_{\mathbf{y}^{A,d}} G(\mathbf{x},\mathbf{y})\phi^{A,d}d\mathbf{y},\ \mathbf{x} \in \mathbf{x}^{B,d}.$$

Stability and accuracy considerations for this method and requirements on the placement of the surfaces are detailed in (Ying, 2004); however, for better understanding, we reproduce figure 2.5 in order to illustrate the basic idea of how the translations work between equivalent surfaces and equivalent check surfaces.

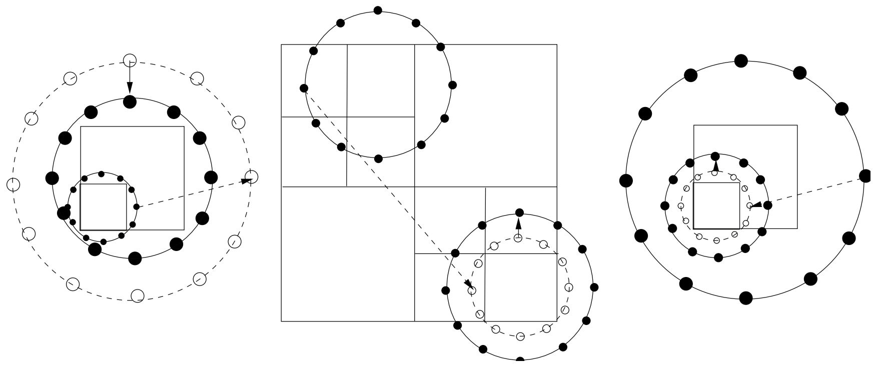


Figure 2.5: Using the same definitions for the solid lines, dashed lines, solid circles and empty circles, the left figure represents a graphical interpretation of the M2M translation, the middle the M2L translation and the right the L2L translations for the kernel independent FMM.

The location of these circles in 2D and boxes in 3D are further discussed in (Ying, 2004), such that appropriate stability is achieved in the potential evaluation and solving of the necessary integral equation.

As before, N_B is the set of all of B's neighbors, L_N^B is the list of neighbors of B, adjacent boxes to B at the same refinement level such that $L_N^B \subset N^B$, and L_I^B is the interaction list, children of B's parent's neighbors which are not in N_B; $L_I^B \subset F^B$, the far-field of B. For a regular distribution of particles, this is enough to build a recursive quadtree or octree until each box B contains s or fewer points of evaluation.

The flow of the algorithm remains the same as in Algorithm 1. First, the tree construction is done in the normal fashion, and for each box, B, in the preorder traversal of the tree, B is subdivided if it has more than s points. Then, another preorder traversal allows us to build the th necessary neighbor and interaction lists. Second, an upward pass is performed using M2M translations. Once the upward equivalent density is known for each box B in the tree, the downward pass is performed. Near-field interactions are taken care of separately.

In 2D, the SVD decomposition is used to accelerate the solving of the integral equation resulting from solving for the equivalent density at the equivalent surface, where the potential is known at the check surface. This operator is in fact precomputable. In 3D as boxes are used for the equivalent and check surfaces, the authors use an FFT-based acceleration which works well with the kernel-independent representations.

Parallelization issues are an additional concern in FMM implementations. Often one processor is inadequate for storing all data if there is a large number of particles; hence, data partitioning must be efficient. To maintain a consistent tree structure, communication needs must be accounted for, and upward and downward passes can result in data needing to be synchronized between processors. In (Ying, 2004), each processor ignores all other processors for computation purposes. Then, an upward equivalent densities are synchronized, so no synchronization occurs at computation time. (Lashuk et al., 2009) updates the previous parallelization work, providing multiple enhancements using GPUs for accelerating near-field interactions and prioritizing matrix operators by type to achieve significant speeds on multiple architectures.

We reserve the discussion for the additional lists and operators needed for non-uniform distributions of forces or points, resulting in possible adaptive trees, for section 2.4. Further, we provide more detail on the nature of the equivalent density representations there as well as the operators involved.

2.3 Motivation for an FMM Volume Solver

In order to adapt existing FMM structures from particle to volume solvers, we need to discuss the concepts of *precomputation tables* and *level-restricted* trees. To motivate the discussion of precomputed tables, we begin with a one-dimensional example, then move on to a two-dimensional version from (Ethridge and Greengard, 2001) for additional motivation and discussion of level-restricted trees. We follow in the next section with changes introduced for three dimensions and kernel independence.

1D Example

We motivate the discussion with an example. We imagine that we have $\Omega \in \mathcal{R}^2$, and a continuous distribution of body force sources along the diagonal line. That is, if $\Omega = [-0.5, 0.5]$, we have defined sources at the set of points $\{(x, y)|x = y$, and $x, y \in [-0.5, 0.5]\}$. As we only know our force f along a single line, we project this line of forces to one dimension or $\mathcal{R}$ as in figure 2.6.

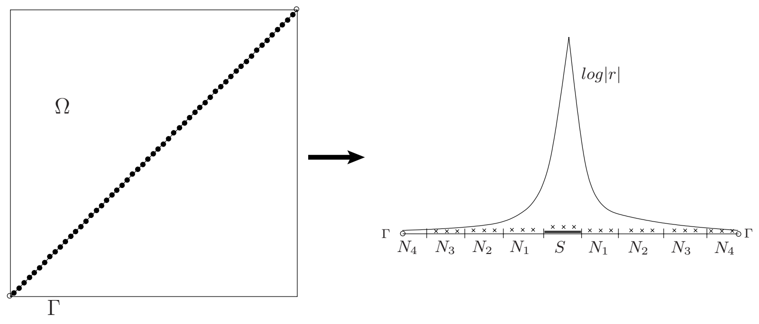


Figure 2.6: On the left we see a regular distribution of particles which we project from 2D into 1D. In 1D, we see that for the evaluation points, marked as X's, they are regularly spaced. For some domain S, contributions from $N(S)$ can have weights precomputed and stored. Every interval has the opportunity to be S, so near-field contribution weights are always stored. Given symmetry, we can save even fewer precomputations.

Imagine now that we discretize this line of forces into disjoint intervals, N_t, of equal size. If we are trying to calculate the portion of the potential at a specific point x_j specifically due to the forces in the interval S (we assume that x_j can be in any interval, including S), then we can write this contribution to $u(x_j)$'s potential as

$$u(x_j)_S = \int\limits_S G(r) f ds(y). \tag{2.14}$$

Here, $r = |x_j - y|$, $y \in S$ and we let our kernel be represented as $G(r) = G(|\mathbf{x} - \mathbf{y}|)$ where we have $G(r) = -\frac{1}{2\pi} log(r)$ in 2D, $G(r) = \frac{1}{4\pi r}$ in 3D. Additionally, let f in S be f_s. As we assumed that f_s was continuously known on each interval, then we know f_s at a discrete set of points, $y_0, y_2, ..., y_{m-1} \in S$. For example, we could choose to represent S at distinct points as in figure 2.6. Then, we can construct a polynomial approximation to f_s as

$$f_s(y) = \sum_{i=0}^{m-1} c_i y^i. \tag{2.15}$$

Equation 2.14 becomes

$$\begin{aligned} u(x_j)_S &\approx \sum_{i=0}^{m-1} c_i \int\limits_S G(r) y^i ds(y) \\ &= \sum_{i=0}^{m-1} c_i \sum_{k=0}^{m-1} G(r) y_k^i. \end{aligned}$$

As we know f_s at all m points $y_k \in S$, we can solve for all of the coefficients c_i in equation 2.15 in a least-squares fashion (Ethridge and Greengard, 2001). This system, unfortunately, may exhibit oscillatory behavior, so it is better to reduce the degree of the polynomial, or increase the number of evaluation points while keeping the polynomial degree constant. That is, keep the m y_k points the same, and decrease the polynomial approximation degree to n, $n < m$ such that our system becomes

$$\begin{bmatrix} 1 & y_0 & y_0^2 & \dots & y_0^{n-1} \\ 1 & y_1 & y_1^2 & \dots & y_1^{n-1} \\ 1 & y_2 & y_2^2 & \dots & y_2^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & y_{m-1} & y_{m-1}^2 & \dots & y_{m-1}^{m-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{m-1} \end{bmatrix}$$

This makes it easier to solve for $\mathbf{c}$ smoothly, using a least-squares SVD decomposition for example. Our system to solve becomes

$$u(x_j)_S \approx \sum_{i=0}^{(n-1)<(m-1)} c_i \sum_{k=0}^{m-1} G(r) y_k^i \tag{2.16}$$

We now wish to evaluate the potential due to interval S, at three distinct points in each interval. We indicate these points with an "X" in figure 2.6. As each interval is of equivalent size, then each evaluation point is equally spaced.

We make one additional definition. Let $N(N_t)$ be the set of intervals directly adjacent to interval N_t. For example, $N(S)$ is the set of intervals to the left and right of S. As these intervals are of equivalent size, evaluating the distance between a point in S and a point in the interval directly to the left of S has a direct analog in the interval to the right. Hence, we can enumerate the intervals in relation to S as in figure 2.6. The set $N(S)$ technically has two intervals as members, but since these two intervals are equivalently distanced, we say simply that $N(S) = N_1$.

For a point $x_j \in \Omega$, we now use the polynomial approximation of f_s to show that the contribution from S to the potential at x_j is:

$$u(x_j)_S \approx \sum_{i=0}^{n-1} c_i \sum_{k=0}^{m-1} G(r) y_k^i \tag{2.17}$$

For the two intervals in $N(S)$, if we imagine there are three target points in each interval, $G(r)$ can be can be precomputed and stored for the six necessary point locations. Since we are approximating f_s at m points, we can precompute $6mn$ values of the form $G(r)y_k^i$. As all intervals are of the same size as S, these precomputed weights can be used for every interval when it gets the chance to be the source interval S for the contribution to its neighbors. In fact, since $N(S)$ has two intervals of equal size, we can exploit the symmetry to precompute only $3mn$ values. If we let $F(x_j, i)$ be the precomputed weight summation $\sum_{k=0}^{m-1} G(r)y_k^i$ for a point $x_j \in S$, equation 2.17 becomes

$$u(x_j)_S \approx \sum_{i=0}^{n-1} c_i F(x_j, i) \tag{2.18}$$

Additionally, for any $x_j \in S$, precompute the contribution to its potential from its own interval.

For stability purposes in the computation of the polynomial approximation, it is better to define a function $\beta_i(y - y_C)$ where y_C is the center of the interval S and β_i is the basis function $(y - y_C)^i$, so we compute the polynomial approximation based on the relative distance from the center of the interval. This is also more sensible for use in fast summation codes, such as FMM, where we are computing contributions from the center of cells. So, for any point $x_j \in \Omega$, we compute the contribution from interval S as

$$u(x_j)_S \approx \sum_{i=0}^{n-1} c_i \sum_{k=0}^{m-1} G(r)\beta_i(y_C - y_k) \tag{2.19}$$

Breaking down equation 2.19 into three situations. For every discrete point, x_j, we calculate the contribution from its own interval, near-field intervals and far-field intervals.

$$\begin{aligned} &\text{if } x_j \in S && \Rightarrow \text{Use precomputed weights as in equation 2.18} \\ &\text{if } x_j \in N(S) && \Rightarrow \text{Use precomputed weights as in equation 2.18} \\ &\text{if } x_j \notin \{S, N(S)\} && \Rightarrow \text{Compute the full summation} \end{aligned}$$

For the last step, we can actually use multipole expansions and the FMM to compute far-field computations, which we address further in the next section.

2D Motivation

The method in 2D is analogous to the 1D example above. This discussion follows the approach in (Ethridge and Greengard, 2001). We will only discuss the case of a non-adaptive force distribution resulting in a fully balanced quadtree. Imagine we have subdivided a domain Ω into equally sized boxes. Then, assume the force f is given on a $k \times k$ grid at a leaf box B for $k = 4$. Then, a 4^{th} order polynomial approximation to f centered at (x_{B_c}, y_{B_c}), the center of B is

$$f_B(x, y) = \sum_{i=1}^{N_k} c_i \gamma_i(\hat{x}, \hat{y}) \tag{2.20}$$

Let $\hat{x} = (x_{B_c} - x)$ and $\hat{y} = (y_{B_c} - y)$ and $N_k = \frac{k(k+1)}{2} = 10$. The basis functions, γ_i, $i = 1, ..., N_k$ are given as the set $\{x^s y^t | s, t \geq 0, (s + t) \leq (k - 1) = 3\}$. Equation 2.20 is overdetermined, so again $\mathbf{c}$ can be solved for in a least-squares fashion. For a point $\mathbf{x} \in N(B)$, the near-field neighbor-list (the set of boxes sharing a boundary with B), we can compute the potential contribution from B as

$$\begin{aligned}
u_B(\mathbf{x}) &= \int\limits_B G(r) f_B(\mathbf{y}) d\mathbf{y} \\
&= \sum_{i=1}^{N_k} c_i \int\limits_B G(r)\gamma_i(\mathbf{y}) d\mathbf{y} \\
&= \sum_{i=1}^{10} c_i F(\mathbf{x}, i)
\end{aligned}$$

The value $F(\mathbf{x}, i)$ represents a precomputable value since $\mathbf{x}$ is assumed to be on a regular grid. So, the total number of values that need to be precomputed are $k^2 N_k = 160$ values for each possible neighbor, where there are 9 possible neighbors. However, due to symmetry, horizontal and vertical neighbors can be stored the same way, as can vertical neighbors.

The advantages to this approach are clear, allowing savings on computation time for near-field computations; however, we have assumed that the quadtree is non-adaptive. For an adaptive tree (such as figure 2.9), this is not possible since near-field neighbors of B can be of any size. In order to fix this, require the following restriction: two leaf nodes which are neighbors must be no more than one refinement level away. That is, if B is at level l, its neighbors must be at level l, $l-1$, or $l+1$. A straightforward method for taking an existing quadtree and fixing it to meet this restriction is available in (Ethridge, 2000) and further outlined here in section A.1. We refer to this as our *level-restricted* tree requirement. Such restrictions are not uncommon, and while we only discuss a straightforward sequential approach, (Sundar et al., 2008) represents the state-of-the-art for parallel balancing.

Once a tree T meets the tree-level restriction, we can precompute the $F(\mathbf{x}, i)$ values more easily. There are 9 possible neighbors of B of the same size, 12 neighbors one level higher, and 12 one level lower. So, for a fourth-order polynomial approximation to f_B, we will have to precompute $160(9 + 12 + 12)$ values for a box B of size H. Again, this seems unrealistic since

B can be of any size. However, these values can be computed on the assumption that B has unit size, centered at the origin. We reproduce the following Lemma from (Ethridge, 2000):

Theorem 2.1. *Let $\tilde{B} = \left[-\frac{1}{2}, \frac{1}{2}\right]^2$. If B is a leaf box at refinement level l and $\mathbf{x}$ is a point being evaluated in $N(B)$, let $\tilde{\mathbf{x}} = 2^{l-1}(\mathbf{x} - \mathbf{x}_{B_c})$. That is, $\tilde{\mathbf{x}}$ is the target evaluation point, $\mathbf{x}$, scaled to the unit square, centered at the origin, $\tilde{B}$. Two new forces for the scaled point are evaluated as*

$$\tilde{f}(\tilde{\mathbf{x}}, i) = \int\limits_{\tilde{B}} \gamma_i(\mathbf{x}) G(|\tilde{\mathbf{x}} - \mathbf{x}|) d\mathbf{x}$$

$$\beta(i, l) = 2^{(d+2)(1-l)} \int\limits_{\tilde{B}} \gamma_i(\mathbf{x}) G(2^{(1-l)}) d\mathbf{x}$$

The variable d represents the degree of the basis function γ_i. Then, for the precomputable value, $F(\mathbf{x}, i)$ becomes

$$F(\mathbf{x}, i) = 2^{(d+2)(1-l)} \tilde{f}(\tilde{\mathbf{x}}, i) + \beta(i, l) \tag{2.21}$$

Hence, weights can be precomputed based on the unit box, $\tilde{B}$ and stored. Equation 2.21 then gives the relationship necessary for adjusting the values for the box B. For far-field interactions, multipole expansions are computed using (2.22); this allows the encoding of volume information from the smooth force distribution, as opposed to singular sources. The local expansion approach remains the same as before in section 2.1.

- A *multipole expansion* about $\mathbf{z}_B$ that represents the influence of sources *inside* B on boxes in the far field $\mathcal{F}^B$ can now be expressed as the real part of a complex Laurent series

  $$u^B_{far}(\mathbf{x}) = \mathcal{R}\left[a_0 \log(x_1 + ix_2 - \mathbf{z}_B) + \sum_{k=1}^{p} \frac{a_k}{(x_1 + ix_2 - \mathbf{z}_B)^k}\right],$$

  where the moments of this expansion are computed from the source distribution as

  $$a_0 = -\frac{1}{2\pi} \int_B f(\mathbf{y}) d\mathbf{y}\,, \quad a_k = -\frac{1}{2\pi} \int_B \frac{f(\mathbf{y})((y_1 + iy_2) - \mathbf{z}_B)^k}{k}. \tag{2.22}$$

The error in the multipole expansion is also of the order $O(\frac{1}{2})^p$.

Adapting this approach to three dimensions, we employ the kernel-independent FMM algorithm as discussed in the next section.

2.4 3D Kernel-Independent FMM Volume Integral Solver

Our algorithm follows the overall structure of the FMM algorithms described above but incorporates changes for three dimensions and kernel independence. Additionally, as the number and size of the operators and precomputed tables grows so rapidly, it is necessary to investigate *symmetries* for computational and storage complexity savings in section 2.5

Given an octree T for our 3D bounding domain D, let $D = \sum\{B_i\}$, $i = 1 \ldots M$ be the set of leaf boxes resulting from hierarchical subdivision. For a single-layer kernel K, we compute the integral (2.2) at some point $\mathbf{x}$ as

$$u(\mathbf{x}) = \sum_{i=1}^{M} K[B_i, g^{B_i}](\mathbf{x}), \tag{2.23}$$

where $K[B, g^B](\mathbf{x}) = \int_B K(\mathbf{x}, \mathbf{y})g(\mathbf{y})d\mathbf{y}$, and g^B represents the restriction of the source distribution to the box B. As in the analytic FMM, the contributions to $u(\mathbf{x})$ from boxes B_i nearest to $\mathbf{x}$ are calculated directly as near-field computations while all other contributions are calculated using the $S2M$, $M2M$, $M2L$, $L2L$ and $L2T$ translation operators in the upward and downward passes of the FMM algorithm.

The principal difference between the approach here and the analytic FMM for point sources is that we use the *sampled equivalent densities*, introduced in section 2.2, instead of classical special functions and series expansions to account for far-field interactions. This requires only a *black-box* kernel evaluation routine and allows for a kernel-independent implementation. A second difference between the current approach and prior kernel-independent FMM schemes is that we are dealing with a continuous source distribution rather than a collection of point-like particles. To extend the method of (Ying et al., 2003; Ying et al., 2004b) to this setting, we use polynomial basis functions to approximate the source distribution g on each leaf box, following the two-dimensional approach of (Greengard and Lee, 1996; Ethridge and Greengard, 2001; Cheng et al., 2006b). More precisely, we assume that the input source is given on each leaf box B by a polynomial g^B of degree $k+1$ with coefficients γ^B,

$$g^B = \sum_{j=1}^{N_k} \gamma_j^B \beta_j \left(2^\ell (\mathbf{x} - \mathbf{c}_B) \right), \tag{2.24}$$

where β_j are polynomial basis functions, ℓ is the depth of the box B ($\ell = 0$ at the root of T), and $\mathbf{c}_B$ is its center. We use monomials for low-order accuracy and tensor-product Chebyshev polynomials for higher-order accuracy. The number of coefficients is $N_k = k(k+1)(k+2)/6$ for each scalar source function g. We describe an interpolation scheme to convert a set of source values defined on a grid of sample points to a polynomial representation in Section 2.4.6. As output, our algorithm can return either point values of the potential at each target point or a polynomial approximation of the potential on each leaf box (which can then be evaluated at arbitrary locations).

To simplify the exposition, we present our algorithm first for a uniformly refined octree of depth ℓ and then discuss the changes necessary for the adaptive octree case separately. The final algorithmic steps are outlined in section 2.4.8, and we briefly discuss how the major loops are optimized for shared-memory parallelization in section 3.3. We begin by explaining our use of equivalent density representations for g^B and γ^B.

2.4.1 Equivalent Densities

The kernel-independent approach to translation operators is based on the following idea, expanded upon from section 2.2. For kernel K, suppose we have an arbitrary (smooth or non-smooth) source distribution g_s in a volume Ω_s with surface Γ_s. Let Γ_t denote an auxiliary surface in the exterior of Γ_s, and let Γ_{check} denote yet another auxiliary surface in the exterior of Γ_t. Finally, let E denote the exterior of Γ_{check}. We will compute a charge density ϕ_t on Γ_t such that the potentials $K[\Omega_s, g_s]$ and $K[\Gamma_t, \phi_t]$ coincide in E. This is always possible if the exterior Dirichlet problem on Γ_t has a unique solution and the exterior field can be represented in terms of a single layer potential[1]

Remark 2.1. *For some problems, such as the Helmholtz equation, a combination of single and double layer sources may be required because of non-physical resonances in the single layer representation, but it is generally sufficient for non-oscillatory kernels (cf. (Kress, 1999) for the Poisson equation, (Ladyzhenskaya, 1964) for the Stokes equations).*

Our goal is to use $K[\Gamma_t, \phi_t]$ to represent the far-field instead of a multipole expansion. For this, we let Γ_{check} approximate the outer boundary of the neighbor list L_N^B and solve a Fredholm integral equation of the first kind for ϕ_t,

$$K[\Gamma_t, \phi_t](\mathbf{x}) = K[\Omega_s, g_s](\mathbf{x}), \quad \text{for all } \mathbf{x} \in \Gamma_{check}. \tag{2.25}$$

Having matched the field on Γ_{check}, the fields will match in the exterior E (with precise estimates depending on the specific kernel). We refer to Γ_t as an *equivalent surface* with *equivalent density* ϕ_t, and Γ_{check} as a *check surface*. In the case when the original density is concentrated on the surface Γ_s, then (2.25) can be written as

$$K[\Gamma_t, \phi_t](\mathbf{x}) = K[\Gamma_s, \phi_s](\mathbf{x}), \quad \text{for all } \mathbf{x} \in \mathbf{x}_t. \tag{2.26}$$

Equations (2.25) and (2.26) form the foundation for the derivation of our translation operators.

Just as the equivalent densities will be used to replace multipole expansions in the FMM, we match the field created by charges *outside* the near neighbors of a box by a discretized layer

[1]For some kernels (Stokes), a low-dimensional nullspace may need to be eliminated.

potential defined on a surface enclosing the box, and a different equivalent density will be used to replace the local expansion. The number of samples used to represent the equivalent density is the analog of the number of expansion terms in a classical FMM. Γ_t and Γ_{check} are cubic surfaces, uniformly sampled at p locations. For a requested FMM precision, $\epsilon_{fmm} = 10^{-n_p}$, $p = n_p^3 - (n_p - 2)^3$ (the nature of these points is discussed in section 2.4.2).

In discretized form, (2.26) can be written as

$$\mathbf{K}^{\Gamma_t, \mathbf{x}_t} \phi_t = \mathbf{K}^{\Gamma_s, \mathbf{x}_t} \phi_s, \tag{2.27}$$

where ϕ_s and ϕ_t are vectors of point-sampled densities, and $\mathbf{K}^{a,b}$ are matrices with entries given by $\mathbf{K}^{a,b}_{ij} = K(a_i, b_j)$ for sample points a_i and a_j on surfaces a and b. For known ϕ_s and solving for ϕ_t, (2.27) is a discretization of a Fredholm equation of the first kind. For large p, linear systems may be poorly conditioned; in such cases, we choose to utilize Tikhonov regularization methods (Kress, 1999) to invert $\mathbf{K}^{\Gamma_t, \mathbf{x}_t}$. Tikhonov regularization replaces $(\mathbf{K})^{-1}$ with $(\alpha I + (\mathbf{K})^* \mathbf{K})^{-1} \mathbf{K}^*$, where the regularization parameter α is chosen to minimize the error in matrix inversion. We discuss this approach and its accuracy in sections 3.2.1 and 3.2.3.

Kernel invariance and matrix precomputation. For all equations we consider, the kernels are invariant with respect to rigid transformations: for scalar kernels, $K(T\mathbf{x}, T\mathbf{y}) = K(\mathbf{x}, \mathbf{y})$ for any rigid transformation T, and for matrix kernels such as those used for the Stokes equations, $K(T\mathbf{x}, T\mathbf{y}) = TK(\mathbf{x}, \mathbf{y})T^T$. Hence, all matrices $\mathbf{K}$ need to be computed only once for each class of pairs of equivalent surfaces, closed with respect to rigid transformations. As we define equivalent surfaces relative to boxes, these classes typically correspond to adjacency relationships between boxes. Furthermore, many (but not all) kernels are *homogeneous*: for any positive c, $K(c\mathbf{x}, c\mathbf{y}) = c^r K(\mathbf{x}, \mathbf{y})$ for some $r \neq 0$. We will refer to r as a *scaling exponent*. In such cases, the number of classes of equivalent surface pairs requiring separate matrices can be further reduced. Similarly, for $S2M$ and near-field calculations, kernel invariance can be used to precompute translation coefficients.

We consider optimizations due to invariance for each translation operator in the next sections. To simplify formulas, we assume scalar kernels in our presentation, although our implementation can handle matrix kernels.

2.4.2 Upward Pass

For the upward pass, we define the source-to-multipole and multipole-to-multipole operators. The analog of sources in the analytical multipole algorithm in our case are polynomials approximating the force on a leaf box. The analog of multipole expansions are *upward equivalent densities*. For consistency with analytic FMM, we use S and M in operator names to denote these quantities.

Source to Multipole (S2M) translations. For each leaf box B, we choose $\mathbf{y}^{B,u}$, the *upward equivalent surface*, to be a box of radius $(1+\delta)r$ and $\mathbf{x}^{B,u}$, the *upward check surface*, to be a box of radius $(3-2\delta)r$. Both surfaces are centered at $\mathbf{c}_B$, the center of B and aligned with B; δ is chosen to satisfy $0 \leq \delta \leq \frac{2}{3}$. By choosing δ to be small, $\mathbf{x}^{B,u}$ and $\mathbf{y}^{B,u}$ are well-separated, ensuring smooth kernels for evaluating equivalent densities (Ying et al., 2004b) (we use $\delta = 0.1$ in practice). Equation (2.25) for upward equivalent density $\phi^{B,u}$ in this case becomes

$$K[\mathbf{y}^{B,u}, \phi^{B,u}](\mathbf{x}) = K[B, g^B](\mathbf{x}), \quad \text{for all } \mathbf{x} \in \mathbf{x}^{B,u}. \tag{2.28}$$

With polynomial coefficients γ^B of g^B, the right-hand side of (2.28) is approximated by

$$K[B, g^B](\mathbf{x}) \approx \sum_{j=1}^{N_k} \gamma_j^B F_j^B(\mathbf{x}), \quad \text{where} \tag{2.29}$$

$$F_j^B(\mathbf{x}) = \int_B \beta_j \left(2^\ell(\mathbf{y} - \mathbf{c}_B)\right) K(\mathbf{x}, \mathbf{y}) d\mathbf{y}, \quad \text{for } \mathbf{x} \in \mathbf{x}^{B,u}, \tag{2.30}$$

and γ_j^B is the j^{th} coefficient of γ^B, β_j is the jth of N_k basis polynomials.

By translation invariance, $F_j^B(\mathbf{x})$ depends only on the choice of β_j and tree level ℓ of B. To evaluate the integrals in expressions for $F_j^B(\mathbf{x}_i)$, we use adaptive Gaussian quadrature (Berntsen et al., 1991). The Nyström discretization (see sections A.3.5 and A.3.6 for more details) of (2.28) at p sample points on $\mathbf{y}^{B,u}$ yields

$$K[\mathbf{y}^{B,u}, \phi^{B,u}](\mathbf{x}_i) = \sum_{j=1}^{N_k} F_j^B(\mathbf{x}_i)\gamma_j^B, \quad \text{for } \mathbf{x}_i \in \mathbf{x}^{B,u}, \tag{2.31}$$

or in matrix form

$$\mathbf{K}_{S2M}^B \boldsymbol{\phi}^{B,u} = \mathbf{F}_{S2M}^B \gamma^B, \tag{2.32}$$

where $\boldsymbol{\phi}^{B,u}$ is the vector of samples of equivalent density of size p, $\mathbf{F}_{S2M}^B$ represents the matrix of precomputed weights $F_j^B(\mathbf{x}_i)$ of size $p \times N_k$, and $\mathbf{K}_{S2M}^B$ is the matrix with entries $K(\mathbf{x}_i, \mathbf{y}_j)$, $i = 1 \ldots p$, $j = 1 \ldots p$. Solving for $\boldsymbol{\phi}^{B,u}$,

$$\boldsymbol{\phi}^{B,u} = (\mathbf{K}_{S2M}^B)^{-1} \mathbf{F}_{S2M}^B \gamma^B = \mathbf{T}_{S2M}^B \gamma^B. \tag{2.33}$$

Since for a uniformly refined tree all leaves are at the same level, the matrix $\mathbf{T}_{S2M}^B$ depends only on the level ℓ due to translation invariance, so only one matrix is computed. Figure 2.7(a) illustrates the computation of $\phi^{B,u}$ from γ^B.

Multipole to Multipole (M2M) translations. M2M translation operators convert the sampled equivalent density representation of a field at a child box C to a sampled equivalent density for the parent box B, shown in figure 2.7(b). The upward equivalent surfaces $\mathbf{y}^{C,u}$, $\mathbf{y}^{B,u}$, and the upward check surface $\mathbf{x}^{B,u}$ are defined in the same way as for $S2M$ translations. For child C of B, we use (2.26) with $\phi_s = \phi^{C,u}$, $\phi_t = \phi^{B,u}$

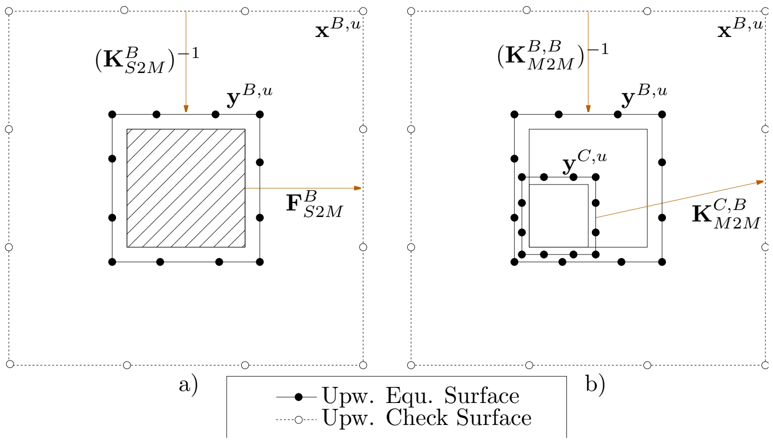


Figure 2.7: $S2M$ and $M2M$ kernel-independent FMM translation operators. a) $S2M$: given a polynomial approximation, γ_B, to a smooth force, g^B inside of a leaf box, B (γ may be computed from grid point locations as indicated by 'x'), using the $S2M$ translation operators and precomputed quadrature weights, an upward check potential, $u^{B,u}$ is computed at the upward check surface, $\mathbf{x}^{B,u}$ (dashed lines). This potential is then used to compute an upward equivalent coefficient density, $\phi^{B,u}$ at the upward equivalent surface, $\mathbf{y}^{B,u}$ (solid lines), by solving a linear system of equations; b) $M2M$: for a non-leaf box, B and children A, we use the $M2M$ translation operator to compute an upward check potential, $u^{B,u}$, at the upward check surface $\mathbf{x}^{B,u}$ (dashed lines), resulting from A's upward equivalent coefficient density, $\phi^{A,u}$. This potential is then used to compute an upward equivalent coefficient density, $\phi^{B,u}$ at the upward equivalent surface, $\mathbf{y}^{B,u}$ (solid lines), just as in the $S2M$ computation stage.

$$K[\mathbf{y}^{B,u}, \phi^{B,u}](\mathbf{x}) = \sum_{C} K[\mathbf{y}^{C,u}, \phi^{C,u}](\mathbf{x}), \quad \text{for all } \mathbf{x} \in \mathbf{x}^{B,u}, \tag{2.34}$$

leading to the discretized equation for B at level ℓ in T

$$\mathbf{K}_{M2M}^{B,B}\boldsymbol{\phi}^{B,u} = \sum_{C} \mathbf{K}_{M2M}^{C,B}\phi^{C,u}.$$

Similar to the $S2M$ computations, these systems are solved as

$$\boldsymbol{\phi}^{B,u} = \sum_{C} (\mathbf{K}_{M2M}^{B,B})^{-1}\mathbf{K}_{M2M}^{C,B}\boldsymbol{\phi}^{C,u} = \sum_{C} \mathbf{T}_{M2M}^{C,B}\phi^{C,u}. \tag{2.35}$$

For any two children C_1 and C_2, there is a rotation R mapping C_1 to C_2; therefore, for kernels invariant with respect to translations and rotations, only *one* matrix $\mathbf{T}_{M2M}^{C,B}$ needs to be computed per level, with the contribution to $\boldsymbol{\phi}^{B,u}$ from any other child obtained by composing this matrix with an appropriate permutation of $\phi^{C,u}$.

For *homogeneous* kernels, only one matrix needs to be stored at a single level ℓ. Indeed, for a box B at depth ℓ, the matrix $\mathbf{K}_{M2M}^{B,B}$ has entries $w_j K(\mathbf{y}_i, \mathbf{x}_j)$, where w_j is the quadrature weight of sample at $\mathbf{x}_j$. If the scaling factor for a matrix $\mathbf{K}$ is r, $K(\mathbf{y}_i, \mathbf{x}_j) = (1/2)^{r\ell} K(\mathbf{y}_i^*, \mathbf{x}_j^*)$ where $\mathbf{y}_i^*$ and $\mathbf{x}_j^*$ are samples corresponding to $\mathbf{y}_i$ and $\mathbf{x}_j$ based on a normalized box $B^* = [-1,1]^*$ at level $\ell = 0$. The quadrature weights scale as area; that is, they need to be multiplied by $1/2^{2\ell}$ such that matrix entries scale as $1/2^{(r+2)\ell}$. As $\mathbf{K}_{M2M}^{B,B}$ and $\mathbf{K}_{M2M}^{C,B}$ scale in the same way, the normalized matrix $\mathbf{T}_{M2M}^{C,B^*}$ does not depend on scale.

For *inhomogeneous kernels*, at most one matrix per level needs to be stored.

2.4.3 Downward Pass

In the downward pass, we compute the analog of local expansions in the analytic FMM, the *downward equivalent densities*. We define the kernel-independent versions of $M2L$ operators (for boxes in the interaction list L_I^B), $L2L$ operators (for translating a parent's local expansion), and $L2T$ operators for final evaluation at target locations. These translation operators are illustrated in Figure 2.8(c-e).

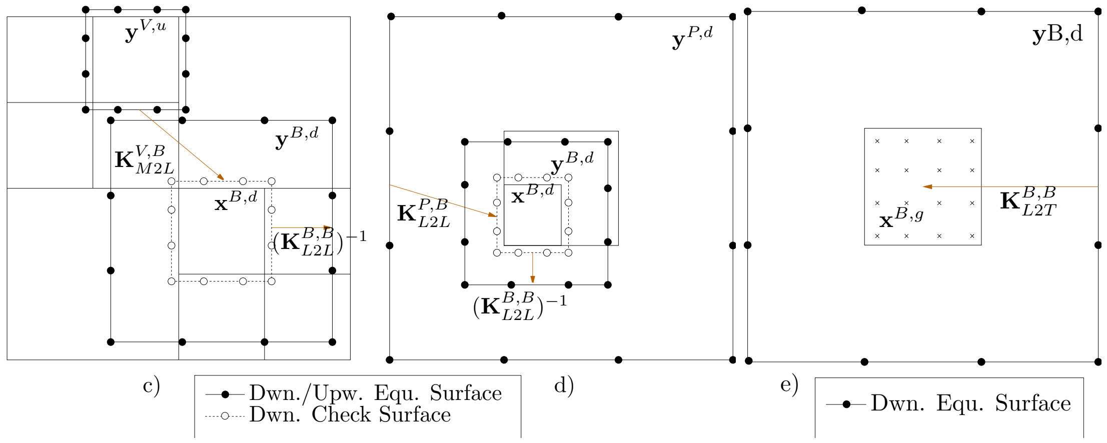


Figure 2.8: $M2L$ and $L2L$ kernel-independent FMM translation operators. c) $M2L$: for box B, to compute the contribution from a box, $V \in L_I^B$, we use the $M2L$ translation operators to compute the induced downward check potential, $u^{B,d}$ at the downward check surface, $\mathbf{x}^{B,d}$, from V's upward equivalent coefficient density, $\phi^{V,u}$ at its upward equivalent density surface, $\mathbf{y}^{B,u}$. $u^{B,d}$ is then matched at B's downward equivalent surface, $\mathbf{y}^{B,d}$ to compute the downward equivalent density, $\phi^{B,d}$; d) $L2L$: for all other boxes in $\mathcal{F}^B L_I^B$, the $L2L$ translation operators are used to take B's parent, P's downward equivalent density, $\phi^{B,d}$ at P's downward equivalent density surface, $\mathbf{y}^{B,d}$ and compute the contribution to B's downward check potential, $u^{B,d}$ at B's downward check surface, $\mathbf{x}^{B,d}$. This potential is then matched at B's downward equivalent surface, $\mathbf{y}^{B,d}$, to obtain B's downward equivalent density, $\phi^{B,d}$; e) $L2T$ translations: for leaf boxes, the $L2T$ translation operator translates B's downward equivalent density $\phi^{B,d}$ at surface $\mathbf{y}^{B,d}$ to its target grid $\mathbf{x}^{B,g}$.

Multipole to Local (M2L) translations. The $M2L$ operator translates an upward equivalent density $\phi^{V,u}$, approximating the field of sources inside a box $V \in L_I^B$, to a *downward equivalent density* $\phi^{B,d}$ for a box B, approximating the influence of these far-field sources inside B. In this case, we seek to have identical potentials *inside* the box B. To satisfy the conditions for check and equivalent surfaces, B is enclosed by the check surface $\mathbf{x}^{B,d}$, which, in turn, is enclosed by the downward equivalent surface $\mathbf{y}^{B,d}$, not overlapping $\mathbf{y}^{V,u}$. This is achieved by swapping upward equivalent and check surfaces to obtain downward equivalent and check surfaces: $\mathbf{y}^{B,d} = \mathbf{x}^{B,u}$ and $\mathbf{x}^{B,d} = \mathbf{y}^{B,u}$. Equation (2.26) takes the form

$$K[\mathbf{y}^{B,d}, \phi^{B,d}](\mathbf{x}) = K[\mathbf{y}^{V,u}, \phi^{V,u}](\mathbf{x}), \quad \text{for all } \mathbf{x} \in \mathbf{x}^{B,d}, \tag{2.36}$$

where $\phi^{B,d}$ is discretized at p uniformly spaced samples on $\mathbf{y}^{B,d}$. The right-hand side of (2.36) is computed and stored as a *downward check potential*, $u^{B,d}$ at $\mathbf{x}^{B,d}$, and $\boldsymbol{\phi}^{B,d}$ is recovered after the $L2L$ contribution is added. For a box B at depth ℓ

$$u^{B,d}_{M2L} = \sum_{V \in L^B_I} \mathbf{K}^{V,B}_{M2L} \boldsymbol{\phi}^{V,u}. \tag{2.37}$$

We efficiently evaluate $u^{B,d}$ with FFTs by treating densities as being defined on extensions of $\mathbf{y}^{V,u}$ and $\mathbf{x}^{B,d}$ to 3D Cartesian grids with zero values in the interior. This results in $O(p^3/2)$ sample locations, and the computational cost of $O(p^3/2 \log(p))$ for evaluation.

Further, there are at most 189 possible locations for $V \in L^B_I$ relative to any particular B; however, using translation and rotation invariance of the kernel as discussed in section 2.5, we store at most 16 total $\mathbf{K}^{V,B}_{M2L}$ matrices for a homogeneous kernel that is we store 16 matrices $\mathbf{K}^{V,B^*}_{M2L}$, one for each class of V from the interaction list $L^{B^*}_I$ of the normalized box $B^* = [-1,1]^3$, with $\mathbf{K}^{V,B}_{M2L}$ obtained by scaling as in the $M2M$ case. For an inhomogeneous kernel, at most 16 matrices are needed for each possible level of T (the actual number is smaller than the maximum number, due to boundary effects at the coarse levels of the tree).

Local to Local (L2L) translations. Contributions from $\mathcal{F}^B \setminus L^B_I$ are captured through the local field computed for B's parent box, P, using $L2L$ operators (Figure 2.8(d)). We translate

$\phi^{P,d}$ at $\mathbf{y}^{P,d}$ to $\phi^{B,d}$ at $\mathbf{y}^{B,d}$ using the equation

$$K[\mathbf{y}^{B,d}, \phi^{B,d}](\mathbf{x}) = K[\mathbf{y}^{P,d}, \phi^{P,d}](\mathbf{x}), \quad \text{for all } \mathbf{x} \in \mathbf{x}^{B,d}. \tag{2.38}$$

The right-hand side is computed as a contribution to $u^{B,d}$, so (2.38) for B at depth ℓ becomes

$$u^{B,d}_{L2L} = \mathbf{K}^{P,B}_{L2L} \boldsymbol{\phi}^{P,d} \text{ such that} \tag{2.39}$$

$$\boldsymbol{\phi}^{B,d} = (\mathbf{K}^{B,B}_{L2L})^{-1} \left(u^{B,d}_{M2L} + u^{B,d}_{L2L} \right). \tag{2.40}$$

The precomputation of matrix $\mathbf{K}^{P,B}_{L2L}$ is completely analogous to $\mathbf{K}^{C,B}_{M2M}$, with parent and child swapped. As in the $M2M$ computations, the inverted operator, $(\mathbf{K}^{B,B}_{L2L})^{-1}$ is precomputed once for homogeneous kernels as a $p \times p$ matrix at $\ell = 0$ with normalized box $B^* = [-1, 1]^3$ and scaled as necessary; for inhomogeneous kernels, at most one matrix is stored for each possible level of T.

Local to Grid Target (L2T) translation. For each leaf box B, we evaluate $u^{B,g}$ at grid locations, $\mathbf{x}^{B,g}$. At depth ℓ, $\phi^{B,d}$ accounts for all contributions from $\mathcal{F}^B$ while direct near-field calculations (discussed in detail in section 2.4.4) account for the contributions from $\mathcal{N}^B$. The far-field potential is computed using $L2T$ operators (Figure 2.8(e)),

$$u(\mathbf{x}) = K[\mathbf{y}^{B,d}, \boldsymbol{\phi}^{B,d}](\mathbf{x}), \mathbf{x} \in \mathbf{x}^{B,g}, \text{ or in matrix form: } u^{B,g} = \mathbf{K}^{B,B}_{L2T} \boldsymbol{\phi}^{B,d}. \tag{2.41}$$

For a uniformly-refined tree, all leaves are at the same level, so we precompute and store one $\mathbf{K}^{B,B}_{L2T}$ matrix.

2.4.4 Near-Field Interactions

After the far-field contributions are computed, the final step is to compute near-field interactions for leaf boxes. This is the most expensive step in the computation, if carried out naïvely, and it is essential to optimize this part of the algorithm. For each leaf box B, we need to compute the influence of the volume density g^U for every box $U \in L_B^N$ (the near field boxes). Given a polynomial approximation γ^U to g^U, we evaluate the potential on an $n \times n \times n$ grid of samples $\mathbf{x}^{B,g}$ on B, which we then add to the far field contribution computed in (2.41).

The principal mechanism to accelerate this step is based on the observation that we may use a regular grid pattern of points in B, permitting the use of precomputation. More precisely,

$$u^{B,g}(\mathbf{x}) = \sum_U K[U,g](\mathbf{x}) = \sum_U \sum_{j=1}^{N_k} \gamma_j^U F_j^{U,B}(\mathbf{x}), \tag{2.42}$$

$$F_j^{U,B}(\mathbf{x}) = \int_U \beta_j \left(2^\ell(\mathbf{y} - \mathbf{c}_U)\right) K(\mathbf{x},\mathbf{y}) d\mathbf{y}, \quad \text{for } \mathbf{x} \in \mathbf{x}^{B,g}, \tag{2.43}$$

where $\mathbf{c}_U$ is the center of box U. We evaluate $u^{B,g}$ on a uniform grid $\mathbf{x}_i^{B,g}$, $i = 1 \ldots n^3$ for $n \leq 6$ and on a tensor product Chebyshev grid for $n > 6$ to avoid condition problems, as discussed in section 3.2.2. In matrix form (2.42) becomes

$$u^{B,g} = \sum_U \mathbf{F}^{U,B} \gamma^U. \tag{2.44}$$

For a uniformly-subdivided octree, there are at most 27 possible locations for $U \in L_N^B$ with respect to B itself; using symmetries, however, only 4 are unique up to translation and rotation (section 2.5). As in the $S2M$ computations, adaptive Gaussian quadrature (Berntsen et al., 1991) is used to precompute and store the weights for these matrices. This can be done to machine precision for the function u and its first or second derivatives.

As each leaf box, B is not dependent on the near-field computations of any leaf box in T, it can quickly be seen how even the simplest approaches can take advantage of parallel architectures in the near-field computations. In section 3.3, we discuss how we use OpenMP (Chapman et al., 2007) and load-balancing approaches to parallelize the near-field and other computational steps of the FMM for shared-memory, multiprocessor architectures.

2.4.5 Polynomial approximation of the solution

In order to compute the value of u^B at an arbitrary point in the box, it is convenient to approximate it as a polynomial υ^B using a least-squares fit, minimizing

$$\sum_{i=1}^{n^3} \|u^B(\mathbf{x}_i) - \sum_{j=1}^{N_n} \upsilon_j^B \beta_j(\mathbf{x}_i - \mathbf{c}_B)\|^2 \quad \text{for } \mathbf{x}_i \in \mathbf{x}^B, \tag{2.45}$$

where $\beta_j \in \{P_a(x)P_b(y)P_c(z), 0 \leq a+b+c \leq n-1\}$ using either a monomial or Chebyshev polynomial basis, depending on the desired order n. For $n \leq 6$ it is more convenient to use regular grids, while for $n > 6$ Chebyshev grid points provide greater stability. In section 3.2.2 we demonstrate the accuracy of equispaced points and Chebyshev points for $n = 4, 6, 8$. For box B at depth ℓ, if Γ is the matrix with entries $\Gamma_{ij} = \beta_j(2^\ell(\mathbf{x} - \mathbf{c}_B))$, (2.45) leads to the equation $\upsilon^B = \Gamma^{(+)}u^{B,g}$, where the pseudoinverse $\Gamma^{(+)}$ needs to be precomputed only once as it does not depend on the kernel and is scale-invariant in *all* cases; that is, $\Gamma_{ij} = \beta_j(\mathbf{x}_i^*)$ where $\mathbf{x}_i^*$ are grid points in $B^* = [-1, 1]^3$. Once the υ_j^B are known, we can evaluate the solution at an arbitrary point $\mathbf{x}_t \in B$ as

$$u(\mathbf{x}_t) = \sum_{j=1}^{N_n} \upsilon_j^B \beta_j(\mathbf{x}_t - \mathbf{c}_B). \tag{2.46}$$

In general, we assume that k, the order of the approximation γ^B of the force g^B, is equal to n, the order of approximation of υ^B; however, as source and target locations need not be the same, k and n can be different.

2.4.6 Polynomial force approximation from grid samples

In our description of the upward pass, we have assumed that the right-hand side is already given as a polynomial. If the force is available in another form (e.g., as samples on an AMR grid or polynomials on an unstructured finite element grid), we can resample it adaptively, to obtain

k^{th}-order approximations with desired error on leaf boxes, then convert it to a polynomial representation. The only requirement on the input force in this case is that we can evaluate it at our grid locations with k^{th}-order accuracy or better, which is nontrivial only in the case of forces given by samples at scattered points.

If the values of the force $f(\mathbf{x})$ are known at k^3 uniformly sampled or Chebyshev points on a leaf B with center $\mathbf{c}_B$, an approximation to the force is constructed as

$$g^B(\mathbf{x}) \approx \sum_{j=1}^{N_k} \gamma_j^B \beta_j(\mathbf{x} - \mathbf{x}_B)$$

for $N_k = k(k+1)(k+2)/6$. As for the evaluation at arbitrary target locations, we use the monomial basis for $k \leq 6$, and Chebyshev basis for $k > 6$.

2.4.7 Non-Uniform Source Distributions and Adaptive FMM

We have thus far assumed that all leaves in T are at the same level. Adaptive refinement of the octree results in leaf boxes at different levels for nonuniform source distributions. This leads to several additional types of interactions between boxes that need to be taken into account.

For arbitrary adaptive octrees, the number of relative positions of boxes one needs to consider can become very large. In order to avoid storing large number of precomputed matrices, we consider *level-restricted refinement*: we require adjacent leaf boxes be within one level of each other, a common restriction in tree codes and structured grids. Many fast approaches exist to convert arbitrary octrees to ones satisfying this constraint (Sundar et al., 2008); we currently use a straightforward sequential algorithm (section A.1) similar to (Ethridge and Greengard, 2001).

We begin by introducing the notation for these lists and then discuss how this affects the S2M and near-field interaction computations.

Lists for adaptive FMM. For adaptively refined trees, we define several lists in addition to the neighbor list L_N^B and interaction list L_I^B used in the uniform case. Our definitions and notation follow (Greengard and Rokhlin, 1987; Greengard and Rokhlin, 1988; Greengard, 1994).

For a leaf box B, we define the U and W lists.

- The *U-list*, L_U^B, is the set of other leaves adjacent (at arbitrary levels) to B, including itself; L_U^B coincides with the neighbor list L_N^B for the uniformly refined case.

- The *W-list*, L_W^B, is the set of descendants of B's neighbors, not adjacent to B, but whose parents are adjacent to B. This list contains boxes at finer levels than B for which B is in their far range, but which are in the near range of the parent of B. For any $W \in L_W^B$, W is at a finer level than B and $W \in \mathcal{N}^B$ (conversely, $B \in \mathcal{F}^B$).

For leaf and non-leaf boxes B, we define V and X lists.

- The *V-list*, L_V^B, is the set of B's parent's neighbor's children, not-adjacent to B. $L_V^B = L_I^B$ for uniformly-refined trees, and if one completes an adaptive tree T by adding all missing boxes, on non-empty levels to a uniformly refined tree T^u, then L_V^B in T is a subset of L_I^B in T^u.

- The *X-list*, L_X^B, is the set of boxes A such that $B \in L_W^A$. In a more geometric manner, one can define X as the set of leaf boxes on levels coarser than B, overlapping a box in the interaction list of B in T^u but not overlapping the neighbors of B in T^u.

The following observations can be made about these lists: $B \in L_U^A$ if and only if $A \in L_U^B$, $V \in L_V^A$ if and only if $A \in L_V^B$, $B \in L_W^A$ if and only if $A \in L_X^B$. A fragment of an adaptively refined level-restricted tree, with U,V,W and X lists for a box B was shown above in Figure 2.9.

By the following lemma, for level-restricted trees, boxes in W and X lists have finite possible positions.

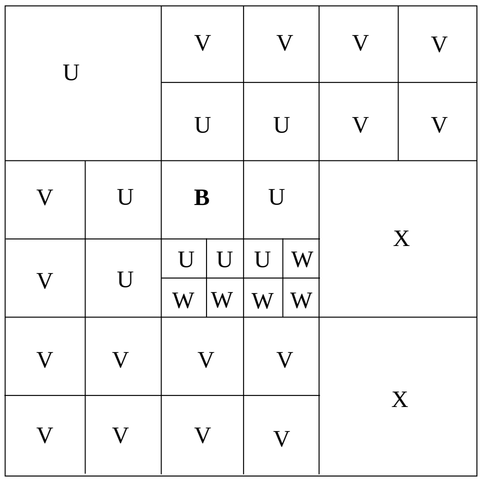


Figure 2.9: A 2d quadtree resulting from a non-uniform source distribution. For box B, correspondingly marked boxes represent the boxes in the U, V, W, and X lists of B.

Lemma 2.1. *For a level-restricted tree T, in which all neighboring leaf boxes are within one level of each other in the octree, for a box, B, all boxes in L_W^B and L_X^B must also be within one level of B.*

Proof. Assume for a box B that there exists a box $W \in L_W^B$ such that $level(W) - level(B) \geq 2$. That implies that for W's parent, P_W, $level(P_W) - level(B) \geq 1$, further implying that for some descendant D of P_W, $D \in L_U^B$ and $level(D) - level(B) \geq 2$, violating our tree-level restriction. So, W must be within one level of B. Since $W \in L_W^B$ implies $B \in L_X^W$, this also means that all boxes $X \in L_X^B$ must also be within one level of B. $\square$

Possible positions of boxes in U, V, W and X lists in a level-restricted tree are shown in Figure 2.10. Boxes in L_U^B and L_V^B are treated exactly in the same way as boxes in L_N^B and L_I^B, respectively in the uniform case: for L_U^B, the near-field interaction operators are used, and for L_V^B, $M2L$ operators are used.

For some leaf box B with parent P_B, if $W \in L_W^B$, then $W \notin \mathcal{F}^B$; therefore, W's contribution to B is not accounted for through P_B. At the same time, since B *is* in $\mathcal{F}^W$, W's contribution to B's potential can be computed by evaluating its upward density potential (the analog of multipole expansions) at target locations in B; hence, using notation analogous to other operators, $M2T$ operators need to be defined. For $X \in L_X^B$, $B \in \mathcal{N}^X$ but $X \in \mathcal{F}^{P_B}$; thus, we need to evaluate contributions of sources from X directly. We can apply these contributions to the downward density of B; that is, we need to define an $S2L$ operator. As explained below, for the local low-order polynomial representations to the force distributions, it may be preferable to use near-field computations mapping polynomial coefficients from boxes in L_X^B and L_W^B to potential values at target locations in B.

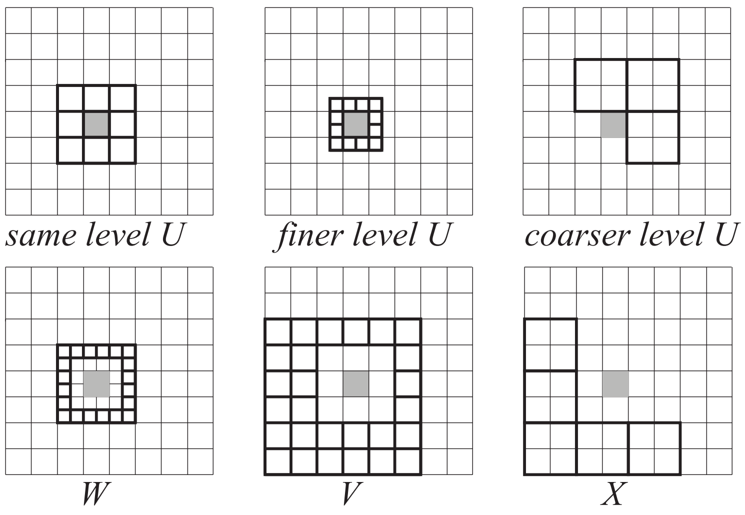


Figure 2.10: Possible box positions for different lists in a level-restricted trees in 2d. The configurations in 3d are analogous.

To summarize, for adaptive FMM, in addition to $M2M$, $M2L$, $L2L$ and $L2T$ already defined, two additional operators, $M2T$ and $S2L$ need to be defined. Further, as leaves of the tree now may exist at arbitrary levels, and boxes $U \in L_U^B$ may be on levels different from B, both $S2M$ and near-field ($S2T$) computations need to be modified. We begin by describing adaptations to the $S2M$ and $S2T$ operators and follow with a discussion of the new $M2T$ and $S2L$ operators.

S2M operators for the adaptive case. For a uniformly-refined domain, all leaves in the octree structure are on the same level, so only one matrix, $\mathbf{T}_{S2M}^{B}$ needs to be computed for a box B at leaf level ℓ. In the adaptive case, however, leaf boxes can be located at multiple levels.

For *homogeneous* kernels, we store a *single* matrix $\mathbf{T}_{S2M}^{B^*}$, scaling for level ℓ as was done for the $M2M$ and $L2L$ operators. Let $B^* = [-1,1]^3$ at $\ell = 0$. Then, for $\mathbf{x} \in B$ at level ℓ, let $\mathbf{x}^* = 2^\ell(\mathbf{x} - \mathbf{c}^B)$ for $\mathbf{x}^* \in B^*$. For scaling exponent, r, $K(\mathbf{x}_i, \mathbf{y}_j) = \left(\frac{1}{2}\right)^{r\ell} K(\mathbf{x}_i^*, \mathbf{y}_j^*)$, and (2.30) becomes

$$F_j^B(\mathbf{x}) = \left(\frac{1}{2}\right)^{(r+2)\ell} K[B^*, \beta_j](\mathbf{x}^*) = \left(\frac{1}{2}\right)^{(r+2)\ell} F_j^{B^*}(\mathbf{x}^*) \quad \text{for all } \mathbf{x} \in \mathbf{x}^{B,g}, \mathbf{x}^* \in \mathbf{x}^{B^*,g}.$$

In matrix form, $\mathbf{F}_{S2M}^{B} = 2^{-(r+2)\ell}\mathbf{F}_{S2M}^{B^*}$ and $\mathbf{K}_{S2M}^{B} = 2^{-r\ell}\mathbf{K}_{S2M}^{B^*}$. Solving for $\boldsymbol{\phi}^{B,u}$, (2.33) becomes

$$\boldsymbol{\phi}^{B,u} = \frac{1}{2}(\mathbf{K}_{S2M}^{B^*})^{-1}\mathbf{F}_{S2M}^{B^*}\gamma^B = \mathbf{T}_{S2M}^{B^*}\gamma^B, \tag{2.47}$$

where $\mathbf{T}_{S2M}^{B^*}$ is precomputed and stored. For inhomogeneous kernels, we store one matrix per level containing leaf boxes.

Neighbor list interactions for adaptive trees. In section 2.4.4, for a box B with neighbor U, the assumption that all leaves are at the same level allows us to use (2.44) with precomputed matrices $\mathbf{F}_{S2T}^{U,B}$ and U's coefficients γ^U to evaluate U's contribution to B's potential at the grid points $\mathbf{x}^{B,g}$. For adaptive level-restricted trees, leaves may exist at any level and $U \in L_U^B$ may exist at one level finer or coarser than B. As above for the $S2M$ operators, for homogeneous kernels with scaling exponent r, we only compute matrices for pairs (B, U) with B scaled to B^* (U is appropriately scaled as well to U^*) such that (2.43) and (2.44) become

$$F_j^{U,B}(\mathbf{x}) = \left(\frac{1}{2}\right)^{(r+2)\ell} K[U^*, \beta_j](\mathbf{x}^*) = \left(\frac{1}{2}\right)^{(r+2)\ell} F_j^{(U,B)^*}(\mathbf{x}^*),$$

for all $\mathbf{x} \in \mathbf{x}^{B,g}$, $\mathbf{x}^* \in \mathbf{x}^{B^*,g}$, and

$$u^{B,g} = \sum_U \mathbf{F}_{S2T}^{U,B} \gamma^U = \left(\frac{1}{2}\right)^{(r+2)\ell} \sum_U \mathbf{F}_{S2T}^{(U,B)^*} \gamma^U, \tag{2.48}$$

where r is the kernel scaling exponent.

As discussed earlier, there are 27 possible same-level neighbors, and due to tree-level restrictions, there are 56 fine-level neighbors (one level deeper in the tree) and 56 coarse-level neighbors (one level higher in the tree), all constituting the 139 possible locations for boxes in L_U^B. As shown in section 2.5, using symmetries of relative positions of U and B, we only precompute and store 10 matrices of size $n^3 \times N_k$.

For inhomogeneous kernels, this set of matrices is precomputed for each level for which leaf boxes exist.

M2T and S2L operators. As explained above, for a leaf box B and $W \in L_W^B$, we need an operator that evaluates the potential represented by $\phi^{W,u}$, the upward equivalent density of W, at the target grid locations on B, $\mathbf{x}^{B,g}$:

$$u^{B,g}(\mathbf{x}) = K[\mathbf{y}^{W,u}, \boldsymbol{\phi}^{W,u}](\mathbf{x}) \quad \text{for } \mathbf{x} \in \mathbf{x}^{B,g},$$

or in matrix form,

$$u^{B,g} = \mathbf{K}_{M2T}^{W,B} \phi^{W,u}, \tag{2.49}$$

where the operators $\mathbf{K}_{M2T}^{W,B}$ are precomputed and stored. Similar to all previous cases, for homogeneous kernels, $\mathbf{K}_{M2T}^{W,B}$ can be computed for the normalized box B^* only and scaled as necessary.

For all boxes B, the list L_X^B contains leaf boxes X, for which contributions to B are computed by evaluating contribution of g^X, represented by coefficients γ^X, on B's downward check surface:

$$u^{B,d}(\mathbf{x}) = K[X, g^X](\mathbf{x}) \approx \sum_{j=1}^{N_k} \gamma_j^X F_j(\mathbf{x}) \quad \text{for } \mathbf{x} \in \mathbf{x}^{B,d},$$

or in matrix form,

$$u^{B,d} = \mathbf{F}_{S2L}^{X,B} \gamma^X. \tag{2.50}$$

For a box B there are 152 possible locations for $W \in L_W^B$; however, due to symmetries, only 6 locations are distinct up to translation and rotation, so only 6 $\mathbf{K}_{M2T}^{W,B}$ matrices of size $n^3 \times p$ are stored. Also, due to the inverse relationship between L_X^B and L_W^B, the number of symmetry classes is the same; that is, 6 matrices $\mathbf{F}_{S2L}^{X,B}$ of size $p \times N_k$ need to be precomputed for each level for which leaf boxes X exist. As in all other cases, for homogeneous kernels, the matrices need to be precomputed only for one level $\ell = 0$ and scaled as necessary. Symmetry classes are discussed in 2.5.

Remark 2.2. *We use one additional optimization which applies in cases when the order of local polynomial approximations of the force is low compared to the order of approximation used for upward and downward check densities. In such cases, the size of $M2T$ and $S2L$ matrices may actually be* larger *than the size of the matrices needed for direct computation of contributions from coefficients on boxes $W \in L_W^B$ or $X \in L_X^B$ to the target grid locations $\mathbf{x}^{B,g}$ on B. Assuming we have a homogeneous kernel, if $W \in L_W^B$ is a leaf for a box B, we can* *replace $\mathbf{K}_{M2T}^{W,B}$ with an $S2T$ operator $\mathbf{F}_{S2T}^{W,B}$, constructed exactly in the same way as for boxes in the neighbor list L_U^B. Similarly, if B is a leaf, for a box $X \in L_X^B$, we can replace $\mathbf{F}_{S2L}^{X,B}$ with $\mathbf{F}_{S2T}^{X,B}$. In other words, for low order polynomial approximations of the force, one treats leaf W boxes in the same way as adjacent boxes of B and for leaf boxes B, L_X^B is also treated as adjacent boxes. Specifically, $\mathbf{F}_{S2T}^{W,B}$ is of size $n^3 \times N_k$ while $\mathbf{K}_{M2T}^{W,B}$ is of size $n^3 \times p$, so when a box W is a leaf and $N_k < p$, we use the faster S2T translations. Similarly, $\mathbf{F}_{S2T}^{X,B}$ is of size $n^3 \times N_k$ and $\mathbf{F}_{S2L}^{X,B}$ is of size $p \times N_k$, so for leaves B, we use $\mathbf{F}_{S2T}^{X,B}$ when $n^3 < p$. Again, symmetries result in 6 matrices needed of each type (section 2.5).*

For inhomogeneous kernels, we must compute and store these matrices for each necessary level ℓ.

2.4.8 Pseudocode and Complexity for Kernel-Independent FMM Volume Solver.

Pseudocode. We assume that a tree-level restricted octree, T, already exists (Ethridge and Greengard, 2001) and that for each box, B, we are given the approximation, γ^B to the force g^B (we discussed how to construct γ from g in section 2.4.6). For clarity, we do not include the optimization of replacing $M2T$ and $S2L$ with $S2T$ operators when more efficient as discussed above for Algorithm 2 below.

Algorithm 2 Kernel-Independent Volume FMM

```
STEP 1 - BUILD LISTS
for each box $B$ in preorder traversal of $T$ do
    build $L_U^B$, $L_W^B$, $L_X^B$, and $L_V^B$ (section 2.4.7)
end for
STEP 2 - UPWARD PASS (section 2.4.2)
for each box $B$ in postorder traversal of $T$ do
    if $B$ is a leaf box then
        Convert local force approximations to upward densities: $\phi^{B,u} := \mathbf{T}_{S2M}^{B}\gamma^{B}$ (2.33)
    else
        Translate children's upward densities: $\phi^{B,u} := \sum_{C} \mathbf{T}_{M2M}^{C,B}\phi^{C_i,u}$ (2.35)
    end if
end for
STEP 3 - DOWNWARD PASS (section 2.4.3)
for each non-root box $B$ in preorder traversal of $T$ do
    Compute potential due to $B$'s parent $P$ and from $L_V^B$ and $L_X^B$ to $B$'s downward potential:
    $u^{B,d} := \mathbf{K}_{L2L}^{P,B}\phi^{P,d} + \sum_{V\in L_V^B} \mathbf{K}_{M2L}^{V,B}\phi^{V,u} + \sum_{X\in L_X^B} \mathbf{F}_{S2L}^{X,B}\gamma^{X}$ (2.39), (2.37), (2.50)
    Translate the check potential to the downward density: $\phi^{B,d} := (\mathbf{K}_{L2L}^{B,B})^{-1}u^{B,d}$ (2.40)
    if $B$ is a leaf box then
        Compute potentials from adjacent and W boxes to the potential at grid locations:
        $u^{B,g} := \sum_{U\in L_U^B} \mathbf{F}_{S2T}^{U,B}\gamma^{U} + \sum_{W\in L_W^B} \mathbf{K}_{M2T}^{W,B}\phi^{W,u}$ (2.44), (2.49)
        Add the potential from the far field: $u^{B,g} := u^{B,g} + \mathbf{F}_{L2T}\phi^{B,d}$ (2.41)
    end if
end for
```

Computational complexity and storage requirements. We analyze the complexity for a uniformly-refined octree. The analysis for the adaptive FMM is similar but slightly more complicated. We assume a homogeneous scalar kernel such as the Laplace kernel in equation (2.10) for analyzing the storage and computational complexities. Further, we assume that there are ℓ levels in the octree T. For a uniform tree, this implies we have $M_\ell = 8^\ell$ leaves and $M_t = (8^{\ell+1} - 1)/7$ total boxes in T. If we are using a k^{th}-order polynomial approximation to the force at each leaf, we further assume there are approximately $N = M_\ell n^3$ total target points and $C = M_\ell N_k$ total coefficients. Let p be the number of coefficients sought in the multipole expansion, affecting the size of the equivalent densities and surfaces; for a desired level of precision, $\epsilon_{fmm} = 10^{-n_p}$ in the expansion, $p = n_p^3 - (n_p - 2)^3$. In table 2.1, we indicate the computational complexity of each step of the non-adaptive FMM algorithm as well as the amount of precomputation and storage used for operators at each step. For non-uniform source distributions, we store additional operators for the near-field interactions in the *U*, *W*, and *X* operators; the complexity of these operators are based on the degree of adaptivity.

Finally, we note that the computational and storage complexities will scale linearly for matrix or inhomogeneous kernels. For example, for the Stokes kernel in equation (2.12), the number of coefficients, p, scales as a results of the matrix kernel size to $p = 9(n_p^3 - (n_p - 2)^3)$. For the Modified Helmholtz kernel in equation (2.11), the inhomogeneous nature of the kernel results in an increased storage complexity, which varies depending on the number of different levels in the tree.

Operator	Computational Complexity	Storage
$S2M$: $\mathbf{T}_{S2M}^{B}$	$O(Cp)$	pN_k
$M2M$: $\mathbf{T}_{M2M}^{C,B}$	$O((M_t - M_\ell)p^2)$	p^2
$M2L$: $\mathbf{K}_{M2L}^{V,B}$	$O(M_t p^{3/2} \log(p) + 189 M_t p^{3/2})$	$16p^{3/2}$
$L2L$: $\mathbf{K}_{L2L}^{P,B}$, $(K_{L2L}^{B,B})^{-1}$	$O(M_t p^2)$	$2p^2$
$L2T$: $\mathbf{K}_{L2T}^{B,B}$	$O(Np)$	pn^3
Near Interaction: $\mathbf{F}_{S2T}^{U,B}$	$O(27NN_k)$	$4N_k n^3$
U-list (*same*, *fine*, *coarse levels*): $\mathbf{F}_{S2T}^{U,B}$		$10N_k n^3$
W-list: $\mathbf{F}_{S2T}^{W,B}$, $\mathbf{K}_{M2T}^{W,B}$		$6n^3(N_k + p)$
X-list: $\mathbf{F}_{S2T}^{X,B}$, $\mathbf{F}_{S2L}^{X,B}$		$6N_k(n^3 + p)$

Table 2.1: Computational complexity and storage requirements for a scalar homogeneous kernel. These values scale linearly for matrix and inhomogeneous kernels.

2.5 Symmetries for precomputed interaction operators

For each of the lists L^B_U,L^B_V,L^B_W and L^B_X, we may need to precompute translation matrices for densities or polynomial coefficients. The number of different relative positions of the box B and a box in one of these lists can be large, and precomputing all possible matrices may require significant time and substantial storage. Performance can also be affected due to the need for random access of large amounts of precomputed data.

The number of matrices we need to precompute can be substantially reduced if we take into account symmetries; that is, many box positions are equivalent in the sense that there is a rigid transformation T, mapping a box Z_1 to Z_2 and the box B to itself. We store a single matrix for a representative box for each symmetry class, obtaining matrices for all elements of the class by applying a transformation T to the matrix for the representative box.

For every list type $Z \in \{U, V, W, X\}$, we define a set of possible box positions $Pos(Z)$ and a set of symmetry classes which form a partition of $Pos(Z)$. For each class, we define a *reference box*, and for each box position in $Pos(Z)$, we need an efficient way to determine its class and a transformation $T(B) : \mathbf{R}^3 \to \mathbf{R}^3$ mapping it to the reference box.

For all lists, the symmetries are related to the transformations of space which map a grid of cubes to itself. Referring to a grid of size $N \times N \times N$ grid as an N^3 grid, we consider grids of sizes 1^3 to 7^3 (we discuss which lists correspond to which cubes in more detail below). Before considering individual lists, we classify all symmetries of such grids.

Grid symmetries. The cubes on the grid N^3 are indexed by (i, j, k) where each index takes values $-M \ldots -1, 0, 1 \ldots M$ for odd $N = 2M + 1$ and $-M \ldots -1, 1 \ldots M$ for even N. We skip index 0 for even grids to ensure that the coordinates of the cube centers and cube indices are transformed by symmetries of the cube in the same way. If the cube size is 1, then the cube center coordinates are exactly the indices (i, j, k) for odd N and differ by $\pm 1/2$ for even N, depending on the index sign.

Each N^3 grid can be partitioned into M (for even N) or $M + 1$ (for odd N) layers, with layer l consisting of cubes (i, j, k) with $max(i, j, k) = l$. Layer 0 consists of one cube and exists only for odd N, and layer M consists of the cubes on the surface of the N^3 grid. We will refer to layers either by their number l or by size. For odd N, layer l has size $(2l + 1)^3$ and for even N, layer l has size $(2l)^3$.

The group of symmetries G_{cube} of a cube has order 48. For a cube centered at zero, transformations in G_{cube} are compositions of rotations and reflections, mapping each axis direction to another, possibly with orientation reversed. Clearly, any permutation of directions is possible, so it is convenient to identify the group with $S_3 \times J^3$, where S_3 is the group of permutations of length 3, and J is the two-element group of reflections. The rotational part of any element of G_{cube} can be specified as a permutation of length 3 on the set of axes $\{x, y, z\}$, with an orientation 1 or -1 specified for each axis. Transformations from G_{cube} encoded in this way can be applied to points very efficiently: for a point $\mathbf{x} \in \mathbf{R}^3$, the permutation is applied to its coordinates, which are then scaled by 1 or -1.

For the N^3 grid, the equivalence classes under the action of G_{cube} can be enumerated combinatorially. Observe that if two triples of indices (i, j, k) and (i', j', k') differ only by signs of components, corresponding cubes are in the same class: they are mapped to each other by reflections. To enumerate all classes, we only need to consider cubes with nonnegative indices. Two cubes with nonnegative indices (i, j, k) and (i', j', k') are in the same class if and only if there is a permutation mapping (i, j, k) to (i', j', k').

If we adopt the convention that i, j, and k represent distinct numbers in the range $1 \dots M$, seven series of equivalence classes are easily enumerated, corresponding to signatures (i, j, k), (i, i, j), (i, i, i), $(0, i, i)$, $(0, 0, i)$ and $(0, 0, 0)$. A reference box in every class is uniquely de-

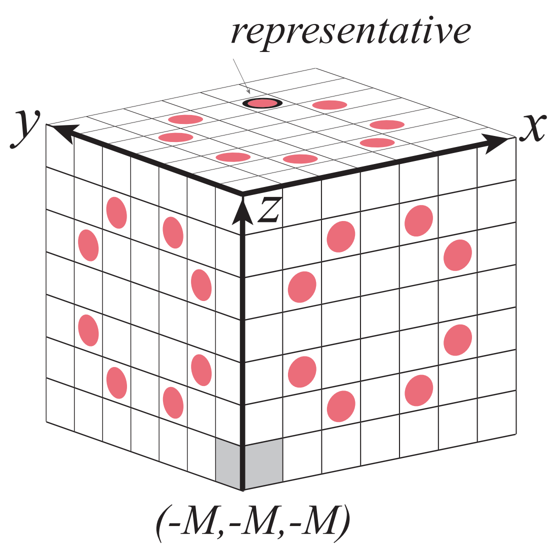


Figure 2.11: A three-dimensional view of the class $(1,2,3)$ in the 7^3 grid, showing the index directions (x,y,z) the representative box, and the box with index $(-M,-M,-M) = (-3,-3,-3)$.

fined by requiring that its three indices are all nonnegative and are in nondecreasing order (Figure 2.11).

The properties of classes in each series are summarized in Table 2.2 with figures illustrating the geometric meaning of each class series. For example, the classes with signature (i,j,k) consist of cubes in the interior of layer faces with centers not on face diagonals or lines connecting edge centers, (i,i,i) are classes of cubes at layer vertices, and $(0,i,i)$ are classes of cubes at layer edge centers.

The total number of classes for $(2M)^3$ layers is $(M+1)M/2$ (classes (i,j,M) with $i,j = 1\ldots M$, $i \leq j$), and for $(2M+1)^3$ layers, it is $(M+2)(M+1)/2$ (classes (i,j,M) with $i,j = 0\ldots M, i \leq j$).

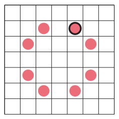
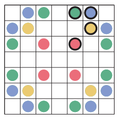
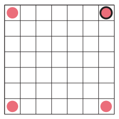
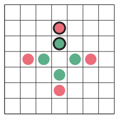
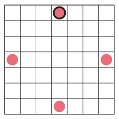
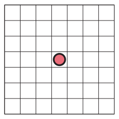
Series signatures	7^3 layer classes	Reference cube	Classes per grid	Classes per layer	Class size
(i, j, k)		$(\|i\|, \|j\|, \|k\|)$, $\|i\| < \|j\| < \|k\|$	$\begin{pmatrix} M \\ 3 \end{pmatrix}$	$\begin{pmatrix} M-1 \\ 2 \end{pmatrix}$	48
(i, i, j)		$(\|i\|, \|i\|, \|j\|)$,$\|i\| < \|j\|$ or $(\|i\|, \|j\|, \|j\|)$	$M^2 - M$	$2M - M$	24
(i, i, i)		$(\|i\|, \|i\|, \|i\|)$	M	1	8
$(0, i, j)$		$(0, \|i\|, \|j\|)$	$\begin{pmatrix} M \\ 2 \end{pmatrix}$	$M - 1$	24
$(0, i, i)$		$(0, \|i\|, \|i\|)$	M	1	12
$(0, i, i)$		$(0, 0, \|i\|)$	M	1	6
$(0, 0, 0)$	—	$(0, 0, 0)$	1	—	1

Table 2.2: Series of equivalence classes of cubes in an N^3 grid. For even $N = 2M$, only the first 3 series of classes may be nonempty. For odd $N = 2M + 1$, all classes are present. For $M \leq 2$, (i, j, k) classes are empty, and for $M = 1$, (i, i, j) and $(0, i, j)$ classes are also empty. Class (0,0,0) corresponding to the center of the grid, exists only in layer 0. In the figures, boxes in different classes in one series are marked with circles of different colors, representative boxes are marked with circles with black border. The view is from the top, with first index direction to the right, second direction up and third towards the viewer, as in Figure 2.11.

For a box Z, with grid index (i, j, k) relative to B, the reference box is obtained by taking absolute values and sorting the indices; sign changes and a permutation mapping (i, j, k) to the reference box index also encode the transformation as explained above.

Next, we show how symmetry classes for different lists can be obtained from symmetry classes of layers of different sizes.

Symmetries of L_U^B. Due to our tree-level restriction, boxes $U \in L_U^B$ are either neighbors of B, neighbors of B's parent and adjacent to B, or children of neighbors of B adjacent to B. We denote these three sublists of L_U^B by $L_{U,n}^B$, $L_{U,p}^B$ and $L_{U,c}^B$ respectively. Note that $A \in L_{U,p}^B$ is equivalent to $B \in L_{U,c}^A$, a duality similar to the duality between W and X lists. It is sufficient to consider $L_{U,n}^B$ and $L_{U,c}^B$: the classes for $L_{U,p}^B$ are obtained by swapping B and U and considering classes of $L_{U,c}^B$. The neighbors of B on the same level as B form a 3^3 grid centered at B, and from table 2.2, it can be immediately seen that the number of classes is 4: (1,1,1), (0,1,1), (0,0,1), and (0,0,0).

Possible locations of $U \in L_{U,c}^B$ can be thought of as the outer layer of a 4^3 grid, with $M = 2$; B in this case corresponds to the 2^3 subgrid in the center, so rotations of the 4^3 grid mapping it to itself also map B to itself. Again, we immediately obtain 3 classes: (1,1,2), (1,2,2), (2,2,2). This gives a total of 10 classes for L_U^B.

Symmetries of L_V^B. Boxes in L_V^B are at the same level as B and are children of neighbors of the parent of B, so they can all be represented by cubes of a 6^3 grid. This grid, however, is not centered at B, so the group of rigid transformations of the grid mapping to itself do not necessarily preserve B. The problem can be avoided if we regard L_V^B as a subset of a 7^3 grid centered at B with $M = 3$. All boxes $V \in L_V^B$ are in layers 2 and 3 of this grid, and there are 10 classes: for layer 3, classes $(i, j, 3)$, $i, j = 1 \ldots 3$, $i \leq j$ and for layer 2, classes $(i, j, 2)$, $i, j = 0, 1, 2$, $i \leq j$.

Because we consider only a subset of the the full 7^3 grid, the class sizes are smaller, but one can easily show that no class becomes empty, so the number is optimal.

Symmetries of L^B_W. For level-restricted trees, boxes $W \in L^B_W$ are children of neighbors of B not adjacent to B; that is,they reside in the surface layer of a 6^3 grid with B identified as the central 2^3 grid. Again, table 2.2 immediately yields the classes of boxes: in this case, $M = 3$, and just as it is the case for outer-layer of possible boxes in L^B_V, we have 6 classes, $(i, j, 3)$, $i, j = 1 \ldots 3$, $i \leq j$. Due to the duality between L^B_X and L^B_W, the number of classes for L^B_X is the same. We note that the class sizes may *not* be the same.

To summarize, the following procedure can be used to obtain a precomputed matrix and transformation to the reference box for a given pair (B, Z). First, if Z is in $L^B_{U,n}$, $L^B_{U,c}$, L^B_V or L^B_W, determine the translation and scaling which map B to the central box or 2^3 subgrid of a larger grid, depending on the list:

- central box of 3^3 for $L^B_{U,n}$,
- central 2^3 subgrid of 4^3 grid for $L^B_{U,c}$,
- central box 7^3 grid for L^B_V,
- and central 2^3 subgrid of 6^3 grid for L^B_W.

Then, we apply the same transformation to the center of Z; resulting coordinates yield the index (i, j, k), which is translated into the reference box and rotation as described above.

For the two lists, $L^B_{U,p}$ and L^B_X, we use duality to lists $L^B_{U,c}$ and L^B_W respectively: instead of mapping a box B to the central 2^3 subgrid, we map $U \in L^B_{U,p}$ (respectively $W \in L^B_X$) to this subgrid and compute the index for B.

2.6 Additional Boundary Conditions for the Box

In order to impose boundary conditions on the box, we turn to using the classic ideas of Lord Rayleigh (Rayleigh, 1892) as described in (Greengard and Rokhlin, 1987; Ethridge and Greengard, 2001). We begin by discussing how to modify these approaches for the kernel-independent FMM with periodic boundary conditions and then discuss how to impose homogeneous Dirichlet boundary conditions. Mixed boundary conditions such as a combination of Dirichlet, periodic and Neumann can also be derived from the methods described below. Additionally, inhomogeneous boundary conditions, while not implemented in our current algorithm, could be performed in a manner similar to (Ethridge and Greengard, 2001)

2.6.1 Periodic Boundary Conditions

Imposing periodic boundary conditions on a domain box B consists of accounting for all copies of B in infinite space. In two dimensions, the problem would be as seen in figure 2.12. In three dimensions, we *tile* $\mathcal{R}^3$ with copies of our original domain B. At the conclusion of the upward pass (M2M - Multipole to Multipole pass) in algorithm 1, we have a multipole expansion for the entire source distribution in the original domain B, or in the case of the kernel-independent FMM in algorithm 2, we have an upward equivalent density $\phi^{B,u}$ representation on the upward equivalent surface, $\mathbf{y}^{B,u}$ for the entire domain, B. Since $\phi^{B,u}$ is independent of location for our kernel, all copies of B have the same equivalent densities at their equivalent surfaces. Therefore, for all copies B_c of B which are not immediately adjacent (copies which are in its far-field), we can compute the influence from $\phi^{B_c,u}$ at B's downward check surface, $\mathbf{x}^{B,d}$. Then at the downward pass (M2L - Multipole to Local) of the FMM, the influence from all of these boxes is accounted for. In figure 2.12 boxes inside of the dotted-line are in the near-field and must be accounted for differently.

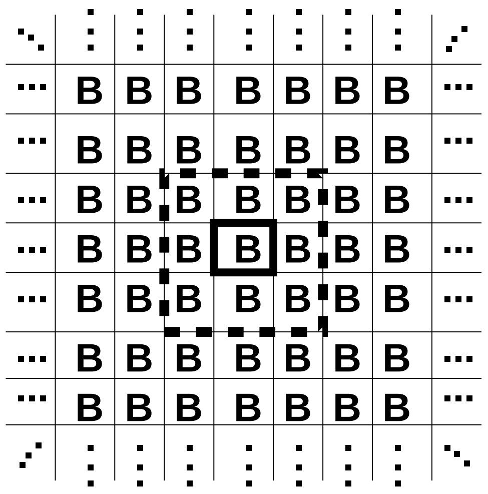


Figure 2.12: The original domain B is tiled on the infinite plane. In order to account for periodic boundary conditions, we must calculate the potential induced on B by a significantly large numbers of copies of itself, an intractable task if attempted directly.

We cannot of course account for an infinite number of copies of B; furthermore, even attempting to account for a large number of them (for example, computing the influence more than even 10 levels outward) is cumbersome and computationally not feasible directly. Instead of using lattice sums which rely on the equation type, we use the existing infrastructure of the kernel-independent FMM to aid us. Assuming that B is of width $H = 2^{-r+1}$ for rootlevel r of T, as before, let $\mathcal{N}^B$ be the near-field of B, containing all copies of itself within a box centered at B with width $3H$. In the first step, we account for the influence of copies of B not in $\mathcal{N}^B$ which are contained within a box of width $9H$ centered at B; for the periodic solver, this represents B's interaction list L_I^B. Since we are computing the check potential, $u_{M2L}^{B,d}$ at B's check surface from copies of B using B's own equivalent density, $\phi^{B,u}$, this is a slight modification of the $M2L$ operation in section 2.4.3. The modified equation is presented in equation (2.51).

$$u_{M2L}^{B,d} = \sum_{V \in L_I^B} \mathbf{K}_{M2L}^{V,B} \phi^{B,u} = \mathbf{K}_{M2L}^{B} \phi^{B,u}, \tag{2.51}$$

where $\mathbf{K}_{M2L}^{B} = \sum_{V \in L_I^B} \mathbf{K}_{M2L}^{V,B}$ is used since $\phi^{B,u}$ is the same equivalent density for all nodes in L_I^B.

We further note that the full domain of B's parent, defined here as B^*, is the box centered at B of width $3H$ containing all boxes in $\mathcal{N}^B$. In figure 2.13, B is at the center of the image, and all boxes in $\mathcal{N}^B = B^*$ are contained within the red outlined box centered at B.

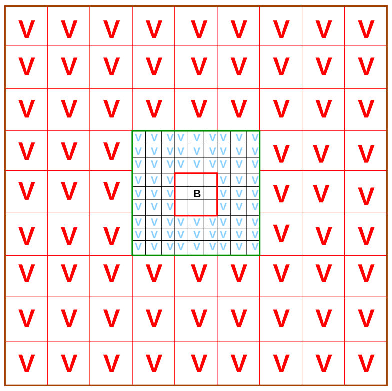


Figure 2.13: For our original domain, denoted by B, the periodic far-field, $\mathcal{F}^B$ is everything outside of the small red box (out to infinity, not just the small sampling here). All boxes within the blue region with green labeled Vs are in B's direct interaction list, L_I^B. B's parent in the periodic domain is considered to be the box in red, and its interaction list consists of all boxes of the same size with boxes labeled with red Vs. B's parent's parent is the green-outlined box, and its interaction list consists of an additional layer of 702 boxes outside of this picture, each of which is of size $(7H)^3$ for H the width of our domain B. In this way, we account for $\left(3^{N+1} - 3\right)/2$ *rings* of B (or $(3^{N+1})^3 - 3^3$ copies of B) after just N steps, going out $\sum_{i=1}^{N} 3^i$ concentric rings of B.

Continuing outward in figure 2.13, all boxes labeled as a blue V inside of the green-outlined box, centered at B, are copies of B and are in L_I^B. All of the boxes with red Vs are copies of B^* and are in $L_I^{B^*}$ such that each of these boxes computes the influence of the copies of B^* at $\mathbf{x}^{B^*,d}$ using $\phi^{B^*,u}$ as above with equation (2.51), substituting B^* for B:

$$u_{M2L}^{B^*,d} = \sum_{V \in L^{B^*}_I} \mathbf{K}_{M2L}^{V,B^*} \boldsymbol{\phi}^{B^*,u} = \mathbf{K}_{M2L}^{B^*} \boldsymbol{\phi}^{B^*,u}. \tag{2.52}$$

We note that in equations (2.51) and (2.52), for a scale-variant kernel, $\mathbf{K}_{M2L}^{V,B}$ and $\mathbf{K}_{M2L}^{V,B^*}$ (and subsequently, the summation kernels, $\mathbf{K}_{M2L}^{B}$ and $\mathbf{K}_{M2L}^{B^*}$) are equal up to a scaling factor for respective members of their interaction lists, as described earlier. Additionally, we compute $\phi^{B^*,u}$ using a slight modification of the $M2M$ process from section 2.4.2. That is, modifying equation (2.35),

$$\boldsymbol{\phi}^{B^*,u} = \sum_B (\mathbf{K}_{M2M}^{B^*,B^*})^{-1} \mathbf{K}_{M2M}^{B,B^*} \boldsymbol{\phi}^{B,u} = \sum_B \mathbf{T}_{M2M}^{B,B^*} \phi^{B,u} \tag{2.53}$$

We note that since B^*'s children are just copies of the original B equation (2.53) is simplified as

$$\boldsymbol{\phi}^{B^*,u} = \mathbf{T}_{M2M}^{B^*} \phi^{B,u} = \left(\sum_B \mathbf{T}_{M2M}^{B,B^*} \right) \phi^{B,u}, \tag{2.54}$$

where $\mathbf{T}_{M2M}^{B,B^*}$ is computed for each translated copy of B. In figure 2.14 we modify our earlier figures to show how the $M2M$ process works for the full domain. In three dimensions, the number of children accounted for is 27 in the $M2M$ process.

This is a slight modification to the M2M upward computations already used in the general structure. For B^*'s parent, we calculate its upward equivalent density as the result of the potential from 27 copies of B^* in three dimensions. We continue this process, building concentrically outward to insure symmetry in all directions, avoiding the need for renormalizing the densities as we move upward (Berman and Greengard, 1994; Rodin and Overfelt, 2004). Iterating this

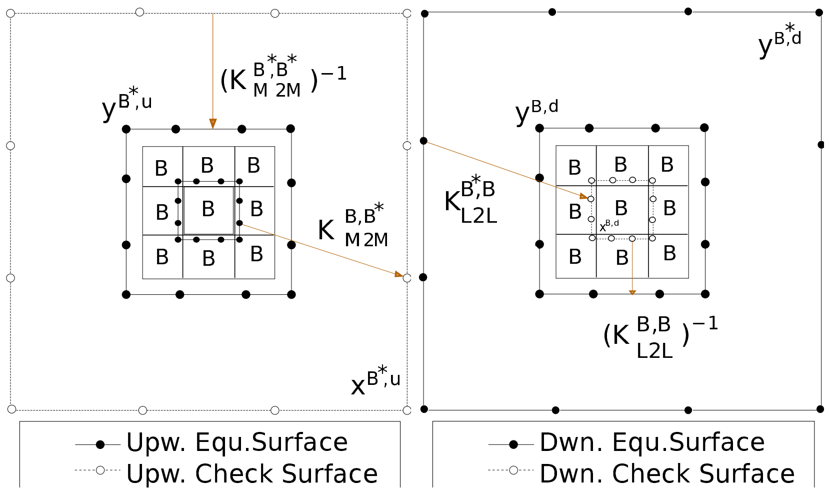


Figure 2.14: For our original domain, denoted by B, we modify the $M2M$ and $L2L$ operations for calculating the far-field interactions. *Left*: We calculate the upward equivalent density induced by the domain and all of its immediate neighbors (simply copies of itself, occupying the space of size $(3H)^d$, centered at the domain of width H) at its parent B^*; this process is continued up N levels. *Right*: For passing the downward equivalent densities downward, we modify the $L2L$ operator to calculate the far-field potential from parent B^* to its center child, B only; at the completion of this step, our original domain has a description of its entire periodic far-field domain in its downward equivalent density, which is in turn passed downward through the tree.

procedure, notice that if we consider each *ring* of copies of B, after N such steps, we have accounted for $\sum_{i=1}^{N} 3^i$ rings of B. We then pass the influences downward in an L2L or local to local fashion (again noting that these matrices will be reused so no additional cost is incurred) as seen in figure 2.14. Here we only need to translate the downward equivalent densities to the *center* child of a box. That is, if B is a box with parent B^* as in figure 2.14, we compute

$$u_{L2L}^{B,d} = \mathbf{K}_{L2L}^{B^*,B} \boldsymbol{\phi}^{B^*,d}. \tag{2.55}$$

We only need to compute a single $\mathbf{K}_{L2L}^{B^*,B}$ operator since we are only interested in computing the far-field influence at our original domain, and hence only compute the induced potential at the box B at the center of the cube of 27 children of B^*. We then can use equation (2.40) to recover the downward equivalent density for B from $u_{M2L}^{B,d}$ and $u_{L2L}^{B,d}$.

Algorithm 3 presents pseudocode for computing all far-field interactions for the full domain up to N steps outward. We ensure the algorithm is performed before the downward $L2L$ pass in Algorithm 2, modifying to make sure that we begin the downward pass of Algorithm 2 at the rootlevel; this guarantees that all interactions in the far-field of our domain have been accounted for. In fact, Algorithm 3 can be performed directly after the $M2M$ phase of Algorithm 2 *or* right before the $L2L$ phase.

Algorithm 3 Far-Field Interaction Computation for Periodic kiFMM

```
Let $B$ be the full domain $D$ at the root level, $r$ with equiv. density $\phi^{B,u}$ and $B^*$ be the parent of $B$
for $i = 1$ to $N$ do
    Compute $u_{M2L}^{B,d}$ from copies of $B$ in $L_I^B$ using equation (2.51)
    Compute $\phi^{B^*,u}$ using equation (2.54)
    Let $B = B^*$ and $B^*$ be its parent
end for
Compute $\boldsymbol{\phi}^{B^*,d} = (\mathbf{K}_{L2L}^{B^*,B^*})^{-1} \left(u_{M2L}^{B^*,d}\right)$ at top-most level.
for $i = N$ to $1$ do
    Compute $u_{L2L}^{B,d}$ from its parent's $\boldsymbol{\phi}^{B^*,d}$ using equation (2.55)
    Compute $\boldsymbol{\phi}^{B,d}$ from $u_{M2L}^{B,d}$ and $u_{L2L}^{B,d}$ using equation (2.40)
    Let $B^* = B$ and $B$ its center-most child.
end for
```

For each level of the algorithm in three dimensions, we perform one step of the $M2M$ computation for the parent box with 27 children for $O(27p^2)$ computational complexity. We perform the FFT and its inverse once at each level as the upward equivalent densities are the same for all box copies at that level, and the number of boxes in the interaction list, L_I, at each level is $9^3 - 3^3 = 702$, so the total cost of the augmented $M2L$ phase at each level is $O(p^{3/2}\log(p) + 702p^{3/2})$. Finally, we perform one $L2L$ computation for a cost of $O(p^2)$, so the total cost of Algorithm 3 with N steps is $O\left(N \cdot (28p^2 + p^{3/2}(log(p) + 702)\right)$. As N is quite small, this additional cost is minimal when compared to the overall cost of Algorithm 2.

Remark 2.3. *No actual locations are stored, and once the matrices are computed, they can be reused for a scale-variant kernel (the densities are scaled for the appropriate levels). At the i^{th} step of the algorithm 3^i rings of the domain are accounted for as described earlier, and the j^{th} ring of the periodic domain contains $(2j+3)^3 - (2j+1)^3$ copies of the original domain, so after N steps, $\left(3^{N+1} - 3\right)/2$ copies of the domain are accounted for. After going up 12 levels, we have computed the influence from nearly $800,000$ rings outward. Every box is accounted for except for those directly adjacent to B. Experiments have shown that setting N to $10-12$ guarantees a high level of accuracy on the order of 10^{-8}.*

In order to account for periodic members of the V-list of a box at any depth in the domain, we note that at the end of the upward pass of the FMM algorithm, we have a description at every level of the multipole expansion (or upward equivalent density), so we only need to modify the interaction lists for every box in B at every level. This can be easily computed during the upward pass, and the number of interactions added is only near the boundary, so the additional computations are minimal, and the $M2L$ phase of the algorithm is left largely unchanged. Near-field interactions are also easily accounted for by augmenting the L_U^B, L_W^B, L_X^B with pointers to existing box structures and which types of near neighbors they are, allowing us to use existing precomputed tables and coefficients. The additional computational complexity does not alter the analysis of our algorithm's complexity from section 2.4.8. In our analysis there, we assumed that every box could have an upper bound of 189 boxes in L_I^B for example. While boundary boxes for the free-space solver have significantly fewer than this, for the periodic solvers, boundary boxes can now have full-sized computation lists; however, the total number of computations still fall within our upper bounds.

Finally, we note that some additional balancing may be necessary across the periodic boundaries, but this is a straightforward modification of our existing tree-balancing scheme in section A.1 as we need to just keep track of periodic nearest neighbors. Another option is to only balance the original domain and compute unbalanced nearest neighbor interactions *on the fly* as described in section A.1.2.

2.6.2 Homogeneous Dirichlet Boundary Conditions

We now consider the following situation for a box D with boundary ∂D:

$$-\Delta u = f \text{ in } D$$

$$u = C \text{ on } \partial D$$

In particular, we will consider $u = 0$ on ∂D. In order to solve this problem for a box B in two dimensions with force f, we can satisfy the boundary condition on its right boundary ($+x$-boundary) by reflecting the source points across the boundary and negating the forces. In order to satisfy the condition on the left boundary ($-x$ boundary), we reflect the source points across that boundary and negate the values as well. However, this now causes a violation of the boundary condition across the right boundary, so we repeat on the right side, then the left, etc. We can also do the same across the top and bottom boundaries. Additionally, we notice that if

$-f_B^R$ and $-f_B^L$ are the reflections of the source points and forces across the right boundary and left boundaries, then by symmetry, $-f_B^R = -f_B^L$. In fact, in two dimensions, the entire plane can be tiled as in figure 2.15. We notice that the pattern begins to repeat itself, and for a general number of d dimensions, we can in fact tile R^d with copies of a *supercell* with a small number of flipped/negated copies of the domain; this supercell is just of size 2^d or 4 in two dimensions and 8 in three dimensions.

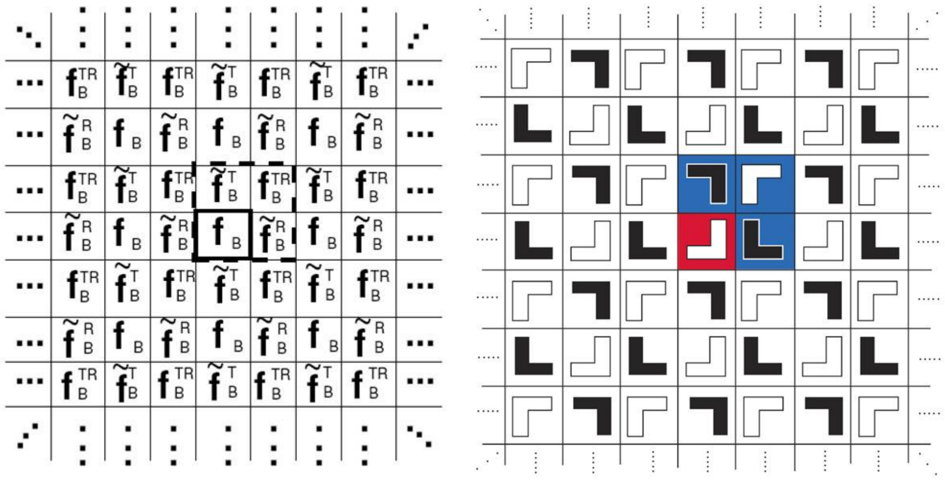


Figure 2.15: *Left*: For a box B with source distribution f_B in the dark black box, we reflect f_B across the right boundary and negate it as $\tilde{f_B^R}$. Repeating this across the other boundaries and considering symmetries results in the tiling above. The group of boxes surrounded by dotted lines, denoted as the *supercell*, tiles the plane, and we can embed this in the periodic solver. In three dimensions, the group is 8 boxes. *Right*: More easily seen in a symbolic format, the box in red represents our original source distribution, with the blue boxes representing the proper reflections (the shape in black implies the force values are also negated). Again, the union of the blue and red boxes denote the supercell which tiles the infinite domain.

We can now solely focus on the sources inside of the supercell since it tiles the infinite domain. As a result, for far-field influences on the supercell, we can embed this group into our periodic solver. Figure 2.15 again shows that this grouping is of size four for two dimensions, and in three dimensions, this supercell will be of size eight as we also have to reflect in the z-direction. On first glance, it looks as if this approach increases our computations and storage by an order of 2^d; however, careful implementation allows us to largely recover this overhead.

In order to see how we can avoid unnecessary computation, we begin by looking at a leaf box B that is on the boundary of our original domain (not the supercell), denoted as D as in figure 2.16 for two dimensions. As is seen, the box B has reflections of itself as direct neighbors as well as members of its interaction list. That is, B is influenced by *reflected* copies of itself.

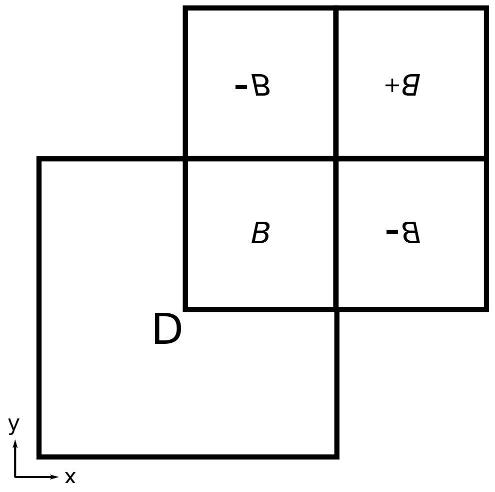


Figure 2.16: For some domain D in two dimensions, we show a single leaf B in D's sub-domain (all other leaves are not drawn here) as well as reflections of B across the boundaries such that the Dirichlet boundary condition of $u = 0$ is satisfied there. Here, we consider that B has itself as well as its reflections as direct neighbors in L_U^B, its near-neighbor interaction list.

We recall that for a box B and $U \in L_B^N$, equation 2.42 dictates that we calculate the influence from U via its polynomial approximation, γ^U. Now we consider if U is simply a reflection of B across the $+x$-boundary; that is, its positions are reflected in the x-direction and sources are negated as described above. We call this box $B' = U$ here. We recall from equation (2.24) that if B has smooth force $g^B(\mathbf{x})$ and center $\mathbf{c}_B$ at level ℓ,

$$g^B(\mathbf{x}) = \sum_{j=1}^{N_k} \gamma_j^B \beta_j \left(2^\ell(\mathbf{x} - \mathbf{c}_B)\right). \tag{2.56}$$

Defining $\mathbf{x} = (x, y, z)$, $\mathbf{c}_B = (cx_B, cy_B, cz_B)$, and $\bar{\mathbf{x}} = \mathbf{x} - \mathbf{c}_B$, then $\beta_j = P_a(\bar{x})P_b(\bar{y})P_c(\bar{z})$ where $a+b+c = j$ ($j < N_k = k(k+1)(k+2)/6$ (for approximation order k and monomial P_α of order α). For a point $\mathbf{x}' \in B'$, which is simply a reflection of $\mathbf{x} \in B$ across the $+x$-boundary, we now have

$$\begin{aligned}
\beta_j \left(2^\ell(\mathbf{x}' - \mathbf{c}_{B'})\right) &= P_a(\bar{x}')P_b(\bar{y}')P_c(\bar{z}') \\
&= (-1)^a P_a(\bar{x})P_b(\bar{y})P_c(\bar{z}) \\
&= (-1)^a \beta_j \left(2^\ell(\mathbf{x} - \mathbf{c}_B)\right),
\end{aligned}$$

where a is simply the order (or parity) of the x-monomial. Therefore, we can very simply compute $\gamma^{B'}$ as

$$\gamma^{B'} = -T_x \gamma^B,$$

for diagonal matrix $T_{jj} = (-1)^a$ for polynomial β_j with a again being the order of P_a. We note that we negate T_x because the sources, g^B are negated along with their locations being reflected. Further, it can easily be seen how to compute reflection matrices T_y and T_z for reflections across the $+y$ and $+z$-boundaries in three dimensions. Further, we construct T_{xy} as T_yT_x, etc. In total, we need only 7 such diagonal matrices in three dimensions: T_x, T_y, T_z, T_{yx}, T_{zx}, T_{zy} and T_{zyx}. Also, T_x, T_y, T_z, and T_{zyx} will always be negated (and are stored as such) due to g^B being negated across boundaries; all other matrices are not negated as the negations across boundaries cancel each other out.

We note that in practice, we need not compute and store all of the T matrices as they are diagonal with each entry being ± 1, so transforming γ^B can be done quickly in $O(N_k)$ operations. Additionally, the number of leaf boxes for which this is necessary is relatively small as it only is needed when a box B has members of L_U^B across the domain boundaries.

The other component we need is a way to quickly construct the upward equivalent densities we will need for a box B's interaction list. That is, consider that $V' \in L_V^B$ is a box which is a reflection of some box V (not necessarily in L_V^B) across one of the domain boundaries; without loss of generality, we will assume V' is a reflection of V across the $+x$ boundary of the domain D. In order to compute the influence of V' on B, we need to quickly construct $\phi^{V',u}$, the upward equivalent density of V'. There are two cases we need to consider: if V' is a leaf or a non-leaf.

First, if V' is a leaf, we normally construct $\phi^{V',u}$ from equations (2.29) and (2.30). As with the near-field interactions, we need to construct a transformation of $\gamma^{V'}$ from γ^V. With the construction of $\phi^{V',u}$, however, we note that the precomputed operator, $\mathbf{F}_{S2M}^{V'}$ (modifying equation (2.30)) has entries

$$F_j^{V'}(\mathbf{x}') = \int_{V'} \beta_j \left(2^\ell(\mathbf{y} - \mathbf{c}_{V'})\right) K(\mathbf{x}', \mathbf{y}) d\mathbf{y}, \quad \text{for } \mathbf{x}' \in \mathbf{x}^{B',u}. \tag{2.57}$$

As above, we can see that $F_j^{V'} = (-1)^a F_j^V$ where a is the order of the monomial P_a used in the construction of $\beta_j(\mathbf{y} - \mathbf{c}_V) = P_a(\bar{x}) P_b(\bar{y}) P_c(\bar{z})$. That is, using the same definitions for T_x, etc. as above:

$$\mathbf{F}_{S2M}^{V'} = T_x \mathbf{F}_{S2M}^V.$$

Further, it can easily be seen from this relationship and equation (2.31),

$$\begin{aligned} K[\mathbf{y}^{V',u}, \phi^{V',u}](\mathbf{x'}_i) &= \sum_{j=1}^{N_k} F_j^{V'}(\mathbf{x'}_i)\gamma_j^{V'} \\ &= \sum_{j=1}^{N_k} F_j^{V}(\mathbf{x}_i)\gamma_j^{V}, \quad \text{for } \mathbf{x'}_i \in \mathbf{x}^{V',u} \text{ and } \mathbf{x}_i \in \mathbf{x}^{V,u} \end{aligned}$$

As $\mathbf{x}'$ is a reflection of $\mathbf{x}$, and the upward equivalent surface $\mathbf{y}^{V',u}$ is also a reflection of $\mathbf{y}^{V,u}$, $\phi^{V',u}$ is a permutation of $\phi^{V,u}$. In figure 2.17, we indicate how the locations of the check and equivalent surfaces are reflected.

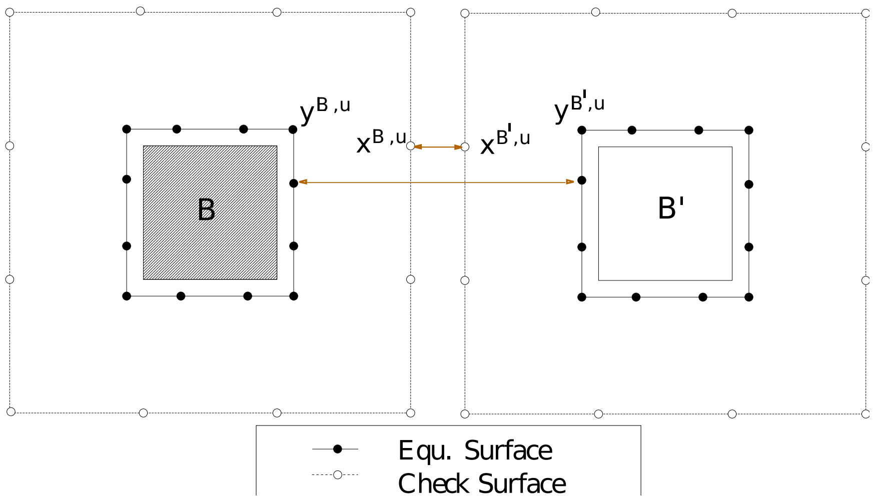


Figure 2.17: For some box B with upward check surface $\mathbf{x}^{B,u}$ and equivalent surface $\mathbf{y}^{B,u}$, let B' be a single reflection of B (and negation of its smooth force distribution) across some boundary. Then, the relative positions of the check and equivalent surfaces, $\mathbf{x}^{B',u}$ and $\mathbf{y}^{B',u}$ respectively are mapped back to their reflections in order to permute $\phi^{B,u}$ to $\phi^{B',u}$

More specifically, assuming we have already computer $\phi^{V,u}$ in the $S2M$ pass, $\phi^{V',u}$ is simply computed as (still assuming we are reflecting across the $+x$ boundary)

$$-\mathcal{P}_x \phi^{V,u},$$

for permutation matrix $\mathcal{P}_x$ where the entry $\mathcal{P}_{ij} = 1$ if $\mathbf{x}'_i$ is the reflected location of $\mathbf{x}_j$. We again note that we need only seven such permutation matrices, corresponding to our reflections: $-\mathcal{P}_x$, $-\mathcal{P}_y$, $-\mathcal{P}_z$, $\mathcal{P}_{yx}$, $\mathcal{P}_{zx}$, $\mathcal{P}_{zy}$, $-\mathcal{P}_{zyx}$. Those which are negative represent a single or odd combination of reflections, in which the smooth source $g^{V'}$ is a negation of g^V along with a reflection of source locations.

Remark 2.4. *The permutation matrices, $\mathcal{P}$ in practice are precomputed, stored, and loaded at runtime for each specific numerical precision, n_p; that is, $\mathcal{P}$ is of size $p \times p$ for p sample locations on the upward check surface. In practice, as these matrices are largely zero, we store them as arrays and do a fast $O(p)$ swap of array entries to build $\phi^{V',u}$ from $\phi^{V,u}$*

We have discussed how to build the reflected upward equivalent densities when a box is a leaf, but when a box is a non-leaf, the discussion is equivalent. If V' lies in a reflected domain of D, so do all of its children, C', which are reflections of some boxes $C \in D$; hence, the operator, $\mathcal{P}$ which permutes $\phi^{C,u}$ to $\phi^{C',u}$ also permutes $\phi^{V,u}$ to $\phi^{V',u}$.

For the upward $M2L$ pass now, we compute the interactions for all boxes B from their interaction lists L_V^B for all boxes enclosed inside of the original domain D at a specific level ℓ. If we notice that a box $V \in L_V^B$ also has a reflection V' which is in some box's interaction list, we permute $\phi^{V,u} \to \phi^{V',u}$ and compute the influence from V'. This allows us to maintain the same general structure for performing shared memory parallelization for the $M2L$ step as described in section 3.3.

For a box B, all of its ancestors in the tree T lie directly inside of the domain D or outside of the supercell that tiles the infinite domain, as in the periodic solver. Hence, in the downward pass, there is no permutation of downward equivalent densities.

Finally, we note that as in the periodic solver, at the completion of the $M2M$ computation, we have a description for the entire domain D's equivalent density, $\phi^{D,u}$. In order to use the periodic solver for computing the potential from the far-field of our domain, we let S be the supercell of D and its rotated/negated copies as described above. We then compute the upward equivalent density of S using equation (2.58).

$$\boldsymbol{\phi}^{S,u} = \sum_{D_i} (\mathbf{K}_{M2M}^{S,S})^{-1} \mathbf{K}_{M2M}^{D_i,S} \boldsymbol{\phi}^{D_i,u} = \sum_{D_i} \mathbf{T}_{M2M}^{D_i,S} \left(\mathcal{P}_i \phi^{D,u} \right). \tag{2.58}$$

That is, for $i \in 0, x, y, z, yx, zx, zy, zyx$, we apply the permutation operator to $\boldsymbol{\phi}^{D,u}$ and apply the $M2M$ operator ($i = 0$ implies an identity operator or no permutation). As the supercell's children have the same orientation as the regular $M2M$ operator in section 2.4.2, we can reuse the existing operators and need no additional storage. We now call Algorithm 3 with S as the root of our domain with density $\boldsymbol{\phi}^{S,u}$. At the end of the algorithm, we have a description for the downward equivalent density at S, $\boldsymbol{\phi}^{S,d}$, which we then translate solely to the original domain D's downward check potential, $u_{L2L}^{D,d}$ using equation (2.39). As before, we recover the downward equivalent density for D using (2.40) applied to $u_{L2L}^{D,d}$ and $u_{M2L}^{D,d}$ (computed as described above). We note that the additional cost to the existing periodic volume solver is minimal: one additional $M2M$ computation of cost $O(8p^2)$ and one $L2L$ operation of cost $O(p^2)$.

Finally, we note that for the Dirichlet solver, there is no need to worry about tree-balancing across the domain boundary as in the periodic case. In order to see why, we consider the extreme case presented in figure 2.18. As can be seen, once all rotations are performed to compose the supercell, a box B will only gain rotated copies of boxes in its existing near-neighbor list; hence, the supercell is already fully-balances, assuming the original domain has been balanced.

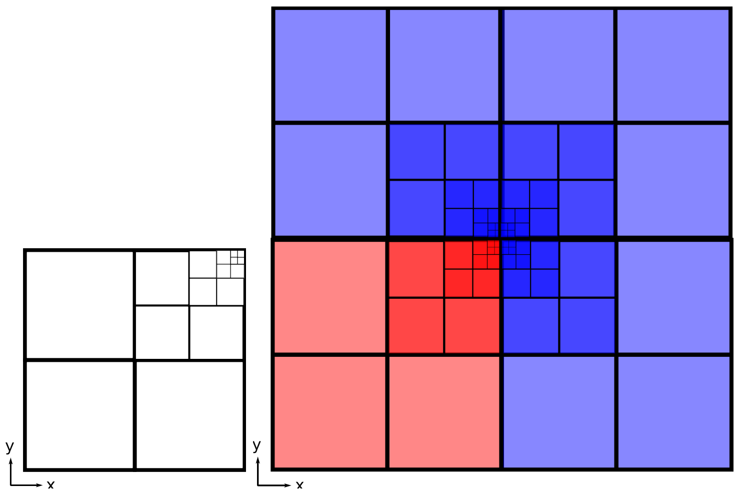


Figure 2.18: *Left*: A fully-balanced domain with multiple levels approaching one corner of the boundary; *Right*: After performing all rotations to the domain, the supercell is fully-balanced across all boundaries.

3 Numerical Results for Volume Solver in the Box

The three-dimensional kernel-independent elliptic PDE volume solver algorithm has been implemented in C++, and we have tested several kernels and source and target point distributions in section 3.1. The accuracy of various operators and the effects of regularization are considered in section 3.2. Our tests were run on an Intel Xeon-based X7560 (2.27GHz 64 bit) system with 16 CPUs and 128GB of RAM; the major computation loops are accelerated with OpenMP (Chapman et al., 2007) as discussed in section 3.3.

3.1 Overall Approximation Error

In order to test the accuracy and speed of the full algorithm, we consider three specific equations: the Poisson equation (2.7), the Modified Helmholtz equation (2.8), and the Stokes equations (2.9). These equations represent our canonical set as the corresponding kernels highlight the ability of our algorithm to handle scalar, scale-variant, and matrix kernels, respectively. Following results for these three equations in free-space, we then consider periodic and Dirichlet boundary conditions on the unit box as well as an example in which we combine smooth and singular sources.

3.1.1 Poisson Equation

We first test the free-space Poisson solver on three different types of problems designed to show how our algorithm handles increasing levels of complexity in the force distribution.

We use an adaptive-refinement strategy similar to (Ethridge and Greengard, 2001). For this, we compute a k^{th}-order polynomial approximation, γ^B, to the force $g^B(\mathbf{x})$ sampled on a $k \times k \times k$ grid. We let $\tilde{g}^B$ be the force evaluated on a refined $2k \times 2k \times 2k$ grid. If $||g^B(\mathbf{x}) - \tilde{g}^B(\mathbf{x})||_2 > \epsilon_{rhs}$, B is subdivided, and the octree is balanced as needed. Three force distributions, used in Examples (1-3) below are shown in Figure 3.1.

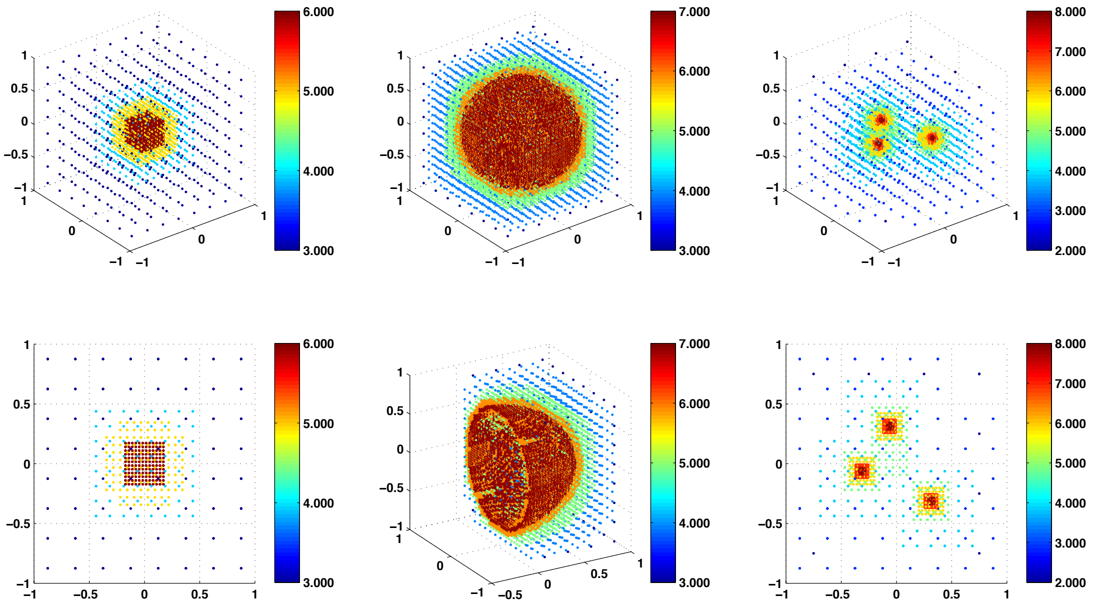


Figure 3.1: Sample force distributions based on adaptive refinement. Each point, colored by its tree level, ℓ, indicates the center of a leaf box, B. *Left*: A single sharply-peaked Gaussian function; *Middle:* A discontinuous force distribution, equal to one inside a sphere and zero outside; *Right:* A discontinuous force distribution involving oscillatory functions restricted to the interiors of a set of three spheres.

Example 1. The first experiment tests the accuracy of our method for solving the Poisson equation (equation (2.7) with kernel (2.10)) with a fast-decaying smooth right-hand side.

$$-\Delta u(\mathbf{x}) = \sum_{i=0}^{8} -e^{-L||\mathbf{x}-\mathbf{x}_i||^2}\left(4L\,||\mathbf{x}-\mathbf{x}_i||^2 - 6L\right), L = 250$$

with solution

$$u(\mathbf{x}) = \sum_{i=0}^{8} -e^{-L||\mathbf{x}-\mathbf{x}_i||^2},$$

where $\mathbf{x}_i = (\pm\frac{3}{40}, \pm\frac{3}{40}, \pm\frac{3}{40})$ inside of the $[-1, 1]^3$ box. This test requires a high degree of adaptivity to achieve good accuracy with a limited number of points.

In Table 3.1, ϵ_{fmm} is the precision of the translation operators, ϵ_{rhs} is the refinement criterion for the adaptive refinement of the source distribution, and M_ℓ is the number of leaves in the tree T with L_T levels. The number of points N_{pts} is computed as $M_\ell k^3$ where k^3 is the number of points per leaf, chosen to be sufficiently large to build the polynomial approximation of order k. The computation time, T_{FMM}, is given in seconds, and the *Rate* is in points per second. E_2 and E_∞ are the relative L^2 and L^∞ errors, respectively. Timings include FMM evaluation times only; when the precision ϵ_{fmm} remains constant, the rate of work per source and target point remains close to constant, as we would expect since the FMM algorithm scales linearly.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	Rate
Fourth-Order Force Approximation								
10^{-2}	10^{-2}	736	47104	6	$2.3E-02$	$2.4E-02$	$9.15E-03$	$5.15E+06$
10^{-4}	10^{-2}	736	47104	6	$1.1E-03$	$6.8E-04$	$2.43E-02$	$1.94E+06$
10^{-4}	10^{-4}	3088	197632	7	$1.1E-04$	$2.7E-04$	$9.66E-02$	$2.05E+06$
10^{-6}	10^{-4}	3088	197632	7	$3.8E-05$	$2.5E-05$	$3.79E-01$	$5.21E+05$
10^{-6}	10^{-6}	19328	1236992	8	$3.8E-06$	$3.6E-06$	$2.39E+00$	$5.18E+05$
10^{-8}	10^{-6}	19328	1236992	8	$3.7E-06$	$1.3E-06$	$6.21E+00$	$1.99E+05$
10^{-8}	10^{-8}	143088	9157632	9	$1.6E-07$	$8.8E-08$	$4.53E+01$	$2.02E+05$
Sixth-Order Force Approximation								
10^{-4}	10^{-4}	1408	304128	6	$1.1E-04$	$2.3E-04$	$1.10E-01$	$2.76E+06$
10^{-6}	10^{-4}	1408	304128	6	$1.4E-05$	$3.5E-05$	$1.87E-01$	$1.62E+06$
10^{-6}	10^{-6}	4936	1066176	7	$9.0E-07$	$2.2E-06$	$6.67E-01$	$1.60E+06$
10^{-8}	10^{-6}	4936	1066176	7	$3.3E-07$	$1.6E-07$	$1.62E+00$	$6.60E+05$
10^{-8}	10^{-8}	20112	4344192	8	$2.4E-08$	$6.2E-08$	$6.87E+00$	$6.33E+05$
10^{-10}	10^{-8}	20112	4344192	8	$1.8E-08$	$1.0E-08$	$1.58E+01$	$2.76E+05$
10^{-10}	10^{-10}	92072	19887552	9	$6.3E-09$	$9.7E-09$	$7.51E+01$	$2.65E+05$
Eighth-Order Force Approximation								
10^{-6}	10^{-6}	2024	1036288	7	$9.2E-07$	$3.5E-06$	$4.67E-01$	$2.22E+06$
10^{-8}	10^{-6}	2024	1036288	7	$3.7E-07$	$8.2E-07$	$7.42E-01$	$1.40E+06$
10^{-8}	10^{-8}	5440	2785280	7	$1.9E-08$	$6.6E-08$	$2.11E+00$	$1.32E+06$
10^{-10}	10^{-8}	5440	2785280	7	$7.7E-09$	$7.6E-09$	$4.36E+00$	$6.40E+05$
10^{-10}	10^{-10}	22800	11673600	8	$4.1E-09$	$5.6E-09$	$1.94E+01$	$6.00E+05$
10^{-12}	10^{-10}	22800	11673600	8	$2.6E-09$	$4.7E-09$	$4.16E+01$	$2.81E+05$
10^{-12}	10^{-12}	50352	25780224	9	$2.1E-09$	$4.6E-09$	$9.21E+01$	$2.80E+05$

Table 3.1: Free-Space Poisson Equation, Example 1: Gaussian bump at the origin numerical results.

Example 2. In this example, we consider a discontinuous right-hand side, with $g(\mathbf{x}) = 1$ inside a sphere of radius $R = 0.75$, and $g(\mathbf{x}) = 0$ outside the sphere. Letting $r = ||\mathbf{x}||$, the problem becomes

$$-\Delta u(\mathbf{x}) = \left\{ \begin{array}{cc} 1 & \text{if } r \leq R \\ 0 & \text{else} \end{array} \right\}$$

with solution

$$-\Delta u(\mathbf{x}) = \left\{ \begin{array}{cc} \left(R^2 - r^2\right)/6 + R^2/3 & \text{if } r \leq R \\ R^3/3r^2 & \text{else} \end{array} \right\}.$$

While this problem can be handled analytically, it serves as a useful test of performance on adaptive data structures that are refined in the neighborhood of a surface. The number of points indicates the total number of points both inside and outside the sphere. Since the coefficient representation of the force for a leaf node entirely outside of the sphere is zero, these boxes are ignored in all evaluation phases; this increases the computed rate somewhat. A greater speedup is achieved from the observation that leaf nodes entirely in the interior have a constant source distribution, so that only one polynomial coefficient is non-zero. This significantly accelerates both the near-field and S2M calculation stages. Results are shown in Table 3.2.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	L^2	L^∞	T_{FMM}	Rate
$k = 4$, Fourth-Order Force Approximation								
10^{-2}	10^{-2}	232	14848	4	$1.1E-02$	$1.6E-02$	$1.83E-03$	$8.1E+06$
10^{-3}	10^{-3}	1184	75776	5	$4.3E-03$	$4.6E-03$	$1.10E-02$	$6.9E+06$
10^{-4}	10^{-4}	5888	376832	6	$1.4E-04$	$2.5E-04$	$1.13E-01$	$3.3E+06$
$k = 6$, Sixth-Order Force Approximation								
10^{-5}	10^{-5}	11432	2469312	7	$1.6E-05$	$3.3E-05$	$6.21E-01$	$4.0E+06$
10^{-6}	10^{-6}	80088	17299008	8	$7.2E-06$	$1.8E-05$	$6.06E+00$	$2.8E+06$
$k = 8$, Eighth-Order Force Approximation								
10^{-7}	10^{-7}	127856	65462272	8	$9.4E-07$	$3.3E-06$	$1.89E+01$	$3.5E+06$
10^{-8}	10^{-8}	528984	270839808	10	$3.7E-07$	$9.8E-07$	$1.23E+02$	$2.2E+06$
10^{-9}	10^{-9}	2074360	1062072320	10	$3.2E-08$	$2.6E-07$	$6.49E+02$	$1.6E+06$

Table 3.2: Free-Space Poisson Equation, Example 2: Discontinuous Force numerical results.

Example 3. For our third example, we replicate an experiment from (McCorquodale et al., 2007) for a highly-oscillatory force with discontinuities along multiple surfaces.

$$f_m(r) = \left\{ \begin{array}{ll} ((r - r^2)\sin(2m\pi r))^2 & \text{if } r < 1 \\ 0 & \text{if } r \geq 1 \end{array} \right\}$$

$$-\Delta u(\mathbf{x}) = \frac{1}{R^3}\sum_{i=0}^{2} f_m(|\mathbf{x} - \mathbf{c}_i|/R), \tag{3.1}$$

where $\mathbf{c}_0 = (3/16, 7/16, 13/16)$, $\mathbf{c}_1 = (7/16, 13/16, 3/16)$, $\mathbf{c}_2 = (13/16, 3/16, 7/16)$, $R = 0.05$, and the wavelength of f_m is $\lambda_m = R/(2m) = (1/40m)$. Defining

$$\begin{aligned} \phi_m(r) &= \frac{(10r^6 - 28r^5 + 21r^4 - 7)}{840} + \frac{(60r - 120)}{r\lambda_m^6} - \frac{9}{\lambda_m^4} \\ &+ \left[\frac{(300r - 120)}{r\lambda_m^6} - + \frac{(30r^2 - 36r + 9)}{\lambda_m^4} + \frac{(r^4 - 2r^3 + r^2)}{2\lambda_m^2}\right]\cos(r\lambda_m) \\ &+ \left[-\frac{360}{r\lambda_m^7} + \frac{(120r^2 - 96r + 12)}{r\lambda_m^5} + \frac{(5r^3 + 8r^2 - 3r)}{\lambda_m^3}\right]\sin(r\lambda_m), \quad \text{and} \\ \theta_m(r) &= \left(\frac{360}{\lambda_m^6} - \frac{12}{\lambda_m^4} - \frac{1}{120}\right)/r, \end{aligned}$$

we write the solution to equation (3.1) as

$$u^{exact}(\mathbf{x}) = \left\{ \begin{array}{ll} \phi(||\mathbf{x} - \mathbf{c}_0||/R) + \sum_{i=1,2}\theta(||\mathbf{x} - \mathbf{c}_i||/R) & \text{if } ||\mathbf{x} - \mathbf{c}_0|| < R \\ \phi(||\mathbf{x} - \mathbf{c}_1||/R) + \sum_{i=0,2}\theta(||\mathbf{x} - \mathbf{c}_i||/R) & \text{if } ||\mathbf{x} - \mathbf{c}_1|| < R \\ \phi(||\mathbf{x} - \mathbf{c}_2||/R) + \sum_{i=0,1}\theta(||\mathbf{x} - \mathbf{c}_i||/R) & \text{if } ||\mathbf{x} - \mathbf{c}_2|| < R \\ \sum_{i=0}^{2}\theta(||\mathbf{x} - \mathbf{c}_i||/R) & \text{else} \end{array} \right\}$$

In order to compare our results to (McCorquodale et al., 2007), we use the error metric introduced there. Let ϵ^B be the vector of errors calculated as the difference between the calculated and exact solutions on B and calculate the following norm over all leaf boxes.

$$||\epsilon^B_{all}||_2 = \sum_B \left(\int \frac{\epsilon^B}{||u^{exact}||_\infty} \right)^{\frac{1}{2}}.$$

As indicated in (McCorquodale et al., 2007), $\forall i, j = 0, 1, 2$, $i \neq j$

$$||u^{exact}||_\infty = \left| \left(-\frac{6}{\lambda_m^4} - \frac{1}{120} \right) / R + \left(\frac{720}{\lambda_m^6} - \frac{24}{\lambda_m^4} - \frac{1}{120} \right) / ||\mathbf{c}_i - \mathbf{c}_j|| \right|$$

Our automatic refinement strategy refines within or near the sphere surfaces, with refinement taking place in the exterior of the spheres only for the purpose of tree-balancing. We build coefficients only on leaf boxes which contain non-zero source distributions: either interior to or intersecting one of the three spheres. Results are shown in Table 3.3.

While the performance of our code cannot be compared easily to the optimized and parallelized scheme presented in (McCorquodale et al., 2007), we have implemented much higher order accurate schemes. Thus, as expected, we are able to reach comparable accuracies with significantly fewer points. To compare the number of points required, we consider the number of points in the finest level solve of their three-level examples. For $m = 7$, we achieve accuracy on par with their most accurate tests with approximately $1/100$ as many points. For $m = 15$, we require approximately $1/5$ as many points, and with $1/4$ as many points, we achieve about two orders of magnitude greater accuracy. For $m = 30$, we achieve equivalent results with approximately $1/4$ as many points. Additionally, we extended the examples for an even higher wavenumber component ($m = 60$), decreasing the wavelength to 4.1710^{-4}, and achieving good results with fewer than 10^9 points.

m	ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	k	$\|\|\epsilon^B_{all}\|\|_2$	$\|\|\epsilon^B_{all}\|\|_\infty$
1	10^{-6}	10^{-6}	3984	254976	8	4	$2.3E-07$	$2.2E-05$
1	10^{-8}	10^{-8}	7296	1575936	9	6	$1.1E-08$	$7.1E-07$
1	10^{-10}	10^{-10}	24144	12361728	9	8	$1.6E-10$	$1.8E-08$
7	10^{-6}	10^{-6}	93816	6004224	10	4	$2.2E-06$	$1.8E-04$
7	10^{-8}	10^{-8}	195984	42332544	10	6	$9.3E-09$	$1.1E-06$
7	10^{-10}	10^{-10}	228312	116895744	10	8	$1.1E-10$	$4.3E-08$
15	10^{-6}	10^{-6}	140568	8996352	10	4	$6.4E-07$	$8.6E-05$
15	10^{-8}	10^{-8}	1092456	235970496	11	6	$7.2E-08$	$4.2E-06$
15	10^{-10}	10^{-10}	1596672	817496064	11	8	$4.8E-10$	$6.1E-08$
30	10^{-6}	10^{-6}	148272	9489408	10	4	$5.8E-07$	$6.2E-05$
30	10^{-8}	10^{-8}	1491216	322102656	11	6	$2.1E-08$	$3.8E-06$
30	10^{-10}	10^{-10}	1720152	880717824	12	8	$4.9E-09$	$2.0E-06$
60	10^{-6}	10^{-6}	150288	9618432	11	4	$6.0E-07$	$9.6E-05$
60	10^{-8}	10^{-8}	1502592	324559872	11	6	$9.2E-08$	$1.4E-05$
60	10^{-10}	10^{-9}	1659312	849567744	11	8	$7.7E-08$	$1.7E-05$

Table 3.3: Free-Space Poisson Equation, Example 3: Discontinuities along several spherical surfaces containing oscillating source distributions numerical results.

3.1.2 Modified Helmholtz Equation

For the Modified Helmholtz equation (equation (2.8) with kernel (2.11)), we use a right-hand side similar to that of Example 1 for the Poisson equation, setting the Helmholtz parameter (inverse Debye length) to $\alpha = \pi$:

$$\alpha\, u(\mathbf{x}) - \Delta u(\mathbf{x}) = \sum_{i=0}^{8} -e^{-L||\mathbf{x}-\mathbf{x}_i||^2}\left(4L\,||\mathbf{x}-\mathbf{x}_i||^2 - 6L - \alpha\right), L = 250$$

with solution

$$u(\mathbf{x}) = \sum_{i=0}^{8} -e^{-L||\mathbf{x}-\mathbf{x}_i||^2},$$

for $\mathbf{x}_i = (\pm\frac{3}{40}, \pm\frac{3}{40}, \pm\frac{3}{40})$ inside of the $[-1,1]^3$ box. All translation matrices are computed to a precision of $\epsilon_{fmm}/10$. These matrices can be computed at run-time in a lazy manner; if α is known before run-time, these tables can be precomputed, stored, and loaded as necessary. Additionally, since the right-hand side is the same as in Example 1 for the Poisson equation, we use the same point distribution; hence, the timings are essentially the same as Experiment 1 and are omitted here. Results are shown in Table 3.4.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞
Fourth-Order Force Approximation						
10^{-2}	10^{-2}	736	47104	6	$2.3E-02$	$2.4E-02$
10^{-4}	10^{-2}	736	47104	6	$6.0E-04$	$5.6E-04$
10^{-4}	10^{-4}	3088	197632	7	$1.1E-04$	$1.8E-04$
10^{-6}	10^{-4}	3088	197632	7	$2.4E-05$	$2.1E-05$
10^{-6}	10^{-6}	19328	1236992	8	$2.3E-06$	$3.4E-06$
10^{-8}	10^{-6}	19328	1236992	8	$2.0E-06$	$8.7E-07$
10^{-8}	10^{-8}	143088	9157632	9	$9.9E-08$	$9.0E-08$
Sixth-Order Force Approximation						
10^{-4}	10^{-4}	1408	304128	6	$1.1E-04$	$2.4E-04$
10^{-6}	10^{-4}	1408	304128	6	$1.4E-05$	$3.5E-05$
10^{-6}	10^{-6}	4936	1066176	7	$8.5E-07$	$2.5E-06$
10^{-8}	10^{-6}	4936	1066176	7	$1.4E-07$	$1.3E-07$
10^{-8}	10^{-8}	20112	4344192	8	$1.5E-08$	$7.5E-08$
10^{-10}	10^{-8}	20112	4344192	8	$8.9E-09$	$7.4E-09$
10^{-10}	10^{-10}	92072	19887552	9	$2.2E-09$	$5.7E-09$
Eighth-Order Force Approximation						
10^{-6}	10^{-6}	2024	1036288	7	$1.9E-06$	$4.7E-06$
10^{-8}	10^{-6}	2024	1036288	7	$1.8E-07$	$4.6E-07$
10^{-8}	10^{-8}	5440	2785280	7	$1.2E-08$	$1.4E-08$
10^{-10}	10^{-8}	5440	2785280	7	$7.7E-09$	$7.6E-09$
10^{-10}	10^{-10}	22800	11673600	8	$3.4E-09$	$5.6E-09$
10^{-12}	10^{-10}	22800	11673600	8	$1.3E-09$	$2.0E-09$
10^{-12}	10^{-12}	50352	25780224	9	$2.0E-09$	$2.6E-09$

Table 3.4: Free-Space Modified Helmholtz Equation example: Gaussian bump at the origin numerical results.

3.1.3 Stokes Equations

We test the ability of our code to handle matrix kernels by solving the Stokes equations (equation (2.9) with kernel (2.12)) with the following divergence-free fast-decaying force.

$$-\mu\Delta u(\mathbf{x}) + \nabla p(\mathbf{x}) = \sum_{i=0}^{8} \left(8L^3 \left|\left|\mathbf{x} - \mathbf{x}_i\right|\right|^2 - 20L^2\right) e^{-L||\mathbf{x}-\mathbf{x}_i||^2} \left[\nabla \times (\mathbf{x} - \mathbf{x}_i)\right]$$

with solution

$$u(\mathbf{x}) = \frac{2L}{\mu} \sum_{i=0}^{8} e^{-L||\mathbf{x}-\mathbf{x}_i||^2} \left[\nabla \times (\mathbf{x} - \mathbf{x}_i)\right]$$

for $\mathbf{x}_i = (\pm\frac{3}{40}, \pm\frac{3}{40}, \pm\frac{3}{40})$, $\mu = 1$, $L = 125$ inside of the $[-1, 1]^3$ box. Errors are again similar to those seen in the fast-decaying experiments from examples 1 and 4; timings are worse, as expected, since we are dealing with nine times as many degrees of freedom per point. Results are shown in in Table 3.5.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	Rate
Fourth-Order Force Approximation								
10^{-2}	10^{-2}	2038	130432	6	$1.3E-01$	$1.5E-01$	$1.35E-01$	$9.67E+05$
10^{-4}	10^{-2}	2038	130432	6	$1.0E-03$	$8.4E-04$	$9.49E-01$	$1.37E+05$
10^{-4}	10^{-4}	10606	678784	7	$9.1E-04$	$9.4E-04$	$5.11E+00$	$1.33E+05$
10^{-6}	10^{-4}	10606	678784	7	$8.4E-06$	$1.1E-05$	$2.24E+01$	$3.04E+04$
10^{-6}	10^{-6}	69140	4424960	8	$7.5E-06$	$1.4E-05$	$1.47E+02$	$3.02E+04$
10^{-8}	10^{-6}	69140	4424960	8	$2.2E-07$	$4.4E-07$	$4.83E+02$	$9.16E+03$
10^{-8}	10^{-8}	484408	31002112	9	$1.4E-07$	$4.4E-07$	$3.23E+03$	$9.61E+03$
Sixth-Order Force Approximation								
10^{-4}	10^{-4}	2696	582336	7	$4.9E-04$	$8.6E-04$	$1.57E+00$	$3.70E+05$
10^{-6}	10^{-4}	2696	582336	7	$4.3E-06$	$9.9E-06$	$5.99E+00$	$9.71E+04$
10^{-6}	10^{-6}	10396	2245536	7	$8.1E-06$	$1.3E-06$	$2.36E+01$	$9.51E+04$
10^{-8}	10^{-6}	10396	2245536	7	$1.6E-07$	$4.1E-07$	$7.79E+01$	$2.88E+04$
10^{-8}	10^{-8}	59830	12923280	8	$1.4E-07$	$4.3E-07$	$4.11E+02$	$3.15E+04$
10^{-10}	10^{-8}	59830	12923280	8	$5.4E-09$	$1.3E-08$	$1.03E+03$	$1.25E+04$
10^{-10}	10^{-10}	295100	63741600	9	$5.2E-09$	$1.1E-08$	$5.31E+03$	$1.20E+04$
Eighth-Order Force Approximation								
10^{-6}	10^{-6}	4894	2505728	7	$4.8E-06$	$1.5E-05$	$1.43E+01$	$1.75E+05$
10^{-8}	10^{-6}	4894	2505728	7	$8.0E-08$	$4.3E-07$	$4.14E+01$	$6.06E+04$
10^{-8}	10^{-8}	12860	6584320	7	$1.5E-07$	$4.5E-07$	$1.03E+02$	$6.41E+04$
10^{-10}	10^{-8}	12860	6584320	7	$6.0E-09$	$1.5E-08$	$2.37E+02$	$2.78E+04$
10^{-10}	10^{-10}	55854	28597248	8	$4.7E-09$	$1.6E-08$	$1.05E+03$	$2.73E+04$
10^{-12}	10^{-10}	55854	28597248	8	$6.3E-09$	$8.8E-09$	$1.96E+03$	$1.46E+04$
10^{-12}	10^{-12}	132490	67834880	9	$5.6E-09$	$7.0E-09$	$4.46E+03$	$1.52E+04$

Table 3.5: Free-Space Stokes Equation example numerical results.

3.1.4 Periodic Boundary Conditions

In section 2.6.1, we discussed how we use a modification of the existing kernel-independent framework to impose periodic boundary conditions. To review, it is straightforward to extend the solver infrastructure described above to the case of periodic boundary conditions, using the the classical method of images of Lord Rayleigh (Rayleigh, 1892), following the discussion of (Ethridge and Greengard, 2001). The influence of all separated image boxes can be incorporated using either a recursive approach (Helsing, 1994) or a scheme based on lattice sums (Greengard and Rokhlin, 1987). In either case, the additional work depends only on ϵ_{fmm} and not on the number of degrees of freedom. The main difference is that the unit cell B now has near neighbors, whose influence must be accounted for. This, too, has relatively little impact on performance. A small number of additional boxes are added to both the interaction and near neighbor lists, but no additional data structures are created; instead, everything is handled via careful book-keeping to minimize additional memory consumption.

As an example, we consider the Poisson problem with periodic boundary conditions

$$-\Delta u(\mathbf{x}) = 3CM^2\pi^2 \sin(\pi Mx)\sin(\pi My)\sin(\pi Mz),$$

for which the solution is

$$u(\mathbf{x}) = C\sin(\pi Mx)\sin(\pi My)\sin(\pi Mz).$$

We conduct our experiments for a non-trivial oscillatory force, choosing $C = 7$ AND $M = 5$ on the domain $[-1, 1]^3$ with varying degrees of depth and precision. Plots of sample slices for the force and solution are shown in figure 3.2, and we check the relative L_2 and L_∞ with results shown in Table 3.6.

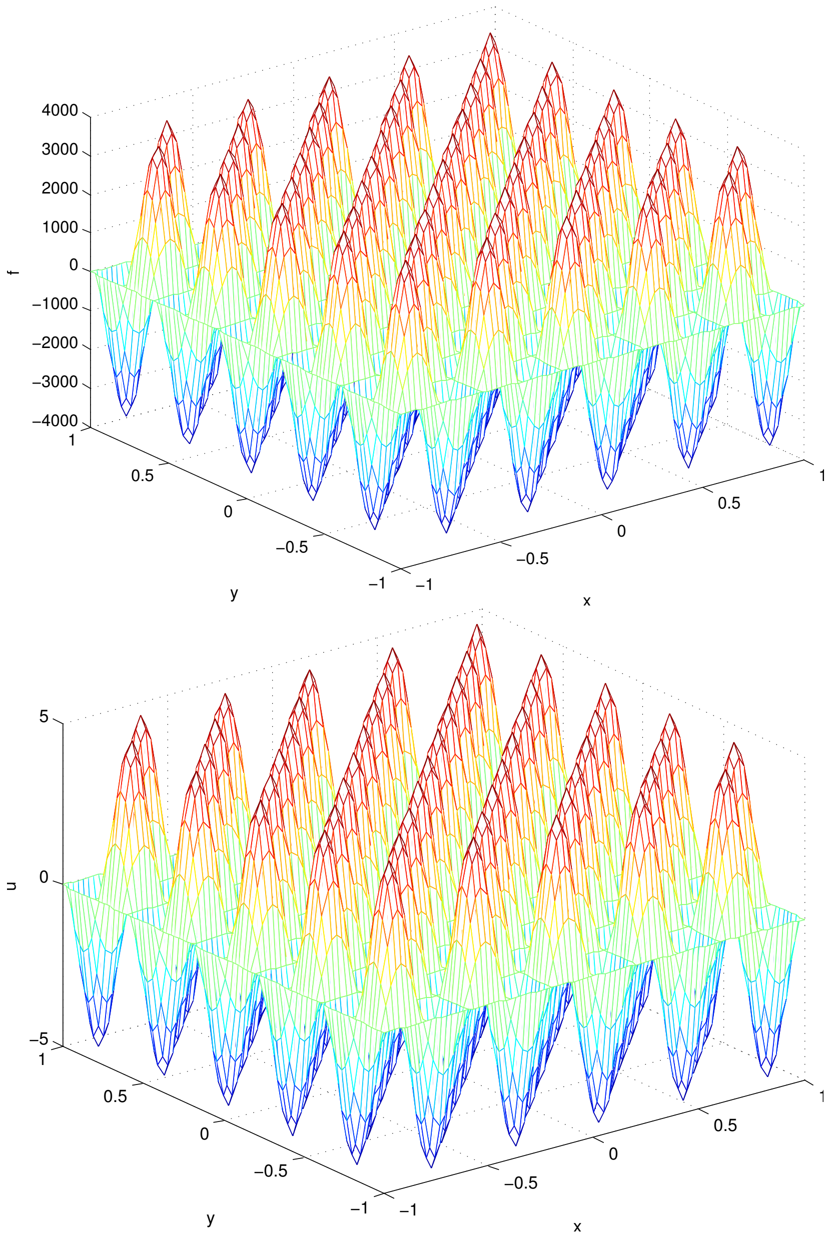


Figure 3.2: Plot of Periodic Boundary Conditions Example 1. We look at a single slice of the force on the top and solution on the bottom for $z = -0.25$. The nearly-uniform nature of this test force results in a low level of adaptive refinement.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	Rate
				Fourth-Order Force Approximation				
10^{-2}	10^{-2}	32768	2097152	6	$2.6E-02$	$3.3E-02$	$6.37E-01$	$3.29E+06$
10^{-4}	10^{-2}	32768	2097152	6	$1.1E-03$	$1.4E-03$	$2.30E+00$	$9.12E+05$
10^{-4}	10^{-4}	262144	16777216	7	$2.9E-04$	$7.6E-04$	$1.55E+01$	$1.08E+06$
10^{-6}	10^{-4}	262144	16777216	7	$1.6E-05$	$2.4E-05$	$3.79E+01$	$4.43E+05$
10^{-6}	10^{-6}	2097152	134217728	8	$5.6E-06$	$1.9E-05$	$3.18E+02$	$4.22E+05$
				Sixth-Order Force Approximation				
10^{-2}	10^{-2}	32768	7077888	6	$2.7E-02$	$3.4E-02$	$9.10E-01$	$7.78E+06$
10^{-4}	10^{-2}	32768	7077888	6	$2.8E-04$	$7.4E-04$	$1.98E+00$	$3.57E+06$
10^{-4}	10^{-4}	37248	8045568	7	$2.7E-04$	$8.2E-04$	$2.30E+00$	$3.50E+06$
10^{-6}	10^{-4}	37248	8045568	7	$1.8E-05$	$3.2E-05$	$5.21E+00$	$1.54E+06$
10^{-6}	10^{-6}	262144	56623104	7	$5.5E-06$	$1.8E-05$	$3.86E+01$	$1.47E+06$
10^{-8}	10^{-6}	262144	56623104	7	$1.5E-07$	$4.6E-07$	$9.66E+01$	$5.86E+05$
10^{-8}	10^{-8}	2097152	452984832	8	$1.0E-07$	$3.0E-07$	$8.05E+02$	$5.63E+05$
				Eighth-Order Force Approximation				
10^{-2}	10^{-2}	4096	2097152	5	$1.8E-02$	$2.8E-02$	$3.22E-01$	$6.51E+06$
10^{-4}	10^{-2}	4096	2097152	5	$6.9E-04$	$1.5E-03$	$4.41E-01$	$4.72E+06$
10^{-4}	10^{-4}	32768	16777216	6	$3.7E-04$	$9.5E-04$	$3.38E+00$	$4.96E+06$
10^{-6}	10^{-4}	32768	16777216	6	$7.0E-06$	$1.9E-05$	$6.28E+00$	$2.67E+06$
10^{-6}	10^{-6}	242432	124125184	7	$6.5E-06$	$2.0E-05$	$4.81E+01$	$2.58E+06$
10^{-8}	10^{-6}	242432	124125184	7	$2.0E-07$	$9.1E-07$	$1.02E+02$	$1.22E+06$
10^{-8}	10^{-8}	262144	134217728	7	$1.4E-07$	$5.7E-07$	$1.04E+02$	$1.29E+06$
10^{-10}	10^{-8}	262144	134217728	7	$2.6E-08$	$6.1E-08$	$2.29E+02$	$5.86E+05$
10^{-10}	10^{-10}	2097152	1073741824	8	$9.0E-09$	$4.3E-08$	$1.87E+03$	$5.75E+05$

Table 3.6: Periodic Boundary Conditions example numerical results.

Compared to non-periodic examples, the rate of the solver is slightly slower, as would be expected due to an increase in the sizes of interaction and near-neighbor lists, especially near the box boundaries. However, we see that we still obtain proper scaling as we increase the problem size and maintain the same ϵ_{fmm} precision, as desired.

It is straightforward to extend this approach to a variety of homogeneous Dirichlet, Neumann or mixed boundary conditions by the method of images as well with very little additional effort as previously described. We show two more examples for the Dirichlet boundary examples, one for verification of the implementation and comparison to the periodic solver, timing-wise, and a second example for a highly-adaptive right-hand side.

3.1.5 Dirichlet Boundary Conditions

We currently have only implemented Dirichlet boundary conditions in cases where $u(\mathbf{x}) = 0$ on $\partial\Omega$, the boundary of our domain, in this case the $[-1,1]^3$ box. We present two test examples. The first is nearly-identical example as the periodic boundary solver above for nearly-uniform refinement to display any affect on the timings of our solver, and the second is a highly-nonuniform right-hand side with a great degree of adaptivity near a corner of the box domain in order to show the periodic algorithm handles more complexity in its interaction lists.

Example 1. As an example, we consider the source function

$$f(\mathbf{x}) = 3CM^2\pi^2 \sin(\pi M(1+x))\sin(\pi M(1+y))\sin(\pi M(1+z)),$$

for which the solution with boundary conditions of $u = 0$ on our domain $[-1,1]^3$ is

$$u(\mathbf{x}) = C\sin(\pi M(1+x))\sin(\pi M(1+y))\sin(\pi M(1+z)).$$

As can be seen, this is simply a shifted form of the example one above. We again choose $C = 5$ AND $M = 7$ and check the relative L_2 and L_∞ with results shown in Table 3.7.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	Rate
Fourth-Order Force Approximation								
10^{-2}	10^{-2}	32768	2097152	6	$2.6E-02$	$3.3E-02$	$6.56E-01$	$3.20E+06$
10^{-4}	10^{-2}	32768	2097152	6	$1.1E-03$	$1.4E-03$	$2.40E+00$	$8.74E+05$
10^{-4}	10^{-4}	262144	16777216	7	$2.9E-04$	$7.6E-04$	$1.78E+01$	$9.42E+05$
10^{-6}	10^{-4}	262144	16777216	7	$1.6E-05$	$2.4E-05$	$4.00E+01$	$4.19E+05$
10^{-6}	10^{-6}	2097152	134217728	8	$5.5E-06$	$1.9E-05$	$3.25E+02$	$4.13E+05$
Sixth-Order Force Approximation								
10^{-2}	10^{-2}	32768	7077888	6	$2.6E-02$	$3.4E-02$	$8.18E-01$	$8.65E+06$
10^{-4}	10^{-2}	32768	7077888	6	$2.8E-04$	$7.4E-04$	$2.53E+00$	$2.80E+06$
10^{-4}	10^{-4}	37248	8045568	7	$2.7E-04$	$8.2E-04$	$2.87E+00$	$2.80E+06$
10^{-6}	10^{-4}	37248	8045568	7	$1.8E-05$	$3.2E-05$	$7.22E+00$	$1.11E+06$
10^{-6}	10^{-6}	262144	56623104	7	$5.5E-06$	$1.8E-05$	$4.77E+01$	$1.19E+06$
10^{-8}	10^{-6}	262144	56623104	7	$1.4E-07$	$4.5E-07$	$1.21E+02$	$4.68E+05$
10^{-8}	10^{-8}	2097152	452984832	8	$1.1E-07$	$5.1E-07$	$8.92E+02$	$5.08E+05$
Eighth-Order Force Approximation								
10^{-2}	10^{-2}	4096	2097152	5	$1.8E-02$	$2.8E-02$	$3.78E-01$	$5.55E+06$
10^{-4}	10^{-2}	4096	2097152	5	$6.9E-04$	$1.5E-03$	$5.72E-01$	$3.67e+06$
10^{-4}	10^{-4}	32768	16777216	6	$3.7E-04$	$9.5E-04$	$4.10E+00$	$4.10E+06$
10^{-6}	10^{-4}	32768	16777216	6	$7.0E-06$	$1.9E-05$	$8.28E+00$	$2.03E+06$
10^{-6}	10^{-6}	242432	124125184	7	$6.5E-06$	$2.1E-05$	$5.60E+01$	$2.22E+06$
10^{-8}	10^{-6}	242432	124125184	7	$2.0E-07$	$9.0E-07$	$1.18E+02$	$1.05E+06$
10^{-8}	10^{-8}	262144	134217728	7	$1.3E-07$	$5.5E-07$	$1.25E+02$	$1.07E+06$
10^{-10}	10^{-8}	262144	134217728	7	$1.3E-08$	$5.2E-08$	$3.03E+02$	$4.43E+05$
10^{-10}	10^{-10}	2097152	1073741824	8	$8.8E-09$	$3.1E-08$	$2.54E+03$	$4.23E+05$

Table 3.7: Dirichlet Boundary Conditions first example numerical results.

Since the test problems are nearly identical, the sizes of the test cases are equivalent, allowing us to see that the timings are slightly larger and the rates are slightly slower. Indeed, this is expected given that we need to rotate and translate upward equivalent densities and polynomial approximations in order to account for the interactions and contributions for boxes across the boundaries and inside of the supercell, as discussed in section 2.6.1. As desired, however, the differences are minimal (and not an eight-fold increase as would occur for a poor implementation).

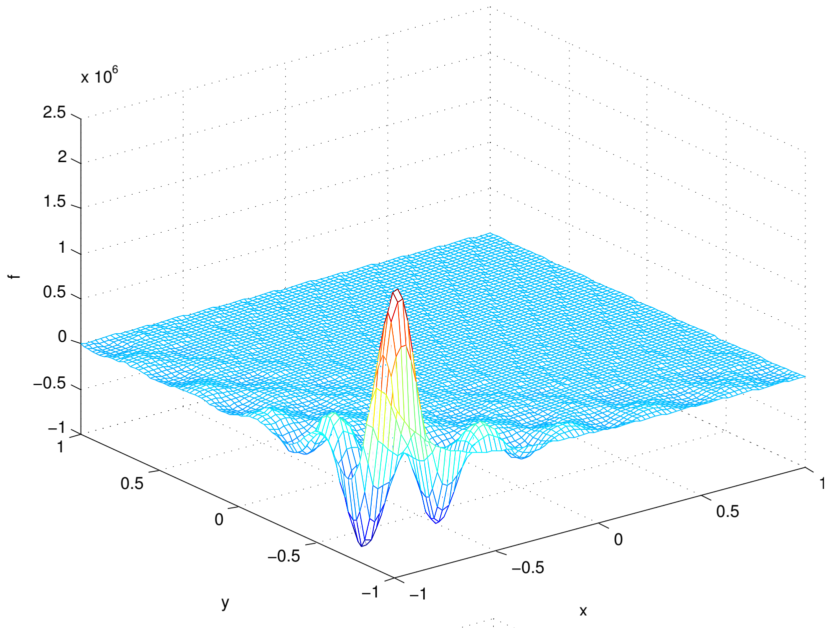
Example 2. We investigate one final test example for a highly-adaptive right-hand side for the Dirichlet solver. For $r^2 = (x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2$, let

$$\begin{aligned} -\Delta u(\mathbf{x}) &= \left(3M^2 - 6L - 4L^2r^2\right) \exp\left(Lr^2\right) \sin(M(x+1)) \sin(M(y+1)) \sin(M(z+1)) \\ &- 4LM \exp\left(Lr^2\right)[\cos(M(x+1)) \sin(M(y+1)) \sin(M(z+1))(x - cx) \\ &+ \cos(M(y+1)) \sin(M(x+1)) \sin(M(z+1))(y - cy) \\ &+ \cos(M(z+1)) \sin(M(x+1)) \sin(M(y+1))(z - cz)], \end{aligned}$$

where $L = \sqrt{2}$, $M = 5\pi$, and $c_x = c_y = c_z = \frac{3}{4}$ The solution with boundary conditions of $u = 0$ on our domain $[-1, 1]^3$ is

$$u(\mathbf{x}) = \sin(M(1 + x)) \sin((1 + y)) \sin(M(1 + z)) \exp\left(Lr^2\right).$$

In figure 3.3 we take a look at a single slice of the force and solution in order to show how a non-uniform refinement is achieved for the octree.



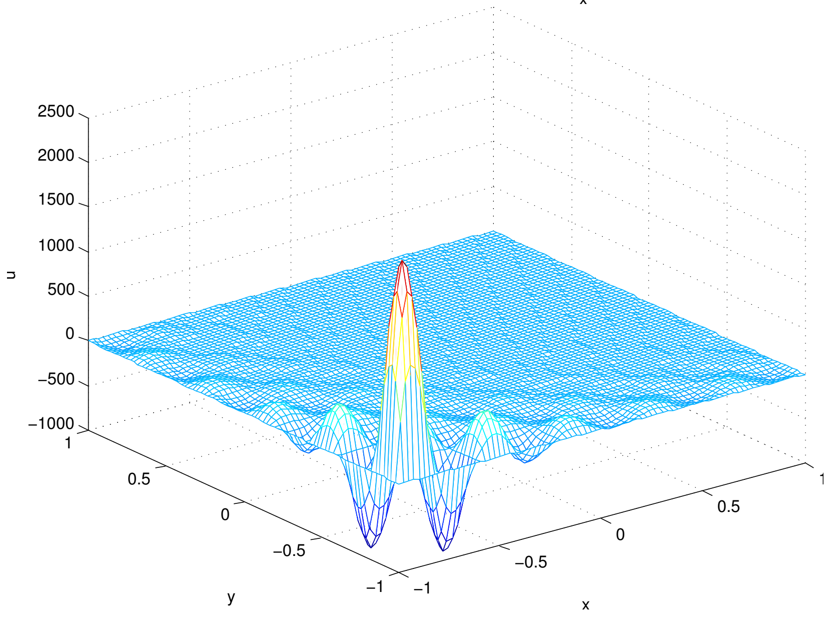


Figure 3.3: Plot of Dirichlet Boundary Conditions Example 2. *Top*: A single slice of the smooth force distribution; *Bottom*: A single slice of the solution. For both, $z = -0.25$, and we note the high-gradient nature near the corner of $x = y = -1$ results in a highly-adaptive refinement near the boundary of the domain.

We look at the numerical results and timings in Table 3.8. As can be seen, we continue to achieve a good rate, comparable to earlier results despite the introduction of many more interactions and much larger L_B^U, L_B^W and L_B^X lists. Additionally, a significantly larger amount of book-keeping is taking place as we must perform many more translations and rotations of densities and polynomials to account for cross-boundary interactions.

ϵ_{fmm}	ϵ_{rhs}	M_ℓ	N_{pts}	L_T	E_2	E_∞	T_{FMM}	Rate
Fourth-Order Force Approximation								
10^{-2}	10^{-2}	7484	478976	7	$6.3E-02$	$7.7E-02$	$3.40E-01$	$1.41E+06$
10^{-4}	10^{-4}	46600	2982400	8	$6.6E-04$	$1.4E-03$	$3.80E+00$	$7.85E+05$
10^{-6}	10^{-6}	316296	20242944	9	$6.5E-06$	$1.7E-05$	$6.44E+01$	$3.14E+05$
10^{-8}	10^{-8}	2111110	135111040	9	$1.0E-07$	$2.6E-07$	$9.90E+02$	$1.36E+05$
Sixth-Order Force Approximation								
10^{-2}	10^{-2}	1142	246672	6	$5.2E-02$	$7.4E-02$	$1.18E-01$	$2.09E+06$
10^{-4}	10^{-4}	4859	1049544	7	$5.9E-04$	$1.3E-03$	$5.84E-01$	$1.80E+06$
10^{-6}	10^{-6}	23122	4994352	8	$5.3E-06$	$1.5E-05$	$5.73E+00$	$8.72E+05$
10^{-8}	10^{-8}	100416	21689856	8	$1.1E-07$	$2.5E-07$	$5.50E+01$	$3.94E+05$
10^{-10}	10^{-10}	417586	90198576	9	$2.3E-08$	$5.2E-08$	$5.42E+02$	$1.66E+05$
Eighth-Order Force Approximation								
10^{-2}	10^{-2}	337	172544	5	$4.8E-02$	$8.0E-02$	$8.49E-02$	$2.03E+06$
10^{-4}	10^{-4}	1100	563200	6	$7.5E-04$	$1.5E-03$	$3.67E-01$	$1.53E+06$
10^{-6}	10^{-6}	3438	1760256	6	$6.3E-06$	$1.7E-05$	$1.45E+00$	$1.21E+06$
10^{-8}	10^{-8}	10802	5530624	7	$1.1E-07$	$2.6E-07$	$8.40E+00$	$6.58E+05$
10^{-10}	10^{-10}	41833	21418496	8	$2.2E-08$	$5.3E-08$	$6.59E+01$	$3.25E+05$
10^{-12}	10^{-12}	133442	68322304	8	$5.0E-09$	$1.1E-08$	$4.65E+02$	$1.47E+05$

Table 3.8: Dirichlet Boundary Conditions second example numerical results.

3.1.6 Combining Smooth and Singular Sources

A number of applications require the modeling of source distributions that contain both a smooth component and a singular component. In electrostatics, for example, positively charged ions are often approximated as point charges and the neutralizing electrons as an inhomogeneous continuous background. We consider such a case here. The relevant Poisson equation has the form

$$\Delta u(\mathbf{x}) = f_{smooth}(\mathbf{x}) + \sum_{i=1}^{N} q_i \delta(\mathbf{x} - \mathbf{x_i}),$$

where the q_i are positive and the neutralizing background takes the form of a sum of Gaussian distributions $f^i_{smooth}(\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(\mathbf{x}-\sigma)^2/2\sigma^2}$ centered on each δ-function.

The smooth portion can be handled as above, while the particle sources can be handled with the corresponding particle-based kernel-independent FMM (Ying et al., 2003). However, it is trivial to modify our solver to incorporate the particle sources into the *S2M* operator 2.4.2 by changing equation (2.32). That is, let

$$\mathbf{K}^B_{S2M} \boldsymbol{\phi}^{B,u} = \mathbf{F}^B_{S2M} \gamma^B + \sum_{i=1}^{N} q_i G(\mathbf{x}, \mathbf{x_i}), \tag{3.2}$$

where G is the kernel used for evaluating the singular component, consisting of the point charges. Once the point charges are incorporated into $\boldsymbol{\phi}^{B,u}$, the rest of the components of the algorithm (i.e., M2M, M2L, and L2L) take care of the far-field interactions. We need only calculate the influence of near-field particle interactions directly, and evaluate both the local expansions (L2T) and the smooth contributions at particle locations. The latter is done by interpolation, as discussed in section 2.4.5.

Our primary concern here is to verify that incorporating particle charges into the volume solver does not add extensive overhead since the accuracy of the kernel-independent particle solver has been established (Ying et al., 2003) as has the volume solver above. Hence, we begin by investigating the timings of the particle solver only below. We use a regular distribution of N_{pts} with varying accuracy ϵ_{fmm}. The results can be seen in table 3.9

ϵ_{fmm}	$S2M/M2M$	$Near$	$M2L$	$L2L/L2T$	T_{FMM}
$N_{pts} = 1.024E+06$, $M_\ell = 4096$, $L_T = 5$					
10^{-2}	$2.51E-02$	$6.00E-01$	$2.74E-02$	$4.24E-03$	$6.57E-01$
10^{-4}	$6.85E-02$	$5.99E-01$	$1.41E-01$	$2.33E-02$	$8.32E-01$
10^{-6}	$1.39E-01$	$6.05E-01$	$4.04E-01$	$6.63E-02$	$1.21E+00$
10^{-8}	$2.50E-01$	$6.00E-01$	$1.09E+00$	$1.58E-01$	$2.10E+00$
$N_{pts} = 4.096E+06$, $M_\ell = 32768$, $L_T = 6$					
10^{-2}	$1.06E-01$	$1.29E+00$	$2.49E-01$	$1.90E-02$	$1.66E+00$
10^{-4}	$2.92E-01$	$1.29E+00$	$1.37E+00$	$9.97E-02$	$3.05E+00$
10^{-6}	$6.15E-01$	$1.29E+00$	$3.51E+00$	$2.92E-01$	$5.71E+00$
10^{-8}	$1.14E+00$	$1.29E+00$	$9.42E+00$	$6.48E-01$	$1.25E+01$
$N_{pts} = 1.638E+07$, $M_\ell = 262144$, $L_T = 7$					
10^{-2}	$1.55E+00$	$4.04E+01$	$2.22E+00$	$2.71E-01$	$4.44E+01$
10^{-4}	$4.23E+00$	$4.06E+01$	$1.13E+01$	$1.49E+00$	$5.76E+01$
10^{-6}	$8.73E+00$	$4.06E+01$	$3.15E+01$	$4.18E+00$	$8.50E+01$
10^{-8}	$1.51E+01$	$4.07E+01$	$8.24E+01$	$1.01E+01$	$1.48E+02$
$N_{pts} = 6.554E+07$, $M_\ell = 2097152$, $L_T = 8$					
10^{-2}	$6.83E+00$	$8.42E+01$	$2.33E+01$	$1.33E+00$	$1.16E+02$
10^{-4}	$1.85E+01$	$8.43E+01$	$9.13E+01$	$6.84E+00$	$2.01E+02$
10^{-6}	$3.89E+01$	$8.52E+01$	$2.65E+02$	$1.95E+01$	$4.09E+02$
10^{-8}	$7.17E+01$	$8.52E+01$	$6.99E+02$	$4.22E+01$	$8.98E+02$

Table 3.9: Poisson equation with singular sources only.

We then look at the volume solver only and notice only evaluation the stages which change significantly are the $S2M$ and $Near$ portions. For this example, we use the singular sources for the building of the octree only; the resulting tree assumes a k^3 grid given on each leaf, from which we construct the polynomial approximation to the smooth force as in our previous examples. Additionally, in these stages, the operators are calculated once each, so as expected, as the size of the tree and accuracy increase, the times for the upward and near computations decrease. We fix the polynomial approximation to a 6^{th} order approximation ($k = 6$), and note that since operators for the downward pass of the algorithm remain the same, these timings remain consistent with the particle-only solver. We present the timings for the volume-only solver in table 3.10.

ϵ_{fmm}	$S2M/M2M$	$Near$	$M2L$	$L2L/L2T$	T_{FMM}
$N_{pts} = 1.024E+06$, $M_\ell = 4096$, $L_T = 5$					
10^{-2}	$2.94E-03$	$3.72E-02$	$2.74E-02$	$4.70E-03$	$7.22E-02$
10^{-4}	$7.56E-03$	$3.77E-02$	$1.61E-01$	$2.86E-02$	$2.34E-01$
10^{-6}	$1.94E-02$	$3.71E-02$	$3.84E-01$	$6.18E-02$	$5.03E-01$
10^{-8}	$4.86E-02$	$3.73E-02$	$1.13E+00$	$1.53E-01$	$1.37E+00$
$N_{pts} = 4.096E+06$, $M_\ell = 32768$, $L_T = 6$					
10^{-2}	$2.23E-02$	$3.10E-01$	$2.60E-01$	$1.85E-02$	$6.11E-01$
10^{-4}	$5.61E-02$	$3.11E-01$	$1.34E+00$	$1.01E-01$	$1.81E+00$
10^{-6}	$1.46E-01$	$3.08E-01$	$3.48E+00$	$2.92E-01$	$4.23E+00$
10^{-8}	$3.65E-01$	$3.12E-01$	$9.76E+00$	$6.51E-01$	$1.11E+01$
$N_{pts} = 1.638E+07$, $M_\ell = 262144$, $L_T = 7$					
10^{-2}	$1.72E-01$	$2.53E+00$	$2.13E+00$	$2.80E-01$	$5.12E+00$
10^{-4}	$4.68E-01$	$2.53E+00$	$1.14E+01$	$1.40E+00$	$1.58E+01$
10^{-6}	$1.21E+00$	$2.53E+00$	$2.94E+01$	$4.24E+00$	$3.73E+01$
10^{-8}	$2.85E+00$	$2.54E+00$	$8.25E+01$	$1.00E+01$	$9.79E+01$
$N_{pts} = 6.554E+07$, $M_\ell = 2097152$, $L_T = 8$					
10^{-2}	$1.19E+00$	$2.05E+01$	$1.76E+01$	$1.52E+00$	$4.08E+01$
10^{-4}	$3.12E+00$	$2.05E+01$	$9.09E+01$	$6.52E+00$	$1.21E+02$
10^{-6}	$9.15E+00$	$2.05E+01$	$2.71E+02$	$1.98E+01$	$3.20E+02$
10^{-8}	$2.28E+01$	$2.05E+01$	$6.92E+02$	$4.21E+01$	$7.77E+02$

Table 3.10: Poisson equation with smooth sources only.

We now combine both the particle and volume solvers, and the performance of our scheme is shown in Table 3.11. The real advantage of this approach is seen in how the timings for all but the $Near$ stages are nearly identical. Further, as noted, the $S2M$ and $L2T$ stages include minimal extra computation and as the equivalent densities include all information for the singular *and* smooth sources, no additional computation is required for the $M2L$, $M2M$, or $L2L$ stages.

ϵ_{fmm}	$S2M/M2M$	$Near$	$M2L$	$L2L/L2T$	T_{FMM}
$N_{pts} = 1.024E + 06$, $M_\ell = 4096$, $L_T = 5$					
10^{-2}	$2.52E - 02$	$6.42E - 01$	$5.09E - 02$	$4.84E - 03$	$7.23E - 01$
10^{-4}	$6.97E - 02$	$6.49E - 01$	$1.29E - 01$	$2.35E - 02$	$8.71E - 01$
10^{-6}	$1.41E - 01$	$6.50E - 01$	$3.77E - 01$	$6.66E - 02$	$1.24E + 00$
10^{-8}	$2.48E - 01$	$6.41E - 01$	$1.03E + 00$	$1.58E - 01$	$2.08E + 00$
$N_{pts} = 4.096E + 06$, $M_\ell = 32768$, $L_T = 6$					
10^{-2}	$1.11E - 01$	$1.58E + 00$	$2.39E - 01$	$1.85E - 02$	$1.95E + 00$
10^{-4}	$3.04E - 01$	$1.61E + 00$	$1.32E + 00$	$1.01E - 01$	$3.34E + 00$
10^{-6}	$6.29E - 01$	$1.61E + 00$	$3.55E + 00$	$2.92E - 01$	$6.08E + 00$
10^{-8}	$1.17E + 00$	$1.60E + 00$	$9.20E + 00$	$6.48E - 01$	$1.26E + 01$
$N_{pts} = 1.638E + 07$, $M_\ell = 262144$, $L_T = 7$					
10^{-2}	$1.58E + 00$	$4.31E + 01$	$2.14E + 00$	$2.58E - 01$	$4.71E + 01$
10^{-4}	$4.31E + 00$	$4.34E + 01$	$1.14E + 01$	$1.48E + 00$	$6.06E + 01$
10^{-6}	$8.79E + 00$	$4.38E + 01$	$3.05E + 01$	$4.20E + 00$	$8.72E + 01$
10^{-8}	$1.55E + 01$	$4.32E + 01$	$8.26E + 01$	$9.96E + 00$	$1.51E + 02$
$N_{pts} = 6.554E + 07$, $M_\ell = 2097152$, $L_T = 8$					
10^{-2}	$7.00E + 00$	$1.04E + 02$	$1.80E + 01$	$1.40E + 00$	$1.31E + 02$
10^{-4}	$1.93E + 01$	$1.05E + 02$	$9.14E + 01$	$6.76E + 00$	$2.23E + 02$
10^{-6}	$4.01E + 01$	$1.07E + 02$	$2.61E + 02$	$1.93E + 01$	$4.27E + 02$
10^{-8}	$7.34E + 01$	$1.06E + 02$	$7.00E + 02$	$4.19E + 01$	$9.21E + 02$

Table 3.11: Poisson equation with a mixture of smooth and singular sources example numerical results.

As noted, the upward pass timings are minimally larger than for the particle-only case, and the downward pass timings are agnostic about the nature of the sources, dependent only on the tree-structure itself. The $Near$ computation timings are simply the sum of the particle and volume-based cases, since there is no amortization of cost in this step.

3.2 Numerical Accuracy of Operator Components

In this section, we first verify numerically that the equivalent density representation yields the expected accuracy, followed by a discussion of how the choice of grids affects the order of convergence and an overall numerical justification for the use of Tikhonov regularization.

3.2.1 Equivalent Density Accuracy

As discussed in section 2.4.1, we invert several matrices of discretized Fredholm equations of the first kind in order to build far-field representations; e.g.,

$$\mathbf{K}^{\mathbf{y}_d,\mathbf{x}_d}\phi_d = \mathbf{K}^{\mathbf{y}_s,\mathbf{x}_d}\phi_s.$$

As in (Ying et al., 2004b), we choose to use Tikhonov regularization (Kress, 1999) when solving these ill-conditioned systems. This solves two problems: in this way, we eliminate the null space in the cases when it is present (Stokes kernel) and we significantly improve accuracy of the inversion for higher numbers of samples (Section 3.2.3). We verify the potential we get from $\phi^{B,u}$, computed using our regularized method in the $S2M$ operation, approximates well $u(\mathbf{x})$, computed directly from a force. We test using $g^B = \sum_{a,b,c}^{(a+b+c)\leq(k-1)} x^a y^b z^c$ for box B of width 2 and compute

$$u(\mathbf{x}) = \int_B K(\mathbf{x},\mathbf{y}) g^B(\mathbf{y}) d\mathbf{y},\ \mathbf{x} \in \mathbf{x}^{B,u},$$

to within 10^{-16} accuracy using adaptive Gaussian quadrature (Berntsen et al., 1991). We then compute $\phi^{B,u}$ at $\mathbf{y}^{B,u}$ using (2.33) where $(\mathbf{K}_{S2M})^{-1}$ is replaced with

$$(\alpha I + (\mathbf{K}_{S2M})^{*}\mathbf{K}_{S2M})^{-1}\mathbf{K}^{*}_{S2M}.$$

For FMM precision, n_p, we choose $\alpha = 10^{-(n_p+1)}$. More details on the choice of α are available in (Ying et al., 2004b). Our algorithm relies on the fact that for surfaces outside the near field of B, $\phi^{B,u}$ is a sufficiently accurate representation of B's volume force. We compute

$$u(\mathbf{x})_{equiv} = \int_{\mathbf{y}^{B,u}} K(\mathbf{x},\mathbf{y})\phi^{B,u}(\mathbf{y})d\mathbf{y}$$

for $\mathbf{x} \in S$, some surface. To evaluate the accuracy of this approximation, we compute

$$u(\mathbf{x})_{exact} = \int_{B} K(\mathbf{x},\mathbf{y})g^{B}(\mathbf{y})d\mathbf{y}$$

up to an accuracy of 10^{-16} (Berntsen et al., 1991). In figure 3.4, we compare the infinity-norm of the resulting error for three different kernels (Laplace, Modified Helmholtz, Stokes) and varying levels of the polynomial approximation and multiple degrees of FMM evaluation precision. For each of the kernels of interest, $\phi^{B,u}$, computed by inverting our ill-conditioned kernels, is recovered on each surface S to within the requested degree of precision. For evaluating the accuracy of the kernel inversion and regularization in the computation of $\phi^{B,d}$, we note that this computation is equivalent to the particle-based FMM, of which numerical analysis for the $M2L$ and $L2L$ operators is available in (Ying et al., 2004b).

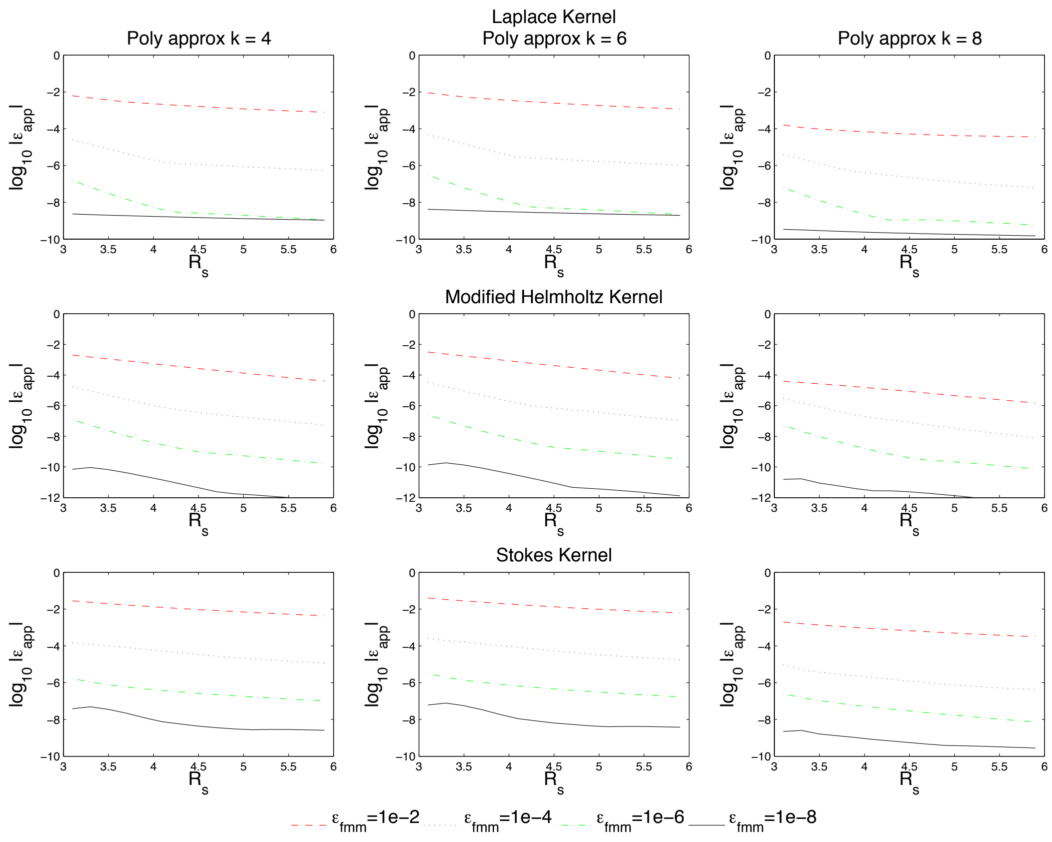


Figure 3.4: Error due to upward equivalent density approximation of the field. From left to right, three columns show the errors for the polynomial force approximations of degree 4, 6 and 8. Each plot shows four levels of FMM precision, $\epsilon_{fmm} = 10^{-n_p}$, $p = n_p^3 - (n_p - 2)^3$ points are used on the surfaces $\mathbf{y}^{B,u}$ and $\mathbf{x}^{B,u}$. For the evaluation surfaces S, we vary the radius R_S from 3.1 to 5.9, the region covering $L_I^B \in \mathcal{F}^B$. The y-axis of each plot is the infinity norm $||u_{equiv} - u_{exact}||_\infty$ computed over 488 samples on S.

3.2.2 Polynomial Basis and Grid Spacing

As discussed in section 2.4.4, we evaluate the solution at a leaf box B on a grid of points, $\mathbf{x}^{B,g}$ and construct an approximating polynomial from these points. Additionally, (section 2.4.6) we construct a k^{th}-order polynomial approximation to B's distributed force if g^B is given on a grid. For consistency with AMR codes and efficiency of implementation, it would have been desirable to use uniform grid samples. This approach works well for $n \leq 6$, but it is well-known for large n that equispaced grids lead to instabilities (Trefethen and Bau, 1997); as a result, for $n > 6$ we use Chebyshev grid points. To show that regularly-spaced grid points perform poorly for $n, k > 6$, we consider the following test case:

$$-\Delta u(\mathbf{x}) = e^{-L(||\mathbf{x}||_2)^2} \left(4L(||\mathbf{x}||_2)^2 - 6L\right), L = 250, \mathbf{x} \in [-1,1]^3$$

This is a simplification of Example 1 in section 3.1.1 where our domain is $[-1,1]^3$ and $||\mathbf{x}||_2$ is measured with respect to the origin. In figure 3.5, we compare the overall relative L^2 error, E_2, for solutions using equispaced and Chebyshev grid points in the evaluation of the solution and construction of the polynomial approximations of degree $4, 6$ and 8. Errors for discretizations using equispaced or Chebyshev grid points are similar for $k \leq$, but for $k = 8$, Chebyshev points are more accurate.

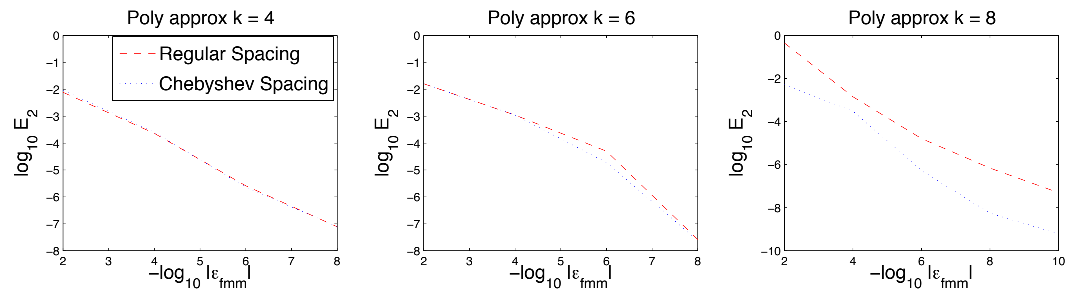


Figure 3.5: For each of the test examples, the x-axis indicates the negative log of the requested FMM accuracy, ϵ_{fmm}, and the y-axis indicates the log of E_2. The number of points chosen for each ϵ_{fmm} is similar to those in Example 1 of section 3.1.1 for $\epsilon_{rhs} = \epsilon_{fmm}$. *Left*: for polynomial approximation of degree 4 and $\mathbf{x}^{B,g}$ of size 4^3 on each leaf B, overall relative error is close for equispaced and Chebyshev points. *Middle:* For $n, k = 6$ differences are visible but insignificant. *Right:* For $n, k = 8$, solutions based on equispaced grid are less accurate.

3.2.3 Tikhonov Regularization

As discussed above in section 3.2.1, we use Tikhonov regularization (Kress, 1999) to invert Fredholm equations of the first kind, specifically the S2M, M2M, and L2L operators in section 2.4. Further, in section 3.2.1, we looked specifically at the accuracy resulting from this inversion process. To justify the overall use of Tikhonov regularization, we consider the following test cases for the Poisson and Stokes equations, respectively for $\mathbf{x} \in [-1,1]^3$.

$$-\Delta u(\mathbf{x}) = e^{-L(||\mathbf{x}||_2)^2}\left(4L(||\mathbf{x}||_2)^2 - 6L\right), L = 250$$

$$-\Delta u(\mathbf{x}) + \nabla p(\mathbf{x}) = \left(8L^3\,||\mathbf{x}-\mathbf{x}_i||^2 - 20L^2\right) e^{-L||\mathbf{x}-\mathbf{x}_i||^2}\left[\nabla \times (\mathbf{x}-\mathbf{x}_i)\right], L = 125$$

In figure 3.6, we compare the overall relative L^2 error, E_2, solutions, resulting from Tikhonov regularization versus no regularization and the construction of polynomial approximations of degree $k = 6$ for the right-hand sides (errors for $k = 4, 8$ are similar). For decreasing levels of ϵ_{fmm}, we choose $\epsilon_{fmm} = \epsilon_{rhs}$.

We notice that for $\epsilon_{fmm} > 10^{-7}$, the effect of not employing regularization is equivalent to using regularization for both the Laplace and Stokes operators. However, as ϵ_{fmm} decreases, the number of sample points on the equivalent and check surfaces increases, resulting in larger linear systems, which as mentioned earlier, may be poorly conditioned. Indeed, for such larger systems resulting from $\epsilon_{fmm} \leq 10^{-7}$, it is necessary to regularize the systems to achieve desirable results.

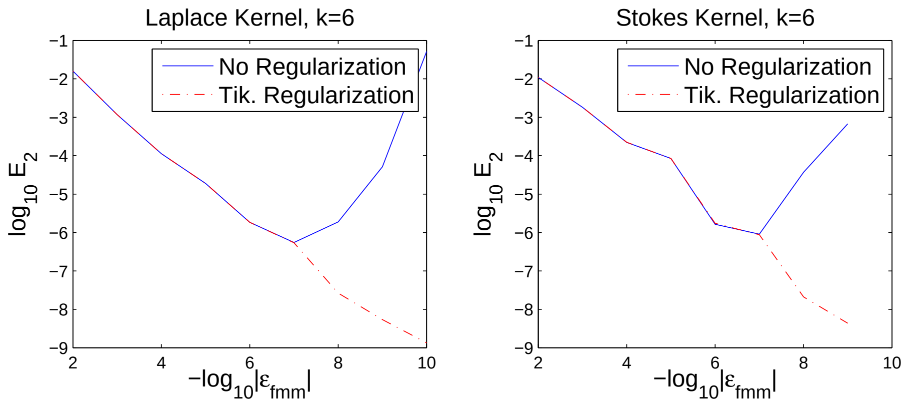


Figure 3.6: For each of the test examples, the x-axis indicates the negative log of the requested FMM accuracy, ϵ_{fmm}, and the y-axis indicates the log of E_2. The number of points chosen for each ϵ_{fmm} is similar to those in Examples 1 of section 3.1.1 and section 3.1.3 for $\epsilon_{rhs} = \epsilon_{fmm}$. *Left*: for polynomial approximation of degree 6 for the Laplace kernel with and without regularization. *Right*: for polynomial approximation of degree 6 for the Stokes kernel with and without regularization.

3.3 Shared-Memory Parallelization and Load-Balancing Effects

We close with a discussion of how we accelerate the major computation loops of our algorithm using OpenMP shared-memory parallelization and load-balancing techniques. As indicated in section 5.1, we have designed the code to take advantage of shared-memory architectures through the use of OpenMP. In particular, we highlight the steps to accelerate the various major steps in Algorithm 1. For details on the nature of OpenMP and its usage, we refer the reader to (Chapman et al., 2007).

S2M and M2M Computations In the upward pass (step 2 of Algorithm 1 and section 2.4.2), we begin by building a list of all leaf boxes, B in the octree T, which have sources. We then do a simple OpenMP parallelization step over these boxes for the $S2M$ step. As all components of equation (2.33) are of the same size for each leaf box, there is no need to rebalance the load among threads

In order to ensure proper order of computation, we proceed by sorting all non-leaves in reverse-order by depth. For each non-leaf level in T, beginning at the deepest level, we translate a box B's children's upward equivalent densities to its own through the $M2M$ computation in (2.35). Again, as each of the components is of the same size, there is no need to rebalance among threads. As we parallelize only among boxes at the same depth in T, level ℓ is not processed until $\ell + 1$ has completed. Further, once we have reached coarse level $\ell = 1$ (which only occurs for periodic or Dirichlet boundary conditions), we discontinue the parallelization.

M2L, L2L, and L2T Computations In the downward pass of Algorithm 1 (section 2.4.3), we perform a similar operation as above for the $M2M$ step. First, we sort all boxes B in T from the shallowest to deepest levels in the tree. For each level, ℓ, we parallelize among the boxes being processed at that level for the $M2L$ and $L2L$ computations. The $L2L$ components in equation (2.39) are of equivalent size for each box B; however, for each box B, the size of L_V^B varies greatly (for example, this list is much smaller for boxes on the edge or corners of our domain). In order to ensure proper balancing among threads, we further sort all boxes for each level, ℓ by the size of L_V^B and then reorder the boxes such that the sum of all L_V^B for each thread is of roughly the same size.

For the $L2T$ computations in equation (2.41), we once again build a list of only leaf boxes, for which the target solution is desired, and we parallelize the computations in this list. The components of the discretized equation are all the same size, as with the $S2M$ computation, so there is no need to rebalance among threads for this step.

Near-Field Computations We focus our discussion here on the U-list computations. Parallelizing the near-field computations in equation (2.44) is the most straightforward in that no leaf box B is dependent on the completion of computations by any other box. That is, we can simply parallelize the computations among leaf boxes, for which the L_U^B exists. However, even more so than with the $M2L$ computations, the sizes of L_U^B can be very different among leaf boxes (especially in the most adaptively-refined octrees). Thus, we sort all leaf boxes B in T by the size of L_U^B and reorder the list of leaves such that the sum of the size of L_U^B among each thread is roughly equivalent, ensuring a relatively well-balanced load among threads. The size of the components and operators are the same for each box B, so balancing by list sizes is optimal. We note that this rebalancing is largely unnecessary for uniformly-refined trees.

Additionally, for matrix kernels (e.g. Stokes) and larger orders of polynomial approximation, constantly loading large matrices into memory results in little speedup as we increase the number of processes. In order to correct this, for each equivalence class as described in section 2.5, we perform all of the operations involving a single class first before performing all computations for other classes of operators. Hence, we only load each matrix operator at most once per processor.

We note that for the $M2L$ step, as we have to process level ℓ before moving to level $\ell - 1$, operators will constantly have to be reloaded. Performing all computations in order for each equivalence class at each level is done, but we have seen little time savings for this in practice as opposed to the near-field computations, where it is essential for good speedup.

Remark 3.1. *As with the near-field computations, for adaptively-refined trees, we rebalance the loads among threads for the X and W lists, which involve additional near-field $S2T$, $M2T$ and $S2L$ computations in equations* (2.50) *and* (2.49)*, based on the sizes of L_X^B and L_W^B, respectively. Additionally, we perform all computations in order of equivalence class, again loading each matrix operator at most once.*

Timing Results Versus Number of Processors In order to see the effect of our use of OpenMP and load-balancing strategies, we investigate the strong scaling of two fixed problems. First, from Example 1 in section 3.1.1, we set the polynomial order, ϵ_{rhs}, and ϵ_{fmm} to 8. The reasoning behind this is to ensure that for a single processor, neither the near-field nor far-field computations fully dominate the timings. In table 3.12 we look at the timings for the different algorithmic steps (note that the near-field computation times include U,W,and X list computation times) as well as plot the decreasing times in figure 3.7.

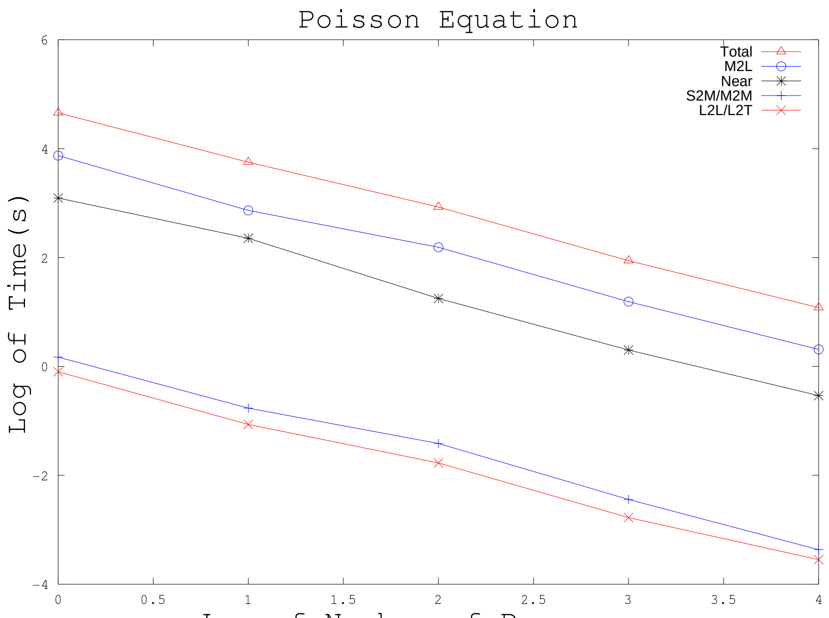
For our second study of the effect of shared-memory parallelization, we look at the Stokes kernel tests from section 3.1.3. We fix the polynomial order to 8 and look at $\epsilon_{rhs} = \epsilon_{fmm} = 6$, again in an effort to not have one step fully dominate the computational time, allowing us to look at the effect of scaling the number of processors. Timing results can be seen in table 3.13 and figure 3.7.

N_{procs}	$S2M/M2M$	$Near$	$M2L$	$L2L/L2T$	T_{FMM}	$Rate$
1	$1.126E+00$	$8.534E+00$	$1.464E+01$	$9.340E-01$	$2.524E+01$	
2	$5.880E-01$	$5.112E+00$	$7.285E+00$	$4.779E-01$	$1.346E+01$	$1.874E+00$
4	$3.750E-01$	$2.377E+00$	$4.559E+00$	$2.928E-01$	$7.604E+00$	$1.770E+00$
8	$1.839E-01$	$1.232E+00$	$2.280E+00$	$1.460E-01$	$3.842E+00$	$1.979E+00$
16	$9.701E-02$	$6.894E-01$	$1.241E+00$	$8.547E-02$	$2.113E+00$	$1.818E+00$

Table 3.12: Timings (all in wall-time seconds) for the various components of the FMM volume solver for a fixed problem size for the Poisson equations. The polynomial-order, ϵ_{rhs}, and ϵ_{fmm} are set to 8. The number of leaves, $M_\ell = 5440$, the tree level, $L_T = 7$, and $N_{pts} = 2785280$ as seen in Example 1. N_{procs} indicates the number of processors, which we scale linearly. We separate the $S2M/M2M$, $Near$ (U,W,X-list computations), $M2L$ (V-list computations), $L2L/L2T$ timings with the total shown as T_{FMM}. The scaling $Rate$ is shown last.

N_{procs}	$S2M/M2M$	$Near$	$M2L$	$L2L/L2T$	T_{FMM}	$Rate$
1	$8.795E+00$	$5.417E+01$	$9.402E+01$	$6.833E+00$	$1.638E+02$	
2	$5.182E+00$	$2.857E+01$	$5.214E+01$	$3.618E+00$	$8.951E+01$	$1.830E+00$
4	$2.8670E+00$	$1.308E+01$	$3.030E+01$	$1.734E+00$	$4.798E+01$	$1.866E+00$
8	$1.570E+00$	$6.781E+00$	$1.614E+01$	$8.367E-01$	$2.532E+01$	$1.894E+00$
16	$8.248E-01$	$3.612E+00$	$9.452E+00$	$3.918E-01$	$1.429E+01$	$1.773E+00$

Table 3.13: Timings for the various components of the FMM volume solver for a fixed problem size for the Stokes equations. The polynomial-order, ϵ_{rhs}, and ϵ_{fmm} are set to 6. The number of leaves, $M_\ell = 4894$, the tree level, $L_T = 7$, and $N_{pts} = 2505728$ as seen in Example 5. N_{procs} indicates the number of processors, which we scale linearly. We separate the $S2M/M2M$, $Near$ (U,W,X-list computations), $M2L$ (V-list computations), $L2L/L2T$ timings with the total shown as T_{FMM}. The scaling $Rate$ is shown last.



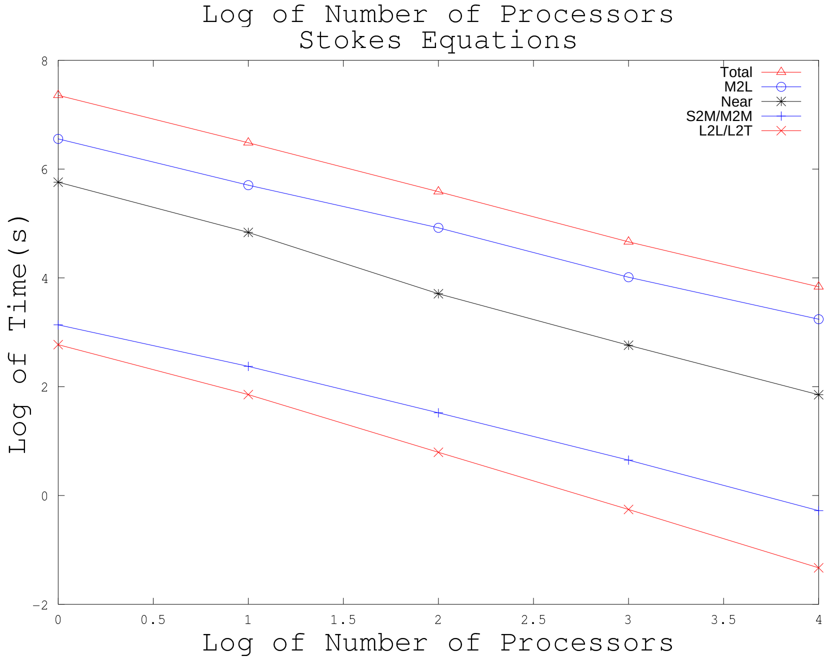


Figure 3.7: *Top*: Log-log plot for timings from table 3.12, *Bottom*: Log-log plot for timings from table 3.13. In both examples, total FMM computation time, T_{FMM} exhibits good, nearly-linear scaling with shared-memory parallelization.

As can be seen in tables 3.12 and 3.13, our scheme exhibits the desirable result of nearly-linear speedup as we scale the number of processors. As indicated in the conclusion, current work is being done to incorporate this work with (Lashuk et al., 2009) in order to achieve parallelization on a significantly larger scale.

4

COMPLEX GEOMETRY SOLVER

As discussed in the Introduction, our goal is to solve PDEs of the form in equation (1) in complex geometries (e.g., figure 1). For this purpose, we review the Embedded Boundary Integral (EBI) approach (Ying et al., 2004a), discussing pre-existing components and new components which allow for complex force distributions.

4.1 Embedded Boundary Integral Method

For an inhomogeneous interior Dirichlet PDE of the form

$$\mathcal{L}(u)(\mathbf{x}) = f(\mathbf{x}) \ \text{ in } \omega, \ u = g \ \text{ on } \partial\omega, \tag{4.1}$$

where ω is some complex geometry with boundary $\partial\omega$ and g is in the Dirichlet boundary condition on $\partial\omega$, the EBI solver performs the following steps:

1. Embed ω in a regular domain, Ω with domain $\partial\Omega$ and solve:

$$\mathcal{L}(u_1)(\mathbf{x}) = f(\mathbf{x}) \ \text{ in } \Omega, \tag{4.2}$$

where the force f is given at possibly irregular locations, depending on the input.

2. Solve a boundary integral problem on $\partial\omega$ where the border data, g is modified using the solution of u_1 derived from step 1.

$$\mathcal{L}(u_2)(\mathbf{x}) = 0, \ u_2 = g - u_1 \ \text{ on } \partial\omega \tag{4.3}$$

3. Add u_1 and u_2 to obtain the full solution.

In previous work, the prism, in which ω is embedded was discretized regularly such that $h_x = h_y = h_z$ (Mayo, 1984; Mayo, 1985; Mayo and Greenbaum, 1992; McKenney et al., 1995).

As suggested in (Mayo, 1984), $\partial\Omega$ is chosen to be far removed from $\partial\omega$. For our approach, u_2 will not have to be evaluated on $\partial\Omega$, so it need not have a regular discretization; $\partial\Omega$ is chosen to be a bounding box of arbitrary size as long as it encloses all of ω. As a graphic example, figure 4.1 takes the shape in figure 1 and embeds it inside of a rectangular prism, which has been regularly discretized (again, this is not necessary).

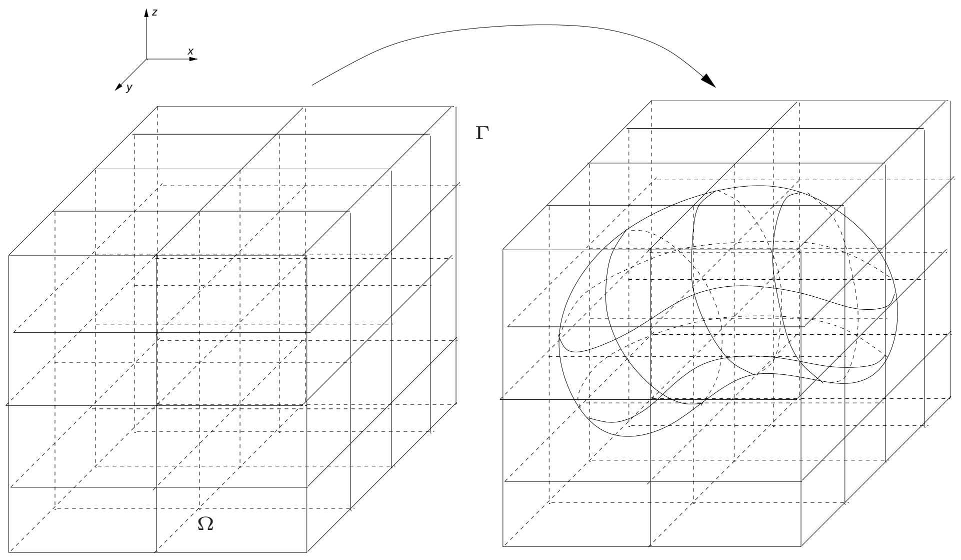


Figure 4.1: The original domain is embedded in a uniformly-discretized domain.

We first seek to solve equation 4.2 on the new regular domain with simple boundary conditions (e.g., $u|_{\partial\Omega} = 0$ or free-space). There are many possible options for fast solutions to this problem including Fast Fourier methods, cyclic reduction methods, multigrid methods, and FMM.

(Ying, 2004) uses FFT methods, assuming that the body force is available everywhere. The same is true for (McKenney et al., 1995; Mayo, 1985). This incorporates the homogeneous boundary conditions; however, one could also use volume integrals methods. That is, one can instead solve for u_1 everywhere using the FMM-based volume integral method from Chapter 2 in free-space by setting radiation boundary conditions of $u_1 \to 0$ as $\mathbf{x} \to \infty$. Further, one can then easily solve for u_1 on $\partial\omega$ by interpolating free-space grid values on leaves to the boundary points, using the octree to locate which leaves contain boundary points. Given that we have a solver perfectly designed for this task for free-space elliptic PDEs, we assume the force is known everywhere in Ω and briefly describe the pre-existing algorithm we use for solving the boundary integral. We then turn to how we perform an extension of the volume force for cases when we only have sources inside of our domain.

4.1.1 Solving the Boundary Integral

Having solved for u_1 on Ω, it is necessary to solve the Laplace equation in step **3** for u_2 in Ω. Require $u_2 = g - u_1$ on $\partial\omega$; hence, there will be discontinuities across $\partial\omega$. The density function on $\partial\omega$ is needed in order to correct for these discontinuities, using theory established in section A.3.

For a Dirichlet Laplace problem inside some domain in $\mathcal{R}^3$, it is possible to solve for the potential u due to some density ϕ on a boundary $\partial\omega$. As the boundary surface is in $\mathcal{R}^3$, consider it as parameterized by the parameters s and t. Consider that the density, ϕ, is defined by $\phi(s,t)$ as opposed to the point $\mathbf{y}$. Further, any point $\mathbf{y}$ on $\partial\omega$ is given by $(x(s,t), y(s,t), z(s,t))$ for some s, t. The density function is described by the following Fredholm equation of the second kind:

$$\phi(s,t) + \frac{1}{2\pi} \int_{\partial\omega} \phi(s,t) \frac{\partial}{\partial \mathbf{n}} \frac{1}{r} d\partial\omega = 2\tilde{g}(s,t) \tag{4.4}$$

In equation 4.4, $\tilde{g}$ represents the new modified boundary condition, $\tilde{g} = g - u_1|_{\partial\omega}$, where u_1 is interpolated from the grid points Ω. Various interpolation strategies are possible. For example, (Ying et al., 2004a) use Lagrange interpolation, and (Mayo, 1985) suggests cubic or quintic spline interpolation. Let equation 4.4 be written as

$$(\phi + G\phi)(\mathbf{x}) = \tilde{g} \tag{4.5}$$

For evaluating the integral equation, (Ying et al., 2006) let the boundary Γ be the union of overlapping patches, P_j, $j = 1, ..., m$ parameterized over an open set $U_j \subset \mathcal{R}^2$ by smooth functions, $g_j : U_j \to \mathcal{R}^3$, $g_j(U_j) = P_j$. They use a partition of unity, $w_j : \Gamma \to \mathcal{R}$ where $w_j \geq 0$ is smooth and supported in P_j and $\forall \mathbf{x} \in \Gamma$, $\sum_{j=1}^{m} w_j(\mathbf{x}) = 1$. They let J_j be the Jacobian of g_j and define $\psi_j(c_j) = w_j(g_j(c_j))\phi(g_j(c_j))J_j(c_j)$, and the integral equation, $(G\phi)(\mathbf{x})$ is restricted to the domain of each U_j such that the integral equation operator can be represented as

$$\begin{aligned}(G\phi)(\mathbf{x}) &= \sum_{j=1}^{m} \int\limits_{U_j} G(\mathbf{x}, g_j(c_j))w_j(g_j(c_j))\phi(g_j(c_j))J_j(c_j)dc_j \\ &= \sum_{j=1}^{m} \int\limits_{U_j} G(\mathbf{x}, g_j(c_j))\psi_j(c_j)dc_j\end{aligned}$$

Note that ψ_j vanishes on ∂U_j. Then, regularly spaced gridpoints are chosen on U_j as the set $\{c_{j,i}\}$ and the set $C = \cup_{j=1}^{m}\{c_{j,i}\}$ is the set of grid points used for approximating the integral equation. That is, these are used as the Nyström points in the quadrature method. The equation is broken into three parts, for non-singular evaluation and singular evaluation, such that they compute integrals of the form

$$\int\limits_{U} G(\mathbf{x}, g(c))\psi(c)dc. \tag{4.6}$$

If $\mathbf{x}$ is not in $g(U)$, then equation 4.6 is non-singular and can be evaluated using the trapezoidal rule and FMM (Ying et al., 2006). Otherwise, if $\mathbf{x} = g(\hat{c})$, $\hat{c} \in U$, introduce the function,

$$\eta_{\hat{c}}(c) = \chi\left(\frac{|c-\hat{c}|}{\sqrt{h}}\right)$$

The function $\chi : [0,\infty) \to [0,1]$ is chosen to be infinitely differentiable and non-increasing such that $\chi(r) = 1$ near $r = 0$ and $\chi(r) = 0$ when $r \geq 1$. Further details are available in (Ying, 2004). Then, equation 4.6 is broken into singular and non-singular parts,

$$\int_U G(g(\hat{c}), g(c))\psi(c)dc = \int_U G(g(\hat{c}), g(c))(1-\eta_{\hat{c}}(c))\psi(c)dc + \int_U G(g(\hat{c}), g(c))\eta_{\hat{c}}(c)\psi(c)dc.$$

The first part is non-singular and can again be evaluated with the trapezoidal rule and FMM. For the second part, a change to polar coordinates is utilized to obtain a smooth solution, where FFT-acceleration is used to interpolate the quadrature points from the Cartesian grid to the polar grid (Ying, 2004).

4.1.2 Solving the Laplace Equation

We have now described how to compute u_1 and ϕ on $\partial\omega$, and we wish to solve the Laplace equation $\Delta u_2 = 0$ in Ω where $u_2 = g - u_1$ on $\partial\omega$. One method involves computing jump conditions as in (McKenney et al., 1995; Ying et al., 2004a); an additional approach is to evaluate u_2 near the boundary using nearly singular integration; (Ying et al., 2006) provide a technique for nearly singular integration near a smooth boundary in 3D. They suggest using the integral equation based on the fundamental solution to the Laplace equation. Away from the boundary, the evaluation is fine and non-singular; however, near the boundary the kernel, $G(\mathbf{x}, \mathbf{y})$ will become singular. This newer approach breaks the domain Ω into three disjoint partitions, Ω_0, Ω_1 and Ω_2. Points in Ω_0 are considered *well-separated* from the boundary, points in Ω_1 are *intermediately near* and points in Ω_2 are *near* the boundary. For surface discretization spacing $h_s < 1$, they define well-separated as being greater than $\sqrt{h_s}$ away from the boundary, intermediately-near as being from h_s to $\sqrt{h_s}$ away, and near as being within h_s distance of the boundary. Well-separated points in Ω_0 are evaluated using direct quadrature as above, where weights are determined using the trapezoidal rule. For intermediate points in Ω_1, they resample on a finer grid using FFTs, and then compute the quadrature. For points $\mathbf{x}$ in Ω_2, they locate a point on $\mathbf{x}_0$ on the boundary such that for $\alpha < 1$ but ≈ 1; the following property thus holds that

$$\frac{\mathbf{x}_0 - \mathbf{x}}{||\mathbf{x}_0 - \mathbf{x}||} \cdot \mathbf{n}(\mathbf{x}_0) \geq \alpha.$$

Then, a set of points $\{\mathbf{x}_j\}$, $j = 1, ..., L$ for some L are located such that

$$\mathbf{x}_j = \mathbf{x}_0 + j \frac{\mathbf{x}_0 - \mathbf{x}}{||\mathbf{x}_0 - \mathbf{x}||} \cdot \mathbf{n}(\mathbf{x}_0) \beta h.$$

The constant β is chosen so that $\alpha\beta < 1$ but ≈ 1. Using the jump conditions and singular integral evaluations, the potential at $\mathbf{x}_0$ can be obtained (Ying, 2004). For points $\mathbf{x}_j$ in Ω_0, normal quadrature is performed, and for points in Ω_1, the above method is used. With the potential at these points known, interpolation is used to obtain the values at $\mathbf{x}$.

The boundary is considered to be built as in the last section 4.1.1, with Nyström points chosen as before. Figure 4.2, reproduced from (Ying et al., 2006), provides an illustration of their method. In general, this method provides a way of using the kernel-independent FMM to accelerate the solution procedure for the Laplace equation.

4.1.3 Putting It Together

Assuming a solution for u_1 is computed using the free-space FMM-based volume integral solver, and u_2 with appropriate jump corrections is solved as discussed above, we can now add the two solutions together such that $u = u_1 + u_2$ is the solution for requested interior Dirichlet PDE in equation (4.1).

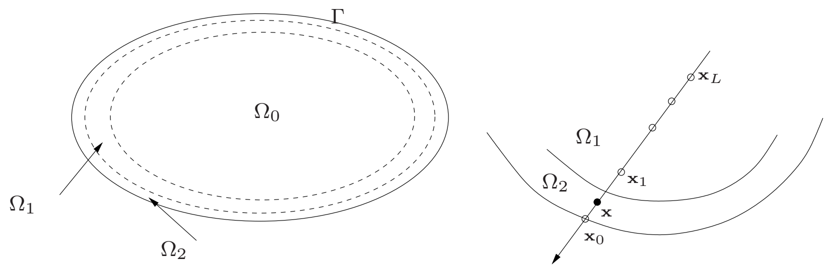


Figure 4.2: Reproduced from (Ying et al., 2006), this figure shows how to perform the nearly singular integration and further details the notation

4.2 Extending the Body Forces When Boundaries are Present

While we have described a modification for taking into account non-uniformly distributed body forces for the EBI solver, this still assumes that the force is supported everywhere in free space while user input may only provide forces for the interior of the desired domain of computation. As we want to approximate the forces in a leaf node at k^{th}-order accuracy, for example, leaf nodes which intersect the boundary may have too few points for the desired accuracy; hence, forces will have to be extended or extrapolated through the boundary. There are different approaches that could be employed, including extrapolation techniques in Ghost Fluid Methods (Fedkiw et al., 1999; Aslam, 2003), seen in level set approaches (Osher and Fedkiw, 2002). However, we can follow the method of (Ethridge, 2000) as we already have the tree restrictions as required. We outline the approach for 2D below.

4.2.1 Previous 2D Approach

Assuming $f(\mathbf{x})$ is given only in the interior of some complex domain, additional considerations need to be made in order to utilize the FMM-based free-space volume solver. In particular, there are four problems: (1): the boundary of the domain for computation must be defined, (2): volume grid points which are inside the domain need to be located, (3): irregularly shaped regions in leaf nodes need to be defined, and (4): forces must be approximated within each irregular subdomain.

There are several ways of shaping the domain, and defining the boundary curve. In (Ethridge, 2000), each curve is approximated by a piecewise cubic polynomial. Further, there are several approaches for determining which leaf nodes curves intersect, and (Ethridge, 2000) uses a Newtonian iteration procedure. Leaf nodes are then divided into three categories: irregular nodes which contain a piece of the boundary, semiregular nodes which are neighbors of irregular nodes, and regular nodes which are neither irregular nor semiregular. Figure 4.3 shows an example of this classification in 2D.

Each regular and semiregular box contains the $k \times k$ (i.e., 4×4) grid of points, but irregular boxes may not. For the volume integral, it is necessary to determine which points are inside of the computational domain. For a bounded domain, Ω with boundary $\partial\Omega$, the following result holds from (Kress, 1999):

$$u(\mathbf{z}) = \frac{1}{2\pi\mathbf{i}} \int_{\partial\Omega} \frac{d\mathbf{w}}{\mathbf{w}-\mathbf{z}} = \begin{cases} 1 & \text{if } \mathbf{w} \in \Omega \\ \frac{1}{2} & \text{if } \mathbf{w} \in \partial\Omega \\ 0 & \text{if } p \in \mathcal{R}^2 \backslash \bar{\Omega} \end{cases}$$

Using FMM, $u(\mathbf{z})$ can be quickly and accurately evaluated to determine irregular grid points in the interior. For unbounded problems, a similar result holds (Kress, 1999; Ethridge, 2000).

Another problem is that each irregular leaf box may contain several intersecting curves, which are located in (Ethridge, 2000) by "walking" along each curve and tracing out interior subregions in irregular leaf nodes. Locating interior subregion points is then performed via a similar method as above.

Once boundaries of irregular regions have been determined and gridpoints have been located,

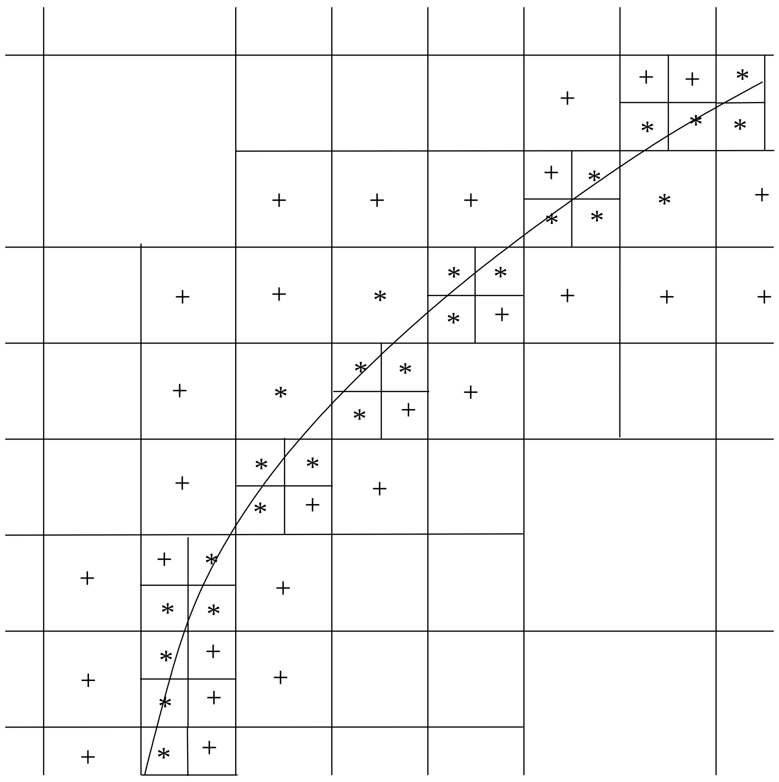


Figure 4.3: Classification of leaf nodes. Nodes marked by an asterisk, $*$ are irregular, those marked with a plus, $+$ are semiregular, and the rest are regular.

there is the issue of approximating the polynomial there as described in the last section. In order to maintain a high-level of accuracy, the force needs to be known at a sufficient number of points; however, situations such as figure 4.4 may exist, where too few points are available.

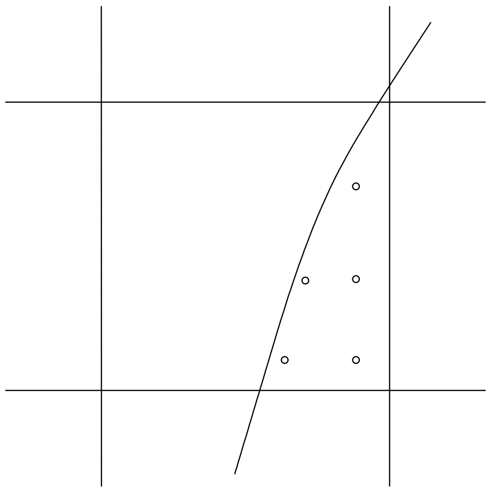


Figure 4.4: An irregular box where only a few source points are known. In order to approximate the force at this node, extrapolation or other techniques are necessary.

If there are enough points, it may be possible to maintain a high-order approximation, but in order to maintain the benefits of the precomputable tables, (Ethridge, 2000) takes a different approach. The irregular region is embedded in a box of the same size of that level of the tree such that k^2 source points are available (16 for $k = 4$) if possible; that is, the portion of the boundary is embedded along with as much of the interior of the domain of computation as possible. The situation is broken into three possibilities, where the irregular domain contains 0, 1, 2, or 3 corners of the bounding box at that level. For example, in figure 4.4, the region contains the bottom-right and top-left corners, so 2 corners are bounded. *Bounding boxes* are built such that they include all of the original corners. This makes interpolation strategies easier (Ethridge, 2000). An example for figure 4.4 is seen in figure 4.5.

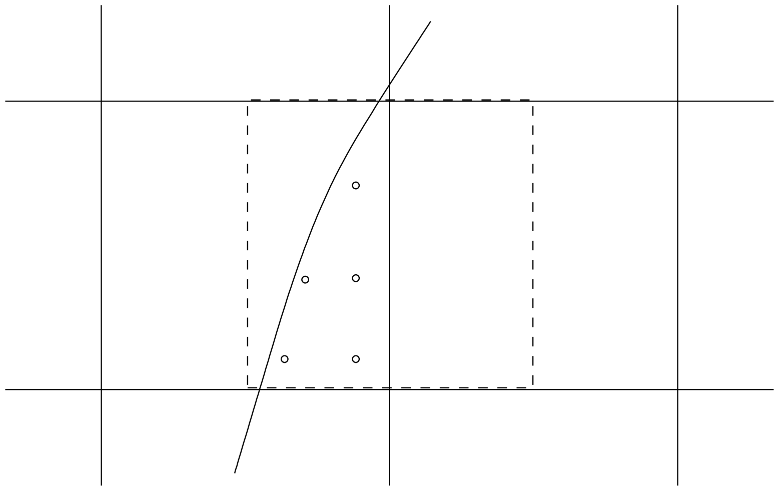


Figure 4.5: Irregular regions are bounded by boxes of the same scale as boxes at that level in the hierarchy, such that all corners of the irregular leaf box are included, and as much of the interior domain is included as well. The bounding box is the dashed box.

Points that are available for interpolation are decided to be those which are within the bounding box, or are in boxes which the bounding box intersects where the points are still inside the interior of the domain, or are in boxes which share a boundary with the bounding box. For the example above, the points available for interpolation are seen in figure 4.6, which expands figure 4.5 and rotates it.

With the locations of all available source points, (Ethridge, 2000) first tries extrapolating forces across the boundary. As the sources are on a grid, 1D extrapolation may be possible for collinear points. This process is repeated until as many possible extrapolation points are generated, and then the precomputed interpolation is used. Otherwise, if not enough points can be generated, a variety of least squares techniques are employed. If this does not work, the

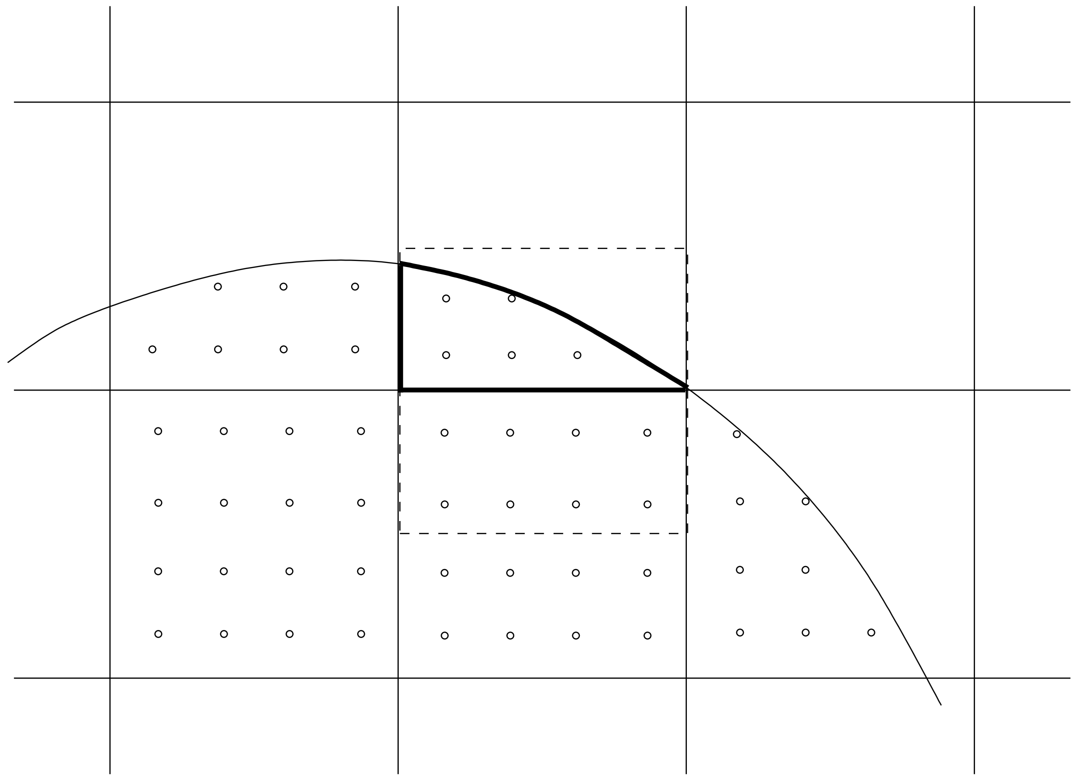


Figure 4.6: The leaf node which includes the irregular region (solid black lines) only has a few available internal points. The points that are available for interpolation and extrapolation purposes are located by seeing which leaf nodes touch the bounding box (dashed lines) of the irregular region.

order of least squares fitting is reduced until some order of accuracy is obtained, and the user is notified.

4.2.2 Current 3D Approach

We re-emphasize that the key point for the volume solver in this context is that we require an accurate polynomial representation at every leaf box B of f^B, B's support of the force f. As before, we refer to this polynomial as γ^B.

To motivate the construction of γ^B, we begin with an example in which f^B is given on grid of size $k \times k \times k$ (hereafter referred to as a k^3 grid). A k^{th} order approximation, γ^B could then be constructed as

$$f^B(\mathbf{x}) \approx \sum_{j=1}^{N_k} \gamma_j^B \beta_j(\mathbf{x} - \mathbf{c}_B), \tag{4.7}$$

where $N_k = k(k+1)(k+2)/6$, and β_j is the j^{th} basis polynomial in our representation (for $k = 3, 4, 5, 6$, we use monomials and require k^3 equispaced points in practice. As discussed earlier, this system is overdetermined (Trefethen and Bau, 1997), so we can solve (4.7) for γ^B using the SVD.

In general, the above approach does not work for obvious reasons. Since forces may not always be represented on a grid, we may not have enough points to accurately and stably compute γ^B, especially since we will require our tree to be balanced; that is, assuming T has been subdivided to have at most s points per leaf box B, subsequent balancing for tree-level restriction may result in some leaves with very few or no points. To compensate for such leaves, we utilize the data structures available to us through the various lists to build γ^B at sufficiently high order. Further, even for forces on a grid, we may have a situation in which a leaf node, B, intersects the curve and only a fraction of its grid points are contained within the computational domain.

For example, consider figure 4.7, where several leaf nodes lack enough grid points to accurately compute the polynomial approximation to the degree requested.

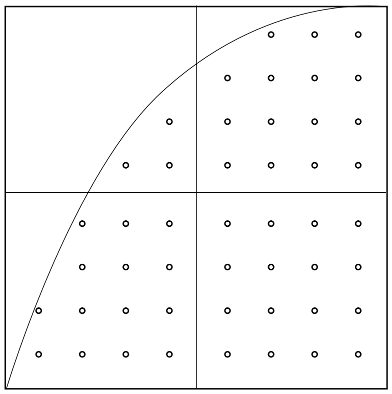


Figure 4.7: A subset of a 2D tree T in which four leaf boxes are shown with an overlapping domain. Points within the domain are represented, and as can be seen, several boxes do not have full grid representations. Here, $k = 4$.

It can be seen that by evaluating γ^B at a limited grid of size $\tilde{k}^3 < k^3$, one of two situations is possible. First, the number of the points used in evaluating γ^B may be too small in order to build a sufficiently smooth kth order polynomial approximation, and forcing it to do so would result in an inaccurate representation of f^B or worse, a discontinuity could result in large oscillations of the force here (Demmel, 1997; Trefethen and Bau, 1997). One option is to decrease the order of the polynomial approximation at such leaf boxes, but this affects the overall requested accuracy of the solver by making it low accuracy at specific locations. In order to see the effects of local low-order approximation, we return to equations (2.4)- (2.6). Again, let $\hat{f}(\mathbf{x})$ be the kth-order polynomial approximation to $f(\mathbf{x})$ such that $|\hat{f}(\mathbf{x}) - f(\mathbf{x})| \leq \delta = O(h^k)$, and let $\hat{u}(\mathbf{x}) = \hat{\mathbf{Q}}[f](\mathbf{x})$ denote the quadrature approximation of $\int_\Omega K(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) dy$. Assuming the near field is computed exactly, the quadrature error satisfies an estimate of the form

$$\left|\hat{\mathbf{Q}}[f](\mathbf{x}) - \int_\Omega K(\mathbf{x},\mathbf{y})f(\mathbf{y})dy\right| \leq \epsilon\|f\|_1,$$

where ϵ is the approximation error in the FMM, controlled by the parameter p, the number of discretization points used for building the equivalent densities, as described in (Ying et al., 2004b; Langston et al., 2011) and $\|f\|_1 = \sum_i |f_i|$ as before for sampled f_i. Then, if ϵ is chosen to be on the same order as δ, we have a k^{th}-order scheme in terms of accuracy. However, if we know that $\delta = O(h^{\tilde{k}})$ locally in some area $\tilde{\Omega} \subset \Omega$, we can make no guarantee on the global accuracy of the scheme. It is therefore important to build smooth extensions to the force f on leaf boxes which overlap our boundary.

4.2.3 Generating List of Boxes for Which Extension Necessary

Before discussing how we perform our extension, we begin by describing for which boxes extension will be necessary. This further requires the discussion of how we build our adaptively-refined trees.

Assuming we are given our right-hand side function $f(\mathbf{x})$ as an input routine, defined on each leaf box B as $f_k^B(\mathbf{x})$, sampled at all k^3 discrete equispaced points which are in the interior of our domain, we compute a k^{th}-order polynomial approximation, γ^B using the same strategy as outlined in (Langston et al., 2011). If B is fully enclosed within our domain, we compute the approximation from the full grid of points; for boxes overlapping the boundary, we use the subset of k^3 points inside of our domain. As in (Ethridge and Greengard, 2001), we evaluate γ^B on a refined $(2k)^3$ grid (or subset of such points that are in the interior of boxes overlapping the domain), denoting this approximated evaluation of f as $f_{2k}^B(\mathbf{x})$, and compute the error approximation $\left|\left|f_{2k}^B(\mathbf{x}) - \tilde{f}_{2k}^B(\mathbf{x})\right|\right|_2$. For a pre-specified tolerance, ϵ_{rhs}, B and its descendants are recursively subdivided as long as the error in the force approximation on any leaf box is greater

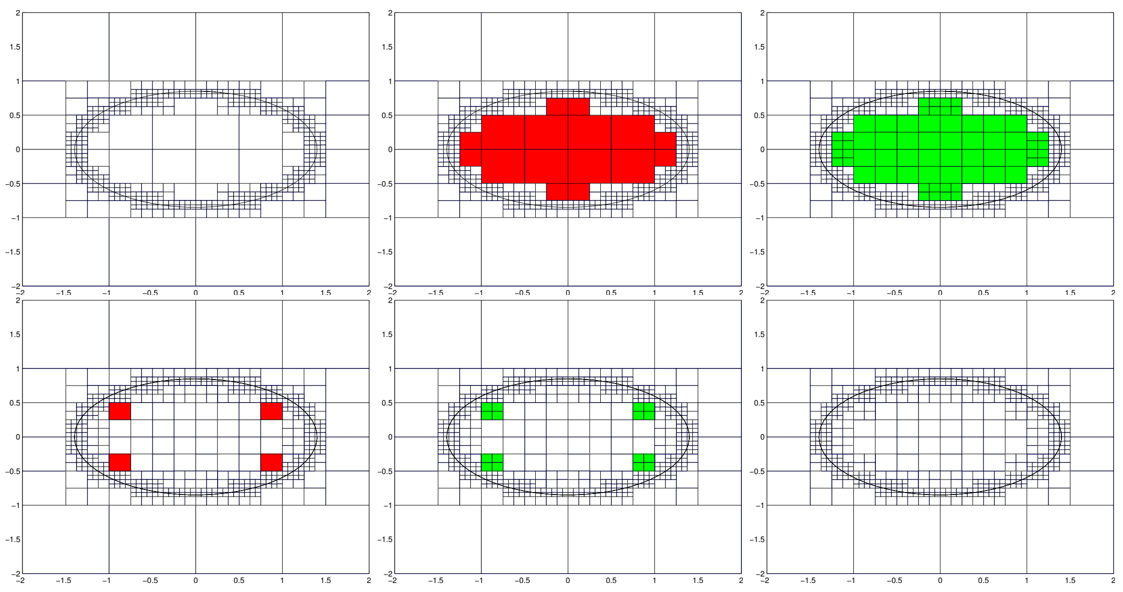


Figure 4.8: *Top left*: We begin with a subdivided domain which is not balanced; *Top middle*: In this example, investigating an interior problem, we are not concerned with exterior boxes yet. We identify all primary violators of the tree-level restriction (in red); *Top Right*: All violators are subdivided and their descendants (in green) are identified; *Bottom left*: All descendants that are in violation of tree-level restriction are identified (in red); *Bottom middle*: All violators subdivided and descendants identified (in green); *Bottom right* All tree-level violations inside of the domain have been taken care of.

than ϵ_{rhs}. As in (Langston et al., 2011), we must further *balance* the resulting octree, T such that all adjacent leaf boxes are within one level of each other in the tree. For an example of our subdivision process, we refer to figure 4.8, which begins by showing a domain in the top-left which has been subdivided until our force approximation tolerance is satisfied but the tree-level restriction is violated. The remaining figures step through the algorithm to subdivide the tree until all the tree-level restriction is satisfied.

We note in figure 4.8 that in the tree-balancing process, we only balance among boxes that are in the interior of our domain or overlap the boundary; this is due to the fact that we only balance among boxes which contain target points in the computation of the free-space computation. Further, for boxes far from the boundary, no extension will be performed, and the polynomial approximation will be zero there, so these boxes do not contribute as sources.

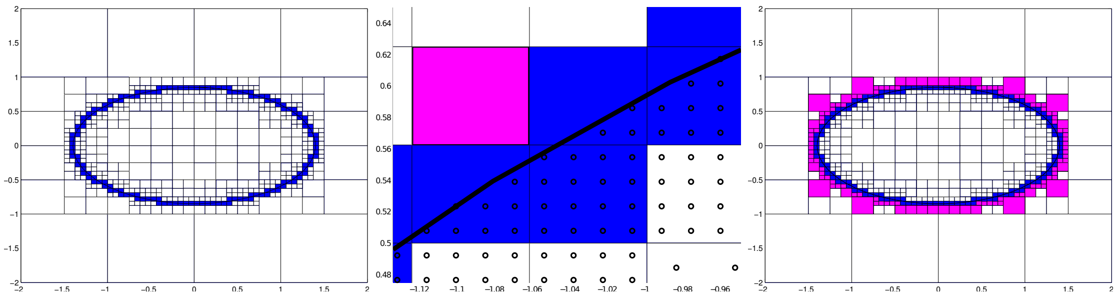


Figure 4.9: We identify all leaf boxes for which extension will need to be performed. *Left*: All leaf boxes which overlap the boundary are tagged (in blue); *Middle*: The boundary may pass very close to an exterior box, so we must extend the force here as well; *Right*: All leaf boxes which are in the exterior of the boundary and are direct neighbors with a leaf box which overlaps the boundary are identified (in purple).

We now turn to locating all boxes for which the extension will be performed. In figure 4.9(*left*), any leaf box overlapping the boundary is marked in blue. For such boxes, if any of the k^3 grid points lay outside of the domain, extension will be performed. However, for some leaf boxes, the curve passes near the corner or edge as in figure 4.9(*middle*). In such cases, the force approximation will experience a sharp-dropoff as a result of not including the exterior box (in purple) as a source box, so we additionally extend the force to exterior boxes.

In figure 4.9(*right*), we identify all boxes in the exterior of our domain which are adjacent to a box overlapping the boundary. However, for a box B, which overlaps the boundary, exterior

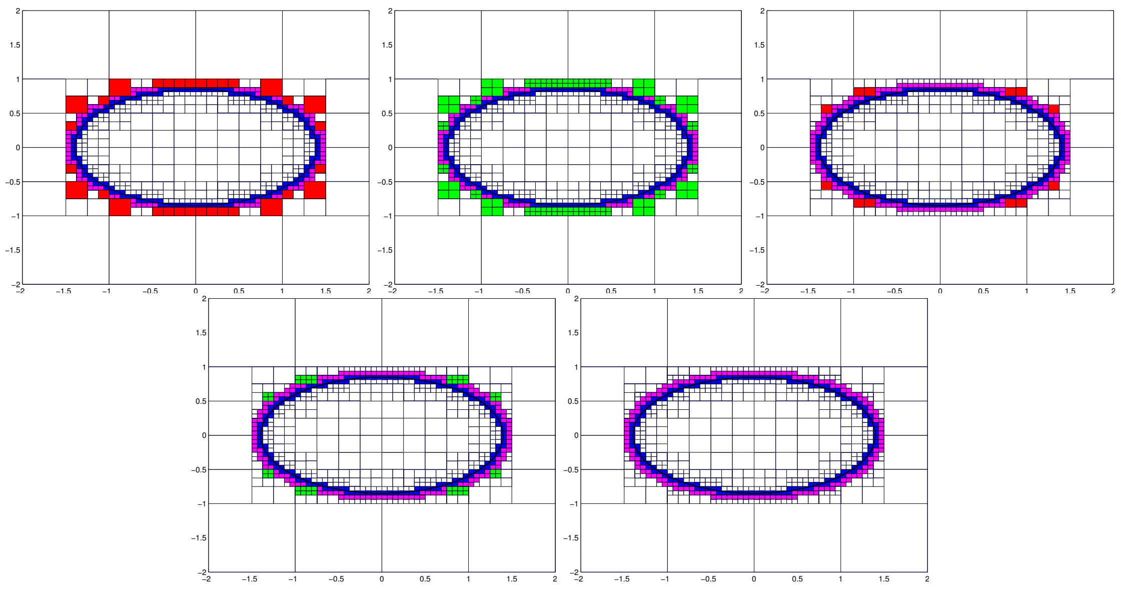


Figure 4.10: *Top left*: All leaf boxes in the exterior, for which extension will be performed but are in violation of our strict 1:1 level restriction are identified (in red) while all exterior leaf boxes not in violation are left as is (in purple); *Top middle*: All violators are subdivided and descendants are identified; *Top right*: All descendants leaf boxes in the exterior which directly touch a leaf box overlapping the boundary and are at a level more shallow are identified as in violation (in red); *Bottom left*: Violators are subdivided and descendants tagged (in green); *Bottom right*: All boxes, for which extension will be performed are identified: blue boxes overlap the boundary while purple boxes have an boundary-overlapping leaf box as a neighbor, now at the same depth in the tree.

boxes, which are adjacent, may be at shallower levels in the tree. Extending to such boxes may result in a non-smooth extension, so we subdivide all exterior boxes which are adjacent to the boundary, and all resulting boxes which are adjacent to an overlapping box are chosen for extension. In figure 4.10, we show the process of subdividing all such exterior boxes and highlighting which resulting boxes are chosen for extension.

By extending by one box outwards, we confront the following issue: as we refine the volume further, we are extending the force by a smaller distance r from the boundary, creating a sharper drop-off. In order to observe the issue, let B_H be a box of width H such that for some target point $\mathbf{q}$

$$P_H(\mathbf{q}) = \int_{B_H} \frac{d\mathbf{y}}{||\mathbf{y} - \mathbf{q}||}.$$

Then we have

$$\int_{B_1} \frac{H^3 d\mathbf{y}}{\left|\left|H\left(\mathbf{y} - \frac{\mathbf{q}}{H}\right)\right|\right|} = H^2 P_1\left(\frac{\mathbf{q}}{H}\right).$$

By the chain rule,

$$\frac{d}{d\mathbf{q}} P_H(\mathbf{q}) = H^2\left(\frac{1}{H}\right)\frac{d}{d\mathbf{q}} P_1\left(\frac{\mathbf{q}}{H}\right).$$

Hence,

$$\left(\frac{d}{d\mathbf{q}}\right)^n P_H(\mathbf{q}) = H^{(2-n)}\left(\left(\frac{d}{d\mathbf{q}}\right)^n P_1\left(\frac{\mathbf{q}}{H}\right)\right)$$

For a box near the edge of the boundary at a distance of r from the closest surface discretization point, one can expect the following rate of decay for $n > 2$:

$$\left(\frac{d}{d\mathbf{q}}\right)^n P_1 \sim \left(\frac{1}{r}\right)^{n-2}.$$

Hence, we obtain

$$\left(\frac{d}{d\mathbf{q}}\right)^n P_H \sim H^{(2-n)}\left(\frac{H}{r}\right)^{n-2} = \left(\frac{1}{r}\right)^{n-2}.$$

Let h_s be the discretization sampling distance on the surface, S, and r be the closest box-boundary distance to S. If m is the interpolation order of the free-space volume solution on S, then the expected error rate is

$$E \sim \left(\frac{1}{r}\right)^{m-2} {h_s}^m = r^2 \left(\frac{h_s}{r}\right)^m .$$

Hence, if we perform an extension outside of S by one box (assuming all boxes on boundary and one step away are of width H), then we have $r \sim H$, implying $E \sim H^2 \left(\frac{h_s}{H}\right)^m$. It is clear that in order to at least achieve good results with an extension of one box distance, H beyond the boundary (again assuming all boxes on the boundary have width H), we must at least have $h_s < H$. For example, if we refine the grid to $H' = H/2$ and leave h_s as is, we now have $E \sim 2^{m-2} H^2 \left(\frac{h_s}{H}\right)^m$. However, if we instead extend by a distance of $2H'$, we obtain, $E \sim (2H')^2 \left(\frac{h_s}{2H'}\right)^m = H^2 \left(\frac{h_s}{H}\right)^m$, so our error remains constant by extending further. So, it is clear that either we need to maintain a relationship of refinement between h_s and H, or extend further as we refine H, or both. This would appear problematic, calling instead for us to extend at a fixed distance. Unfortunately, extending in such a way would be unstable as we refine the volume, so we must guarantee that we refine h_s accordingly with H in order to maintain an appropriate ratio for h_s/H.

4.2.4 Extension Beyond the Boundary

Given a box, B, assume we have an under-resolved grid of points, $\hat{\mathbf{x}}^{B,g}$, a subset of a full grid, $\mathbf{x}^{B,g}$, such that $\hat{\mathbf{x}}^{B,g}$ only includes points in the interior of our boundary; we assume we are given the force values, $f(\hat{\mathbf{x}}^{B,g}) = \hat{f}^{B,g}$ solely at these interior points. For example, consider the partial grid of points highlighted in red in figure 4.11(*left*). For such a leaf box B, construct a *coefficient-to-coefficient mapping* approach. Let B have width $H = 2r$ and center $\mathbf{c}_B$ such that B has fewer than N_{min} points (in practice, we set $N_{min} >= Ck^3$ for k being the order of approximation and C some constant > 2). We then attempt to construct a region, $\tilde{B}$ such that $B \subset \tilde{B}$, $\tilde{B}$ contains at least N_{min}. For example, in figure 4.11(*middle, right*), we highlight regions in red as we search outward for points.

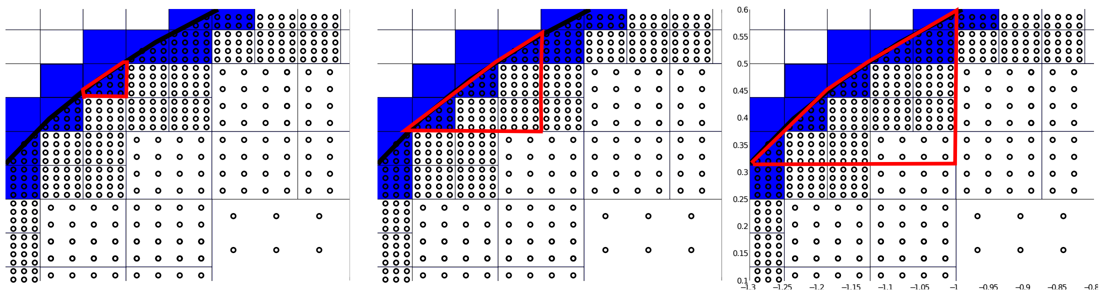


Figure 4.11: *Left*: Select a box B which overlaps the boundary and has an under-resolved grid of points in the interior of the domain; *Middle*: Locate immediately adjacent neighbors; *Right*: Locate boxes further away for inclusion in extrapolation

Locate all boxes within $3H$ of $\mathbf{c}_B$ which are interior or overlapping leaf boxes. Fortunately, L_U^B, L_W^B, L_X^B, and L_V^B provide most of the information needed and a quick way of traversing T. If there are not enough points in these boxes, investigate the boxes in L_V^P for B's parent, P, since it is clear that L_V^P is disjoint from the union of L_U^B, L_W^B, L_X^B, and L_V^B. While we may obtain more points than we need, for the set of located points, X, we quicksort all $\mathbf{x} \in X$ and ignore all points where $||\mathbf{x} - \mathbf{c}_B||_\infty > 3H$. For all other points, denote this region as $\tilde{B}$ and construct $\tilde{\gamma}^{\tilde{B}}$ from all $\mathbf{x} \in \tilde{B}$ with source values $f^{\tilde{B}}$:

$$f^{\tilde{B}}(\mathbf{x}) \approx \sum_{j=1}^{N_k} \gamma_j^{\tilde{B}} \beta_j(\mathbf{x} - \mathbf{c}_B). \tag{4.8}$$

Since $B \subset \tilde{B}$, $\gamma^{\tilde{B}}$ can be evaluated inside of B. That is, let $f^{B,g}$ be the restriction of $f^{\tilde{B}}$

onto a k^3 grid on B, located at grid points $\mathbf{x}^{B,g}$ such that

$$f^{B,g}(\mathbf{x}) = \sum_{j=1}^{N_k} \gamma_j^{\tilde{B}} \beta_j(\mathbf{x} - \mathbf{c}_B) \quad \text{for all } \mathbf{x} \in \mathbf{x}^{B,g}. \tag{4.9}$$

For any points $\mathbf{x} \in \mathbf{x}^{B,g}$ which are interior points, we use the original values, not the interpolated ones as our overdetermined system does not pass directly through the known f values there. With our newly-constructed $f^{B,g}$, we now use (4.7) to construct γ^B, and given the size of $f^{B,g}$, we are guaranteed that the system is sufficiently over-determined. We outline the steps algorithmically in the following pseudocode. We assume that T has already been built as well as all lists for a box B.

Algorithm 4 Build Coefficients for a box B with width H

```
STEP 1 - LOCATE INTERPOLATION POINTS
Let $N = 0$ and iterate through $\mathcal{N}^B$ and $L_I^B$
for each list $L = L_U^B, L_W^B, L_X^B, L_V^B$ do
    while $N < N_{min}$ do
        for $\mathbf{x} \in A$ with force density $f^B(\mathbf{x})$ for each box $A \in L$ do
            store $\mathbf{x}$ in $P[\lfloor \mathbf{x} - \mathbf{c}_B \rfloor / H]$ and set N = N+1
        end for
    end while
end for
if $N < N_{min}$ then
    Let $P = Parent(B)$
    while $P \neq Root(T)$ AND $N < N_{min}$ do
        for $\mathbf{x} \in A$ with force density $f^B(\mathbf{x})$ for each box $A \in L_V^P$ do
            store $\mathbf{x}$ in $P[\lfloor \mathbf{x} - \mathbf{c}_B \rfloor / H]$ and set N = N+1
        end for
        $P = Parent(P)$
    end while
end if
for $i = 0$ to $\ell$ = length($P$) do
    Quicksort $\mathbf{x} \in P[i]$ relative to $\mathbf{x}$'s distance from $\mathbf{c}_B$
    if $||\mathbf{x} - \mathbf{c}_B||_\infty < 3H$ then
        Store $\mathbf{x}$ in $X$
    end if
end for
Let $\tilde{B}$ be the region enclosing all points $\mathbf{x} \in X$ with force values $f^{\tilde{B}}(\mathbf{x})$
STEP 2 - COMPUTE APPROXIMATION FOR $B$
Compute $\gamma^{\tilde{B}}$ from $f^{\tilde{B}}$ using (4.8)
Evaluate $f^B$ at $\mathbf{x}^{B,g}$ from $\gamma^{\tilde{B}}$ using (4.9)
Compute $\gamma^B$ from $f^{B,g}$ using (4.7)
```

In practice, we search B's parent's interaction list, and if unable to collect enough points after traveling this far in the tree, we reduce the order of our polynomial as necessary. If we cannot even locate a single point, B's coefficients are all zero, and B is ignored in the computation stage.

For a uniformly-refined tree, figure 4.12 illustrates the coefficient construction process.

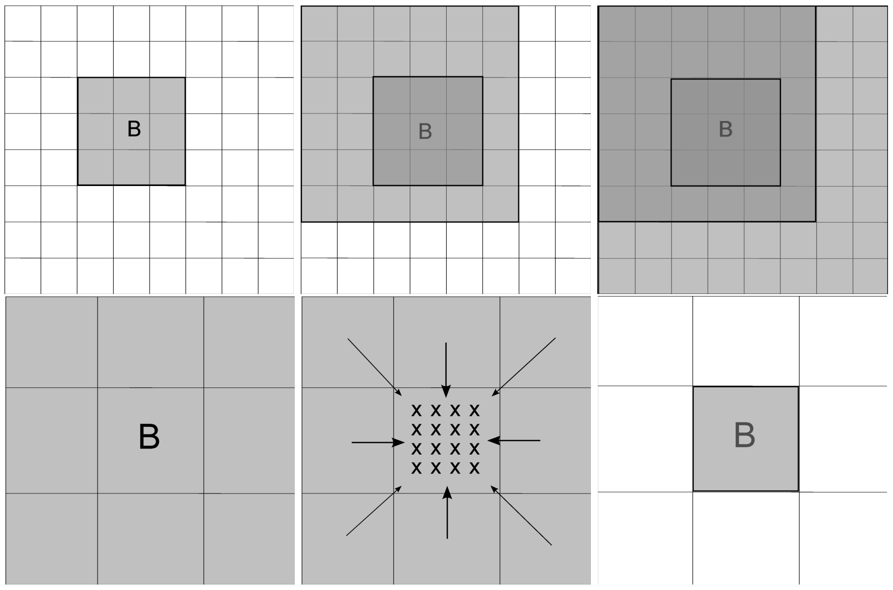


Figure 4.12: From Top-Left to Top-right. For a box B, we assume B does not have a sufficient number of points, so we look at the boxes in its near-field (marked on the *a*). If there are still not enough points, we search through B's interaction list (*b*). While there are still not enough points to build an interpolant, we recursively search B's ancestors' interaction lists (*c*). Assuming the region marked contains a sufficient number of points (in this case, this region is $\mathcal{N}^B$), let this larger region be $\tilde{B}$. Build an interpolant to the forces in $\tilde{B}$ (*d*). In (*e*), we evaluate this polynomial approximation at grid points $\mathbf{x}^{B,g}$ in B. In (*f*), we use the evaluated grid-located forces to compute B's local approximation to the force distribution.

4.2.5 Interpolating Solution Values to Boundary

Given our final solution, $u(\mathbf{x})$ at all $\mathbf{x} \in \mathbf{x}^{B,g}$ for some box B, we need to evaluate u at locations not on our grid. Fortunately, we can use the existing mechanism to build a polynomial approximation to our solution value at the grid locations and interpolate to random locations in B. Given our solution $u^{B,g}$ of size k^3 on all $\mathbf{x} \in \mathbf{x}^{B,g}$, we compute a k^{th} order polynomial approximation, α^B,

$$u^{B,g}(\mathbf{x}) \approx \sum_{j=1}^{N_k} \alpha_j^B \beta_j(\mathbf{x} - \mathbf{c}_B), \tag{4.10}$$

and evaluate u at a location, $\mathbf{x}_t \in B$, $\mathbf{x} \notin \mathbf{x}^{B,g}$

$$u^B(\mathbf{x}_t) = \sum_{j=1}^{N_k} \alpha_j^B \beta_j(\mathbf{x}_t - \mathbf{c}_B) \tag{4.11}$$

If a target point $\mathbf{x}_t$ passes near an edge or corner of a box, the polynomial approximation can evaluate poorly at this location, especially if the solution values change rapidly near box boundaries. This is especially true for points which represent the discretization of the boundary, for which we need to calculate u_1 from equation (4.2) for modifying the boundary value condition for the homogeneous PDE solver in equation (4.3). In order to correct for this, interpolate u from a *cell-centered* grid of points. That is, locate all target grid points in B and its nearby neighbors such that $\mathbf{x}_t$ is in or near the center of a box with surrounding u values. Refer to this box with $\mathbf{x}_t$ near its center as $\tilde{B}$. In figure 4.13, we show how this can occur at a target point and the sort of domain, from which we wish to interpolate the solution at $\mathbf{x}_t$.

Locating the cell-centered box of points in $\tilde{B}$ is straightforward, and we present pseudocode in Algorithm 5.

We look at the effects of performing this cell-centered approach to interpolating to surface values. As an example, consider the solution value, u on a sphere of radius $R = 0.8$:

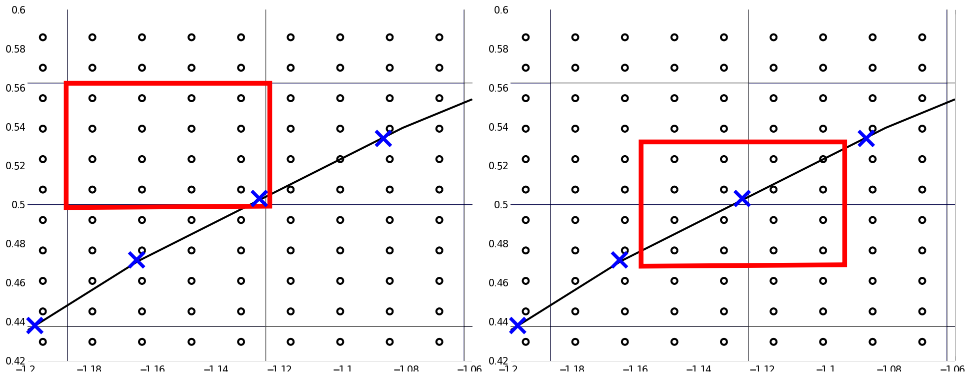


Figure 4.13: *Left*: For surface discretization points (in blue) and for a box such as B in red with grid solution locations (represented by circles), we can see that interpolating to B's surface target point, $\mathbf{x}_t$ from B's solution locations is unstable; *Right*: We locate a cell-centered box $\tilde{B}$ with k^3 points with the target point near the center of $\mathbf{c}_{\tilde{B}}$.

Algorithm 5 Compute solution u at point $\mathbf{x}_t \in B$, $\mathbf{x}_t \notin \mathbf{x}^{B,g}$ for box B with radius r_B

```
STEP 1 - LOCATE INTERPOLATION POINTS
for each box $U \in L_U^B, L_W^B, L_X^B$ do
    if $||\mathbf{c}_B - \mathbf{c}_U||_\infty \leq 2r_B$ then
        for $\mathbf{x} \in U$ do
            if $||\mathbf{x}_t - \mathbf{x}||_\infty \leq r_B$ then
                Store $\mathbf{x}$ in $\mathbf{x}^{\tilde{B},g}$ and $u(\mathbf{x})$ in $u^{\tilde{B},g}$
            end if
        end for
    end if
end for
STEP 2 - COMPUTE VALUES FOR $u$ at $\mathbf{x}_t$
Compute $\gamma^{\tilde{B}}$ from $u^{\tilde{B},g}$ using (4.10) with $\mathbf{c}_B$ replaced with the calculated center $\mathbf{c}_{\tilde{B}}$
Evaluate $u$ at $\mathbf{x}_t$ from $\gamma^{\tilde{B}}$ using (4.11) and $\mathbf{c}_{\tilde{B}}$ in place of $\mathbf{c}_B$
```

$$u(\mathbf{x}) = \frac{1}{\pi}\left[atan\left(sR^2 - s\,||\mathbf{x}||^2\right) + \frac{\pi}{2}\right], \quad s = 20.$$

For $\mathbf{x} = (x, y, z)$, we place 100 equispaced points along the surface of the sphere at the slice $z = 0$. Further, we build a uniformly subdivided tree for depths $d = 4, 5, 6$. For each depth, we interpolate the function values u from the k^3 grid of points in the box B, in which each target point, $\mathbf{x}_t$ is located. We then build a cell-centered box around each target point and then interpolate from those points. We compare the difference in interpolated values in figure 4.14.

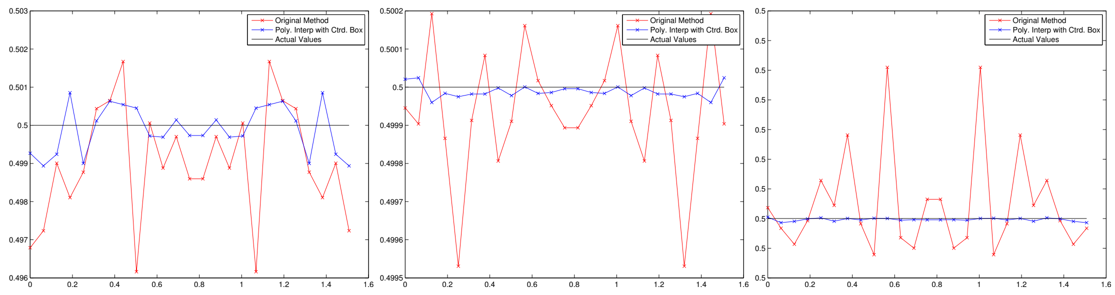


Figure 4.14: For a uniformly-refined tree to depth d, we place 100 surface points on a sphere at $z = 0$ and interpolate from the box B, in which each target point $\mathbf{x}_t$ is located, first using just B's points and then a cell-centered box $\tilde{B}$, built using the approach above. The target points can be designated as $\mathbf{x} = R(\cos\theta, \sin\theta, 0)$ for $\theta \in [0, 2\pi)$, so we only plot a single quadrant of error values for $\theta \in [0, \pi/2]$ as the symmetric nature of our test function produces a repetition in the plotted error values. *Left*: We set $d = 4$. Absolute maximum error for non-centered approach is $3.9E - 03$ and for centered approach is $1.1E - 03$; *Middle*: We set $d = 5$. Absolute maximum error for non-centered approach is $4.7E - 04$ and for centered approach is $4.0E - 05$; *Right*: We set $d = 6$. Absolute maximum error for non-centered approach is $2.5E - 05$ and for centered approach is $6.9E - 07$.

It is clear that performing interpolation, using this more centered approach is necessary to achieve reasonable accuracy, specifically at boundary locations, where the free-space volume solution is required.

5 Numerical Results for Embedded Boundary Solver

The above algorithm has been implemented in C++, and we have tested it with several kernels, different shapes, and for uniform and non-uniform source and target point distributions. Our tests were run on an Intel Xeon X5650 (2.67GHz 64 bit) system with 8 CPUs and 96GB of RAM; the major FMM computation loops are accelerated with OpenMP (Chapman et al., 2007) as discussed in (Langston et al., 2011).

We focus on three specific kernels here: Laplace, Modified Helmholtz, and Stokes; for our purposes, these are representative of the strength of our approach as the Laplace kernel is a scale-invariant, scalar kernel, Modified Helmholtz is scale-variant, and Stokes is a matrix kernel. We further tests these kernels on various shapes from the simple analytic sphere in figure 5.1(a) to the more complex shapes in figures 5.1(b-d).

We begin by looking at the Poisson equation with Laplace kernel, first verifying the boundary integral solver and the homogeneous solver. We then look at a uniformly-refined tree and a high-gradient volume force on a simple sphere to investigate the effect of extending the force versus no extension beyond the boundary. For this same high-gradient force, we also look at the effect of adaptive refinement with the extension. We further investigate the Poisson equation on more complex geometries. We then look at the Modified Helmholtz and Stokes equations with different test examples.

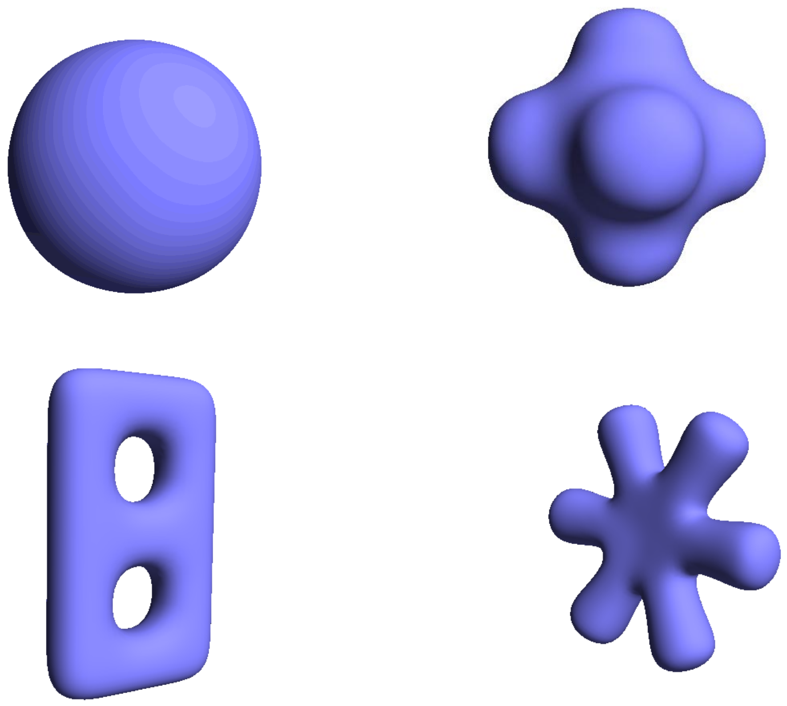


Figure 5.1: 4 shapes for which we test our solvers. a) Analytic sphere; b) Closed pipe joint; c) Two-hole torus; d) Starfish. All 4 shapes are enclosed within a box of size $[-1, 1]^3$ and all computations are run on a domain of size $[-2, 2]^3$ to guarantee adequate spatial room for solvers.

5.0.6 Poisson Equation

For the Poisson equation (equation (2.7) with kernel (2.10)), we investigate the accuracy and timings of our algorithm. We begin first by verifying the accuracy of the boundary integral solver as well as the embedded boundary solver in the absence of a volume force. We then proceed to look at the effects of our extension on a high-gradient force, first for uniform refinement and then for adaptive refinement on a simple sphere. We then look at two additional examples: one simple volume force and one more complex force for non-trivial geometries.

Boundary Integral Solver Test Verification We test the boundary integral equation using the following special test case, recreating the test from (Ying et al., 2006). Specifically

$$\Delta u(\mathbf{x}) = 0, \quad u(\mathbf{x}) = 1 \quad \text{for } \mathbf{x} \in \Omega.$$

As in the previous work, we compute the error of the singular quadrature algorithm on N boundary points from the exact double-layer density, and we compute the error at N points very close to the interior of the domain to test the nearly-singular quadrature. The results here show the effect of our improved blending functions as well as speedup from shared-memory parallelization efforts despite the method's staying largely unchanged. In table 5.1, results for the complex geometry in figure 5.1(b) are shown. Timings for other shapes are similar, and we show the error results for figures 5.1(a)-(c) are shown in figure 5.2.

h_s	N	T_{bdset}	T_{bisset}	T_{ns}	T_s	E_{2s}	E_{2ns}	$Rate_s$	$Rate_{ns}$
0.192	968	1.42e-01	2.48e-02	1.54e-01	1.14e-01	3.70e-03	8.11e-03		
0.096	2896	1.70e-01	4.66e-02	5.40e-01	2.70e-01	2.09e-04	1.99e-04	4.16	5.35
0.048	11132	6.72e-01	1.91e-01	3.54e+00	1.57e+00	7.70e-06	2.72e-06	4.76	6.19
0.024	43632	1.92e+00	1.14e+00	2.71e+01	1.20e+01	4.90e-07	2.52e-07	3.97	3.43
0.012	174780	6.06e+00	4.51e+00	2.05e+02	8.48e+01	6.10e-08	5.67e-08	3.01	2.16
0.006	694668	2.42e+01	1.91e+01	2.19e+03	7.89e+02	8.82e-09	1.67e-08	2.79	1.76

Table 5.1: Boundary Integral Solver test case for Laplacian with shape 5.1(b). These results show an increase in accuracy, rate of convergence, and speedup from (Ying et al., 2006) through minor improvements and enhancements to the existing method.

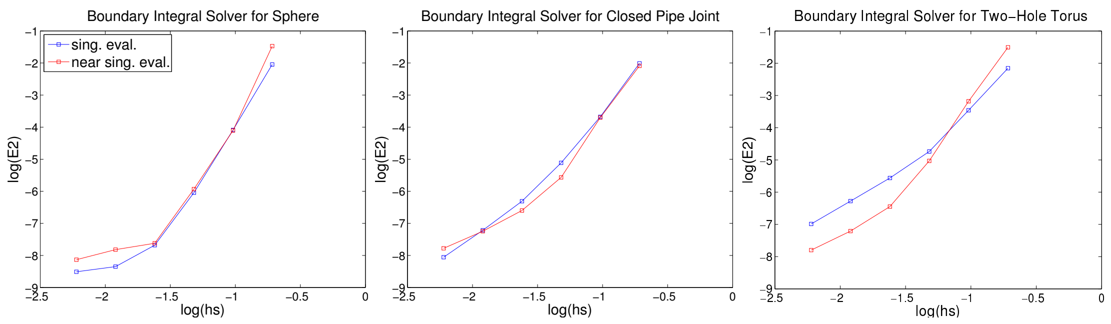


Figure 5.2: For shapes 5.1(a), (b), and (c), we plot the log of the relative error in the boundary integral solver test case versus the log of the surface discretization, h_s for the Laplace equation solver. The number of surface points for equivalent h_s values for each shape are similar. As the shape becomes more complicated, the convergence rate decreases.

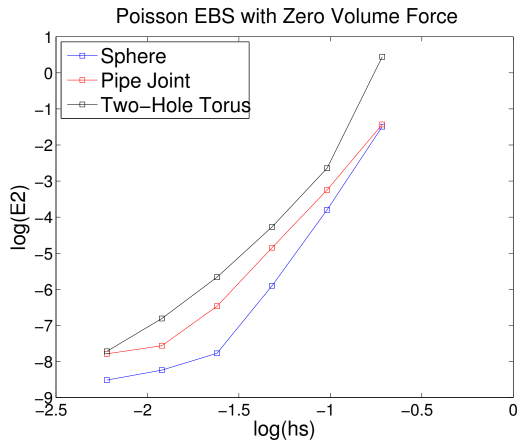
EBI Solver with Zero Volume Force Test Verification We now test the EBI solver for the case in which the volume force is zero. This uses the same test case as the last set of experiments, but we now solve everywhere on a regular grid inside of Ω with discretization h_{vol}. We time the different components for varying levels of fineness for the volume grid, looking at all points on an M_{vol}^3 grid for $M_{vol} = 64, 128, 256$ and 512 in $[-2, 2]^3$ which are inside of the shape. In table 5.2 we look at the timings and results for figure 5.1(b) with $M_{vol} = 512$.

h_s	N_s	t_{bis}	n_{its}	t_{bisit}	t_{far}	t_{near}	t_{tot}	E_∞	E_2
0.192	968	2.13e+00	11	1.93e-01	0.00e+00	3.54e+02	3.56e+02	4.84e+00	3.71e-02
0.096	2896	5.93e+00	10	5.92e-01	7.58e-01	2.80e+02	2.87e+02	5.56e-03	5.66e-04
0.048	11132	3.54e+01	8	4.42e+00	1.11e+01	2.70e+02	3.18e+02	4.30e-04	1.42e-05
0.024	43632	2.01e+02	7	2.87e+01	2.76e+01	3.71e+02	6.03e+02	1.14e-05	3.40e-07
0.012	174780	1.61e+03	6	2.68e+02	8.68e+01	3.69e+02	2.07e+03	4.13e-07	2.74e-08
0.006	694668	1.93e+04	5	3.86e+03	1.97e+02	1.13e+03	2.07e+04	3.45e-07	1.63e-08

Table 5.2: Results for full solver in the absence of an underlying volume force for the interior Dirichlet Poisson equation and figure 5.1(b). The total number of target points is 2231752, consisting of all points lying on a regular grid of discretization size $h_{vol} = 0.0078125$ inside of $[-2, 2]^3$. For larger h_{vol}, error values are nearly equivalent while timings can vary greatly as seen in figure 5.3(b).

For other shapes, such as figures 5.1(a) and (c), the number of surface points, N_s are on the same order for equivalent h_s; however, depending on the complexity of the shape, the error results can vary. In figure5.3(a), we plot the relative errors for figures 5.1 (a)-(c).

Additionally, we note that for smaller M_{vol}, the error results are nearly identical and are omitted here; however, as there is a larger density of points, the far-field and near-field evaluations for the boundary integral solver become, the timings can be significantly different. For example, in figure 5.3, we look at the total time of the solver for different numbers of target points, corresponding to finer h_{vol}. For coarse surface discretizations, there is a significant increase in the total time of the full solver as we increase the number of target points; however, as h_s becomes smaller, the total solve times approach a constant value for finer h_{vol}. This is to be expected, as the boundary integral solver begins to dominate the total computation as $h_s \to 0$, and the volume solver evaluation time becomes relatively less expensive computationally.



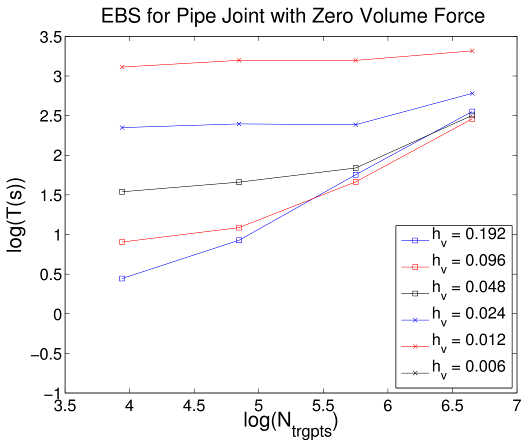


Figure 5.3: For Poisson equation solver, *Left*: Log-log plot of relative error, E_2 versus surface discretization, h_s for the three shapes in figure 5.1(a)-(c) for full embedded boundary solver in the absence of a volume force and for target discretization $h_{vol} = 0.0078125$. For the sphere shape (figure 5.1(a)), the number of target points, $N_{pts} = 4498024$; for the closed pipe joint shape (figure 5.1(b)), $N_pts = 2231752$; for the two-hole torus shape (figure 5.1(c)), $N_pts = 1298880$. *Right*: For figure 5.1(b), we investigate the full solver as we increase the number of volume and surface points. For small h_s, the boundary solver dominates the computation time.

EBI Solver with Volume Force and Extension Poisson Example 1 Having tested the boundary integral solver and its coupling with the embedded boundary solver in the absence of a background volume force, we turn to the case when the underlying volume force is non-zero. We begin by looking at purely uniform-refinement cases. That is, for a chosen level, ℓ_{rhs}, we refine until all leaves are at depth $d = \ell_{rhs}$. Further, we also initially assume that we know the volume force everywhere within our enclosing domain Ω in figure 4.1 such that we can see how well the extension performs in comparison.

In order to test how well the extension performs, we also want to guarantee that we choose a force which changes quickly at the boundary, so for figure 5.1(a) and a sphere of radius $R = 0.8$, let

$$-\Delta u(\mathbf{x}) = \frac{6s^3(||\mathbf{x}||^2 - R^2)^2 - 8s^3(||\mathbf{x}||^2) * (||\mathbf{x}||^2 - R^2) + 6s}{\pi s^4(||\mathbf{x}||^2 - R^2)^4 + 2\pi s^2(||\mathbf{x}||^2 - R^2)^2 + \pi} \quad \text{with solution}$$

$$u(\mathbf{x}) = \frac{1}{\pi}\left(atan(s(R^2 - ||\mathbf{x}||^2)) + \frac{\pi}{2}\right). \tag{5.1}$$

An example slice, showing the high-gradient nature of this force near the surface of the boundary can be seen in figure 5.4.

We begin by looking at results for uniform refinement without extension of the body force beyond the boundary (assuming we know the exact force as input) versus with extension. As the shape is of radius $R = 0.8$, while we embed the sphere in a box of size $[-2, 2]^3$, we restrict the domain of the volume solver's computation to $[-1, 1]^3$. As the right-hand side moves quickly towards zero, this does not affect the solver without extension. Further, for use with the extension, we do not select any boxes outside of this domain, so the effect of the restriction is irrelevant. In table 5.3 we look at the number of leaves used in the volume solver for no extension versus extension cases. As discussed above, when the extension is present, along with interior and overlapping leaves, only exterior leaves one step away from the boundary are included as sources.

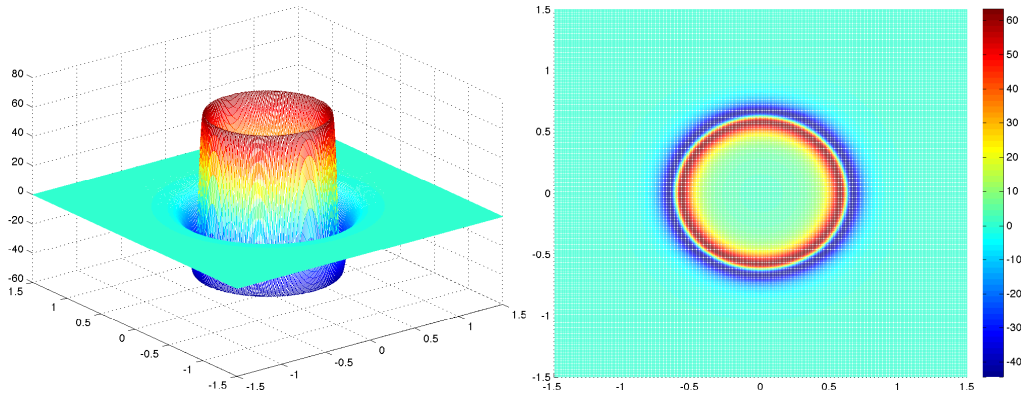


Figure 5.4: For sphere shape 5.1(a) of radius $R = 0.8$ and high gradient force in equation 5.1 with $s = 10$, we look at a particular slice for $z = 0$. The force is spherically symmetric about the origin.

In figures 5.5and 5.6, we compare the plots for uniform refinement without the extension versus using the extension. The complete tables for these results are compiled in the Appendix section A.2 in tables A.1- A.4, for polynomial orders $k = 3$ to $k = 6$. As can be seen in the plots below, our extension performs well in comparison to knowing the force everywhere as we increase the depth of the uniform refinement as well as the scale for the high-gradient test function. As the scaling variable, s increases, we see that the rate of convergence slows quickly for the solver; however, the extension tests continue to perform as well as the non-extension cases.

			Leaf Count for No Extension				Leaf Count for Extension			
T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	L_{in}	L_{crv}	Lvs_{out}	L_{tot}
2	0.096	2400	0	56	8	64	0	56	8	64
3	0.048	9600	56	224	232	512	56	224	128	408
4	0.024	38400	696	776	2264	4096	696	776	512	1984
5	0.012	149784	7280	3056	22432	32768	7280	3056	1872	12208
6	0.012	149784	64288	12368	185488	262144	64288	12368	6728	83384

Table 5.3: For the full solver and uniform refinement using figure 5.1(a) with radius $R = 0.8$, we solve the Poisson equation with high-gradient right-hand side. When no extension is used, we assume all leaves in $[-1, 1]^3$ are used; whereas, when we introduce our extension, only exterior leaves with a neighbor leaf overlapping the boundary are used in the computation. The corresponding leaf counts are shown for varying tree depths, T_d, surface discretization, h_s and surface points, N_s. L_{in}, L_{crv}, and L_{out} refer to the total number of interior, overlapping, and exterior leaves used in the volume solver computation, respectively.

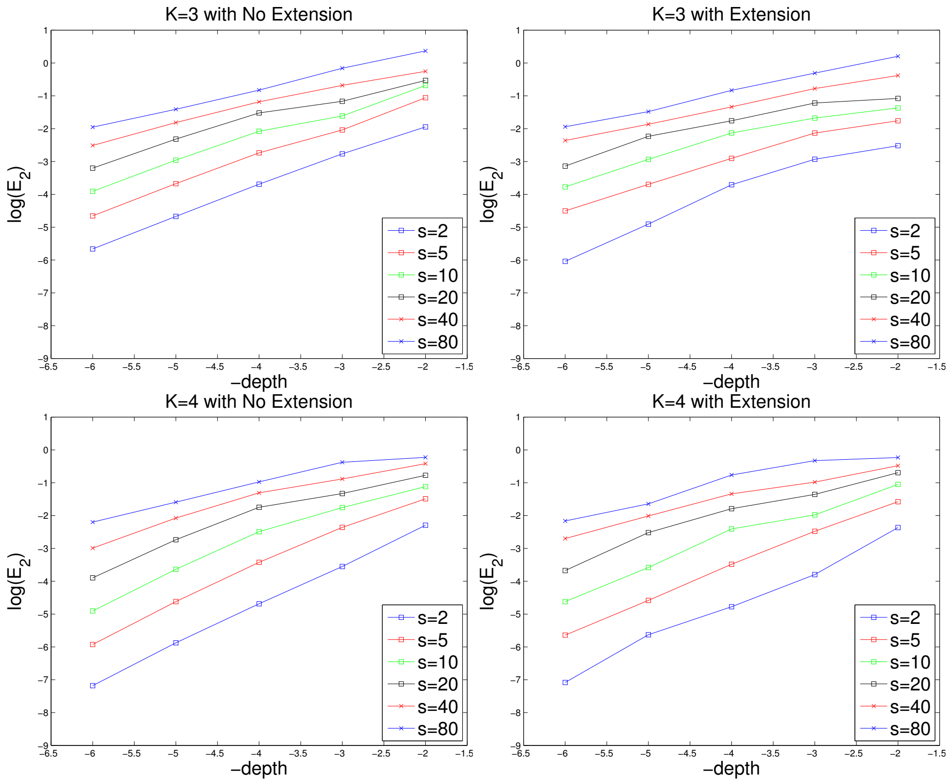


Figure 5.5: Uniform refinement E_2 versus $depth$ plots for (*left*) no extension (force known everywhere) and (*right*) extension to overlapping and exterior leaves for polynomial approximation of orders $k = 3, 4$ (*top* and *bottom*), results from tables A.1 and A.2, respectively.

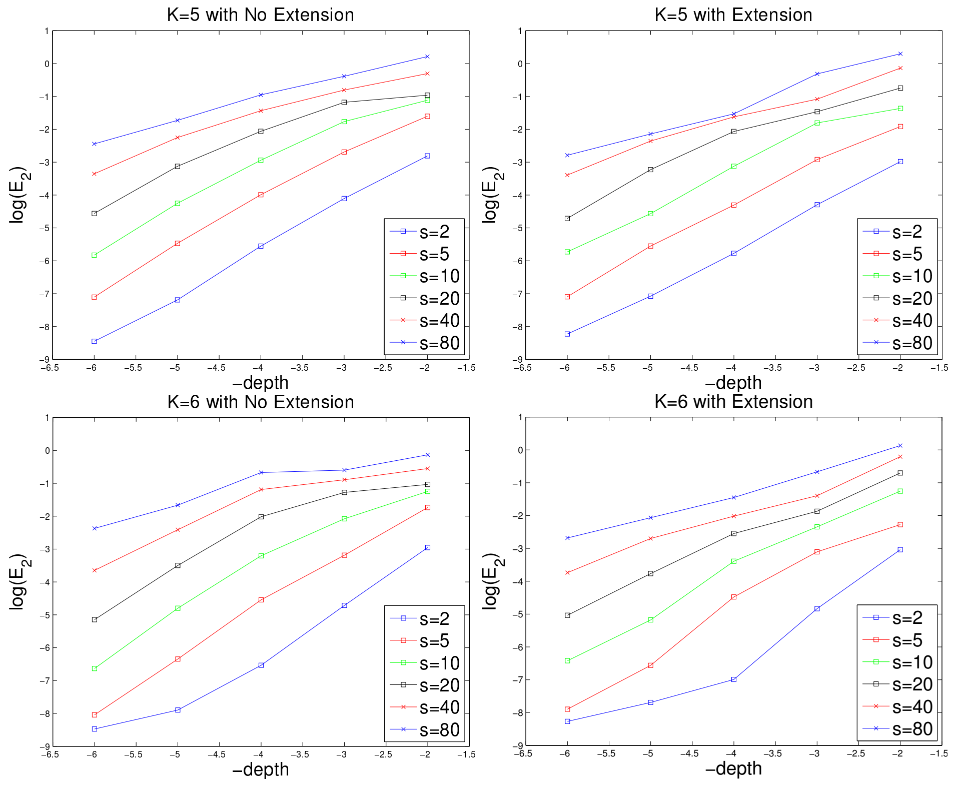


Figure 5.6: Uniform refinement E_2 versus $depth$ plots for (*left*) no extension (force known everywhere) and (*right*) extension to overlapping and exterior leaves for polynomial approximation of orders $k = 5, 6$ (*top* and *bottom*), results from tables A.3 and A.4, respectively.

We now turn to looking at what happens when we extend our solver with adaptive-refinement. We use an adaptive-refinement strategy similar to (Ethridge and Greengard, 2001). For this, we compute a k^{th}-order polynomial approximation, γ^B, to the force $g^B(\mathbf{x})$ sampled on a $k \times k \times k$ grid. We let $\tilde{g}^B$ be the force evaluated on a refined $2k \times 2k \times 2k$ grid. If $\left|\left|g^B(\mathbf{x}) - \tilde{g}^B(\mathbf{x})\right|\right|_2 > \epsilon_{rhs}$, B is subdivided, and the octree is balanced as needed. For overlapping leaves, we only utilize the subsets of full grids, where the subset lies in the interior of the domain. As the L_2 evaluation comparison is effectively scaled by the volume of the corresponding leaf box, we are guaranteed that refinement will terminate, even for leaves with few interior gridpoints. When the subset of points is empty, we compare the volume of the leaf box to ϵ_{rhs}. For example, in figure 5.7, we see how the finest refinement takes place near the boundary.

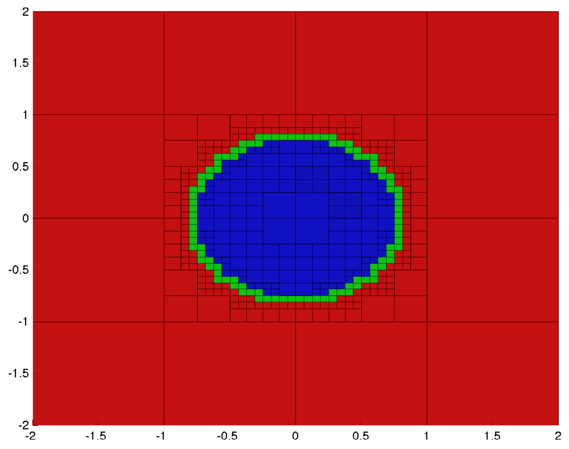


Figure 5.7: Example slice of the octree for high-gradient force in equation 5.1 at $z = 0$ and $s = 10$ for $\epsilon_{rhs} = 4$, $k = 6$. Leaves in the octree marked as interior are indicated in blue; exterior leaves are indicated in red; and overlapping leaves are indicated in green.

For polynomial orders $k = 3$ to $k = 6$, we look at how our solver performs as we decrease ϵ_{rhs}. From the uniform refinement tests, we can see that for $s > 10$, the solver does not perform as well for $k = 3$, so we only look at adaptive refinement for $k \geq 4$ for $s > 10$. Similarly, we only look at results for $k = 4$ up to $s = 20$ and for $k = 5$ up to $s = 40$. We compile the results for all of these tests in the Appendix section A.2 in tables A.5- A.10, and we plot the results for $k = 3$ and $k = 4$ in figure 5.8 and for $k = 5$ and $k = 6$ in figure 5.9.

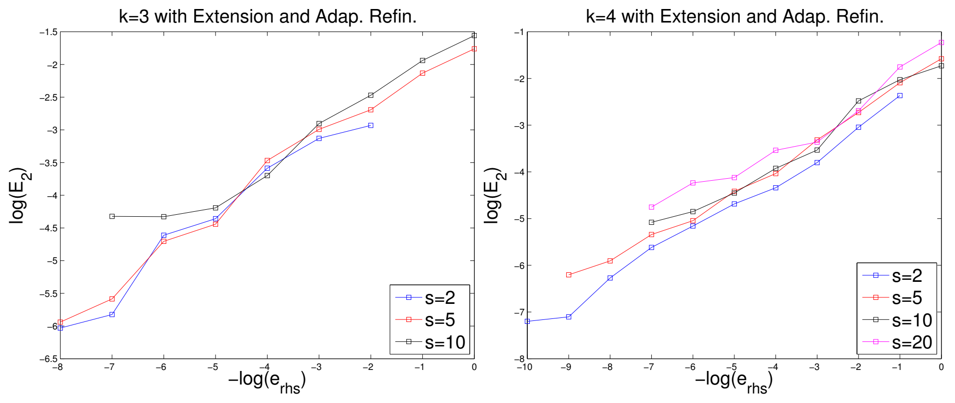


Figure 5.8: Log-log plot for adaptive refinement and E_2 versus ϵ_{rhs} plots using our extension to overlapping and exterior leaves for polynomial approximation of order $k = 3$ (*Left*) and $k = 4$ (*Right*), from results tables A.5 and A.6, respectively.

For the adaptive solver, we can clearly solve to greater accuracy due to increased memory savings as we need fewer leaves to resolve the interior leaves. In order to compare how the adaptive solver performs to the uniform refinement solver, we could look at each solvers results for specific depths; however, for the adaptive solver, multiple values of ϵ_{rhs} can result in a single octree maximum depth. We instead look at how the resulting tree, T, approximates the full volume force through the following metric; let E_2^T be the sum of the error in the volume force among all boxes, B. First, we evaluate

$$e^B = \left(\int_B |f_{(2k)}(\mathbf{x}) - \tilde{f}_{(2k)}(\mathbf{x})|^2 dx \right)^{1/2},$$

where $\tilde{f}_{(2k)}(\mathbf{x})$ is the evaluation of the polynomial approximation,γ, to $f(\mathbf{x})$, restricted to each

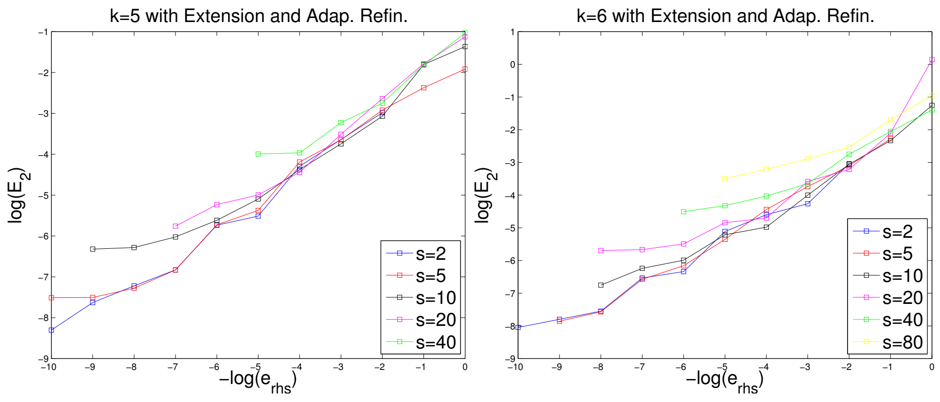


Figure 5.9: Log-log plot for adaptive refinement and E_2 versus ϵ_{rhs} plots using our extension to overlapping and exterior leaves for polynomial approximation of order $k = 5$ (*Left*) and $k = 6$ (*Right*), from results tables A.8 and A.10, respectively.

leaf box B, on a refined grid of size $(2k)^3$ for some polynomial order, k. We further note that as described above, γ^B, on each leaf box B is build from a uniform grid of size k^3 (or subset thereof for leaves overlapping the boundary as discussed above). Now, to evaluate E_2^T, we sum e^B among all boxes and then compute f_{comp} by normalizing by the $L2$ norm of the volume force over the entire tree:

$$E_2^T = \sum_{B \in T} e^B \quad \text{, and}$$

$$f_{comp} = E_2^T / \, ||f||_2^T \, .$$

In figures 5.10-5.16, we plot the results for uniform refinement against adaptive refinement by comparing the relative error results relative to the f_{comp} metric. Again, for $k = 3$ in figure 5.10, we only look at $s = 2, 5, 10$; in figure 5.11 and 5.12, we only look at $s = 2, 5, 10, 20$ for $k = 4$; in figures 5.13 and 5.14, we look at $s = 2, 5, 10, 20, 40$ for $k = 5$; finally, for $k = 6$, we look at all computed values of s in figures 5.15 and 5.16.

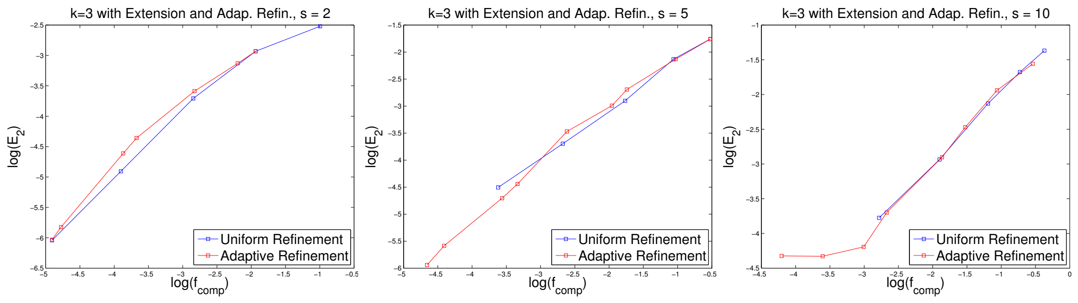


Figure 5.10: Log-log plots comparing the results for uniform refinement against adaptive refinement by comparing relative error results for both solvers versus the error metric, f_{comp}, computed at all interior and overlapping leaf boxes, B in our octree T. For $k = 3$: *Left*: $s = 2$; *Middle*: $s = 5$; *Right* $s = 10$.

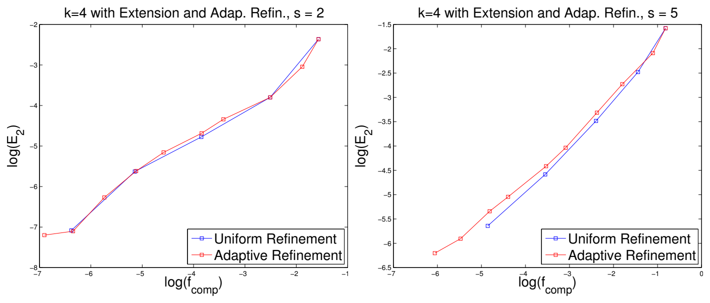


Figure 5.11: Log-log plots comparing the results for uniform refinement against adaptive refinement by comparing relative error results for both solvers versus the error metric, f_{comp}. For $k = 4$: *Left*: $s = 2$; *Right*: $s = 5$.

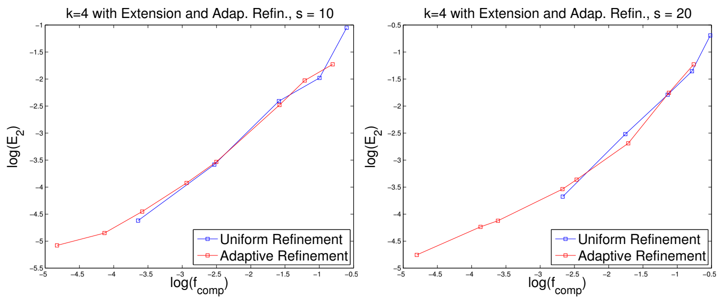


Figure 5.12: Log-log plots comparing the results for uniform refinement against adaptive refinement by comparing relative error results for both solvers versus the error metric, f_{comp}. For $k = 4$: *Left*: $s = 10$; *Right*: $s = 20$.

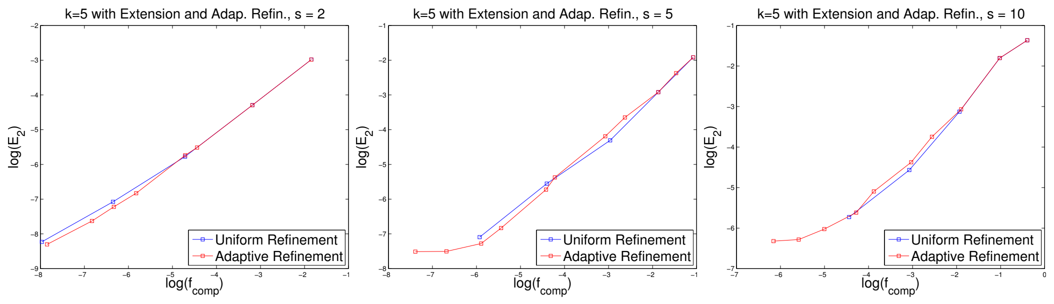


Figure 5.13: Log-log plots comparing the results for uniform refinement against adaptive refinement by comparing relative error results for both solvers versus the error metric, f_{comp}. For $k = 5$: *Left*: $s = 2$; *Middle*: $s = 5$; *Right*: $s = 10$.

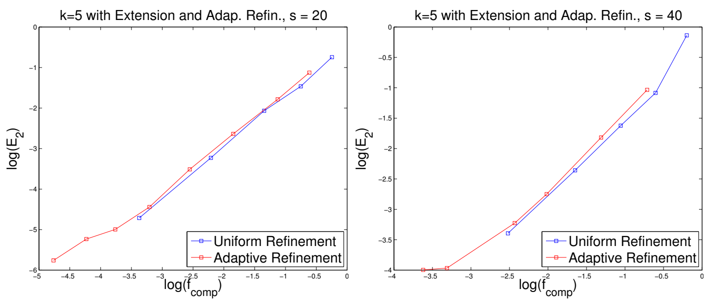


Figure 5.14: Log-log plots comparing the results for uniform refinement against adaptive refinement by comparing relative error results for both solvers versus the error metric, f_{comp}. For $k = 5$: *Left*: $s = 20$; *Right*: $s = 40$.

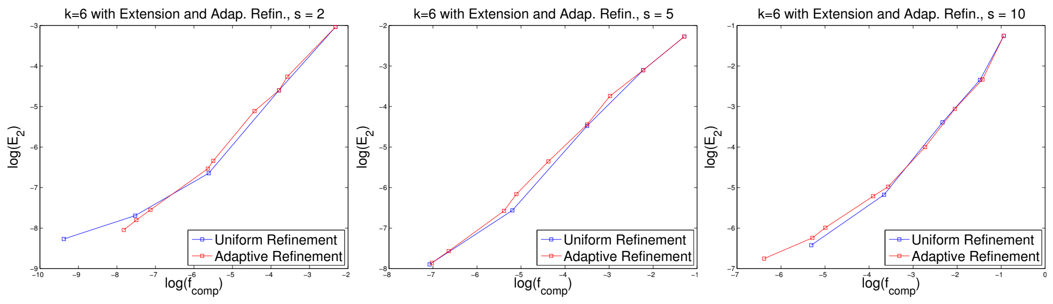


Figure 5.15: Log-log plots comparing the results for uniform refinement against adaptive refinement by comparing relative error results for both solvers versus the error metric, f_{comp} For $k = 6$: *Left*: $s = 2$; *Middle*: $s = 5$; *Right*: $s = 10$.

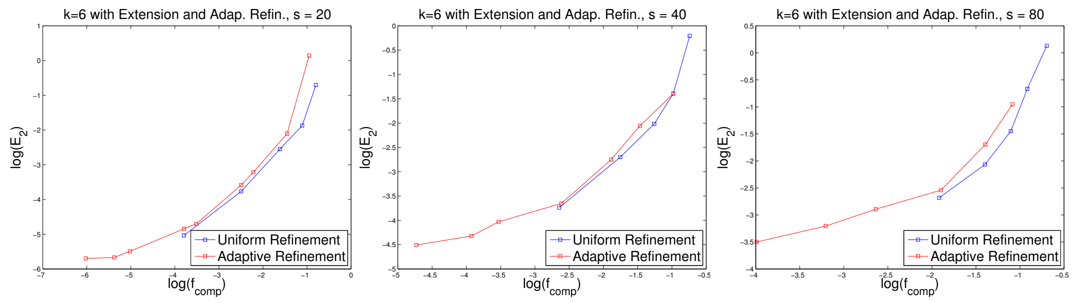


Figure 5.16: Log-log plots comparing the results for uniform refinement against adaptive refinement by comparing relative error results for both solvers versus the error metric, f_{comp} For $k = 6$: *Left*: $s = 20$; *Middle*: $s = 40$; *Right*: $s = 80$.

As can be seen, our adaptive solver returns results that are of an equivalent nature to the uniform solver when we compare how well both types of solvers approximate the volume force. Additionally, as can be seen, we can often return equivalent results using significantly fewer leaf boxes with adaptivity, especially as we increase the gradient near the boundary, and the force becomes smoother in the interior of the sphere. Further, as a result of needing fewer boxes to resolve the force, we can achieve greater accuracy and trees of greater depth. In particular, for $s > 2$ for all orders k of the polynomial approximation, we can exceed the accuracy of the uniform solver.

EBI Solver with Volume Force and Extension Poisson Example 2 For this experiment, we replicate an experiment from (Ying et al., 2006) in order to show the increased accuracy and speed of the current implementation. Additionally, the current work assumes that the force is only given in the interior of the domain, so we are performing the extension here. For $\mathbf{x} = (x_1, x_2, x_3)$

$$-\Delta u(\mathbf{x}) = (x_1 + x_2 + x_3),$$

$$u(\mathbf{x}) = exp(\sqrt{2}\pi x_1)\sin\pi(x_2 + x_3) + \left(x_1^3 + x_2^3 + x_3^3\right)/6. \qquad (5.2)$$

As can be seen, the force, $f(\mathbf{x}) = -(x_1 + x_2 + x_3)$ is very simple, and our polynomial approximation will evaluate it exactly. In such cases, the adaptive refinement strategy used for building the needs to take this into account. Namely, when $\left|\left|g^B(\mathbf{x}) - \tilde{g}^B(\mathbf{x})\right|\right|_2 = 0$, but $\left|\left|g^B(\mathbf{x})\right|\right|_\infty > 0$, we refine B when $\int_B d\mathbf{y} > \epsilon_{rhs}$. For this test, we fix the number of target points at a fine grid inside of the shape in figure 5.1(b), enclosed within $[-0.8, 0.8]^3$. Additionally, we fix the polynomial approximation to $k = 6$ for all tests. Refinement occurs solely inside of the domain with balancing across the domain as necessary to include all external boxes needed for the volume solver. We compile the results in table 5.4 and figure 5.18(a).

h_s	ϵ_{rhs}	t_{vfmm}	t_{bis}	n_{its}	t_{far}	t_{near}	t_{tot}	E_∞	E_2
0.192	10^{-2}	1.72e+00	2.48e+00	13	0.00e+00	4.01e+02	4.06e+02	2.15e+00	8.22e-02
0.096	10^{-4}	2.22e+00	9.36e+00	13	8.01e-01	3.21e+02	3.33e+02	1.27e-02	3.58e-03
0.048	10^{-6}	2.86e+00	4.63e+01	12	1.12e+01	2.36e+02	2.98e+02	7.60e-04	6.00e-05
0.024	10^{-8}	9.46e+00	3.56e+02	12	2.85e+01	2.71e+02	6.69e+02	1.80e-05	1.98e-06
0.012	10^{-10}	8.61e+01	3.01e+03	12	8.66e+01	3.50e+02	3.54e+03	8.86e-07	1.053-07
0.006	10^{-10}	8.13e+01	2.15e+04	12	2.42e+02	1.02e+03	2.29e+04	1.80e-07	2.65e-08

Table 5.4: EBI Poisson closed pipe joint. Non-zero volume force. Here, we set the volume target discretization to $h_{vol} = 0.0078125$ with target point size $N_{vol} = 2231752$ for all tests from example problem in equation 5.2. We fix the polynomial approximation at $k = 6$. For a specific h_s, the number of surface points is equivalent to those given in Table 5.1. For ϵ_{rhs}, the depth of the tree T_d and the number of corresponding volume source points, N_{src} used in computing the extension for the free-space volume solver is given in Table 5.5.

ϵ_{rhs}	T_d	N_{src}
10^{-2}	1	232
10^{-4}	2	1784
10^{-6}	4	117600
10^{-8}	5	941640
10^{-10}	6	7530176

Table 5.5: For Table 5.4, for Poisson Test Example 2, for a given ϵ_{rhs}, the resulting octree is of depth T_d. We fix the polynomial approximation to order $k = 6$, so each leaf has either a full grid of size 6^3 or a fraction thereof (for leaves overlapping the boundary). The number of volume source points from these full or partial grids is given by N_{src}. This is the number of points available for building the polynomial approximation for interior leaves or building the extension for overlapping or exterior leaves.

EBI Solver with Volume Force and Extension Poisson Example 3 We introduce an example in which the force is non-trivial such that the polynomial approximation does not exactly recover the right hand side. Using the shape in figure 5.1(c), we solve

$$\begin{aligned} -\Delta u(\mathbf{x}) &= \sum_{i=1}^{7} 3\pi^2 \sin(\pi(x-x_i))\sin(\pi(y-y_i))\sin(\pi(z-z_i)) \\ &\quad -exp(q\,||\mathbf{x}-\mathbf{x}_i||^2)(4q\,||\mathbf{x}||^2+6q) \quad \text{with solution} \end{aligned}$$

$$u(\mathbf{x}) = \sum_{i=1}^{7} \sin(\pi(x-x_i))\sin(\pi(y-y_i))\sin(\pi(z-z_i)) + exp(q\,||\mathbf{x}-\mathbf{x}_i||^2), \tag{5.3}$$

where $\mathbf{x}_i = (0,0,c)$ for $c = 0, \pm 0.2, \pm 0.6, \pm 1.0$ and $q = -10$. We choose theses coordinates as they represent the edges of each hole in the shape (and the outside edges and center), guaranteeing the boundary is an area of interest. Again, we fix the polynomial approximation to order $k = 6$. An example slice of the adaptively-refined octree is shown in figure 5.17. Numerical results are seen in table 5.8 and figure 5.18(b).

h_s	N_s
0.192	1048
0.096	3434
0.048	12024
0.024	45074
0.012	175816
0.006	693093

Table 5.6: For figure 5.1(c), given surface discretization h_s, N_s is the corresponding number of surface points.

ϵ_{rhs}	T_d	N_{src}
10^{-2}	3	7576
10^{-4}	4	62352
10^{-6}	5	93384
10^{-8}	5	546880
10^{-10}	6	4374712

Table 5.7: For Table 5.8; given ϵ_{rhs}, the resulting octree is of depth T_d. We fix the polynomial approximation to order $k = 6$, and the definition of N_{src} follows the same as in Table 5.5.

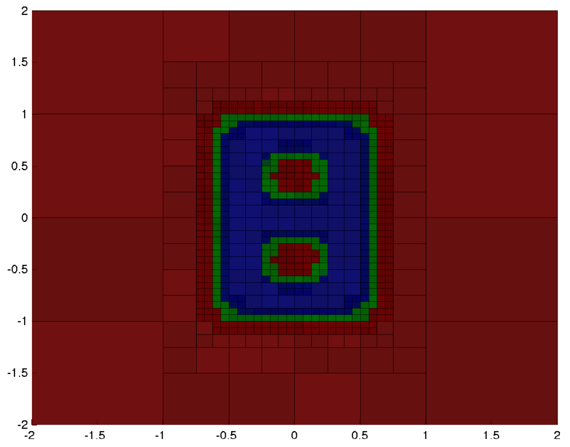


Figure 5.17: Example slice of the octree for Poisson example 3 with nonzero volume force in equation 5.3 for shape 5.1(c). Here, we set $\epsilon_{rhs} = 10^{-6}$ and the polynomial order $k = 6$ for refinement. Leaves in the octree marked as interior are indicated in blue; exterior leaves are indicated in red; and overlapping leaves are indicated in green.

h_s	ϵ_{rhs}	t_{vfmm}	t_{bis}	n_{its}	t_{far}	t_{near}	t_{tot}	E_∞	E_2
0.192	10^{-2}	2.16e+00	3.42e+00	17	0.00e+00	2.89e+02	2.95e+02	6.21e+00	1.23e-01
0.096	10^{-4}	2.34e+00	1.24e+01	16	0.00e+00	2.68e+02	2.83e+02	1.92e-02	4.58e-03
0.048	10^{-6}	2.77e+00	7.34e+01	16	2.360e-01	2.30e+02	3.08e+02	9.68e-04	9.21e-05
0.024	10^{-8}	8.50e+00	3.96e+02	14	3.08e+00	2.47e+02	6.57e+02	3.86e-05	2.02e-06
0.012	10^{-10}	6.11e+01	2.60e+03	14	1.78e+01	3.12e+02	3.00e+03	1.93e-06	1.440e-07
0.006	10^{-10}	6.03e+01	2.77e+04	14	8.16e+01	1.09e+03	2.90e+04	4.92e-07	1.74e-08

Table 5.8: Full EBI solver for Poisson equation example 3 for figure 5.1(c) and non-trivial volume force. Here, we set the volume target discretization to $h_{vol} = 0.0078125$ with target point size $N_{vol} = 1298880$ for all tests from example problem in equation 5.3. We fix the polynomial approximation order to $k = 6$. Table 5.6 gives the number of surface points used in the boundary integral solver for a specific surface discretization, h_s and Table 5.7 gives the corresponding volume source points, N_{src} used in computing the extension for the free-space volume solver.

.

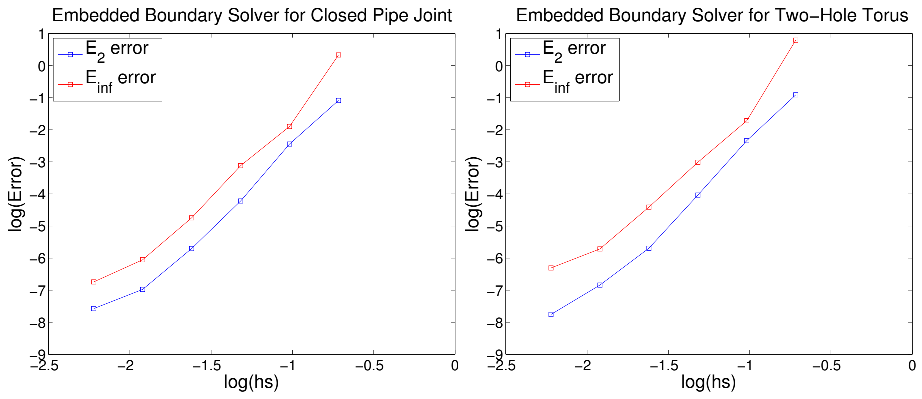


Figure 5.18: *Left*: Log-log plot of relative error versus surface discretization for the EBI solver with non-zero volume force in example 2 and table 5.4 above for the closed pipe joint shape in figure 5.1(b). *Right*: Log-log plot of relative error versus surface discretization for embedded boundary solver with non-zero volume force in example 3 and table 5.8 above for the two-hole torus shape in figure 5.1(c).

5.0.7 Modified Helmholtz Equation

For the Modified Helmholtz equation (equation (2.8) with kernel (2.11)), we investigate the numerical accuracy of our algorithm. For this scalar kernel, we do not investigate the boundary integral solver or full solver with no volume force as the results are nearly identical as for the Poisson equation. To see this, we can simply set $\alpha = 0$ and obtain the Poisson equation. We note that as the kernel is inhomogeneous, matrices are built for all levels of the octree in the FMM for both particle and volume solvers; however, as we compute the matrices once and store them for later use, the additional computation time is negligible, especially if we precompute all interaction matrices and load them at run-time.

We look at two test cases with adaptive refinement and complex geometries, fixing the polynomial approximation to 4^{th} and 6^{th} order.

EBI Solver with Volume Force and Extension Modified Helmholtz Example 1 For this test case, we look at a simple volume force that can be approximated exactly by our polynomials. Specifically,

$$\alpha u(\mathbf{x}) - \Delta u(\mathbf{x}) = \alpha(1 + xyz), \quad u(\mathbf{x}) = 1 + xyz.$$

We choose $\alpha = 1$ and precompute the interaction matrices needed for the volume solver; for all tests and for a given polynomial order k and FMM precision ϵ_{rhs}, we compute the interaction matrices up to precision $\epsilon_{rhs}/10$. For this example as with the second Poisson example, since our polynomial approximates the force exactly, the extension is trivial with our approach.

In order to increase the difficulty, we use the starfish shape in figure 5.1(d). Additionally, we fix the underlying target grid discretization at $h_{vol} = 0.0078125$, for which $N_{trgpts} = 621996$. Results for $k = 4$ and $k = 6$ can be seen in table 5.9.

h_s	ϵ_{rhs}	t_{vfmm}	t_{bis}	n_{its}	t_{far}	t_{near}	t_{tot}	E_∞	E_2
Polynomial approximation of order $k = 4$									
0.048	10^{-4}	1.59e+00	3.27e+01	15	0.00e+00	1.28e+02	1.64e+02	6.22e-03	5.56e-04
0.024	10^{-6}	2.16e+00	2.30e+02	14	1.88e+00	1.56e+02	3.92e+02	1.81e-04	8.51e-06
0.012	10^{-8}	2.44e+00	1.64e+03	14	1.12e+01	1.91e+02	1.85e+03	1.67e-05	4.18e-07
0.006	10^{-10}	6.28e+00	1.78e+04	14	5.92e+01	5.65e+02	1.84e+04	3.62e-06	7.52e-08
Polynomial approximation of order $k = 6$									
0.048	10^{-4}	1.93e+00	3.16e+01	15	0.00e+00	1.27e+02	1.62e+02	5.20e-03	4.71e-04
0.024	10^{-6}	3.28e+00	2.71e+02	14	1.94e+00	1.64e+02	4.42e+02	1.81e-04	8.50e-06
0.012	10^{-8}	6.38e+00	1.90e+03	14	1.13e+01	1.88e+02	2.14e+03	1.67e-05	4.15e-07
0.006	10^{-10}	1.650e+01	1.70e+04	14	6.02e+01	5.50e+02	1.77e+04	3.62e-06	7.49e-08

Table 5.9: EBI solver for the Modified Helmholtz equation for figure 5.1(d) and non-zero volume force. Non-zero Here, we set the volume target discretization to $h_{vol} = 0.0078125$ with target point size $N_{vol} = 621996$ and vary the polynomial approximation order for $k = 4, 6$. Table 5.10 gives the number of surface points used in the boundary integral solver for a specific surface discretization, h_s and Table 5.11 gives the corresponding volume source points, N_{src} used in computing the extension for the free-space volume solver.

h_s	N_s
0.048	7088
0.024	29062
0.012	123466
0.006	452792

Table 5.10: For figure 5.1(d), given surface discretization h_s, N_s is the corresponding number of surface points.

ϵ_{rhs}	T_d	N_{src}	T_d	N_{src}
	$k = 4$		$k = 6$	
10^{-4}	3	1184	2	528
10^{-6}	4	9564	4	32756
10^{-8}	5	77748	5	262232
10^{-10}	6	621996	6	2099108

Table 5.11: For Table 5.9, for Helmholtz Test Example 1, for a given ϵ_{rhs}, the resulting octree is of depth T_d. We fix the polynomial approximation to order $k = 6$, and the definition of N_{src} follows the same as in Table 5.11.

EBI Solver with Volume Force and Extension Modified Helmholtz Example 2 For our second example with the Modified Helmholtz kernel, we look at a more complicated force. Again, this is similar to the second Poisson example in terms of the solution; whereas the right-hand side is more complex due to the introduction of the additional term. Namely, we solve

$$\alpha u(\mathbf{x}) - \Delta u(\mathbf{x}) = \alpha \left(exp(\sqrt{2}\pi x) \sin \pi(y+z) + \left(x^3+y^3+z^3\right)/6 \right) + (x+y+z),$$
$$u(\mathbf{x}) = exp(\sqrt{2}\pi x) \sin \pi(y+z) + \left(x^3+y^3+z^3\right)/6. \tag{5.4}$$

We choose $\alpha = \frac{1}{4}$ and solve this problem inside of the shape in figure 5.1(b); we note that the nature of this force leads to a highly nonuniform octree, especially along the boundaries. An example slice of the octree can be seen in figure 5.19.

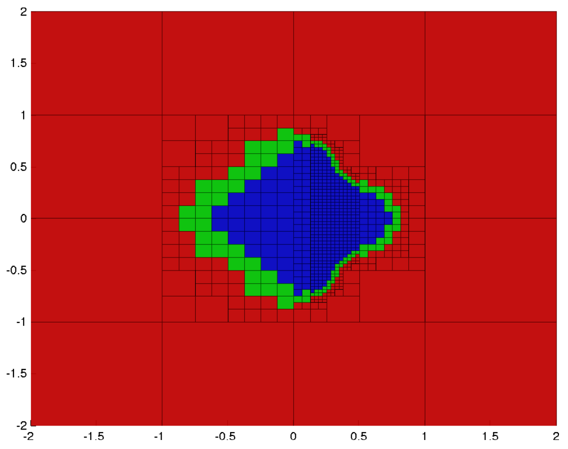


Figure 5.19: Example slice of the octree for the nonuniform force in equation 5.4 at $z = 0$ for $\alpha = 1/4$, $k = 4$ and $\epsilon_{rhs} = 10^{-8}$. Leaves in the octree marked as interior are indicated in blue; exterior leaves are indicated in red; and overlapping leaves are indicated in green.

We again fix the target volume discretization at the fine level of $h_{vol} = 0.0078125$ for $N_{trgpts} = 2231752$ for this shape.

h_s	ϵ_{rhs}	t_{vfmm}	t_{bis}	n_{its}	t_{far}	t_{near}	t_{tot}	E_∞	E_2
Polynomial approximation of order $k = 4$									
0.192	10^{-2}	2.34e+00	2.73e+00	13	0.00e+00	4.18e+02	4.23e+02	1.56e+00	7.83e-02
0.096	10^{-4}	3.14e+00	1.20e+01	13	1.65e+00	3.18e+02	3.35e+02	1.27e-02	3.58e-03
0.048	10^{-6}	4.86e+00	6.13e+01	13	1.62e+01	2.93e+02	3.77e+02	7.57e-04	5.99e-05
0.024	10^{-8}	9.77e+00	3.92e+02	13	4.69e+01	3.33e+02	7.86e+02	1.78e-05	1.99e-06
0.012	10^{-10}	1.13e+01	3.26e+03	13	1.55e+02	4.31e+02	3.88e+03	8.74e-07	1.06e-07
0.006	10^{-10}	1.56e+01	2.79e+04	13	4.00e+02	1.35e+03	2.97e+04	1.82e-07	3.02e-08
Polynomial approximation of order $k = 6$									
0.192	10^{-2}	2.45e+00	2.43e+00	13	0.00e+00	3.99e+02	4.58e+02	1.58e+00	7.88e-02
0.096	10^{-4}	2.48e+00	8.67e+00	13	1.41e+00	3.08e+02	3.25e+02	1.26e-02	3.55e-03
0.048	10^{-6}	3.71e+00	5.36e+01	13	1.59e+01	2.69e+02	3.43e+02	7.58e-04	5.99e-05
0.024	10^{-8}	5.02e+00	3.94e+02	13	4.65e+01	3.30e+02	7.77e+02	1.78e-05	1.99e-06
0.012	10^{-10}	4.99e+01	2.99e+03	13	1.54e+02	4.12e+02	3.62e+03	8.99e-07	1.07e-07
0.006	10^{-10}	5.30e+01	2.90e+04	13	4.06e+02	1.34e+03	3.08e+04	1.84e-07	3.23e-08

Table 5.12: EBI solver for the Modified Helmholtz equation for figure 5.1(b) and non-zero volume force. Non-zero Here, we set the volume target discretization to $h_{vol} = 0.0078125$ with target point size $N_{vol} = 2231752$ and vary the polynomial approximation order for $k = 4, 6$.

h_s	N_s
0.0192	968
0.096	2896
0.048	11132
0.024	43632
0.012	174780
0.006	694668

Table 5.13: For figure 5.1(b), given surface discretization h_s, N_s is the corresponding number of surface points.

ϵ_{rhs}	T_d	N_{src}	T_d	N_{src}
	$k = 4$		$k = 6$	
10^{-2}	3	508	2	1008
10^{-4}	3	732	3	2464
10^{-6}	5	29100	4	14016
10^{-8}	5	232448	4	97976
10^{-10}	6	1729068	6	2377264

Table 5.14: For Table 5.12, for Helmholtz Test Example 1, for a given ϵ_{rhs}, the resulting octree is of depth T_d. We fix the polynomial approximation to order $k = 6$, and the definition of N_{src} follows the same as in Table 5.14.

5.0.8 Stokes Equations

For the Stokes equations (equation (2.9) with kernel (2.12)), we investigate the accuracy and timings of our algorithm. We begin first by verifying the accuracy of the boundary integral solver as well as the EBI solver in the absence of a volume force. We then proceed to look at the effects of our extension on a high-gradient force as with the Poisson equation; however, we only look at the adaptive refinement case as the extension algorithm operates in the same manner as before.

We then look at one additional example, a complex force for a non-trivial geometry. For all examples, we set $\mu = 1$. As the kernel is homogeneous, we can use the same interaction matrices, scaling as necessary if a different μ value is desired.

Boundary Integral Solver As the Stokes equations present us with a matrix kernel, we retest the boundary integral equation using the following special test case, again from (Ying et al., 2006) in order to look at improved results and timings. Specifically,

$$\Delta u(\mathbf{x}) = 0, \quad \mathbf{u}(\mathbf{x}) = \omega \times \mathbf{x} \quad \text{for } \mathbf{x} \in \Omega,$$

where ω is the angular velocity vector of the rotation with $||\omega|| = 1$.

As in the previous work, we compute the error of the singular quadrature algorithm on N boundary points from the exact double-layer density, and we compute the error at N points very close to the interior of the domain to test the nearly-singular quadrature. Again, we show results for a single shape in table 5.15 for figure 5.1(b) in order to highlight increased accuracy and speedup over the previous work; timings for other shapes are similar for equivalent h_s values, and we show error results for this and additional shapes in figure 5.20.

h_s	N	T_{bdset}	T_{bisset}	T_{ns}	T_s	E_{2s}	E_{2ns}
0.192	968	1.765e-01	2.571e-02	2.359e-01	3.052e-01	6.479e-03	4.550e-02
0.096	2896	2.641e-01	5.790e-02	1.025e+00	9.536e-01	4.363e-04	7.646e-04
0.048	11132	6.817e-01	3.204e-01	5.619e+00	6.814e+00	1.498e-05	8.689e-06
0.024	43632	1.884e+00	1.154e+00	4.270e+01	1.190e+02	8.009e-07	3.810e-07
0.012	174780	6.123e+00	4.806e+00	3.307e+02	9.119e+02	9.380e-08	7.263e-08
0.006	694668	2.397e+01	2.691e+01	4.960e+03	5.320e+03	1.400e-08	1.960e-08

Table 5.15: Boundary Integral Solver test case for the Stokes kernel with shape 5.1(b). These results show an increase in accuracy, rate of convergence, and speedup from (Ying et al., 2006) through minor improvements and enhancements to the existing method.

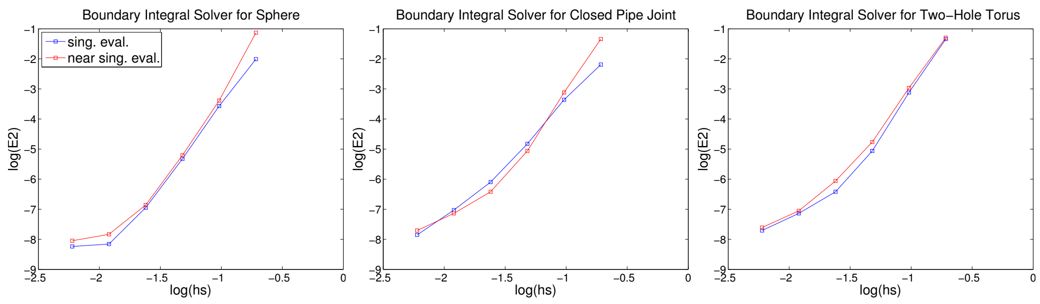


Figure 5.20: For shapes 5.1(a), (b), and (c), we plot the log of the relative error in the boundary integral solver test case versus the log of the surface discretization, h_s for the Stokes equations solver. The number of surface points for equivalent h_s values for each shape are similar. As the shape becomes more complicated, the convergence rate decreases.

EBI Solver with Zero Volume Force We now test the EBI solver for the case in which the volume force is zero. This uses the same test case as the last set of experiments, but we now solve everywhere on a regular grid inside of Ω with discretization h_{vol}. We time the different components for varying levels of fineness for the volume grid, looking at all points on an M_{vol}^3 grid for $M_vol = 64, 128, 256$ and 512 in $[-2, 2]^3$ which are inside of the shape.

h_s	N_s	t_{bis}	n_{its}	t_{bisit}	t_{far}	t_{near}	t_{tot}	E_∞	E_2
$N_{vol} = 2231752$, $h_{vol} = 0.0078125$									
0.192	968	8.38e+00	21	3.99e-01	0.00e+00	7.14e+02	7.15e+02	9.78e+00	6.97e-02
0.096	2896	2.62e+01	18	1.46e+00	5.06e+00	7.00e+02	7.32e+02	5.75e-03	1.08e-03
0.048	11132	1.11e+02	15	7.38e+00	5.41e+01	6.56e+02	8.23e+02	4.44e-04	4.05e-05
0.024	43632	6.51e+02	13	5.01e+01	2.15e+02	1.09e+03	1.96e+03	1.11e-05	6.16e-07
0.012	174780	3.72e+03	11	3.38e+02	1.09e+03	1.90e+03	6.72e+03	3.57e-06	2.20e-07

Table 5.16: EBI Stokes closed pipe joint. Zero volume force.

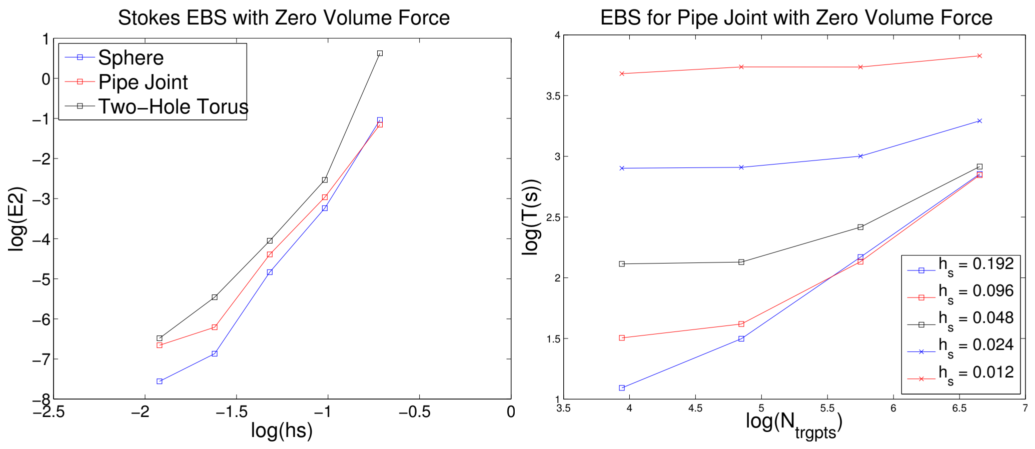


Figure 5.21: For Stokes equations solver, *Left*: Log-log plot of relative error, E_2 versus surface discretization, h_s for the three shapes in figure 5.1(a)-(c) for full embedded boundary solver in the absence of a volume force and for target discretization $h_{vol} = 0.0078125$. For the sphere shape (figure 5.1(a)), the number of target points, $N_{pts} = 4498024$; for the closed pipe joint shape (figure 5.1(b)), $N_pts = 2231752$; for the two-hole torus shape (figure 5.1(c)), $N_pts = 1298880$. ***Right***: For figure 5.1(b), we investigate the full solver as we increase the number of volume and surface points. For small h_s, the boundary solver dominates the computation time.

EBI Solver with Volume Force and Stokes Equations Example 1 For this example, we look at a high-gradient force with adaptive-refinement, varying the polynomial approximation, the degree of the gradient force while attempting to extend the force beyond the boundary as needed. As in the first Poisson example for the high-gradient force, we look at the simple sphere shape in figure 5.1(a) with radius $R = 0.8$:

$$-\mu\Delta\mathbf{u}(\mathbf{x}) + \nabla p(\mathbf{x}) = \left(\frac{10s^3(||\mathbf{x}||^2 - R^2)^2 - 8s^3\,||\mathbf{x}||^2\,(||\mathbf{x}||^2 - R^2) + 10s}{\pi s^4(||\mathbf{x}||^2 - R^2)^4 + 2\pi s^2(||\mathbf{x}||^2 - R^2)^2 + \pi}\right) \cdot (\omega \times \mathbf{x})$$

$$\mathbf{u}(\mathbf{x}) = \frac{1}{\pi}\left(atan(s(R^2 - ||\mathbf{x}||^2)) + \frac{\pi}{2}\right) \cdot (\omega \times \mathbf{x}). \tag{5.5}$$

This right-hand side exhibits interesting behavior on the boundary as well as away from the boundary in the interior of our sphere. For example, looking at the z-component of the force, near the boundary, we observe the high-gradient nature of the force in figure 5.22, for which extension will be necessary. Away from the boundary, we also see adaptive refinement will be necessary in some areas as evident in figure 5.23.

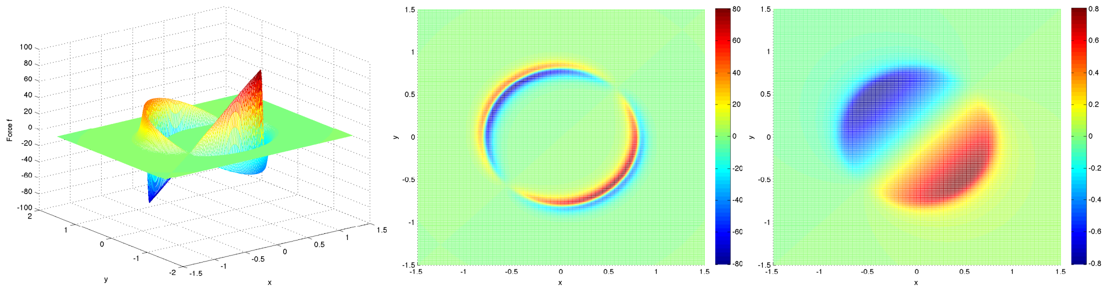


Figure 5.22: For sphere shape 5.1(a) of radius $R = 0.8$ and high gradient force in equation 5.5 with $s = 10$, we look at a particular slice of the force for $z = 0$ for the z-component of the force. The force is symmetric across axes. *Left and middle*: Volume force; *Right*: Solution, u_z.

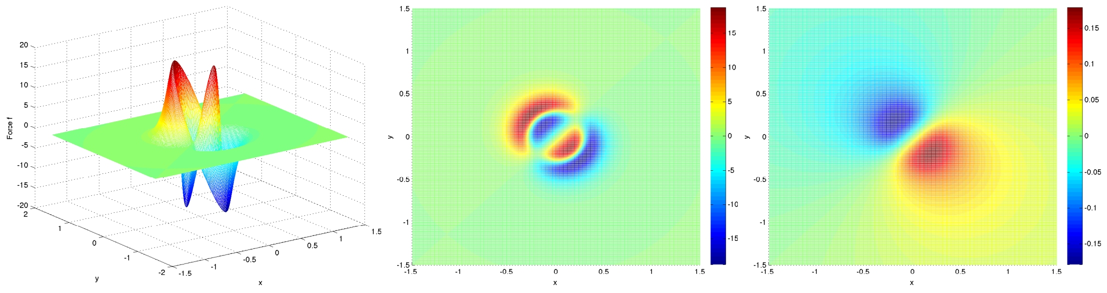


Figure 5.23: For sphere shape 5.1(a) of radius $R = 0.8$ and high gradient force in equation 5.5 with $s = 10$, we look at a particular slice of the force for $z = 0.375$ for the z-component of the force. The force is symmetric across axes. *Left and middle*: Volume force; *Right*: Solution, u_z.

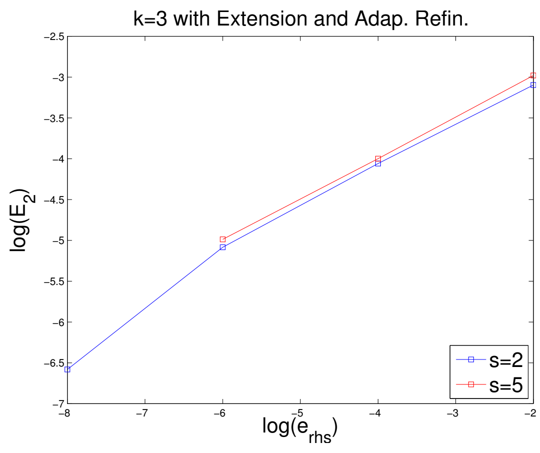
We compute the solution for four polynomial approximation values as before for $k = 3, 4, 5, 6$ using adaptive refinement with our extension. For all tests, we set $\epsilon_{fmm} = \epsilon_{rhs}$. Further, we choose the surface discretization value, h_s by using our observations from figure 5.20(a) and figure 5.21(a). From those tests, we observe the expected relative error when the values on the surface are smooth, so by choosing appropriate h_s, we guarantee the error for full solver will not be dominated by the boundary integral solver. The number of target points in the full solver is 4498024 with the number of target points for the volume solver given by $L_{tot}k^3$. Results are shown in tables 5.17- 5.20 and figures 5.24- 5.25.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 2$										
10^{-2}	3	0.048	9600	42	224	128	394	3270	1.822e-03	7.991e-04
10^{-4}	4	0.024	38400	598	776	464	1838	26818	3.228e-04	8.730e-05
10^{-6}	5	0.012	38400	7126	3056	1872	12054	232826	2.764e-05	8.231e-06
10^{-8}	6	0.012	149784	64288	12368	9728	86384	232826	1.466e-06	2.621e-07
scale $s = 5$										
10^{-2}	4	0.024	38400	266	752	410	1428	17268	1.682e-02	1.046e-03
10^{-4}	5	0.012	149784	2940	3056	1872	7868	119804	4.435e-04	1.001e-04
10^{-6}	6	0.012	149784	32718	12356	6630	51704	1044914	4.577e-05	1.028e-05

Table 5.17: High-gradient Stokes test case for polynomial-approximation of order $k = 3$. For each specific level of precision in the right-hand side, ϵ_{rhs}, we set the FMM precision, $\epsilon_{fmm} = \epsilon_{rhs}$ (except for $\epsilon_{rhs} = 1$, for which $\epsilon_{fmm} = 10^{-2}$) in order to guarantee FMM precision does not dominate the error. Further, we choose the surface discretization, h_s such that the error from the boundary solver is more precise than ϵ_{rhs} as well (using previous results from boundary and embedded boundary solver without volume force to choose proper h_s for sphere shape). For adaptively refined octree from specific ϵ_{rhs}, L_{in} denotes the number of leaves fully inside of the sphere; L_{crv} denotes the number of leaves overlapping leaves; L_{out} denotes number of exterior leaves which are used in the computation of the free-space volume source; L_{tot} is the total number of leaves used in the free-space evaluation. $L_{crv} + L_{out}$ are the number of leaves for which extension may be necessary, whereas all interior leaves use their full k^3 grid for computing k^{th}-order approximation. N_{svol} is the total number of source points used to build the coefficients for the force approximation.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 2$										
10^{-2}	3	0.048	9600	0	176	74	250	4118	4.316e-02	9.778e-04
10^{-4}	4	0.024	38400	350	758	482	1590	47018	2.864e-04	4.233e-05
10^{-6}	5	0.012	149784	2886	2984	1838	7708	278514	8.708e-06	3.189e-06
10^{-8}	6	0.012	149784	22902	12098	9806	44086	1841032	2.667e-07	2.771e-07
scale $s = 5$										
10^{-2}	4	0.024	38400	128	524	206	858	24320	8.305e-03	1.423e-03
10^{-4}	5	0.012	149784	1558	2864	1820	6242	187936	2.692e-04	1.089e-04
10^{-6}	6	0.012	149784	16520	11780	6288	34588	1424490	1.681e-05	5.471e-06
10^{-8}	7	0.012	149784	134068	43604	24498	198570	10006768	6.984e-07	4.017e-07
scale $s = 10$										
10^{-2}	4	0.024	38400	296	770	512	1578	44354	2.076e-02	2.905e-03
10^{-4}	6	0.012	149784	4896	7538	4278	16712	538720	9.614e-04	1.058e-04
10^{-6}	7	0.012	149784	46198	32612	19490	98300	3984232	2.200e-05	5.657e-06

Table 5.18: High-gradient Stokes test case for polynomial-approximation of order $k = 4$. Other header details are available in table 5.17.



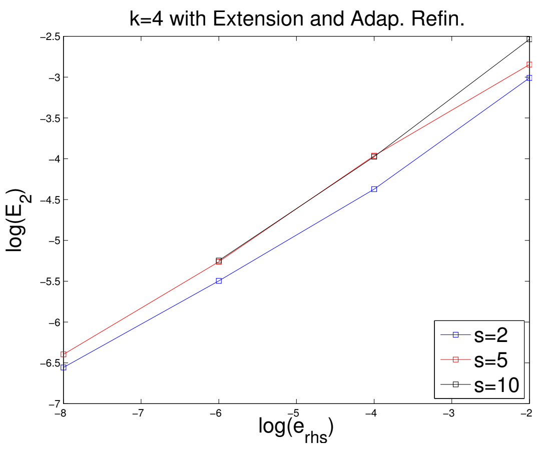


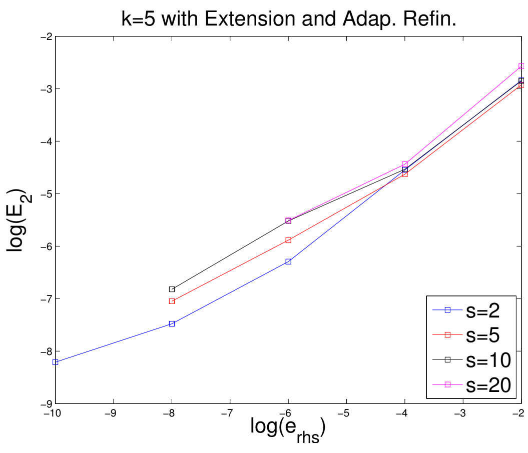
Figure 5.24: For Stokes equations adaptive solver with high-gradient extension. *Left*: Log-log plot of relative error, E_2 versus surface discretization, ϵ_{rhs} for shape figure 5.1(a) and polynomial order of $k = 3$ for results in table 5.17. *Right*: Log-log plot of relative error, E_2 versus surface discretization, ϵ_{rhs} for shape figure 5.1(a) and polynomial order of $k = 4$ for results in table 5.18.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 2$										
10^{-2}	2	0.096	2400	0	56	8	64	2176	2.193e-03	1.446e-03
10^{-4}	4	0.048	9600	56	236	200	492	18312	6.340e-04	2.836e-05
10^{-6}	4	0.024	38400	640	776	512	1928	130376	1.935e-06	5.097e-07
10^{-8}	5	0.012	149784	6160	3056	1872	11088	959136	1.736e-07	3.325e-08
10^{-10}	6	0.012	149784	49210	12368	6988	68566	6899098	6.202e-08	6.172e-09
scale $s = 5$										
10^{-2}	3	0.048	9600	42	224	128	394	15508	4.663e-03	1.185e-03
10^{-4}	5	0.012	149784	442	1166	1022	2630	125984	2.031e-04	2.352e-05
10^{-6}	6	0.012	149784	4074	3680	2538	10292	725830	3.242e-06	1.302e-06
10^{-8}	6	0.012	149784	35070	12368	6632	54070	5131598	2.577e-07	8.945e-08
scale $s = 10$										
10^{-2}	4	0.024	38400	266	758	446	1470	81390	1.599e-02	1.416e-03
10^{-4}	5	0.012	149784	2604	3056	1872	7532	514636	1.523e-04	2.917e-05
10^{-6}	6	0.012	149784	20454	12368	6632	39454	3304598	8.272e-06	3.026e-06
10^{-8}	7	0.012	149784	152246	49472	24968	226686	22027750	7.554e-07	1.511e-07
scale $s = 20$										
10^{-2}	5	0.012	149784	1114	2762	1646	5522	310250	1.796e-02	2.695e-03
10^{-4}	6	0.012	149784	11568	12338	6510	30416	2191756	3.486e-04	3.650e-05
10^{-6}	7	0.012	149784	80124	49466	24932	154522	13011836	1.662e-05	3.096e-06

Table 5.19: High-gradient Stokes test case for polynomial-approximation of order $k = 5$. Other header details are available in table 5.17.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
10^{-2}	2	0.096	2400	0	56	8	64	3648	1.123e-02	5.921e-04
10^{-4}	3	0.048	9600	56	224	128	408	29464	2.299e-04	1.699e-05
10^{-6}	4	0.024	38400	388	776	512	1676	170456	8.989e-07	3.165e-07
10^{-8}	5	0.012	149784	3364	3038	2552	8954	1048358	6.064e-07	3.147e-08
10^{-10}	6	0.012	149784	16572	8354	6544	31470	4417126	7.647e-08	9.762e-09
scale $s = 5$										
10^{-2}	3	0.048	9600	42	224	128	394	26448	6.660e-03	1.371e-03
10^{-4}	4	0.024	38400	380	770	512	1662	167494	6.840e-05	2.214e-05
10^{-6}	5	0.012	149784	3360	3056	1872	8288	1050688	1.930e-06	5.830e-07
10^{-8}	6	0.012	149784	20216	12368	6710	39294	5659846	2.486e-07	7.196e-08
scale $s = 10$										
10^{-2}	4	0.024	38400	176	722	308	1206	113050	1.910e-02	4.005e-03
10^{-4}	5	0.012	149784	1810	3038	1874	6722	712448	1.881e-04	8.017e-05
10^{-6}	6	0.012	149784	11988	12338	6714	31040	3878950	1.251e-05	1.746e-06
10^{-8}	7	0.012	149784	70338	46214	23420	139972	19977592	1.365e-06	2.251e-07
scale $s = 20$										
10^{-2}	5	0.012	149784	854	2414	1484	4752	439588	2.753e-02	6.150e-03
10^{-4}	6	0.012	149784	7812	12110	9296	29218	2950828	2.497e-04	9.229e-05
10^{-6}	7	0.012	149784	50568	48128	23756	122452	15975700	3.046e-05	3.778e-06

Table 5.20: High-gradient Stokes test case for polynomial-approximation of order $k = 6$. Other header details are available in table 5.17.



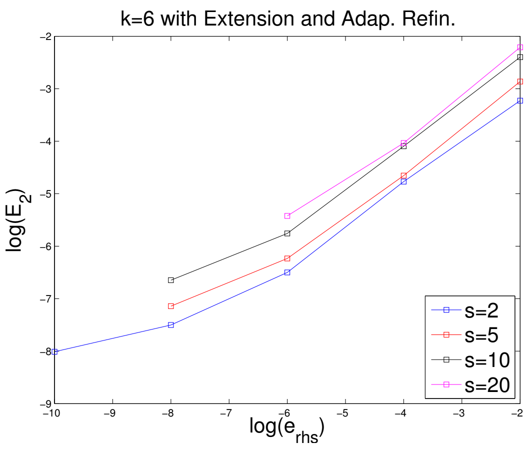


Figure 5.25: For Stokes equations adaptive solver with high-gradient extension. *Left*: Log-log plot of relative error, E_2 versus surface discretization, ϵ_{rhs} for shape figure 5.1(a) and polynomial order of $k = 5$ for results in table 5.19. *Right*: Log-log plot of relative error, E_2 versus surface discretization, ϵ_{rhs} for shape figure 5.1(a) and polynomial order of $k = 6$ for results in table 5.20.

EBI Solver with Volume Force and Stokes Equations Example 2 To further test the EBI solver for the Stokes equations, we use the non-trivial shape in figure 5.1(b) with a more complex volume force at the boundaries (i.e., our polynomial approximation is not exact at interior nodes). The divergence-free force is chosen to have decay and oscillating features. We choose eight points, $\mathbf{x}_i = (x_i, y_i, z_i) = \pm(1/4, 1/4, 1/4)$ for building the following test case:

$$-\mu\Delta\mathbf{u}(\mathbf{x}) + \nabla p(\mathbf{x}) = \sum_{i=1}^{8}(\omega \times (\mathbf{x} - \mathbf{x}_i)) \qquad \left(4q^2 e^{(q||\mathbf{x}-\mathbf{x}_i||^2)}(\Delta(\mathbf{x}-\mathbf{x}_i) + 3/2 + 2/q)\right) + \frac{\pi}{2}\begin{bmatrix} \sin\left(\frac{\pi}{4}((z-z_i)-(y-y_i))\right) \\ \sin\left(\frac{\pi}{4}((x-x_i)-(z-z_i))\right) \\ \sin\left(\frac{\pi}{4}((y-y_i)-(x-x_i))\right) \end{bmatrix},$$

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^{8}(\omega \times (\mathbf{x}-\mathbf{x}_i))e^{(q||\mathbf{x}-\mathbf{x}_i||^2)} + \begin{bmatrix} \sin\left(\frac{\pi}{4}((z-z_i)-(y-y_i))\right) \\ \sin\left(\frac{\pi}{4}((x-x_i)-(z-z_i))\right) \\ \sin\left(\frac{\pi}{4}((y-y_i)-(x-x_i))\right) \end{bmatrix}.$$

We compile the results for this test in tables 5.21-5.23 and figure 5.26.

h_s	ϵ_{rhs}	t_{vfmm}	t_{bis}	n_{its}	t_{far}	t_{near}	t_{tot}	E_∞	E_2
0.192	10^{-2}	1.49e+01	9.22e+00	23	2.50e-01	1.14e+03	1.17e+03	2.72e+00	6.24e-02
0.096	10^{-4}	1.67e+01	3.08e+01	21	5.06e+00	6.79e+02	7.32e+02	2.86e-03	5.43e-04
0.048	10^{-6}	3.16e+01	1.25e+02	18	5.22e+01	6.44e+02	8.55e+02	3.93e-04	3.41e-05
0.024	10^{-8}	1.89e+02	7.93e+02	16	2.05e+02	1.08e+03	2.28e+03	1.02e-05	5.54e-07
0.012	10^{-10}	1.79e+03	5.45e+03	16	9.42e+02	1.96e+03	1.02e+04	3.44e-06	2.00e-07

Table 5.21: EBI solver for the Stokes equations with figure 5.1(b) and non-zero volume force. Here, we set the volume target discretization to $h_{vol} = 0.0078125$ with target point size $N_{vol} = 2231752$ for all tests.

ϵ_{rhs}	T_d	N_{src}
10^{-2}	2	13824
10^{-4}	3	65664
10^{-6}	5	266112
10^{-8}	5	1467072
10^{-10}	6	9588672

Table 5.23: For Table 5.21, for Stokes Test Example 2, for a given ϵ_{rhs}, the resulting octree is of depth T_d. We fix the polynomial approximation to order $k = 6$, and the definition of N_{src} follows the same as in Table 5.23.

h_s	N_s
0.0192	968
0.096	2896
0.048	11132
0.024	43632
0.012	174780

Table 5.22: For figure 5.1(b), given surface discretization h_s, N_s is the corresponding number of surface points.

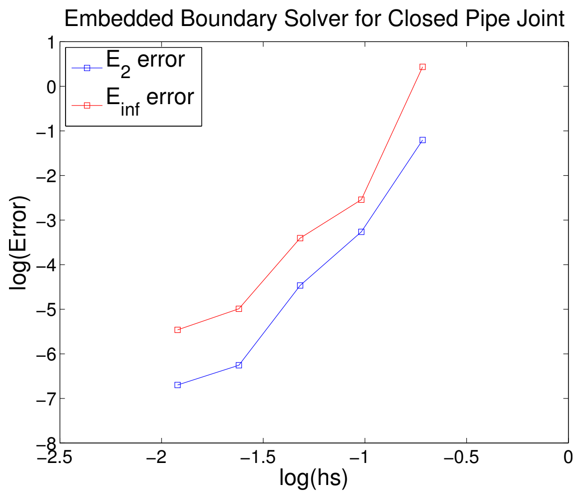


Figure 5.26: Plots of E_2 and E_∞ errors for results from table 5.21.

CONCLUSION

5.1 Conclusions

We have presented an adaptive kernel-independent Fast Multipole Method-based volume integral solver for linear non-oscillatory partials differential equations in three dimensions. Our fast volume solver allows for highly nonuniform force distributions with free-space, periodic and Dirichlet boundary conditions with easy extension to Neumann and more complex boundary conditions in the cube. By incorporating this volume solver with a pre-existing boundary integral solver, we have exhibited the ability to adaptively solve elliptic PDEs with Dirichlet boundary conditions for a variety of complex geometries, where a smooth extension beyond the boundary must be computed.

Future plans for the stand-along volume solver include adding inhomogeneous boundary conditions as in (Ethridge and Greengard, 2001) as well as additional kernels for greater public usability. We are also currently incorporating our periodic boundary conditions and volume solver capabilities into the state-of-the-art kiFMM++ (Lashuk et al., 2009).

For the embedded boundary solver going forward, to obviate the need for interpolating from grid to random locations using a cell-centered approach, we can instead choose a more appropriate solution grid for (i.e., Chebyshev solution points). Currently the solution grid for the volume solver is aligned with the input density locations. The free-space volume solver implementation actually keeps these two grids separate (unless requested), and we are currently extending this feature to the full boundary solver; among the added benefits is the increased accuracy in the final interpolant to target values. Additionally, we are currently working on a hierarchical approach to building the force extension as opposed to the current implementation. Such an approach gives greater control over error estimates.

A

Appendix

A.1 Tree-Level Restriction

As discussed, in order to use our precomputed weights, it is necessary to have an octree which is level-restricted such that all leaf box neighbors are within one level of each other in the tree structure. Alternatively, it is possible to compute additional interactions for unbalanced trees *on-the-fly*. We begin by discussing how we balance octrees to satisfy our tree-level restriction, followed by a discussion of the alternative approach.

A.1.1 Sequential Tree Balancing

There are many significant approaches to balancing trees, most recently and significantly, (Sundar et al., 2008) for parallel balancing. We describe a simple sequential method similar to that described in (Ethridge, 2000) to take a tree which violates the tree level restrictions to one that does not. To begin, all leaf boxes B which have neighbors, U where $depth(U) - depth(B) \geq 2$ are labeled as *primary violators*. Any box B, which is itself not a primary violator but is adjacent to a primary violator that is deeper in the tree than itself, is labeled a *secondary violator*. We then subdivide all primary and secondary violators once and then label all of the new children of primary violators as *possible violators* (secondary violators are split exactly once as they are buffered from any potential additional violations). If these possible violators are indeed adjacent to boxes more than 1 level deep than themselves in the tree, we subdivide them, label their children as possible violators, and continue iterating this procedure until no possible violators are left. More specific details are outlined in Algorithm 6.

Figure A.1 shows an example quadtree in which our space has been discretized around a line

Algorithm 6 Tree Balance for Input T

```
STEP 1 - GENERATE VIOLATORS
for each leaf box $B$ in postorder traversal of $T$ do
    for each box $U \in L_U^B$, if $depth(U) - depth(B) \geq 2$ do
        label $B$ as a primary violator
    end for
end for
for each leaf box $B$ in postorder traversal of $T$, if $B$ is not a primary violator do
    for each box $U \in L_U^B$, if $U$ is a primary violator and $depth(U) > depth(B)$ do
        label $B$ as a secondary violator
    end for
end for
STEP 2 - SUBDIVIDE VIOLATORS
for each leaf box $B$ in postorder traversal of $T$, if $B$ is a primary violator do
    subdivide $B$ and label its children as possible violators. Otherwise, if $B$ is a secondary violator, subdivide it.
end for
STEP 3 - ITERATE ON DESCENDANTS OF VIOLATORS
for each box $B$ in postorder traversal of $T$, if $B$ is a leaf and $B$ is a possible violator do
    for each box $U \in L_U^B$, if $depth(U) - depth(B) \geq 2$ do
        subdivide $B$ and label its children as possible violators
    end for
end for
Repeat Step 3 until there are no boxes left labeled as possible violators
```

traveling through a unit box. We can see that there are many violations of boxes touching other boxes more than one level away.

Step 1 of the tree-balancing algorithm identifies which boxes are *primary violators* and which are *secondary violators*, the results of which can be seen in figure A.2.

As indicated, secondary violators are all split exactly once, and we split primary violators once at first and keep track of their new children. The results of this can be seen in figure A.3. In

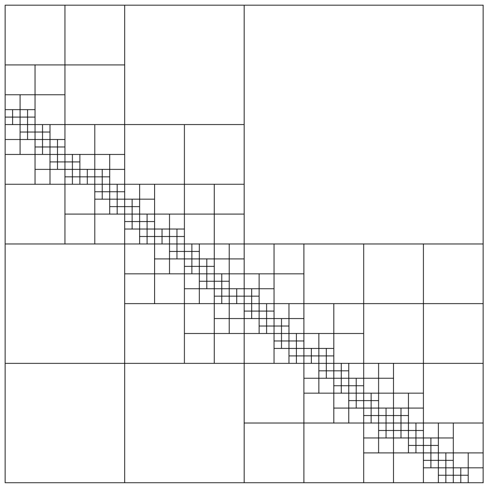


Figure A.1: A 2D example of a quadtree with tree-level violations.

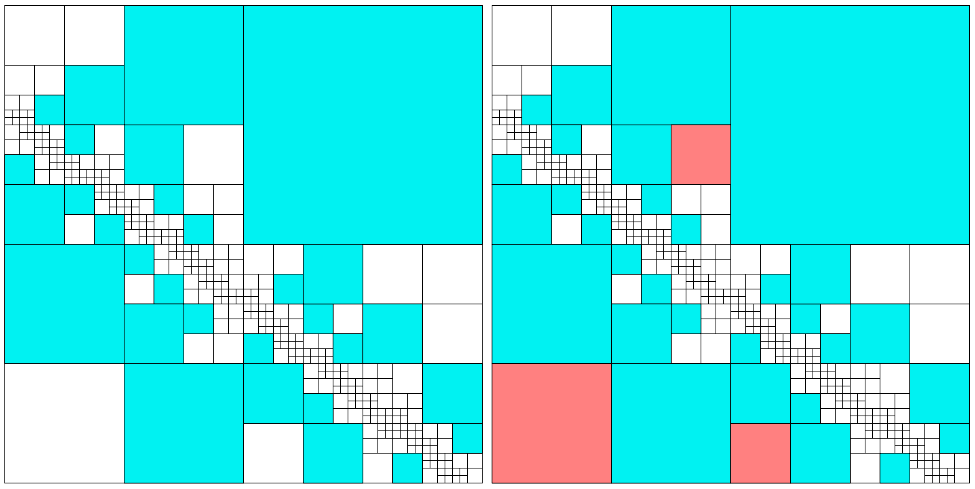


Figure A.2: Identifying primary violator boxes on the left (blue) and secondary violators on the right (red) for Step 1 of the tree-balancing algorithm.

the image on the left, we have marked all new children of primary violators as *possible violators*, and in the image on the right, after we have checked to see if any of these children are violators, we now mark the ones that are indeed violators as needing to be split.

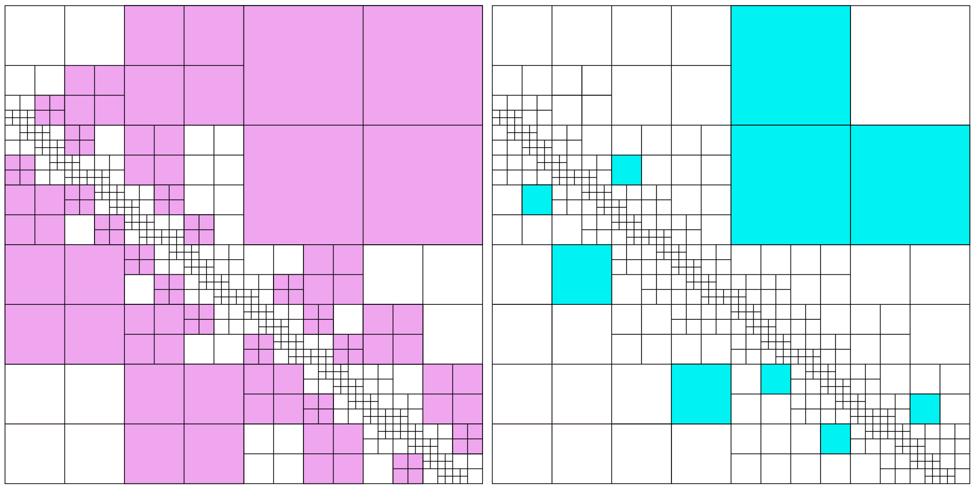


Figure A.3: The image on the left shows the effect of splitting secondary violators and primary violators once for Step 2 of the tree-balancing algorithm. The descendants of primary violators are identified in purple on the left. On the right, we identify in blue which of these descendants are themselves violators. Descendants of secondary violators need not be tracked as secondary violators only need be subdivided once.

In figure A.4, we split all new violators and mark their children as possible violators on the left. Of these, only one child is an actual violator of the tree-level restriction, so we mark it on the right for subdivision.

The splitting of the final violator in figure A.4 leaves a tree which conforms to the tree-level restriction, which can be seen in figure A.5.

The procedure in 3D is completely analogous to the example we have shown above in 2D, the difference being of course that when we subdivide a box B, we have 8 new children. Ad-

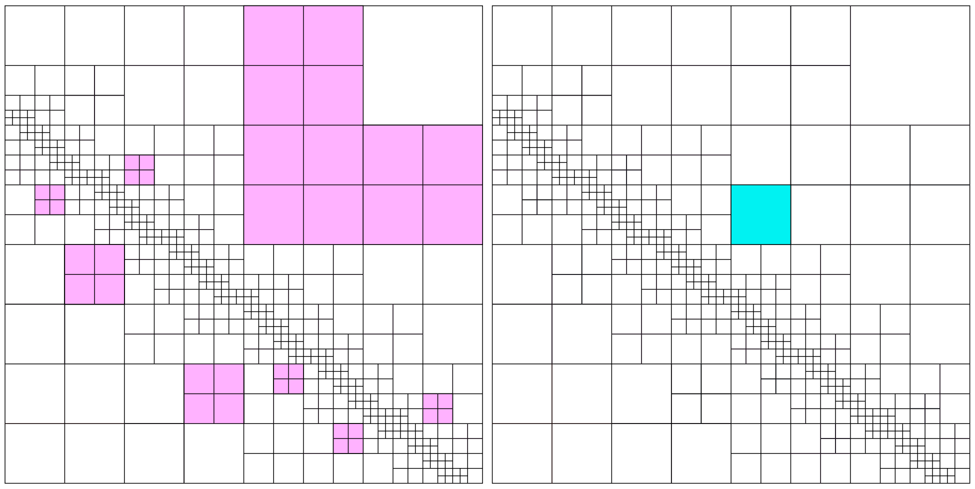


Figure A.4: The image on the left shows the effect of splitting descendants of primary violators, which are themselves violators for the first iteration of Step 3 of the tree-balancing algorithm. Split boxes are indicated in purple, so we can see which of these descendant boxes are still violators. The remaining violator is seen on the right in blue, and the second iteration of Step 4 results in the final tree in figure, which obeys the tree-level restriction.

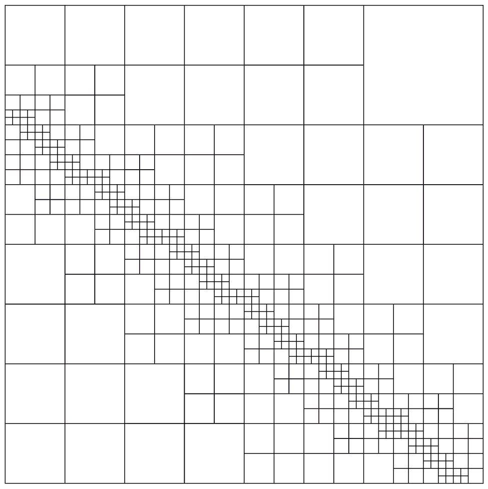


Figure A.5: The level-restricted tree resulting from the full tree-balancing algorithm.

ditionally, instead of only considering violations on four edges and four corners in 2D, in 3D, violations have to be considered across six faces, twelve edges, and eight corners. This last point we consider is why we may want to consider a different approach.

A.1.2 Computing *On-the-Fly* Interactions

As indicated above, subdividing a box in 3D results in 8 new children instead of 4. To put this in context, consider the largest violator on the bottom-right of the original 2D quadtree in figure A.1. This violator resulted in 16 new boxes being generated. Consider an analogous situation for a 3D octree, in which a single primary violator occupies $1/8$th of our entire domain. If we were to split this box once, we get 8 new children. Then, if we needed to split six of these boxes once more, we get 48 new boxes. Now, if we had to split only four of these boxes once more, we get 32 new boxes. The results of doing 3 iterations of splitting on a box B and some of its children. The result in this case is that one box becomes 78. This may not seem like too many, but consider the *worst-case*: if a single box B were to be completely subdivided just two times, one box could result in up to 64 boxes, and if we had to do complete subdivision three times, this would result in up to 512 boxes in 3D (for comparison, in 2D, *worst-case* would result in 16 boxes for two levels of subdivision and 64 boxes for three levels of subdivision).

To put the increased computation in further context, consider the cost for a box B computing the potential from a ring of boxes around its border which are 4 levels deeper in the tree. An extreme scenario can be seen in figure A.6. Before refinement, if we were to have the interaction matrices for computing the influence of boxes $U \in L_U^B$, which are 4 levels deeper than B in the 2D case, we would have to compute $69k^2 N_k = \frac{69k^3(k+1)}{2}$ near-interaction values. After applying our tree-balancing algorithm in 2D (as seen in figure A.6 on the right), our original box becomes 52 boxes, so the total number of precomputed near-field interactions would become $\frac{492k^3(k+1)}{2}$, or an approximately 4-fold increase in the computations (not including the cost of building the new polynomial coefficients for the new boxes).

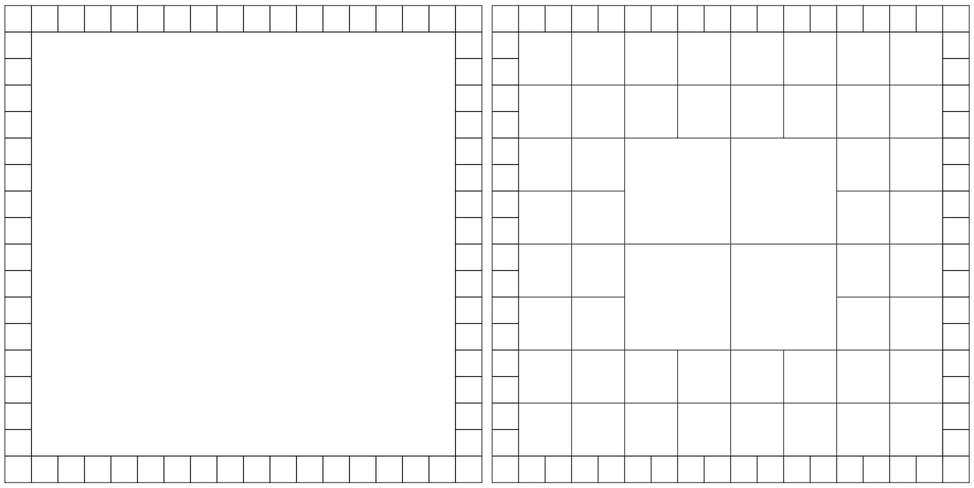


Figure A.6: An extreme scenario for a box in 2D that needs 3 levels of tree-refinement on the left, and the result of applying the sequential tree-balancing algorithm on the right.

The additional cost in 2D is not insignificant, but the case we have provided is quite extreme, so this cost could be considered acceptable. In 3D, however, the additional computation would be much larger. An analogous example to figure A.6 would be a 3D box B in which all of the boxes $U \in L_U^B$ are 4 levels deeper in our octree. In such a case, the number of precomputed near-field interactions on B (again assuming the pre-built tables exist) would be $1737k^3N_k = \frac{1737k^4(k+1)(k+2)}{6}$ since there are 1736 such possible near neighbors, and the number of points in $\mathbf{x}^{B,g}$ is k^3. For $k = 4$, this is already a nearly 200-fold increase in computations from 2D to 3D due to the larger size of the grid, N_K and the larger number of such neighbors. Now, if we do the tree-balancing refinement for such a box B, B would be transformed into 484 boxes, and the total number of computations would jump to $\approx \frac{19360k^4(k+1)(k+2)}{6}$. This is nearly a 12-fold jump in the number of computations (again not including the cost of building the new polynomial coefficients for the 484 new boxes, which is an additionally significant cost), and now more than 300 times the cost of the analogous situation in 2D.

The above situation is quite extreme and very unlikely, but it is meant to point out that the additional cost of balancing beyond one level in 3D is significant. As such, we have developed a method for computing interaction matrices *on the fly*. That is, if there exists some box $U \in L_U^B$ for box B with center $\mathbf{c}^B$ such that $depth(U) - depth(B) \geq 2$, we compute the interactions $F_j(\mathbf{x})$ such that for all $\mathbf{x} \in \mathbf{x}^{U,g}$,

$$\begin{aligned} q(\mathbf{x}) &= \frac{1}{4\pi}\int_B \frac{f(\mathbf{y})}{|\mathbf{x}-\mathbf{y}|}d\mathbf{y} \\ &\approx \sum_{j=1}^{N_k} \gamma_j^B F_j(\mathbf{x}), \end{aligned}$$

where

$$F_j(\mathbf{x}) = \int_B \beta_j(\mathbf{y}-\mathbf{c}_B)\frac{1}{|\mathbf{x}-\mathbf{y}|}d\mathbf{y}.$$

Once these interactions have been computed, we store them to disk in case we need them again later. Additionally, we compute the F_j to a level of $p+1$ precision. The only tricky detail is in how these interaction matrices are stored and looked up such that they can be called upon later. To solve for this, all files are stored in directories indicating the value of $|depth(U)-depth(B)|$ (this only works for scale-variant kernels; scale-invariant kernels are handled by creating an additional level of subdirectories) as well whether U is a finer or coarser neighbor, and the value of precision, p. Following this, we store a lookup variable, λ by first computing an index, $i^B = \frac{\mathbf{c}^B - \mathbf{c}^U}{r_U}$, where $\mathbf{c}^U = (c_0^U, c_1^U, c_2^U)$ is the center of U and r_U is its radius. Then, $n = \frac{r_B + r_U}{r_U} + 1$ (this value is in fact equal to $2^{|depth(U)-depth(B)|+2}$). We then let $\lambda = i_z^B n^2 + i_y^B n + i_x^B$. It is easy to verify that this value along with an indicator of U's depth relative to B is unique, and we use these two values to store the interactions computed on the fly.

As a final note, the above process can also be used for computing interactions from the L_W^B and L_X^B lists, where boxes in those lists also violate the tree-level restrictions of being more than 1 level deeper or shallower in the octree, respectively. Additionally, as discussed previously, symmetries can be developed to further reduce the number of required precomputations.

Remark A.1. *As indicated, computing interactions on the fly can be preferable in many of the most extreme cases for octrees which violate tree-level restrictions. In all of our examples, we perform complete refinement; however, the option of complete rebalancing versus computing interactions on the fly is let as an option for the user in our implementation.*

A.2 Full Tables for High-Gradient Poisson Examples

Here, we put the full tables for polynomial orders $k = 3, 4, 5, 6$ for uniform refinement and adaptive refinement, comparing no extension with the use of the extension for the high-gradient volume force in (5.1) for the Poisson equation. In figures 5.5- 5.6, we plot the E_2 results versus the depth of the uniform tree, comparing no extension to extension. For subsequent tables, we look at the effects of adaptive refinement with use of the extension. The results of these tables were plotted in section 5.0.6.

T_d	E_∞	E_2	E_∞	E_2	E_∞	E_2	E_∞	E_2
	No Ext.		Ext.		No Ext.		Ext.	
	$s = 2$				$s = 5$			
2	7.70E-02	1.13E-02	3.25E-02	3.02E-03	3.67E-01	8.75E-02	1.30E-01	1.74E-02
3	1.91E-02	1.71E-03	5.33E-03	1.17E-03	1.32E-01	9.16E-03	2.68E-02	7.37E-03
4	3.36E-03	2.04E-04	1.94E-03	1.96E-04	3.31E-02	1.83E-03	8.43E-03	1.25E-03
5	3.25E-04	2.12E-05	6.09E-04	1.24E-05	4.16E-03	2.11E-04	1.92E-03	2.01E-04
6	2.22E-05	2.17E-06	2.73E-05	9.10E-07	2.96E-04	2.21E-05	7.75E-05	3.12E-05
	$s = 10$				$s = 20$			
2	8.57E-01	2.07E-01	3.38E-01	4.28E-02	1.53E+00	2.94E-01	7.22E-01	8.34E-02
3	3.33E-01	2.44E-02	9.42E-02	2.10E-02	5.66E-01	6.82E-02	2.67E-01	6.04E-02
4	1.29E-01	8.35E-03	4.71E-02	7.41E-03	3.25E-01	3.01E-02	1.47E-01	1.74E-02
5	2.55E-02	1.10E-03	8.32E-03	1.16E-03	1.15E-01	4.82E-03	3.51E-02	5.85E-03
6	2.15E-03	1.23E-04	2.82E-04	1.68E-04	1.39E-02	6.29E-04	6.20E-03	7.22E-04
	$s = 40$				$s = 80$			
2	2.24E+00	5.57E-01	1.97E+00	4.16E-01	1.15E+01	2.33E+00	7.81E+00	1.60E+00
3	1.47E+00	2.08E-01	6.70E-01	1.66E-01	5.00E+00	6.93E-01	1.55E+00	4.92E-01
4	7.87E-01	6.56E-02	3.51E-01	4.60E-02	1.21E+00	1.49E-01	1.53E-00	1.47E-01
5	3.20E-01	1.53E-02	9.73E-02	1.36E-02	5.09E-01	3.88E-02	3.64E-01	3.31E-02
6	6.61E-02	3.07E-03	2.46E-02	4.36E-03	2.00E-01	1.11E-02	1.56E-01	1.14E-02

Table A.1: For the uniform leaf counts from table 5.3 for use with no extension versus extension, and a polynomial approximation of $k = 3$, we compare the errors for the high-gradient Poisson test case for various values of s.

T_d	E_∞	E_2	E_∞	E_2	E_∞	E_2	E_∞	E_2
	No Ext.		Ext.		No Ext.		Ext.	
	$s = 2$				$s = 5$			
2	4.70E-02	5.07E-03	3.80E-02	4.31E-03	1.85E-01	3.23E-02	1.48E-01	2.64E-02
3	6.35E-03	2.81E-04	5.76E-04	1.59E-04	5.59E-02	4.36E-03	1.33E-02	3.32E-03
4	4.94E-04	2.06E-05	1.85E-04	1.67E-05	9.36E-03	3.76E-04	1.50E-03	3.28E-04
5	2.09E-05	1.33E-06	7.12E-05	2.34E-06	5.78E-04	2.39E-05	3.15E-04	2.60E-05
6	4.83E-07	6.60E-08	4.97E-07	8.26E-08	1.38E-05	1.18E-06	9.66E-06	2.28E-06
	$s = 10$				$s = 20$			
2	3.38E-01	7.60E-02	4.67E-01	8.88E-02	8.06E-01	1.68E-01	1.21E+00	2.02E-01
3	2.32E-01	1.75E-02	2.20E-02	1.05E-02	5.85E-01	4.69E-02	5.41E-01	4.41E-02
4	6.83E-02	3.23E-03	4.62E-03	3.90E-03	2.42E-01	1.79E-02	2.69E-02	1.62E-02
5	6.23E-03	2.30E-04	2.16E-03	2.61E-04	4.87E-02	1.83E-03	2.09E-02	3.02E-03
6	2.00E-04	1.24E-05	1.32E-04	2.40E-05	2.51E-03	1.27E-04	4.33E-03	2.11E-04
	$s = 40$				$s = 80$			
2	2.03E+00	3.80E-01	2.17E+00	3.30E-01	9.74E+00	5.92E-01	3.96E+00	5.85E-01
3	1.72E+00	1.30E-01	5.82E-01	1.04E-01	3.48E+00	4.20E-01	2.46E+00	4.70E-01
4	5.20E-01	4.91E-02	2.56E-01	4.57E-02	1.94E+00	1.06E-01	1.07E+00	1.72E-01
5	1.77E-01	8.37E-03	5.12E-02	9.73E-03	3.98E-01	2.56E-02	3.53E-01	2.27E-02
6	1.84E-02	1.02E-03	7.77E-03	1.99E-03	9.28E-02	6.32E-03	6.70E-02	6.85E-03

Table A.2: For the uniform leaf counts from table 5.3 for use with no extension versus extension, and a polynomial approximation of $k = 4$, we compare the errors for the high-gradient Poisson test case for various values of s

T_d	E_∞	E_2	E_∞	E_2	E_∞	E_2	E_∞	E_2
	No Ext.		Ext.		No Ext.		Ext.	
	$s = 2$				$s = 5$			
2	1.98E-02	1.55E-03	1.26E-02	1.04E-03	2.50E-01	2.49E-02	1.40E-01	1.21E-02
3	2.02E-03	7.81E-05	4.49E-04	5.07E-05	5.42E-02	2.03E-03	1.11E-02	1.20E-03
4	1.02E-04	2.79E-06	3.63E-05	1.68E-06	4.09E-03	1.02E-04	4.25E-04	4.93E-05
5	1.84E-06	6.44E-08	7.91E-07	8.39E-08	1.25E-04	3.40E-06	2.22E-05	2.80E-06
6	2.50E-08	3.54E-09	2.06E-07	5.90E-09	1.73E-06	7.83E-08	1.58E-06	8.04E-08
	$s = 10$				$s = 20$			
2	6.88E-01	7.63E-02	1.04E+00	4.31E-02	1.32E+00	1.09E-01	4.31E+00	1.80E-01
3	2.38E-01	1.71E-02	1.23E-01	1.56E-02	5.35E-01	6.61E-02	2.92E-01	3.43E-02
4	3.73E-02	1.14E-03	6.43E-03	7.47E-04	1.78E-01	8.65E-03	7.39E-02	8.56E-03
5	2.36E-03	5.60E-05	3.61E-04	2.71E-05	2.25E-02	7.46E-04	3.36E-03	5.90E-04
6	2.31E-05	1.48E-06	2.90E-05	1.87E-06	4.69E-04	2.74E-05	5.34E-04	1.94E-05
	$s = 40$				$s = 80$			
2	3.42E+00	4.94E-01	1.51E+01	7.28E-01	6.48E+00	1.63E+00	3.75E+01	1.98E+00
3	1.07E+00	1.56E-01	1.71E+00	8.23E-02	2.22E+00	4.09E-01	4.06E+00	4.83E-01
4	5.89E-01	3.66E-02	6.88E-01	2.38E-02	1.40E+00	1.11E-01	2.10E-01	2.95E-02
5	1.07E-01	5.60E-03	1.47E-02	4.39E-03	2.68E-01	1.87E-02	8.67E-02	7.20E-03
6	5.75E-03	4.40E-04	5.29E-03	4.03E-04	6.04E-02	3.59E-03	1.36E-02	1.62E-03

Table A.3: For the uniform leaf counts from table 5.3 for use with no extension versus extension, and a polynomial approximation of $k = 5$, we compare the errors for the high-gradient Poisson test case for various values of s

T_d	E_∞	E_2	E_∞	E_2	E_∞	E_2	E_∞	E_2
	No Ext.		Ext.		No Ext.		Ext.	
	$s = 2$				$s = 5$			
2	1.65E-02	1.11E-03	2.09E-02	9.13E-04	1.17E-01	1.83E-02	1.16E-01	5.30E-03
3	9.31E-04	1.93E-05	2.97E-04	2.51E-05	1.98E-02	6.49E-04	5.70E-03	7.86E-04
4	1.41E-05	2.92E-07	2.03E-05	2.26E-07	1.31E-03	2.85E-05	5.07E-04	3.33E-05
5	1.71E-07	1.27E-08	2.48E-07	2.03E-08	1.62E-05	4.50E-07	5.25E-06	2.74E-07
6	2.58E-08	3.35E-09	8.29E-08	5.38E-09	1.52E-07	9.04E-09	3.20E-07	1.27E-08
	$s = 10$				$s = 20$			
2	5.31E-01	5.65E-02	1.24E+00	5.55E-02	1.26E+00	9.23E-02	5.13E+00	1.98E-01
3	1.11E-01	8.35E-03	2.31E-02	4.54E-03	4.39E-01	5.28E-02	4.50E-02	1.35E-02
4	2.38E-02	6.26E-04	3.89E-03	4.06E-04	1.49E-01	9.58E-03	1.02E-02	2.83E-03
5	5.55E-04	1.58E-05	9.63E-05	6.64E-06	1.05E-02	3.17E-04	1.89E-03	1.71E-04
6	7.22E-06	2.32E-07	5.24E-06	3.79E-07	1.08E-04	7.12E-06	1.31E-04	9.21E-06
	$s = 40$				$s = 80$			
2	1.99E+00	2.80E-01	1.47E+01	6.22E-01	8.52E+00	7.37E-01	3.50E+01	1.35E+00
3	1.11E+00	1.28E-01	3.12E-01	4.02E-02	2.24E+00	2.51E-01	1.14E+00	2.15E-01
4	7.28E-01	6.46E-02	1.87E-01	9.60E-03	2.30E+00	2.13E-01	2.83E-01	3.54E-02
5	6.87E-02	3.87E-03	1.68E-02	1.65E-03	2.63E-01	2.16E-02	7.56E-02	8.59E-03
6	4.09E-03	2.24E-04	1.85E-03	1.82E-04	8.87E-02	4.25E-03	3.44E-03	2.08E-03

Table A.4: For the uniform leaf counts from table 5.3 for use with no extension versus extension, and a polynomial approximation of $k = 6$, we compare the errors for the high-gradient Poisson test case for various values of s.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 2$										
10^{-2}	3	0.048	9600	56	224	128	408	3648	5.33E-02	1.17E-03
10^{-3}	4	0.024	38400	56	512	296	864	6120	4.657E-03	7.403E-04
10^{-4}	4	0.024	38400	472	776	464	1712	23416	3.346E-03	2.591E-04
10^{-5}	5	0.012	149784	3728	2912	1920	8560	138488	2.806E-04	4.377E-05
10^{-6}	5	0.012	149784	5488	3056	1872	10416	188600	1.883E-04	2.443E-05
10^{-7}	6	0.012	149784	42000	12368	6632	61000	1295632	8.413E-06	1.499E-06
10^{-8}	7	0.012	149784	62328	12368	7064	81760	1844536	2.710E-06	3.305E-07
scale $s = 5$										
10^{0}	2	0.096	2400	0	56	8	64	480	1.300E-01	1.735E-02
10^{-1}	3	0.048	9600	56	224	128	408	2136	2.680E-02	7.371E-03
10^{-2}	4	0.024	38400	304	776	464	1544	18880	2.871E-02	2.026E-03
10^{-3}	5	0.012	149784	544	2072	1496	4112	41392	7.180E-03	1.018E-03
10^{-4}	5	0.012	149784	3024	3056	1872	7952	122072	3.087E-03	3.395E-04
10^{-5}	6	0.012	149784	19392	11624	6632	37648	668296	3.247E-04	3.612E-05
10^{-6}	6	0.012	149784	30968	12368	6632	49968	997768	6.750E-05	1.974E-05
10^{-7}	7	0.012	149784	202728	49472	24872	277072	6117976	6.338E-06	2.598E-06
10^{-8}	8	0.006	149784	341648	115352	82008	539008	10605000	9.230E-06	1.146E-06
scale $s = 10$										
10^{-0}	3	0.048	9600	0	200	32	232	2064	1.13E-01	2.76E-02
10^{-1}	4	0.024	38400	184	728	200	1112	14320	9.02E-02	1.15E-02
10^{-2}	5	0.012	149784	1032	2600	1608	5240	62480	4.06E-02	3.38E-03
10^{-3}	5	0.012	149784	2352	3056	1872	7280	103928	1.05E-02	1.25E-03
10^{-4}	6	0.012	149784	14784	12368	6632	33784	560800	2.51E-03	2.00E-04
10^{-5}	7	0.012	149784	35688	42032	17840	95560	1492792	7.02E-04	6.41E-05
10^{-6}	7	0.012	149784	148856	49472	24872	223200	4663432	6.14E-04	4.70E-05
10^{-7}	8	0.006	599136	658880	197624	99960	956464	20418744	5.27E-04	4.75E-05

Table A.5: High-gradient adaptive Poisson tests for $k = 3$.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 2$										
10^{-1}	2	0.096	2400	0	56	8	64	1088	3.80E-02	4.31E-03
10^{-2}	3	0.048	9600	0	200	32	232	5048	3.20E-03	9.01E-04
10^{-3}	3	0.048	9600	56	224	128	408	8744	5.75E-04	1.58E-04
10^{-4}	4	0.024	38400	152	704	512	1368	30992	6.91E-03	4.56E-05
10^{-5}	4	0.024	38400	640	776	512	1928	66736	2.76E-04	2.06E-05
10^{-6}	5	0.012	149784	2240	3056	1872	7168	239544	4.04E-05	6.94E-06
10^{-7}	5	0.012	149784	5992	3056	1872	10920	479672	7.12E-05	2.41E-06
10^{-8}	6	0.012	149784	20608	12368	6728	39704	1702504	2.24E-06	5.36E-07
10^{-9}	6	0.012	149784	53144	12368	6728	72240	3784808	2.93E-07	7.88E-08
10^{-10}	7	0.012	599136	173792	49472	25640	248904	12658752	5.14E-07	6.31E-08
scale $s = 5$										
10^{-0}	2	0.096	2400	0	56	8	64	1088	1.48E-01	2.64E-02
10^{-1}	3	0.048	9600	0	200	32	232	5048	9.77E-02	8.11E-03
10^{-2}	4	0.024	38400	152	632	248	1032	29888	1.66E-02	1.87E-03
10^{-3}	4	0.024	38400	304	776	512	1592	45232	7.34E-03	4.84E-04
10^{-4}	5	0.012	149784	1560	3008	1872	6440	192976	5.75E-04	9.21E-05
10^{-5}	5	0.012	149784	3920	3056	1872	8848	347064	2.89E-04	2.84E-05
10^{-6}	6	0.012	149784	17640	12368	6728	36736	1512552	1.46E-04	9.01E-06
10^{-7}	6	0.012	149784	34160	12368	6728	53256	2569832	1.58E-04	4.55E-06
10^{-8}	7	0.012	149784	135208	43808	20456	199472	10073072	8.78E-05	1.24E-06
10^{-9}	7	0.012	149784	289824	49472	25640	364936	20084800	1.29E-05	6.26E-07

Table A.6: High-gradient adaptive tests for $k = 4$, continued in Table A.7.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 10$										
10^{-0}	3	0.048	9600	0	200	32	232	5048	2.13e+00	1.87E-02
10^{-1}	4	0.024	38400	152	608	104	864	28760	6.35E-02	9.40E-03
10^{-2}	4	0.024	38400	304	776	512	1592	45232	4.77E-02	3.31E-03
10^{-3}	5	0.012	149784	2352	3056	1872	7280	246712	2.63E-03	2.93E-04
10^{-4}	6	0.012	149784	5976	9656	5792	21424	672528	1.01E-03	1.19E-04
10^{-5}	6	0.012	149784	18928	12368	6728	38024	1594984	6.72E-04	3.53E-05
10^{-6}	7	0.012	149784	53216	37904	22136	113256	4585128	8.57E-05	1.41E-05
10^{-7}	7	0.012	149784	160016	49472	25640	235128	11777088	3.20E-05	8.33E-06
scale $s = 20$										
10^{-0}	3	0.048	9600	0	224	128	352	5168	3.25E-01	5.90E-02
10^{-1}	4	0.024	38400	304	776	368	1448	45232	1.11E-01	1.76E-02
10^{-2}	5	0.012	149784	1848	3056	1872	6776	214456	3.31E-02	2.04E-03
10^{-3}	6	0.012	149784	8832	12224	6632	27688	942408	5.51E-03	4.34E-04
10^{-4}	7	0.012	149784	13968	14816	8576	37360	1352952	5.10E-03	2.90E-04
10^{-5}	7	0.012	149784	73496	49472	25640	148608	6239808	4.44E-03	7.54E-05
10^{-6}	8	0.006	599136	139616	80888	65000	285504	11532048	1.06E-03	5.81E-05
10^{-7}	8	0.006	599136	613464	197624	101184	912272	45482424	3.76E-04	1.76E-05

Table A.7: High-gradient adaptive tests for $k = 4$, continued from Table A.6.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 2$										
10^{-2}	2	0.096	2400	0	56	8	64	2176	1.26E-02	1.04E-03
10^{-4}	3	0.048	9600	56	224	128	408	17256	3.12E-04	5.07E-05
10^{-5}	4	0.024	38400	304	776	512	1592	88376	1.54E-04	3.06E-05
10^{-6}	4	0.024	38400	640	776	512	1928	130376	3.70E-05	1.83E-06
10^{-7}	5	0.012	149784	3024	3056	1872	7952	567136	1.88E-06	1.48E-07
10^{-8}	5	0.012	149784	6328	3056	1872	11256	980136	1.59E-07	6.00E-08
10^{-9}	6	0.012	149784	17840	7472	6680	31992	2620536	3.11E-07	2.35E-08
10^{-10}	6	0.012	149784	51016	12368	6632	70016	7124848	2.89E-08	4.98E-09
scale $s = 5$										
10^{-0}	2	0.096	2400	0	56	8	64	2176	1.40E-01	1.21E-02
10^{-1}	3	0.048	9600	0	200	32	232	9976	5.71E-02	4.24E-03
10^{-2}	3	0.048	9600	56	224	128	408	17256	6.01E-03	1.20E-03
10^{-3}	4	0.024	38400	184	728	368	1280	67448	1.39E-03	2.25E-04
10^{-4}	5	0.012	149784	496	1424	1208	3128	146480	8.09E-04	6.45E-05
10^{-5}	5	0.012	149784	2856	3056	1872	7784	546136	4.99E-05	4.22E-06
10^{-6}	6	0.012	149784	4256	3776	2528	10560	754416	7.34E-06	1.88E-06
10^{-7}	6	0.012	149784	17752	12368	6632	36752	2966848	3.34E-06	1.47E-07
10^{-8}	6	0.012	149784	36008	12368	6632	55008	5248848	2.17E-07	5.28E-08
10^{-9}	7	0.012	149784	113776	47384	24968	186128	16989488	2.50E-07	3.13E-08
10^{-10}	7	0.012	149784	277168	49472	24968	351608	37643000	2.51E-07	3.08E-08

Table A.8: High-gradient adaptive tests for $k = 5$, continued in Table A.9.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 10$										
10^{-0}	2	0.096	2400	0	56	8	64	2176	1.04e+00	4.31E-02
10^{-1}	3	0.048	9600	56	224	128	408	17256	8.26E-02	1.57E-02
10^{-2}	4	0.024	38400	304	776	512	1592	88376	6.37E-03	8.53E-04
10^{-3}	5	0.012	149784	1224	2960	1776	5960	336064	1.30E-03	1.79E-04
10^{-4}	5	0.012	149784	2856	3056	1872	7784	546136	4.35E-04	4.22E-05
10^{-5}	6	0.012	149784	9744	12368	6632	28744	1965848	1.89E-04	8.01E-06
10^{-6}	6	0.012	149784	21000	12368	6632	40000	3372848	2.51E-05	2.41E-06
10^{-7}	7	0.012	149784	66344	44768	21752	132864	10971400	9.14E-06	9.45E-07
10^{-8}	7	0.012	149784	163208	49472	24968	237648	23398000	5.23E-06	5.21E-07
10^{-9}	8	0.006	599136	300472	167648	92520	560640	47542312	5.13E-06	4.77E-07
scale $s = 20$										
10^{-0}	3	0.048	9600	0	224	128	352	10264	4.80E-01	7.47E-02
10^{-1}	4	0.024	38400	184	728	368	1280	67448	1.68e+00	1.64E-02
10^{-2}	5	0.012	149784	1224	2960	1776	5960	336064	1.67E-02	2.30E-03
10^{-3}	6	0.012	149784	5400	9608	4520	19528	1246856	3.36E-03	3.07E-04
10^{-4}	6	0.012	149784	12096	12368	6632	31096	2259848	5.66E-04	3.62E-05
10^{-5}	7	0.012	149784	35816	44384	22424	102624	7177672	1.46E-04	1.01E-05
10^{-6}	7	0.012	149784	86768	49472	24968	161208	13843000	8.48E-05	5.84E-06
10^{-7}	8	0.006	599136	175792	142712	60384	378888	30617704	2.59E-05	1.74E-06
scale $s = 40$										
10^{-0}	4	0.024	38400	152	680	104	936	58320	4.63e+00	9.25E-02
10^{-1}	5	0.012	149784	864	2696	1560	5120	273688	7.06E-01	1.52E-02
10^{-2}	6	0.012	149784	5520	11048	4592	21160	1342928	1.22E-02	1.78E-03
10^{-3}	7	0.012	149784	10464	13232	7280	30976	2109632	4.73E-03	5.94E-04
10^{-4}	7	0.012	149784	46712	49376	24824	120912	8824192	2.92E-03	1.08E-04
10^{-5}	8	0.006	599136	81904	103424	45432	230760	16626424	2.87E-03	1.01E-04

Table A.9: High-gradient adaptive tests for $k = 5$, continued from Table A.8.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 2$										
10^{-2}	2	0.096	2400	0	56	8	64	3648	2.09E-02	9.13E-04
10^{-3}	3	0.048	9600	0	224	128	352	17400	8.89E-04	5.46E-05
10^{-4}	3	0.048	9600	56	224	128	408	29464	2.97E-04	2.51E-05
10^{-5}	4	0.048	9600	184	728	368	1280	116168	5.38E-05	7.73E-06
10^{-6}	4	0.024	38400	472	776	512	1760	188600	6.93E-06	4.57E-07
10^{-7}	4	0.024	38400	640	776	512	1928	224888	2.07E-06	2.92E-07
10^{-8}	5	0.012	149784	3920	3056	1872	8848	1171648	1.72E-07	2.82E-08
10^{-9}	5	0.012	149784	6328	3056	1872	11256	1691776	1.53E-07	1.58E-08
10^{-10}	6	0.012	149784	13696	10208	4544	28448	4010008	6.18E-08	8.99E-09
scale $s = 5$										
10^{-1}	2	0.096	2400	0	56	8	64	3648	1.16E-01	5.31E-03
10^{-2}	3	0.048	9600	56	224	128	408	29464	5.69E-03	7.87E-04
10^{-3}	4	0.024	38400	184	728	368	1280	116168	5.07E-03	1.82E-04
10^{-4}	4	0.024	38400	472	776	512	1760	188600	1.07E-03	3.62E-05
10^{-5}	5	0.012	149784	1728	2960	1776	6464	688600	1.50E-05	4.43E-06
10^{-6}	5	0.012	149784	3528	3056	1872	8456	1086976	5.75E-06	6.91E-07
10^{-7}	6	0.012	149784	5336	5840	3480	14656	1764304	5.21E-06	2.66E-07
10^{-8}	6	0.012	149784	21168	12368	6728	40264	5865472	3.47E-07	2.70E-08
10^{-9}	6	0.012	149784	36568	12368	6728	55664	9191872	3.39E-07	1.39E-08
scale $s = 10$										
10^{-0}	2	0.096	2400	0	56	8	64	3648	1.24e+00	5.55E-02
10^{-1}	3	0.048	9600	0	224	128	352	17400	2.93E-02	4.65E-03
10^{-2}	4	0.024	38400	184	728	248	1160	115856	6.67E-03	8.74E-04
10^{-3}	5	0.012	149784	640	2312	1352	4304	382952	5.09E-04	9.97E-05
10^{-4}	5	0.012	149784	2016	3056	1872	6944	760384	2.15E-04	1.05E-05
10^{-5}	6	0.012	149784	4296	7112	3848	15256	1644376	1.60E-05	6.11E-06
10^{-6}	6	0.012	149784	12432	12368	6728	31528	3978496	6.96E-06	1.02E-06
10^{-7}	6	0.012	149784	23352	12368	6728	42448	6337216	5.03E-06	5.75E-07
10^{-8}	7	0.006	599136	81056	48584	24896	154536	22554984	1.08E-06	1.77E-07

Table A.10: High-gradient adaptive tests for $k = 6$, continued in Table A.11.

rhs	T_d	h_s	N_s	L_{in}	L_{crv}	Lvs_{out}	L_{tot}	N_{svol}	E_∞	E_2
scale $s = 20$										
10^{-0}	3	0.048	9600	0	200	32	232	16968	1.77e+01	1.39e+00
10^{-1}	4	0.024	38400	152	704	248	1104	104008	3.94E-01	7.87E-03
10^{-2}	5	0.012	149784	1104	2864	1584	5552	542584	1.34E-02	6.09E-04
10^{-3}	5	0.012	149784	2352	3056	1872	7280	832960	2.48E-03	2.62E-04
10^{-4}	6	0.012	149784	7968	12296	6344	26608	2998840	1.58E-04	2.99E-05
10^{-5}	7	0.012	149784	13200	13568	7328	34096	4271872	2.13E-04	1.44E-05
10^{-6}	7	0.012	149784	33080	41024	19712	93816	11434896	1.86E-04	3.21E-06
10^{-7}	7	0.012	149784	97184	49472	25064	171720	26173416	4.05E-05	2.15E-06
10^{-8}	8	0.012	149784	228768	167864	95688	492320	67200704	3.92E-05	2.01E-06
scale $s = 40$										
10^{-0}	3	0.024	38400	56	224	128	408	29464	3.12E-01	4.02E-02
10^{-1}	5	0.012	149784	640	2168	1136	3944	378752	6.63E-02	8.78E-03
10^{-2}	6	0.012	149784	3456	5720	2912	12088	1365880	2.03E-02	1.78E-03
10^{-3}	6	0.012	149784	9744	12368	6728	28840	3397888	1.85E-03	2.23E-04
10^{-4}	7	0.012	149784	34448	48488	20840	103776	12528624	7.85E-04	9.32E-05
10^{-5}	7	0.012	149784	60560	49472	25064	135096	18262632	6.34E-04	4.76E-05
10^{-6}	8	0.012	149784	174448	186488	90648	451584	57640040	5.87E-04	3.11E-05
scale $s = 80$										
10^{-0}	5	0.012	149784	272	824	344	1440	152896	6.70e+00	1.11E-01
10^{-1}	6	0.012	149784	2256	3152	2400	7808	811456	1.78E-01	2.01E-02
10^{-2}	7	0.012	149784	8472	12464	6776	27712	3125992	2.46E-02	2.88E-03
10^{-3}	7	0.012	149784	33824	48272	20264	102360	12382056	1.59E-02	1.28E-03
10^{-4}	8	0.006	599136	98752	154712	53832	307296	37845104	1.44E-02	6.22E-04
10^{-5}	8	0.006	599136	216808	197624	100704	515136	67967096	7.70E-03	3.16E-04

Table A.11: High-gradient adaptive tests for $k = 6$, continued from Table A.10.

A.3 Derivation of Fundamental Solutions

For the different boundary value problems seen in potential theory, we want to be able to find solutions which we can state explicitly. In the discussion of solutions, we will introduce the idea of *kernels*, specifically of the form $K(\mathbf{x}, \mathbf{y}) : D \times D \to \Re$ for some domain D. We say that K is *singular* when it performs like r^-2, $r = |\mathbf{x} - \mathbf{y}|$ as $\mathbf{x} \to \mathbf{y}$ and *weakly singular* when K looks like r^{-1}. For example, we will see that the fundamental solution to the Laplace equation in 3D, looks like $G(\mathbf{x}, \mathbf{y}) = -\frac{1}{4\pi r}$, which is weakly singular. One must be careful when using singular and weakly singular kernels to avoid "blow-up" issues when using them for solutions when $\mathbf{x} \to \mathbf{y}$; that is $r \to 0$ (this is further discussed in (Kress, 1989)).

Given some function $f(\mathbf{x}) : D \to \Re$, we define the following integral equations:

$$\int\limits_D K(\mathbf{x}, \mathbf{y})\phi(\mathbf{y})d\mathbf{y} = f(\mathbf{x}) \tag{A.1}$$

$$\phi(\mathbf{x}) + \int\limits_D K(\mathbf{x}, \mathbf{y})\phi(\mathbf{y})d\mathbf{y} = f(\mathbf{x}) \tag{A.2}$$

Equation A.1 is known as the *Fredholm integral equation of the first kind* and equation A.2 is the *Fredholm integral equation of the second kind* (Kress, 1989).

We further introduce the notation of an *integral operator*, A:

$$A\phi = \int\limits_D K(\mathbf{x}, \mathbf{y})\phi(\mathbf{y})d\mathbf{y}$$

Rewrite equations A.1 and A.2 as

$$A\phi = f \text{ and}$$

$$\phi + A\phi = (I + A)\phi = f.$$

This notation will be useful when discussing solutions to boundary value problems and the construction of boundary integrals. We now discuss how we build the kernels, fundamental to the solution of Laplace and Poisson equations.

A.3.1 Derivation of Laplace Fundamental Solution

We rewrite the Laplace equation in two dimensions in polar coordinates as

$$\Delta u = \frac{\partial^2 u}{\partial r^2} + \frac{\partial u}{r \partial r} + \frac{\partial^2 u}{r^2 \partial \theta^2} = 0. \tag{A.3}$$

Assume we are solving this equation on a circular plate of radius a with Dirichlet boundary conditions, such that we have $\Delta u = 0$ for $0 \leq r < a$ and $u(a, \theta) = g(\theta)$. Note that g is periodic in the form $g(\theta) = g(\theta + 2\pi)$. Using separation of variables, we can rewrite equation A.3 as

$$\begin{gathered} \left(\frac{\partial^R}{\partial r^2} + \frac{\partial R}{r \partial r} \right) \Theta + \frac{R \partial^2 \Theta}{r^2 \partial \theta^2}, \\ u(r, \theta) = R(r) \Theta(\theta). \end{gathered}$$

Solving the equation as in (Guenther and Lee, 1988), we solve for Θ and R and superimpose them and use the Fourier expansion of g to obtain

$$u(r, \theta) = \frac{\alpha_0}{2} + \sum_{n=1}^{\infty} \{\alpha_n \cos n\theta + \beta_n \sin n\theta\}$$

$$\alpha_n = \int_0^{2\pi} \frac{g(\gamma)\cos(n\gamma)}{\pi} d\gamma, \;\; \beta_n = \int_0^{2\pi} \frac{g(\gamma)\sin(n\gamma)}{\pi} d\gamma$$

Combine the above three equations to obtain the following formulation

$$u(r, \theta) = \int_0^{2\pi} \frac{g(\gamma)}{\pi} \left\{ \frac{1}{2} + \sum_{n=1}^{\infty} \left(\frac{r}{a} \right)^n \cos\left[n(\gamma - \theta \right] \right\} d\gamma.$$

This can be simplified into *Poisson's formula in 2D*:

$$u(r,\theta) = \int_0^{2\pi} \frac{g(\gamma)}{\pi} \left\{ \frac{a^2 - r^2}{a^2 + r^2 - 2ar\cos\gamma - \theta} \right\} d\gamma, \ |r| < a$$

In 3D, we can also formulate Poisson's formula for a similar Laplace problem:

$$\begin{aligned} \Delta u &= 0 \quad \text{if } |\mathbf{x}| < a \\ u &= g(\mathbf{x}) \quad \text{if } |\mathbf{x}| = a \end{aligned}$$

In this case, we are solving the Laplace equation on a sphere, instead of a disk. Where $|\mathbf{x}| = a$, we denote this "surface" as Γ. Then, for $\mathbf{y} \in \Gamma$, Poisson's formula takes the form:

$$u(\mathbf{x}) = \int_\Gamma \frac{a^2 - ||\mathbf{x}||^2}{4\pi a\, ||\mathbf{x} - \mathbf{y}||^3} g(\mathbf{y}) d\gamma(\mathbf{y}$$

It is actually more straightforward to construct the 3D Poisson's formula using the fundamental solution to the Laplace equation in unbounded space. In the next sections, we will consider the Laplace equation, $\Delta u = 0$ on $\Omega \subset \mathcal{R}^n$ where we have no actual physical boundaries, but we stipulate that $u \to 0$ as $\mathbf{x} \to \infty$. We will formulate a *fundamental solution* to this equation using the *dirac-delta function*, resulting in a *Green's function*. Much of this discussion follows (Guenther and Lee, 1988; Hunter and Pullan, 2002).

A.3.2 Dirac-Delta Function

Define a sequence, $\delta_n(x)$ (Hunter and Pullan, 2002) as

$$\delta_n(x) = \begin{cases} \frac{n}{2} & \text{if } |x| < \frac{1}{n} \\ 0 & \text{else.} \end{cases} \quad n = 1, 2, ...$$

Consider this to be a sequence of force distributions where, for each δ_n, $\int\limits_{-\infty}^{\infty} d_n(x)dx = 1$ is the total force over all x. As n grows, the maximum total force grows while the total force remains constant at 1. The dirac-delta function is defined as the limit of this sequence:

$$\delta(x) = \lim_{n\to\infty} \delta_n x$$

That is, in theory:

$$\delta(x) = \begin{cases} \infty & \text{if } x = 0 \\ 0 & \text{if } x \neq 0. \end{cases}$$

On its own, this is not useful, but for any $h(x) \in C^1$,

$$\int\limits_{-\infty}^{\infty} \delta(a - x)h(x)dx = h(a), a \in \mathcal{R}.$$

Another way to define the dirac-delta function is to use the Heaviside function:

$$H(b - t) = \begin{cases} 0 & \text{if } b < t \\ 1 & \text{else.} \end{cases}$$

The dirac-delta function is the slope of the Heaviside function, or $\delta(a - x) = H'(a - t)$ (Hunter and Pullan, 2002). Define the higher-dimensional dirac-delta functions in d dimensions as

$$\delta(\mathbf{a} - \mathbf{x}) = \prod_{i=0}^{d} \delta(a_i - x_i)$$

Green's Functions

In two dimensions, Φ is a fundamental solution to the Laplace equation if Φ satisfies $\Delta\Phi + \delta(a - x, b - y) = 0$ where δ is the Dirac-Delta function as described above, and there is a singularity at (a, b) (Guenther and Lee, 1988; Kress, 1989). In open-space, Φ will be symmetric about the singularity, so change to polar coordinates where r is the distance of any point from the singularity. Therefore, the Laplace equation becomes (Hunter and Pullan, 2002):

$$\Delta\Phi = \frac{\partial}{r\partial r}\left(r\frac{\partial\Phi}{\partial r}\right) + \frac{\partial^2\Phi}{r^2\partial\theta}$$

Looking at $r > 0$, and the fact that symmetry implies $\frac{\partial^2\Phi}{\partial\theta} = 0$,

$$\frac{\partial}{r\partial r}\left(r\frac{\partial\Phi}{\partial r}\right) = 0.$$

This is a 1D differential equation which can easily be integrated to find the solution

$$\Phi = C\ln r + E \text{ where } C \text{ and } E \text{ are constants} \in \mathcal{R} \tag{A.4}$$

For any domain, $\hat{\Omega} \subset \Omega$ such that $(a, b) \in \hat{\Omega}$, using the properties of the last subsection, $\int\limits_{\hat{\Omega}} \Delta\Phi dx = \int\limits_{\hat{\Omega}} \delta dx = -1$. Further, assume $\hat{\Omega}$ represents a circular disk of radius $r_{max} > 0$ as can be seen in figure A.7.

Using the divergence theorem,

$$\begin{aligned}
\int\limits_{\hat{\Omega}} \Delta\Phi dA &= \int\limits_{\partial\hat{\Omega}} \frac{\partial\Phi}{\partial n} ds \\
\Rightarrow \quad -1 &= \int\limits_{\partial\hat{\Omega}} \frac{\partial\Phi}{\partial r} ds \\
\Rightarrow \quad -1 &= 2\pi C \text{ using equation A.4} \\
\Rightarrow \quad C &= -\frac{1}{2\pi}
\end{aligned}$$

Using $C = -\frac{1}{2\pi}$, set $E = 0$ and arrive at the fundamental solution to the Laplace equation:

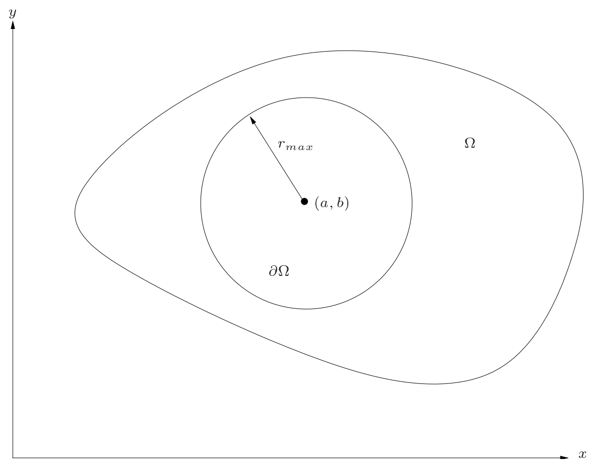


Figure A.7: $\hat{\Omega} \subset \Omega$ represents a circular disk with center at the point of singularity, (a, b) and radius of r_{max}.

$$\Phi = -\frac{1}{2\pi} \ln r \tag{A.5}$$

Equation A.5 is appropriately often referred to as the Green's or fundamental solution to the Laplace equation. In 3D, rewrite the Laplace equation in spherical coordinates as:

$$\Delta\Phi = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial\Phi}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial\Phi}{\partial\theta}\right) + \frac{1}{r^2\sin\theta^2}\frac{\partial^2\Phi}{\partial\Phi^2}.$$

A similar approach results in the fundamental solution in 3D:

$$\Phi = -\frac{1}{4\pi r} \tag{A.6}$$

Together, equations A.5 and A.6 represent the fundamental solutions to the Laplace equation in two and three dimensions, respectively. In fact, these equations, in electrostatics, simply represent the potential at some point as a result of a unit charge at another point. Hence, if $-\Delta u = f$, then we can solve for u as:

$$u(\mathbf{x}) = \int G(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') d\mathbf{x}' \tag{A.7}$$

$G(\mathbf{x}, \mathbf{x}')$ is the Green's function in 2D and 3D. The solution u can be seen in electrostatics as the potential due to some charge distribution, described by f (Chorin and Marsden, 1993). Additionally, note that the Green's functions are symmetric, i.e., $G(\mathbf{x}, \mathbf{x}') = G(\mathbf{x}', \mathbf{x})$.

A.3.3 Derivation of Stokes Fundamental Solution

The following is based on notes from (Ying, 2004). Given a point source at $\mathbf{x}_0$ of the form $\mathbf{f}(\mathbf{x}_0)\delta(\mathbf{x})$, we seek a solution $\mathbf{u}(\mathbf{x})$ for the Stokes equations,

$$\begin{aligned}
-\mu \Delta \mathbf{u} + \nabla p &= \delta_{\mathbf{x}} \mathbf{f} \\
\nabla \cdot \mathbf{u} &= 0
\end{aligned}$$

Applying the fact that in 3d, for $r = |\mathbf{y} - \mathbf{x}|$ (or $\mathbf{r} = (r_x, r_y, r_z) = |\mathbf{y} - \mathbf{x}$—), $\phi = \frac{1}{4\pi r}$ is the fundamental solution to the Laplace equation, so if we take the divergence of the first equation and use our divergence condition on $\mathbf{u}$, we obtain:

$$\begin{aligned}
-\mu \nabla \cdot \Delta \mathbf{u} + \nabla \cdot \nabla p &= \nabla \cdot (\delta_{\mathbf{x}} \mathbf{f}) \\
\frac{\partial^2 (\nabla \cdot \mathbf{u})}{\partial x^2} + \frac{\partial^2 (\nabla \cdot \mathbf{u})}{\partial y^2} + \frac{\partial^2 (\nabla \cdot \mathbf{u})}{\partial z^2} + \Delta p &= \nabla \cdot (-\Delta \phi \mathbf{f}) \\
\Delta p &= \Delta(-\nabla \phi \cdot \mathbf{f}) \\
p &= -\nabla \phi \cdot \mathbf{f}
\end{aligned}$$

Using this result, and plugging back into the Stokes equations:

$$\begin{aligned}
-\mu\Delta\mathbf{u} + \nabla(-\nabla\phi\cdot\mathbf{f}) &= 2A\delta_{\mathbf{x}}\mathbf{f} \\
\mu\Delta\mathbf{u} &= \Delta\phi\cdot\mathbf{f} - (\nabla\otimes\nabla)\phi\mathbf{f} \\
\mu\Delta\mathbf{u} &= (\Delta I - \nabla\otimes\nabla)\phi\mathbf{f} \\
\mathbf{u} &= \frac{\Delta^{-1}(\Delta I - \nabla\otimes\nabla)\phi\mathbf{f}}{\mu} \\
&= \frac{(\Delta I - \nabla\otimes\nabla)(\Delta^{-1}\phi)\mathbf{f}}{\mu}
\end{aligned}$$

We can easily check that since $\Delta\left(\frac{r}{8\pi}\right) = \frac{1}{4\pi r}$, then $\Delta^{-1}\phi = \frac{r}{8\pi}$. Plugging this back in, we can find a fundamental solution for $\mathbf{u}$ as:

$$\begin{aligned}
\mathbf{u} &= \frac{1}{8\pi\mu}\left((\Delta r)I - (\nabla\otimes\nabla)r\right)\mathbf{f} \\
&= \frac{1}{8\pi\mu}\left(\left(\frac{\partial}{\partial x}\left(\frac{\partial r}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial r}{\partial y}\right) + \frac{\partial}{\partial z}\left(\frac{\partial r}{\partial z}\right)\right)I - (\nabla\otimes\nabla)r\right)\mathbf{f} \\
&= \frac{1}{8\pi\mu}\left(\left(\frac{\partial}{\partial x}\frac{x}{r} + \frac{\partial}{\partial y}\frac{y}{r} + \frac{\partial}{\partial z}\frac{z}{r}\right)I - (\nabla\otimes\nabla)r\right)\mathbf{f} \\
&= \frac{1}{8\pi\mu}\left(\left(\frac{3}{r} - \frac{x^2+y^2+z^2}{r^3}\right)I - (\nabla\otimes\nabla)r\right)\mathbf{f} \\
&= \frac{1}{8\pi\mu}\left(\frac{2}{r}I - \left[\frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}\right]r\right)\mathbf{f},\ i,j = 1...3 \\
&= \frac{1}{8\pi\mu}\left(\frac{1}{r}I + \frac{\mathbf{r}\otimes\mathbf{r}}{r^3}\right)\mathbf{f}
\end{aligned}$$

We call the above the fundamental solution, K_s^u, to the velocity component of the Stokes equations due to a point force. For a distributed body force, $\mathbf{F}(\mathbf{x})$, for which $\mathbf{F}\to 0$ as $\mathbf{x}\to\infty$, we construct the volume integral for the solution at all points $\mathbf{x}$ as:

$$\mathbf{u}(\mathbf{x}) = \int K_s^u(\mathbf{x}-\mathbf{y})\mathbf{F}(\mathbf{y})d\mathbf{y}$$

In order to recover the pressure term of the Stokes equations, we simply recall that $p = -\nabla\phi \cdot \mathbf{f}$, allowing us to construct a fundamental solution, K_s^p for the pressure component of the Stokes equations:

$$\begin{aligned} K_s^p &= (-\nabla\phi) \cdot \mathbf{f} \\ &= \frac{1}{4\pi r^3} \left[r_x, r_y, r_z\right]^T \cdot \mathbf{f} \\ &= \frac{\mathbf{r} \cdot \mathbf{f}}{4\pi r^3} \end{aligned}$$

Hence, for our distributed body force $\mathbf{F}(\mathbf{x})$ which decays to zero at infinity, we recover the pressure for the Stokes equations as:

$$p(\mathbf{x}) = \int K_s^p(\mathbf{x} - \mathbf{y})\mathbf{F}(\mathbf{y}) d\mathbf{y}$$

We turn to discussing more of the necessary theorems of potential theory which provide theoretical background for the problems we address.

A.3.4 Properties of Harmonic Functions

Harmonic functions are twice-differentiable real functions, which are solutions to the Laplace equation, and there are several important properties and theorems surrounding them, necessary for uniqueness and existence of solutions to different boundary value problems. We discuss these here, beginning with Green's integral theorems and Green's formula. Much of this discussion is derived from (Kress, 1989).

Green's Integral Theorems and Green's Formula

We begin by stating two theorems from (Kress, 1989) which we be necessary to the formulation of solutions to the boundary value problems of potential theory.

Theorem A.1. *(***Green's 1st, 2nd and 3rd Theorems/Identities***) For a bounded domain Ω with boundary Γ and outward unit normal to the boundary, $\mathbf{n}$ and for all vectors $\Psi_i \in C^1(\bar{\Omega})$, $\Psi_j \in C^2(\bar{\Omega})$, $\bar{\Omega} = \Omega \cup \Gamma$, we have Green's First Theorem:*

$$\int\limits_{\Omega} \left[\Psi_i \Delta \Psi_j + (\nabla \Psi_i) \cdot (\nabla \Psi_j)\right] d\mathbf{x} = \int\limits_{\Gamma} \Psi_i \frac{\partial \Psi_j}{\partial \mathbf{n}} ds$$

where ds is a segment of the boundary Γ. Stipulate $\Psi_i, \Psi_j \in C^2(\bar{\Omega})$. Then Green's Second Theorem is

$$\int\limits_{\Omega} \left[\Psi_i \Delta \Psi_j - \Psi_j \Delta \Psi_i\right] d\mathbf{x} = \int\limits_{\Gamma} \left[\Psi_i \frac{\partial \Psi_j}{\partial \mathbf{n}} - \Psi_j \frac{\partial \Psi_i}{\partial \mathbf{n}}\right] ds$$

Green's Third Theorem is a corollary, following from above for a harmonic function, $\Psi_j \in C^2(\bar{\Omega})$.

$$\int\limits_{\Gamma} \frac{\partial \Psi_i}{\partial \mathbf{n}} = 0$$

We prove the first theorem using the divergence theorem,

$$\int\limits_{\Omega} \nabla \cdot A d\mathbf{x} = \int\limits_{\Gamma} A \cdot \mathbf{n} ds.$$

Let $A = \Psi_i \nabla \Psi_j \in C^1(\bar{\Omega})$ such that $\nabla \cdot A = \nabla \Psi_i \cdot \nabla \Psi_j + \Psi_i \Delta \Psi_j$ and $A \cdot \mathbf{n} = \Psi_i \frac{\partial \Psi_j}{\partial \mathbf{n}}$. This proves Green's First Theorem. To prove Green's Second Theorem, let $B = \Psi_j \nabla \Psi_i \in C^1(\bar{\Omega})$. Computing $\int\limits_{\Omega} (\nabla \cdot A - \nabla \cdot B)\, d\mathbf{x} = \int\limits_{\Gamma} (A \cdot \mathbf{n} - B \cdot \mathbf{n})\, ds$ proves Green's Second Theorem. For Green's Third Theorem, choose $\Psi_i = 1 \in C^1(\bar{\Omega})$ and let $\Psi_j \in C^2(\bar{\Omega})$ be a harmonic function and apply these to Green's First Theorem, thus proving Green's Third Theorem.

From Green's Theorems follows Green's formula. Let $u \in C^2(\bar{\Omega})$ as above where u is harmonic in Ω. Then,

$$u(\mathbf{x}) = \int_{\Gamma} \{u_n(\mathbf{y})G(\mathbf{x},\mathbf{y}) - u(\mathbf{y})G_n(\mathbf{x},\mathbf{y})\}\, d\gamma(\mathbf{y}),\ x \in \Omega \tag{A.8}$$

In equation A.8, let $u_n = \frac{\partial u}{\partial \mathbf{n}}$ and similarly for G_n, where G represents the fundamental solution kernel or Green's function in 2D or 3D. G is referred to as a *single-layer kernel* while G_n is a *double-layer kernel*. Green's formula says harmonic functions can be represented as a convolution with a single-layer kernel, double-layer kernel, or combination of both. A potential, u derived from a single-layer kernel is referred to as a *single-layer potential* and similarly for a double-layer kernel. We look at these potentials in more depth once we introduce the boundary value problems in the next section. For a proof of Green's formula, we refer to (Kress, 1989).

For a function, $u \in C^2(\Omega) \cap C^1(\bar{\Omega})$, where u is not harmonic, it can be shown (Guenther and Lee, 1988) that u is a combination of single-layer, double-layer and volume potentials:

$$u(\mathbf{x}) = \int_{\Gamma} \{u_n(\mathbf{y})G(\mathbf{x},\mathbf{y}) - u(\mathbf{y})G_n(\mathbf{x},\mathbf{y})\}\, d\gamma(\mathbf{y}) - \int_{\Omega} G(\mathbf{x},\mathbf{y})\Delta u d\gamma(\mathbf{y}) \tag{A.9}$$

If we force u to be harmonic, then equation A.9 becomes equation A.8, and if we remove the boundaries, and make the problem unbounded, then equation A.9 becomes the volume integral for the Poisson equation in open space ($u \to 0$ as $\mathbf{x} \to \infty$).

Green's Theorems and Green's formula have several consequences in terms of analysis, which are necessary for understanding the existence and uniqueness for the boundary value problems we will present in the next section.

Mean-Value Theorem and Max-Min Principle

One of the first consequences of Green's formula is the fact that harmonic functions are also analytic functions (Ahlfors, 1979; Kress, 1989). For such functions, we can establish a Mean-Value Theorem. If we solve for a harmonic function on a disk or a ball, then the potential at the center of the domain will be the mean value over its domain and over its boundary.

Theorem A.2. *Let u be a harmonic function in the domain $\Omega_\rho = \{\mathbf{x} \in \mathcal{R}^m| \, ||x|| < \rho\}$, centered at the origin with boundary Γ_ρ. If u is continuous in the closure, $\bar{\Omega}_\rho$, then in 2D and 3D, we have the following respective mean value results:*

$$u(\mathbf{x}) = \frac{1}{\pi\rho^2}\int\limits_{\bar{\Omega}_\rho} u(\mathbf{y}d(\mathbf{y}) = \frac{1}{2\pi\rho}\int\limits_{\Gamma_\rho} u(\mathbf{y}d\gamma(\mathbf{y})$$

$$u(\mathbf{x}) = \frac{3}{4\pi\rho^3}\int\limits_{\bar{\Omega}_\rho} u(\mathbf{y}d(\mathbf{y}) = \frac{1}{4\pi\rho^2}\int\limits_{\Gamma_\rho} u(\mathbf{y}d\gamma(\mathbf{y})$$

The Mean-Value Theorem (MVT) for the boundary result can be achieved from Green's Third Theorem and Green's Formula. For the domain result it is a consequence of integration (Guenther and Lee, 1988; Kress, 1989). We state the Maximum-Minimum Principle for a general function, $u \in c^2(\Omega) \cap C^1(\bar{\Omega})$.

Theorem A.3. *If Ω is bounded with boundary Γ, $u \in C^2(\Omega) \cap C^1(\bar{\Omega})$, one of three results is possible.*

1. *If $\Delta u \geq 0$, $u(\mathbf{x}) \leq max_y u(\mathbf{y})$, $\mathbf{x} \in \Omega$, $\mathbf{y} \in \Gamma$.*
2. *If $\Delta u \leq 0$, $u(\mathbf{x}) \geq min_y u(\mathbf{y})$, $\mathbf{x} \in \Omega$, $\mathbf{y} \in \Gamma$.*
3. *If $\Delta u = 0$, $min_y u(\mathbf{y}) \geq u(\mathbf{x}) \leq max_y u(\mathbf{y})$, $\mathbf{x} \in \Omega$, $\mathbf{y} \in \Gamma$.*

For existence and uniqueness properties of boundary value problems, the Maximum-Minimum Principle will be necessary, and we leave the proof to (Guenther and Lee, 1988; Kress, 1989), based largely on the Mean-Value Theorem. In particular, for a harmonic function u with domain Ω and boundary Γ, the Maximum-Minimum Principle implies that if u attains its maximum or minimum value in its domain, u is constant over its domain. Further, if u is continuous on $\bar{\Omega}$, the maximum and minimum values must be reached on Γ.

For functions satisfying the Laplace equation, the Maximum-Minimum Principle is sensible for the following reasons. As stated earlier, the Laplace equation models the steady-state temperature of a body; hence, there are no sources or sinks in the internal domain, since otherwise, the temperature distribution would not be steady. Therefore, the extreme values must be on the boundaries, or the values are constant throughout.

A.3.5 Boundary Value Problems and Surface Potentials

We have introduced some of the theory surrounding integral equations when discussing fundamental solutions. We look at different boundary value formulations for the Poisson and Laplace equations and different surface potentials for bounded domains.

Boundary Value Problems and Uniqueness

As discussed, solutions to the Laplace equation in open space are harmonic and have fundamental solutions, $G(\mathbf{x}, \mathbf{y})$ in equations A.5 and A.6 for 2D and 3D respectively. Assume now we have a *bounded* domain $\Omega \subset \mathcal{R}^3$ with a sufficiently smooth boundary Γ. Let $\bar{\Omega} = \Omega \cup \Gamma$ and $\Omega^c = \mathcal{R}^3 \backslash \bar{\Omega}$. Further, let $\mathbf{n}$ be the normal at Γ such that $\frac{\partial}{\partial \mathbf{n}}$ indicates differentiation in the direction of the normal. We are concerned with *boundary value problems*, and we formalize the four types for integral equations and their notation.

Interior Dirichlet Problem:

Find $u \in \Omega \cap \Gamma$ where $u = g$ on Γ, or $u|_\Gamma = g$, $g \in C(\Omega)$.

Interior Neumann Problem:

Find $u \in \Omega \cap \Gamma$ where $u_n = g$ on Γ, or $\frac{\partial u}{\partial \mathbf{n}}|_\Gamma = g$, $g \in C(\Omega)$.

Exterior Dirichlet Problem:

Find $u \in \Omega^c \cap \Gamma$ where $u = g$ on Γ, or $u|_\Gamma = g$, $g \in C(\Omega^c)$.

Exterior Neumann Problem:

Find $u \in \Omega^c \cap \Gamma$ where $u_n = g$ on Γ, or $\frac{\partial u}{\partial \mathbf{n}}|_\Gamma = g$, $g \in C(\Omega^c)$.

We present theorems for uniqueness properties of these boundary value problems.

Theorem A.4. *There exists at most one solution to the interior Dirichlet, exterior Dirichlet and exterior Neumann problems. Multiple solutions to the interior Neumann problem exist, but differ at most by a constant.*

The proofs of these uniqueness properties are a result of the Maximum-Minimum Principle (Kress, 1989).

Surface Potentials and Existence of Boundary Value Problem Solutions

Away from the boundary, evaluating equation A.8 presents few problems. However, for a point, $\mathbf{x} \to \Gamma$, problems arise. Several theorems concerning potentials at surfaces tell us how to proceed (Kress, 1989), and we discuss them here from (Kress, 1989).

Theorem A.5. *If Γ is twice-differentiable and the density function ϕ is continuous there, the single-layer potential u is continuous such that on Γ*

$$u(\mathbf{x}) = \int\limits_\Gamma u(\mathbf{y}) G(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}),\ \ x \in \Gamma$$

$$\frac{\partial u\pm}{\partial \mathbf{n}}(\mathbf{x}) = \mp\frac{\phi(\mathbf{x})}{2} + \int\limits_\Gamma \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}),\ \ x \in \Gamma$$

where $\frac{\partial u\pm}{\partial \mathbf{n}}$ is the limit as we approach Γ from outside (+) or inside (-) our domain Ω. Additionally, for a double-layer potential, v, we have away from Γ and near Γ

$$v(\mathbf{x}) = \int\limits_{\Gamma} v(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}),\ x \in \Omega \backslash \Gamma,$$

$$v \pm (\mathbf{x}) = \pm \frac{\phi(\mathbf{x})}{2} + \int\limits_{\Gamma} \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}),\ x \in \Gamma$$

As we take the limit from both side of Γ, *indicate potential jump as* $[[v]]$ *such that it is the limit in the potential taken from both sides. Hence, we can see that* $[[v(\mathbf{x})_n]] \to \left[\frac{\phi(\mathbf{x})}{2} - \left(-\frac{\phi(\mathbf{x})}{2}\right)\right] = \phi(\mathbf{x})$, $\mathbf{x} \in \Gamma$.

Finally, we have the relationship:

$$[[v_n]]\,(\mathbf{x}) \to 0, \mathbf{x} \in \Gamma$$

This theorem is the basis for the jump relations necessary for constructing smooth solutions, especially numerically. The proof details are available in the literature such as (Kellogg, 1967).

Finally, we comment on the existence of solutions to the various boundary value problems by using theorems from (Kress, 1989). The proofs are left to (Kellogg, 1967; Kress, 1989), using the same notation for density ϕ, domains Ω, Ω^c and boundary Γ. For each boundary value problem, we give the solution type and the form of the integral equation for ϕ.

Theorem A.6. *The following double-layer potential is a solution to the* **interior Dirichlet** *problem*

$$u(\mathbf{x}) = \int\limits_{\Gamma} \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}),\ x \in \Omega$$

$$-2g(\mathbf{x}) = \phi(\mathbf{x}) - 2 \int\limits_{\Gamma} \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}),\ x \in \Gamma.$$

The following modified double-layer potential is a solution to the **exterior Dirichlet** *problem*

$$u(\mathbf{x}) = \int\limits_{\Gamma} \left\{ \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) + \frac{1}{r^{d-2}} \right\} d\gamma(\mathbf{y}), \;\; x \in \Omega^c \subset \mathcal{R}^d, \, d = 2, 3$$

$$2g(\mathbf{x}) = \phi(\mathbf{x}) + 2 \int\limits_{\Gamma} \left\{ \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) + \frac{1}{r^{d-2}} \right\} d\gamma(\mathbf{y}), \;\; x \in \Gamma \subset \mathcal{R}^d, \, d = 2, 3.$$

Solutions to the interior and exterior Dirichlet problems are unique.

The following single-layer potential is a solution to the **interior Neumann** *problem*

$$u(\mathbf{x}) = \int\limits_{\Gamma} \phi(\mathbf{y}) G(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}), \;\; x \in \Omega$$

$$2g(\mathbf{x}) = \phi(\mathbf{x}) + 2 \int\limits_{\Gamma} \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}), \;\; x \in \Gamma$$

The solution is unique when

$$\int\limits_{\Gamma} g(\mathbf{y}) d\gamma(\mathbf{y}) = 0.$$

The following single-layer potential is a solution to the **exterior Neumann** *problem*

$$u(\mathbf{x}) = \int\limits_{\Gamma} \phi(\mathbf{y}) G(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}), \;\; x \in \Omega^c$$

$$-2g(\mathbf{x}) = \phi(\mathbf{x}) - 2 \int\limits_{\Gamma} \phi(\mathbf{y}) G_n(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{y}), \;\; x \in \Gamma,$$

where in $\mathcal{R}^2$

$$\int\limits_{\Gamma} \phi(\mathbf{y}) d\gamma(\mathbf{y}) = 0.$$

Further, the solution is unique when

$$\int_{\Gamma} g(\mathbf{y}) d\gamma(\mathbf{y}) = 0.$$

For all four boundary value problem types, we are presented with either a general Fredholm integral equation of the first kind $A\phi = g$ or second kind $(I + A)\phi = g$. As would be expected, approximating solutions to these integral equations is of major concern. We will briefly discuss several approaches to solving integral equations of these sorts.

A.3.6 Solving Integral Equations

When discretizing an integral equation, like the ones seen in the last section for the Fredholm equations, there are two main methods for discretizing the integral equation. The first method is known as a *quadrature method* or *Nyström method*. The second type is a *projection method* such as the *collocation method* and *Galerkin method*. We begin by discussing quadrature (Nyström) methods and then move onto projection methods. There are many references available on the theory of integral equations including (Mikhlin, 1964; Kress, 1989; Hackbusch, 1995).

For smooth 2D problems, the Nyström method is straightforward. For piecewise smooth 2D problems and 3D problems in general, the picture becomes more complicated and we discuss numerical considerations when using the Galerkin method. In Chapter 3, we discuss a current method for 3D smooth problems using the Nyström methods.

Nyström Method

The idea of numerical quadrature is commonly seen when approximating integrals over some domain, D ($\subset \mathcal{R}^d$, $d = 1, 2, 3$) which include a weight function $w(x)$ and a function $f(x)$. For example, consider the integral

$$\int_D w(x)g(x)dx$$

We can do a discrete approximation of this integral by a quadrature method of index n using quadrature points $x_1, x_2, ..., x_n$ and quadrature weights $\alpha_1, \alpha_2, ..., \alpha_n$ of the form:

$$\sum_{j=1}^{n} \alpha_j g(x_j)$$

This rule dictates that the value of g at discrete points x_j is known, or we are able to somehow obtain appropriate approximations to g at these points. A basic example seen in most introductory texts is a trapezoidal rule approximation to the integral of a function $g(x)$, $x \in \mathcal{R}$ over some domain $[a, b] \in \mathcal{R}$ (i.e., $g(x) = \sin x$ over $[0, \pi]$). If we are trying to approximate this function, assuming that g is smooth, we could do so by discretizing $[a, b]$ into n discrete equidistant points where $x_{j+1} - x_j = h$ and letting our weights $\alpha_j = h$ at interior points and $\alpha_0 = \frac{h}{2}$ at $x_0 = a$ and $\alpha_{n+1} = \frac{h}{2}$ at $x_{n+1} = b$. Then our approximation would look like:

$$\begin{aligned} \int_a^b g(x)dx &\approx \sum_{j=0}^{n} \frac{1}{2}\left[g(x_j) + g(x_{j+1})\right] \\ &= \frac{h}{2}g(x_0) + hg(x_1) + hg(x_2) + ... + hg(x_{n-1}) + hg(x_n) + \frac{h}{2}g(x_{n+1}) \end{aligned}$$

If g has many zeros over $[a, b]$ or periods of high oscillation, equally spaced discretization points may lead to a poor solution, or we may need n to be so large, that it is preferable to choose different quadrature points, spaced unevenly. Other rules exist, incorporating different weights and quadrature points, such as Simpson's rules, based on Newton-Cotes formulas for numerical integration (Iserles, 1996; Trefethen and Bau, 1997).

For an integral operator of the first kind, $K\phi(\mathbf{x}) = \int_D K(\mathbf{x}, \mathbf{y})\phi(\mathbf{y})d\mathbf{y}$, $\mathbf{x} \in D$, Nyström's method consists of approximating this as follows:

$$K\phi(\mathbf{x}_i) \approx \sum_{j=1, j\neq i}^{n} \alpha_j K(\mathbf{x}, \mathbf{x}_j)\phi(\mathbf{x}_j) \tag{A.10}$$

Equation A.10 approximates the Fredholm integral equation of the first kind. The integral equation of the second kind can be approximated by

$$(I + K)\phi(\mathbf{x}_i) \approx \phi(\mathbf{x}_i) + \sum_{j=1}^{n} \alpha_j K(\mathbf{x}, \mathbf{x_j})\phi(\mathbf{x_j}) \tag{A.11}$$

The trapezoidal rule can also be used for Nyström's method in 2D, and there are many quadrature rules available (Ying, 2004). These allow for building a system of linear equations, solvable by a variety of methods. Since the resulting matrix can often be dense, fast methods are preferable. For example, (Ying et al., 2004b) use a GMRES solver with a preconditioner, described in (Greengard et al., 1996); GMRES and various other approach for solving such systems are discussed in section A.5

Convergence results for Nyström's are available in (Kress, 1999). For example, for a function $g \in C^2(\Omega)$, the trapezoidal rule for the numerical quadrature is $O(h^2)$ for equidistant discretizations of points. Similarly, Simpson's rule has a convergence rate of $O(h^4)$. In general, let the error in the quadrature from a trapezoidal rule be denoted as $R_T(g)$ for function g. If g is a periodic function, then $|R_T(g)| = O(e^{-2n})$, or the result is an exponential decrease in error. In general, for a function $g \in C^m(\Omega)$, which is 2π periodic, (Kress, 1999) provides the following error estimate:

$$|R_T(g)| \leq \frac{C}{n^m} \left|\left| g^{(m)} \right|\right|_\infty = O(n^{-m}).$$

For 3D, quadrature rules for the types of integral equations we are interested in can be quite complicated, especially if higher-order is desired. Further, if dealing with surfaces which have high curvature in areas, choosing appropriate quadrature points can be tricky to avoid not having to use an excessive number of points, resulting in high numerical costs. Additional difficulties occur when surfaces are no longer smooth and the function defined by the surface is not continuously differentiable, as required by the above error estimates. Another available approach is to use projection methods.

Projection Methods

The idea behind projection methods is to project the linear operators seen in integral equations into subspaces. Since one will be performing numerical calculations on these projections, the subspaces must be of finite dimension (Kress, 1989). To give a full definition, we first define *Banach spaces*, which in turn, requires several other definitions. Further, if a sequence $\{\psi_n\} \in X$ where X is a normed space (normed spaces are discussed further in the Appendix). We say that $\{\psi_\epsilon\}$ is a *Cauchy sequence* if it obeys the definition of sequence with a limit. A sequence is Cauchy if

$$\forall \epsilon > 0, \exists N \text{ such that } \forall n, m \geq N \text{ then } ||\psi_n - \psi_m|| < \epsilon.$$

If $U \subset X$, then if every Cauchy sequence in U converges to an element in U and U is *complete*. If the normed space X is itself complete, then we say that it is also a *Banach* space. If X and Y are two Banach spaces where the linear integral operator, $K : X \to Y$, let $X_n \subset X$, $Y_n \subset Y$ be two finite dimensional subspaces of dimension n and let P_n be a projection operator that projects $Y \to Y_n$ and $X \to X_n$ for example.

For the integral equation $K\phi = g$, we approximate with the projected version of this equation,

$$P_n K \phi_n = P_n g.$$

For an integral equation of the second kind $(I + K)\phi = g$, $K : X \to X$, the projection operators acts as $P_n : X \to X_n$. The projection method approximates the integral equation with a finite dimensional equation of the form:

$$(I + P_n K)\phi_n = P_n g.$$

Discussions of the convergence properties for projection methods are available in (Kress, 1989). The two main projection methods are the collocation methods and the Galerkin method. The Galerkin method is discussed in detail for finite element methods in section A.4. For general descriptions of collocation and Galerkin we refer to (Baker, 1977; Kress, 1989; Sloan, 1992; Hackbusch, 1995; Atkinson, 1997). For further discussions in 3D, specifically for boundary elements, we will refer to (Brebbia et al., 1984; Hackbusch, 1995; Chen and Zhou, 1992; Wendland, 1985).

Collocation Method

For an integral equation of the first kind, $K\phi = g$, the collocation method consists of requiring that the residual, $r = K\phi - g$ vanishes at a set of *collocation points*, $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ such that $\{\mathbf{x}_i\} \in X_n$ by approximating the equation with an element $\phi_n \in X_n$:

$$(K\phi_n)(\mathbf{x}_j) = g(\mathbf{x}_j), j = 1, ..., n.$$

We define the span of X_n as a set of n linearly independent vectors $\{\mu_1, ...\mu_2\}$. Then, we define ϕ_n as a linear combination of these vectors, using real coefficients β_j, $j = 1, ...n$,

$$\phi_n = \sum_{j=1}^{n} \beta_j \mu_j.$$

Our integral equation becomes

$$\sum_{k=1}^{n} \beta_k (K\mu_k)(\mathbf{x}_j) = g(\mathbf{x}_j), j = 1, ..., n.$$

Therefore, we are trying to solve for the μ_j to build ϕ_n. One of the keys to success in this type of method is in the choice of collocation points and appropriate coefficients. More information is available in (Kress, 1989).

Finally, we state the collocation formulation for an integral equation of the second kind: $(I + K)\phi = g$:

$$\sum_{k=1}^{n} \beta_k \left\{\mu_k(\mathbf{x}_j) + (K\mu_k)(\mathbf{x}_j)\right\} = g(\mathbf{x}_j), j = 1, ..., n.$$

Again, examples and numerical results are available in (Kress, 1989).

Galerkin Method

We discuss much of the Galerkin method in section A.4, so we refer there for general Galerkin method development and discussion of Hilbert spaces and discuss how the Galerkin method is applied here, in much the same way.

For the integral equation of the first kind, $K\phi = g$, we require that the residual, $r = K\phi - g$ is orthogonal to a finite dimensional space $\Psi_n = span\{\psi_1, ...\psi_n\}$. We define for $f, g \in H_n$ for some finite dimensional Hilbert space of size n over some domain Ω:

$$(f, g) = \int_{\Omega} fg d\mathbf{x} \text{ for } f, g \in H_n(\Omega).$$

Solving the integral equation for a minimized residual is equivalent to solving the following equation for ϕ_n,

$$(K\phi_n, \psi) = (g, \psi), \ \forall \psi \in \Psi.$$

The resulting linear system (again writing ϕ_j as a linear combination of linearly independent vectors μ_j and coefficients β_j) is

$$\sum_{k=1}^{n} \beta_k (K\mu_k, \psi_j) = (g, \psi_j), j = 1, ..., n.$$

For an integral equation of the second kind $(I + K)\psi = g$, the resulting system of linear equations is

$$\sum_{k=1}^{n} \beta_k \left\{(\mu_k, \psi_j) + (K\mu_k, \psi_j)\right\} = (g, \psi_j), j = 1, ..., n.$$

As discussed for finite elements in section A.4, appropriately choosing the evaluation points, weights, and basis functions play a large role in the development of an accurate finite element method. In fact, much of the research surrounding Galerkin methods involves designing appropriate bases as seen in (Baker, 1977; Brebbia et al., 1984; Wendland, 1985; Kress, 1989; Chen and Zhou, 1992; Sloan, 1992; Hackbusch, 1995; Atkinson, 1997; Braess, 2001).

A.4 Review of Finite Elements

In section 1.1 and (Iserles, 1996), it is shown how to easily discretize an equation such as the Poisson equation in a regular domain using finite difference techniques in multiple dimensions. These schemes work best when we can discretize our domain in a regular way and easiest to do when the domain is a simple shape (rectangular, circular, etc.). However, more complicated domains (L-shaped domains, irregular geometries, etc.) cannot be so easily discretized. For this, we begin by looking at finite elements. Finite element methods work well for more complicated domains, but also work for regular domains. For these reasons, this method is quite popular in general, and we will begin our discussion of the basic topics with a 1D problem. Much of the discussion here is derived directly from (Iserles, 1996; Braess, 2001), condensed or expanded where deemed necessary.

A.4.1 Finite Element Motivation in 1D

Consider the following 1D Poisson equation with Dirichlet boundary conditions:

$$
\begin{aligned}
-u''(x) = f(x), 0 < x < L \qquad \text{(A.12)}\\
u(0) = u(L) = 0.
\end{aligned}
$$

For piece-wise continuous real bounded functions, we introduce an inner-product, linear space, V, and linear functional, $F : V \to \mathcal{R}$, as follows:

$$
(v, u) = \int_0^1 v(x)u(x)dx, \qquad \text{(A.13)}
$$

$$
V = \{\, v \mid v \text{ is continuous on } [0,1]\,, v' \text{ piecewise continuous and bounded on } [0,1]\,, v(0) = v(1) = 0 \,\},
$$

$$
F(v) = \frac{(v', v') - (f, v)}{2}.
$$

We define two problems, a *minimization* problem and a *variational* problem, respectively,

$$\forall v \in V, \text{find } u \in V \mid F(u) \leq F(v) \quad \text{and} \tag{A.14}$$

$$\forall v \in V, \text{find } u \in V \mid (u', v') = (f, v) \tag{A.15}$$

The solution to equation A.12 is also a solution to equations A.14 and A.15. To see the relationship between equations A.12 and A.15, first multiply equation A.12 by some function $v \in V$ and integrate over $(0, 1)$, resulting in the use of the inner-product formulation (equation A.13): $-(u'', v) = (f, v)$. Integration by parts, using the boundary conditions shows that $-(u'', v) = (u', v')$. Hence, if u solves equation A.12, then it solves equation A.15.

Now, assume that u is a solution to the variational problem, equation A.15. We arbitrarily choose some $v \in V$ and let $w = v - u$, $w \in V$ such that

$$\begin{aligned}
F(v) &= F(u + w) \\
&= \frac{1}{2}(u' + w', u' + w') - (f, u + w) \\
&= \frac{1}{2}(u', u') - (f, u) + (u', w') - (f, w) + \frac{1}{2}(w', w') \\
&= \frac{1}{2}(u', u') - (f, u) + \frac{1}{2}(w', w') \text{ since } u \text{ solves equation A.15} \Rightarrow (u', w') = (f'w) \\
&\geq F(u) \text{ since } (w', w') \geq 0.
\end{aligned}$$

Hence, if u is a solution to the Poisson equation A.12, it is a solution to the variational problem, A.15, which, in turn, implies that it is a solution to the minimization problem, equation A.14. Similarly, we can show that if u solves the minimization problem, then it solves the variational problem, and if it solves the variational problem, then it solves the Poisson equation. The details of this are further explained in (Johnson, 1987).

Knowing that the Poisson equation problem is equivalent to the variational and minimization problem, we will show how we explicitly build the set of functions in v. We seek to construct $V_h \subset V$. As with the finite-difference methods, partition the interval $[0, L]$:

$$0 = x_0 \leq x_1 \leq x_2 \leq x_3 ... \leq x_{N-1} \leq x_N \leq x_{N+1} = L.$$

Let $h_i = x_i - x_{i-1}$ for $1 \leq i \leq N+1$ and $h = \max_i h_i$ and define V_h as the set of all functions which are linear on each subinterval, (x_{i-1}, x_i), continuous on $[0, 1]$ and equal to zero at $x = 0$ and $x = 1$. This is very similar to our previous definition for the set of functions, V, but now we are concerned about the subintervals, and how each function, v is linear there. An example function from V_h can be seen in Figure A.8. Notice that in this figure, subintervals do not have to be of equal length. That is h_i and h_j are not necessarily equal for $i \neq j$.

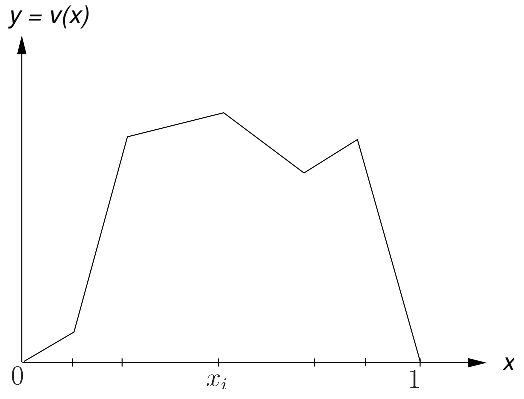


Figure A.8: Example of the type of piece-wise continuous function, $v \in V_h$, which is linear on each subinterval, (x_{i-1}, x_i). The y-axis represents the value of $v(x_i)$ at each point x_i.

Since each function, v, is linear on each subinterval, we define variables $\alpha_i = v(x_i)$. To that end, define the following *basis functions*, $\phi_j \in V_h$:

$$\phi_j(x_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j. \end{cases} \quad i, j = 1, 2, ...N$$

Figure A.9 shows explicitly how the basis functions look over two subintervals. In general, ϕ is referred to as the *hat function*. Additionally, since ϕ_i is linear over any subinterval, (x_{i-1}, x_i), we can evaluate the derivative of ϕ_i, or $\phi_i' = \frac{d\phi_i}{dx}$ at some point x as

$$\phi_i'(x) = \begin{cases} \frac{1-0}{x_i - x_{i-1}} = \frac{1}{h_i} & \text{if } x_{i-1} < x < x_i \\ \frac{0-1}{x_i - x_{i+1}} = \frac{1}{h_{i+1}} & \text{if } x_i < x < x_{i+1}. \end{cases}$$

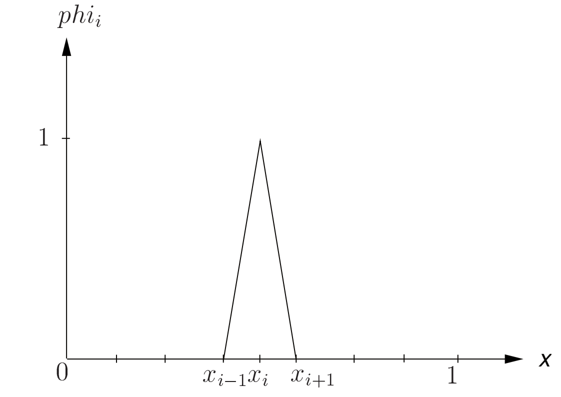


Figure A.9: The basis function, ϕ_i, as defined.

Using these basis functions, we represent any function $v \in V_h$ as a linear combination of the values of $\alpha_i = v(x_i)$ at every point x_i and the basis functions, ϕ_i for all $i = 1, ..., N$:

$$v(x) = \Sigma_{i=1}^{N} \alpha_i \phi_i(x), x \in [0, L].$$

The minimization problem in equation A.14 and the variational problem in equation A.15 can now be written respectively as

$$\forall v \in V_h, \text{find } u \in V_h \mid F(u) \leq F(v) \quad \text{and} \tag{A.16}$$

$$\forall v \in V_h, \text{find } u_h \in V \mid (u', v') = (f, v). \tag{A.17}$$

Methods for solving the minimization problem A.16 are known as *Ritz* methods, and methods for solving the variational problem A.17 are known as *Galerkin* methods. Either approach can be used to construct a finite element approximation to the Poisson equation A.12. In particular, if $u_h \in V_h$ is a solution to problem A.17, then $u_h(x) = \Sigma_{i=1}^{N} \alpha_i \phi_i(x)$ where $\alpha_i = u_h(x_i)$, then the Galerkin method implies that $\Sigma_{i=1}^{N} \alpha_i (\phi_i', \phi_j') = (f, \phi_j)$ where j goes from $1...N$. As both methods are essentially equivalent, most literature refers simply to the Galerkin method, typically the preferred finite element method.

Let $b_j = (f, \phi_j)$, $A_{i,j} = (\phi_i', \phi_j')$, and $\alpha = [\alpha_1 \alpha_2 .... \alpha_N]^T$. We rewrite the variational problem as a system of N linear equations in matrix-vector form $A\alpha = b$ where the entries of the matrix A are the $A_{i,j}$ values. We note that $A_{i,j} = 0$ if x_i and x_j are more than one interval away from each other; i.e., $|i - j| > 1$. Therefore, A is a tridiagonal matrix, where only the entries in the main diagonal and one entry away are non-zero. Also, by the definition of our inner-product, $(\phi_i', \phi_{i-1}) = (\phi_{i-1}', \phi_i) \Rightarrow A_{i,i-1} = A_{i-1,i}$. Evaluate the entries of A as follows:

$$\begin{aligned}
A_{i,i} &= \int_0^1 \phi_i' \phi_i' dx = \int_{x_{i-1}}^{x_{i+1}} \phi_i' \phi_i' dx = \int_{x_{i-1}}^{x_i} \frac{1}{h_i} \frac{1}{h_i} dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_{i+1}} \frac{1}{h_{i+1}} dx \\
&= \int_{x_{i-1}}^{x_i} \frac{1}{h_i^2} dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_{i+1}^2} dx \\
&= \frac{1}{h_i} + \frac{1}{h_{i+1}} \quad \text{and}
\end{aligned}$$

$$\begin{aligned} A_{i,i-1} = A_{i-1,i} &= \int_0^1 \phi_i'\phi_{i-1}' dx = \int_{x_{i-1}}^{x_i} \phi_i'\phi_{i-1}' dx = \int_{x_{i-1}}^{x_i} \frac{1}{h_i}\frac{1}{h_i} dx \\ &= \int_{x_{i-1}}^{x_i} \frac{1}{h_i^2} dx = -\frac{1}{h_i}. \end{aligned}$$

A is commonly referred to as the *stiffness* matrix, and it is symmetric and positive definite ($A_{i,j} = A_{j,i}$ and $(v', v') \geq 0$). From Linear Algebra, we know that any positive-definite matrix is invertible and not singular, so the system $A\alpha = b$ has a unique solution.

If we were to partition our domain into equal pieces (i.e., $h_j = h = \frac{1}{N}$), then our matrix, A, would have the same form as for finite-differences. Specifically, $A\alpha = b$ has the form

$$\frac{1}{h}\begin{bmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_{N-3} \\ \alpha_{N-2} \\ \alpha_{N-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{N-3} \\ b_{N-2} \\ b_{N-1} \end{bmatrix}$$

One of the main differences between the finite-difference discretization and this one is that the matrix A is multiplied by $\frac{1}{h}$ here as opposed to $\frac{1}{h^2}$ in the finite-difference case. This is accounted for in the fact that the right-hand side in this finite element version is of the form $b = \Sigma b_i = \Sigma(f, \phi_i)$. If we were to work this out, we would see that in fact, the formulation here does agree with finite-difference approximations, where the Dirichlet boundary conditions are $g = 0$ on $\partial\Omega$. That is, $\frac{b_i}{h} = f_i$ when averaged over the interval, (x_{i-1}, x_{i+1}). Hence, $b_i \approx hf_i$.

Understanding the basic idea of variational or Galerkin finite element methods as well as the idea of a minimization problem, we move on to look at how we can extend the idea into 2D.

A.4.2 Finite Elements in 2D

We wish to solve the following Poisson equation in 2D with Dirichlet boundary conditions set to zero:

$$-\Delta u = f \text{ in } \Omega \tag{A.18}$$

$$u = 0 \text{ on } \partial\Omega.$$

Define a new set of functions as in the previous section:

$V_2 = \{\, v \mid v \text{ is continuous on } \Omega, \dfrac{\partial v}{\partial x} \text{ and } \dfrac{\partial v}{\partial y} \text{ piecewise continuous and bounded on } \Omega, v = 0 \text{ on } \partial\Omega \,\}.$

Further, we modify our inner-product from before to be $(v, w) = \int\limits_{\Omega} vw\mathbf{x}$. Using the *Divergence Theorem* in two dimensions, we develop one of *Green's* formulas (see (Johnson, 1987) and section A.3 for further details). Let $\mathbf{i}$ represent a unit-vector in the x-direction and $\mathbf{j}$ be a unit-vector in the y-direction. Applying the Divergence Theorem to the following vector in the x-direction, $(v\frac{\partial w}{\partial x})\mathbf{i}$,

$$\int\limits_{\Omega} \nabla \cdot \left(v\frac{\partial w}{\partial x}\right)\mathbf{i}d\mathbf{x} = \int\limits_{\Omega} \left(\frac{\partial v}{\partial x}\frac{\partial w}{\partial x} + v\frac{\partial^2 w}{\partial x^2}\right)d\mathbf{x} = \int\limits_{\partial\Omega} v\frac{\partial w}{\partial x}n_x ds.$$

Applying the Divergence Theorem to a similar vector in the y-direction,

$$\int\limits_{\Omega} \nabla \cdot \left(v\frac{\partial w}{\partial y}\right)\mathbf{j}d\mathbf{x} = \int\limits_{\Omega} \left(\frac{\partial v}{\partial y}\frac{\partial w}{\partial y} + v\frac{\partial^2 w}{\partial y^2}\right)d\mathbf{x} = \int\limits_{\partial\Omega} v\frac{\partial w}{\partial y}n_y ds.$$

Letting $\mathbf{n} = (n_x, n_y)$ represent the normal to $\partial\Omega$ at the point $\mathbf{x} = (x, y)$, add the above two formulations:

$$\int\limits_{\Omega} \left(\frac{\partial v}{\partial x}\frac{\partial w}{\partial x} + v\frac{\partial^2 w}{\partial x^2} + \frac{\partial v}{\partial y}\frac{\partial w}{\partial y} + v\frac{\partial^2 w}{\partial y^2}\right)d\mathbf{x} = \int\limits_{\partial\Omega} \left(v\frac{\partial w}{\partial x}n_x + v\frac{\partial w}{\partial y}n_y\right)ds.$$

Rearranging the terms and using the facts that $\frac{\partial w}{\partial x}n_x + \frac{\partial w}{\partial y}n_y = \frac{\partial w}{\partial \mathbf{n}}$ and $\Delta w = \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} \Rightarrow$

$$\int\limits_{\Omega} \left(\frac{\partial v}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial w}{\partial y} \right) d\mathbf{x} = \int\limits_{\partial\Omega} v \frac{\partial w}{\partial \mathbf{n}} ds - \int\limits_{\Omega} v \Delta w d\mathbf{x}.$$

Rearranging the left-hand-side of the above equation using the gradient operator, ∇,

$$\int\limits_{\Omega} \nabla v \cdot \nabla w d\mathbf{x} = \int\limits_{\partial\Omega} v \frac{\partial w}{\partial \mathbf{n}} ds - \int\limits_{\Omega} v \Delta w d\mathbf{x}. \tag{A.19}$$

Multiplying equation A.18 by any $v \in V$ and integrating over our domain, Ω,

$$-\int\limits_{\Omega} \Delta u v d\mathbf{x} = \int\limits_{\Omega} f v d\mathbf{x}.$$

We can use equation A.19 and notice that by definition, $v = 0$ on $\partial\Omega$. Define $(f, v) = \int\limits_{\Omega} f v d\mathbf{x}$ and $\alpha(u, v) = \int\limits_{\Omega} \nabla v \cdot \nabla w d\mathbf{x}$. Thus, equation A.18 can be written in *variational* form as A.20:

$$\alpha(u, v) = (f, v). \tag{A.20}$$

We can show, via a more complicated integration by parts, that if u satisfies equation A.18, then it satisfies the variational problem A.20 and vice versa. Similarly, we define a *minimization* problem which has the same solution as the original Poisson equation and variational problem (A.18 and A.20, respectively). The details of this can be seen in (Johnson, 1987).

Construct a set $V_h \subset V$, similar to before, where V_h will be a set of functions which are linear over some discretization of a domain, Ω. In order to discretize the domain, assume the boundary $\partial\Omega$ forms a polygonal shape (if it is a smooth curve, we can discretize the curve into segments). Then, we *triangulate* Ω as described in (Johnson, 1987). Triangulation of a polygonal domain constitutes a large research field, and examples of how polygons can be triangulated include Delauney Triangulation, convex hull differencing, and Seidel's algorithm (Tor and Middleditch, 1984; Seidel, 1991; Bloomenthal, 1994; Fournier and Montuno, 1984; Lischinski, 1994). We will not discuss triangulation specifically, but imagine that we are able to triangulate some domain as in Figure A.10, showing how one can break a domain into a series of triangles, K_i, based on discretization points, N_j. Both triangles and points are enumerated. Define h to be the length of the longest side of all triangles and define $V_h \subset V$ as the set of all function which are continuous on Ω, linear on each triangle, K, and disappear on the boundary, $\partial\Omega$.

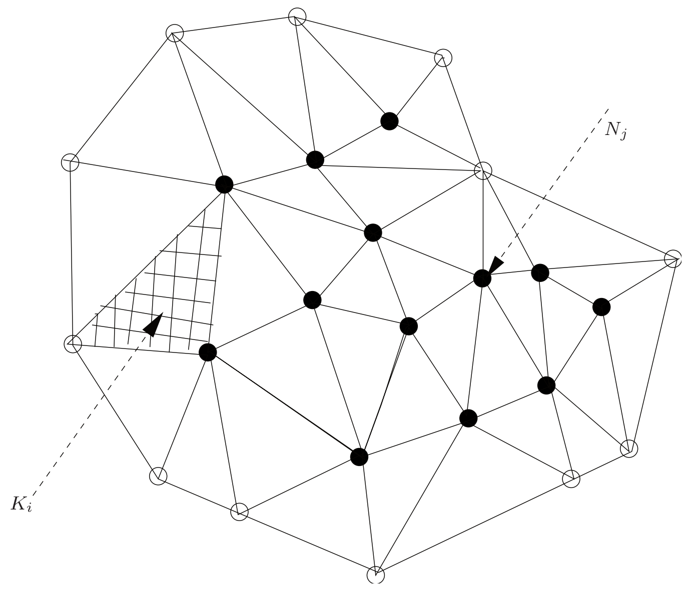


Figure A.10: A domain triangulated into separated triangles, K_i for $i = 1, ..., m$ (m = number of triangles in the domain). Each node is labeled N_j for $j = 1, ..., n$, corresponding to a discretization point for our domain.

To further represent a function $v \in V_h$, we introduce a basis function (or hat function), similar to the one we defined for the 1D problem but now define the basis functions of the nodes, N_i:

$$\phi_j(N_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j. \end{cases} \quad i, j = 1, 2, ...n$$

For example, figure A.11 shows how each basis function, ϕ_j is supported only on triangles which have node N_j as one of its vertices.

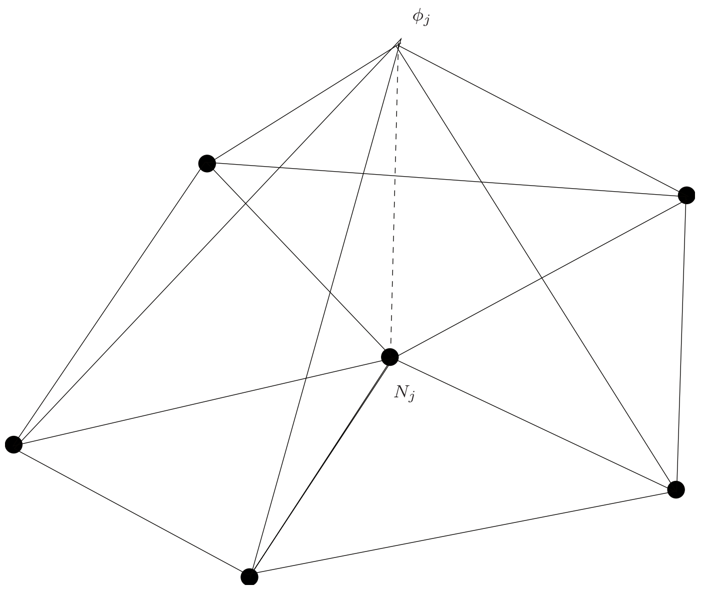


Figure A.11: Example basis function, ϕ_j, supported over triangles K_i, which have node, N_j, as a vertex.

We can write any function $v \in V_h$ as $v(x) = \Sigma_{i=1}^{n} \beta_i \phi_i$ where $\beta_i = v(N_i)$. Further, we can solve for the Poisson equation A.18 by developing a finite element formulation of the Galerkin problem:

$$\forall v \in V, \text{ find } u_h \in V \mid \alpha(u_h, v) = (f, v). \tag{A.21}$$

We can build a system of equations of the form: $A\zeta = b$ where A is the stiffness matrix whose entries are formed by setting $A_{i,j} = a(\phi_i, \phi_j)$, $\zeta_i = u_h(N_i)$ are the solution values, and $b_i = (f, \phi_i)$ is the right-hand side.

If Ω is square, we can triangulate it in a regular fashion as in figure A.12. Here, we say that h is the length of one of the sides of a smaller squares. For an arbitrary node, we indicate its support.

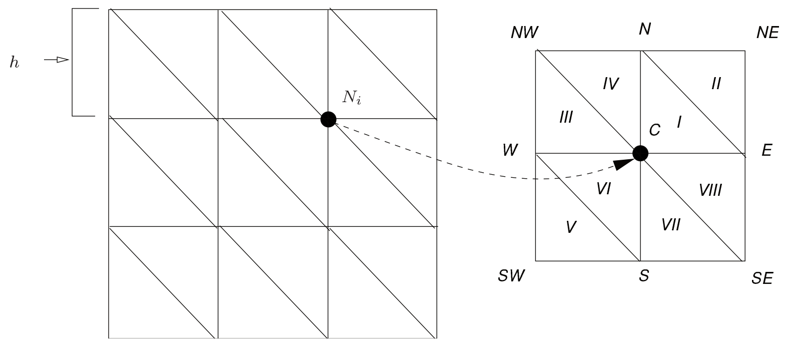


Figure A.12: Example of a small regular 2D problem using finite element methods from (Braess, 2001).

In figure A.12, we focus on node N_i and indicate its center as the center C. The surrounding triangles re labeled using Roman numerals I-VIII as done in (Braess, 2001). Since any interior node i can be magnified to a view of C as the center, we will indicate ϕ_i as ϕ_C. Referring back to figure A.11, we evaluate $\frac{\partial \phi_C}{\partial x}$ and $\frac{\partial \phi_C}{\partial x}$ over triangle I, II, etc., knowing that the length of the perpendicular sides of any triangle is h. The result of these partial derivatives is recorded in the following table:

	I	II	III	IV	V	VI	VII	$VIII$
$\frac{\partial \phi_C}{\partial x}$	$-\frac{1}{h}$	0	$\frac{1}{h}$	0	0	$\frac{1}{h}$	0	$-\frac{1}{h}$
$\frac{\partial \phi_C}{\partial y}$	$-\frac{1}{h}$	0	0	$-\frac{1}{h}$	0	$\frac{1}{h}$	$\frac{1}{h}$	0

For any interior node, i, of our domain we evaluate $A_{i,i} = a(\phi_i, \phi_i) = a(\phi_C, \phi_C)$, where

$$\begin{aligned}
a(\phi_C,\phi_C) &= \int\limits_{I-VIII} \nabla\phi_C\cdot\nabla\phi_C\mathbf{x} \\
&= \int\limits_{I-VIII} \left[\left(\frac{\partial\phi_C}{\partial x}\right)^2 + \left(\frac{\partial\phi_C}{\partial y}\right)^2\right] dxdy \\
&= 2\int\limits_{I-IV} \left[\left(\frac{\partial\phi_C}{\partial x}\right)^2 + \left(\frac{\partial\phi_C}{\partial y}\right)^2\right] dxdy \quad \text{by symmetry} \\
&= \frac{2}{h^2}\left[\int\limits_{I,III} \frac{\partial\phi_C}{\partial x} dxdy + \int\limits_{I,IV} \frac{\partial\phi_C}{\partial x} dxdy\right] \quad \text{by table above} \\
&= \frac{2}{h^2}\left[\frac{h^2}{2}+\frac{h^2}{2}+\frac{h^2}{2}+\frac{h^2}{2}\right] \\
&= 4.
\end{aligned}$$

Similarly, we can compute the product $a(\phi_C,\phi_N)$. By symmetry, $a(\phi_C,\phi_N) = a(\phi_C,\phi_W) = a(\phi_C,\phi_S) = a(\phi_C,\phi_E)$. Notice for example that ϕ_C and ϕ_N only overlap in the regions I and IV. ϕ_C and the other basis functions at E,S, and W will overlap on different regions, but as stated, symmetry will result in the same product. For $a(\phi_C,\phi_N)$, we obtain

$$\begin{aligned}
a(\phi_C, \phi_N) &= \int\limits_{I,IV} \nabla\phi_C \cdot \nabla\phi_N dxdy \\
&= \int\limits_{I,IV} \left[\frac{\partial\phi_C}{\partial x}\frac{\partial\phi_N}{\partial x} + \frac{\partial\phi_C}{\partial y}\frac{\partial\phi_N}{\partial y}\right] dxdy \\
&= \int\limits_{I} \left[\frac{-1}{h} * 0 + \frac{-1}{h} * \frac{1}{h}\right] dxdy + \int\limits_{IV} \left[0 * \frac{1}{h} + \frac{-1}{h} * \frac{1}{h}\right] dxdy \\
&= \frac{-1}{h^2}\left[\int\limits_{I} dxdy + \int\limits_{IV} dxdy\right] \\
&= \frac{-1}{h^2}\left[\frac{h^2}{2} + \frac{h^2}{2}\right] = -1.
\end{aligned}$$

By symmetry, $a(\phi_C, \phi_W) = a(\phi_C, \phi_S) = a(\phi_C, \phi_E) = -1$. Additionally, $a(\phi_C, \phi_{NE}) = a(\phi_C, \phi_{SW}) = 0$ since there is no overlap, and $a(\phi_C, \phi_{NW}) = a(\phi_C, \phi_{SE}) = 0$.

For this unit square, in fact, if we were to enumerate over all nodes on our square domain Ω from the bottom-left to the top-right and apply this finite-element formulation to each node, N_i, the resulting stiffness matrix would have the same coefficients as the matrix A in the five-point finite-difference scheme from section 1.1. The only difference is that in the finite-difference scheme, the matrix was multiplied by $\frac{1}{h^2}$. In fact, here, the right-hand side, b where $b_i = (f, \phi_i)$ is the weighted average of f over each node. Thus, in fact, we will get out a h^2 term over our regular square domain. So, the 5-point finite-difference formulation and the finite element scheme over a regular square domain agree.

We will continue looking at finite elements from the Galerkin method/variational problem point-of-view by discussing more complex elements than those resulting from a triangulation of our domain. First, however, it is important to formalize our function spaces (V and V_h) a more. For this, we introduce and discuss briefly *Hilbert Spaces*.

A.4.3 Overview of Linear, Hilbert and Sobolev Spaces

Galerkin's method is better understood analytically by using more sophisticated function spaces than we have been using. The analysis has been dealt with extensively for finite elements in (Braess, 2001). Here, we will give some broad overview similar to some of what is covered in (Johnson, 1987; Iserles, 1996; Braess, 2001).

If V_1, V_2 are linear spaces (Weir, 1973; Iserles, 1996) over some field F then if $V_1 \subset V_2$, then V_1 is a subspace of V_2. We can also discuss how a linear space has dimension, d, meaning its set of basis functions consists of d linearly independent functions. In general for finite elements, we assume our field is the field of real numbers, $\mathcal{R}$. Examples of linear spaces over $\mathcal{R}$ are the vector space, $\mathcal{R}^d$, and P_n, the set of all polynomials of degree $\leq n$. If we have a linear space, L over the field, $\mathcal{R}$, then we say that a *norm* , $||\cdot||$ is a function mapping from $L \rightarrow \mathcal{R}$ and observes the following properties ($a \in \mathcal{R}$, $f, g \in L$):

$$||f|| \geq 0$$

$$||f|| = 0 \Leftrightarrow f = 0$$

$$||af|| = |a| \, ||f||$$

$$||f + g|| \leq ||f|| \, | \, ||g|| \, .$$

Some of the most general and important norms are known as *p-norms*:

$$||f||_p = \left(\int\limits_{\Omega} |f(\mathbf{x}|)^p d\mathbf{x} \right)^{\frac{1}{p}}, 1 < p < \infty \tag{A.22}$$

$$||f||_\infty = \sup_{\mathbf{x} \in \Omega} |f(\mathbf{x})| \, .$$

The domain, Ω, is the domain over which a set of functions is defined. If a space is equipped with a p-norm with domain Ω, it is denoted as the linear space, $L_p(\Omega)$. If Ω is closed, then $L_p(\Omega)$

is a *normed Banach space*. Some closed linear spaces are also equipped with an *inner product* or *scalar product* of some sort. For example, for linear space, L over $\mathcal{R}$, $f, g, h \in L$, (f, g) is an inner product with the following properties ($a \in \mathcal{R}$):

$$(f, g) = (g, f)$$
$$(af + bg, h) = a(f, h) + b(g, h)$$
$$(f, f) \geq 0$$
$$(f, f) = 0 \Leftrightarrow f = 0.$$

A closed linear space with an inner product is referred to as a *Hilbert space*. For example, $L_2(\Omega)$ is a Hilbert space with the following norm (the 2-norm or *Euclidean* norm) where Ω is closed:

$$||f|| = \left(\int\limits_{\Omega} |f(\mathbf{x}|)^p d\mathbf{x} \right)^{\frac{1}{2}}. \tag{A.23}$$

The inner product is defined as

$$(f, g) = \int\limits_{\Omega} fg d\mathbf{x} \text{ for } f, g \in L_2(\Omega).$$

We further simplify the structure of the norm for $L_2(\Omega)$ in equation A.23 using the inner product formulation as $||f||_{L_2(\Omega)} = \sqrt{(f, f)}$. This is referred to as the L_2 *norm*. *Cauchy's inequality* dictates the following relationship:

$$|(f, g)| \leq ||f||_{L_2(\Omega)} + ||g||_{L_2(\Omega).}$$

In general, $H^m(\Omega)$ is the set of all functions which are in $L_2(\Omega)$, but whose 1st, 2nd, ...,m^{th} derivatives are also in $L_2(\Omega)$. That is, we define the Hilbert spaces, $H^m(\Omega) \subset L_2(\Omega)$ where

$$\begin{aligned} H^m(\Omega) &= \{ v \in L_2(\Omega) \mid \frac{\partial v}{\partial x_i}, \frac{\partial^2 v}{\partial x_i^2}, ..., \frac{\partial^m v}{\partial x_i^m} \in L_2(\Omega), i = 1, ...d \\ & \text{where } \Omega \text{ is bounded on } \mathcal{R}^d,\ d = 1, 2, \text{ or } 3\}. \end{aligned}$$

These Hilbert spaces are specific examples of what are known as *Sobolev* spaces, on which more information (specifically for finite elements) is available in (Braess, 2001). For our purposes we are most concerned about the following Hilbert spaces:

$$\begin{aligned} H^1(\Omega) &= \{ v \in L_2(\Omega) \mid \frac{\partial v}{\partial x_i} \in L_2(\Omega), i = 1, ...d \\ & \text{where } \Omega \text{ is bounded on } \mathcal{R}^d,\ d = 1, 2, \text{ or } 3\}, \end{aligned}$$

$$\begin{aligned} H_0^1(\Omega) &= \{ v \in L_2(\Omega) \mid \frac{\partial v}{\partial x_i} \in L_2(\Omega), i = 1, ...d \\ & \text{where } \Omega \text{ is bounded on } \mathcal{R}^d,\ d = 1, 2, \text{ or } 3, v = 0 \text{ on } \partial\Omega\}. \end{aligned}$$

$H^1(\Omega)$ and $H_0^1(\Omega)$ are also equipped with the following scalar product:

$$(v, w)_{H^1(\Omega)} = \int\limits_{\Omega} (vw + \nabla v \cdot \nabla w)\, d\mathbf{x}, v, w \in H^1(\Omega) \quad \text{where}$$

$$||v||_{H^1(\Omega)} = \left[\int\limits_{\Omega} vw + \nabla v \cdot \nabla w d\mathbf{x} \right]^{\frac{1}{2}}, v \in H^1(\Omega)$$

Additionally, we explicitly define two common Hilbert spaces for finite element methods:

$$\begin{aligned} H^2(\Omega) &= \{ v \in L_2(\Omega) \mid \frac{\partial v}{\partial x_i}, \frac{\partial^2 v}{\partial x_i^2} \in L_2(\Omega), i = 1, ...d \\ & \text{where } \Omega \text{ is bounded on } \mathcal{R}^d,\ d = 1, 2, \text{ or } 3 \quad \text{and}\} \end{aligned}$$

$$H_0^2(\Omega) \quad = \quad \{\, v \in L_2(\Omega) \mid \frac{\partial v}{\partial x_i}, \frac{\partial^2 v}{\partial x_i^2} \in L_2(\Omega), i = 1, ...d$$
$$\text{where } \Omega \text{ is bounded on } \mathcal{R}^d,\ d = 1, 2, \text{ or } 3, v = 0 \text{ on } \partial\Omega\,\}.$$

Another set of functions worth defining explicitly are linear spaces of continuous functions. Commonly used in analytical texts such as (Weir, 1973), they are helpful in explaining the properties of different finite elements. Let $\bar{\Omega} = \Omega \cap \partial\Omega$.

$$C^0(\bar{\Omega}) = \{\, v \mid v \text{ continuous on } \bar{\Omega}\},$$

$$C^1(\bar{\Omega}) = \{\, v \in C^0(\bar{\Omega}) \mid \frac{\partial v}{\partial x_i} \in C^0(\bar{\Omega})\} \quad \text{and}$$

$$C^m(\bar{\Omega}) = \{\, v \in C^0(\bar{\Omega}) \mid \frac{\partial v}{\partial x_i}, \frac{\partial^2 v}{\partial x_i^2}, ..., \frac{\partial^m v}{\partial x_i^m} \in C^0(\bar{\Omega})\}.$$

Using Hilbert spaces and better definitions of continuity, we can give better formulations for variational problems for the Poisson equation. This leads to nice formulations of the problem and allows for better analysis of the existence and uniqueness of solutions (Johnson, 1987).

In particular, solving the Poisson equation:

$$-\Delta u = f \text{ on } \Omega$$

$$u = 0 \text{ on } \partial\Omega$$

is equivalent to the variational problem:

$$\forall v \in H_0^1(\Omega), \text{find } u \in H_0^1(\Omega) \mid (u', v') = (f, v)$$

The variational form of this problem is called a *weak formulation*, and the solution is known as a *weak solution*. Weak solutions are not necessarily classical solutions, and in fact, weak solutions can be shown to exist even if the existence of classical solutions can not be shown. However, using the tools defined here, we can show that a weak solution exists and is unique, and that the solution for the variational and minimization (Galerkin and Ritz) problems coincide.

In order to show this, assume that some Hilbert space, V, with product $(u, v)_V$ and norm $||u||_V$ for $u, v \in V$, and assume $a(u, v)$ acts linearly on $V \times V$, L is a linear operator on V, and require the following properties for all $u, v \in V$:

$$a(u, v) \text{ is symmetric,}$$

$$\mid (a(u, v) \mid \leq \alpha \, ||u||_V \, ||v||_V \, , \alpha > 0,$$

$$a(u, u) \geq \beta \, ||u||_V^2 \, , \beta > 0 \quad \text{and}$$

$$\mid L(u) \mid \leq \gamma \, ||u||_V \, , \gamma > 0.$$

The first property dictates that $a(.,.)$ is *continuous*, the second that it is *V-elliptic*, and the third that $L(.)$ is *continuous* (Johnson, 1987). Given these three properties, the solution to the variational form of the Poisson equation exists and is unique. Further, it is true that $||u_h|| \leq \frac{\gamma}{\beta}$ where u_h is the *weak solution*. The proof of this is not shown here, but it is available in (Johnson, 1987). The existence proof is based on the *Lax-Milgram* Theorem (Kress, 1989). This theorem states that any Hilbert space, V with a V-elliptic (or *strictly coercive*) operator has an inverse, whose proof is also based on use of the Cauchy-Schwarz inequality (Kress, 1989).

If we can show that the variational formulation for a partial differential equation has operators which observe the properties above, then a solution exists and is unique. Consider the 2D Poisson equation A.18 with homogeneous Dirichlet boundary conditions set to zero. The variational form of our problem can be seen as the following:

$$\text{Find } u \in V = H_0^1(\Omega), \Omega \subset \mathcal{R}^2 \text{ such that } a(u,v) = L(v) \forall v \in V \text{ where}$$

$$a(u,v) = \int_\Omega \nabla u \cdot v d\mathbf{x}, L(v) = \int_\Omega f v d\mathbf{x}, f \in L_2(\Omega).$$

Proving that $a(u,v)$ is symmetric, continuous and V-elliptic/strictly coercive is relatively straightforward. The V-elliptic aspect is slightly more complicated, and the details are left to (Johnson, 1987). $L(v)$ is straightforwardly continuous. Therefore, the solution to this problem exists, is unique, and is equivalent to the solution of the minimization formulation of equation A.18.

With a basic understanding of the analysis behind finite elements, we will discuss some of the actual types of finite elements used. Thus far, we have only discussed basic triangular elements. In the next section, we will formalize different finite element spaces, based on element shape.

A.4.4 Examples of Finite Element Spaces

Different finite element spaces of finite size consist of polynomial functions defined on some sort of division of our domain Ω. For example, if $\Omega \subset \mathcal{R}$, we can *triangulate* our space as mentioned before, and then define our finite element space to consist of *triangular* elements. Additionally, a space can be *subdivided* into quadrilateral elements (Zorin et al., 2000). If V_h is the finite element space, we require that $V_h \subset H^k(\Omega)$ where k depends on the order of our boundary value problem ($k = 1$ for first order). Using notation from the previous section, (Johnson, 1987) makes the following observations:

$$V_h \subset H^1(\Omega) \Leftrightarrow V_h \subset C^0(\bar{\Omega}) \quad \text{and}$$
$$V_h \subset H^2(\Omega) \Leftrightarrow V_h \subset C^1(\bar{\Omega}).$$

A finite element space, V_h is now specified by the type of triangulation or subdivision of the domain (the domain is made of disjoint *elements* K_i such as triangles or quadrilaterals), the way each element, $v \in V_h$ acts on each element K_i (linearly, bilinearly, quadratically, cubicly, etc.), and parameters describing each function.

For example, if we have $\Omega \subset \mathcal{R}^2$, triangulated into triangles, K_i, let $\mathbf{P}_j \subset H^1$ be the following space:

$$\mathbf{P}_j = v \mid v \text{ is a polynomial of degree j on each triangle } K_i.$$

For example, $\mathbf{P}_1$ in $\mathcal{R}^2$ consists of all polynomials of the form $v(\mathbf{x}) = a_0 + a_1x + a_2y$, $\mathbf{x} \in K_i$ for each triangle K_i. A clear basis for $\mathbf{P}_1$ is the set of functions $\{1, x, y\}$ from which any function, $v \in \mathbf{P}_1$ can be constructed. $\mathbf{P}_1$ is exactly the type of function we used in the basic 2D example, seen in figure A.11, where each basis function, ϕ_i, descends linearly from its main support node, N_i, to the other vertices of each triangle which it supports.

Instead of descending linearly, a basis function could descend quadratically. That is, let $\mathbf{P}_2$ in $\mathcal{R}^2$ consists of all polynomials of the form $v(\mathbf{x}) = a_0 + a_1x + a_2y + a_3x^2 + a_4y^2 + a_fxy$, $\mathbf{x} \in K_i$ for each triangle K_i. Now, each function has six coefficients, and a simple basis is the set $\{1, x, y, x^2, y^2, xy\}$. In order to construct a basis function, ϕ_i, of this form at some point N_i, we actually need each triangle, K_j to be defined by points at its vertices and points at the midpoints of each edge. Hence, the number of nodes where we have to define support functions has doubled. For example, figure A.13 shows how basis functions defined at the vertex of a triangle look. However, now that we have nodes at the midpoints of edges, we have to be able to define basis functions for nodes at edges. This can be seen in figure A.14. Obviously, the support for this type of node is smaller.

We can also build triangular elements whose basis functions are in $\mathbf{P}_3$, $\mathbf{P}_4$, etc., but we need more and more points on each triangle to construct higher-order (i.e., cubic) functions. In general, for triangular elements, the degrees of freedom required for $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ can be seen in figure A.15.

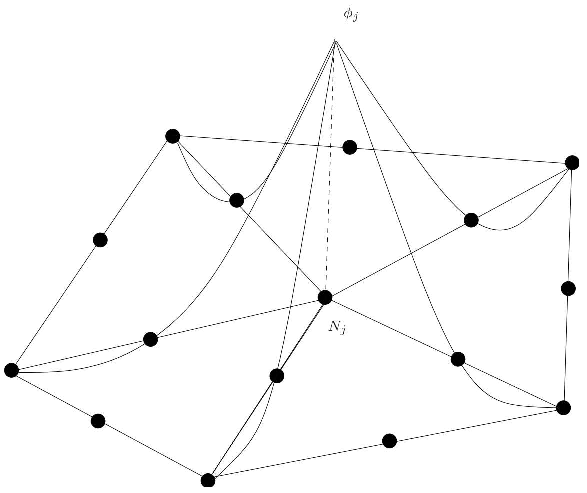


Figure A.13: Example quadratic basis function, $\phi_j \in \mathbf{P}_2$, supported over triangles K_i, which have node, N_j, as a vertex. This an example of a *vertex type* basis function in $\mathbf{P}_2$.

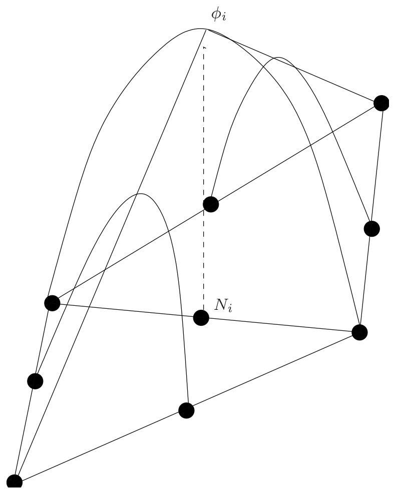


Figure A.14: Example quadratic basis function, $\phi_j \in \mathbf{P}_2$, supported over triangles K_i, which have node, N_j, as a vertex. This an example of a *edge type* basis function in $\mathbf{P}_2$.

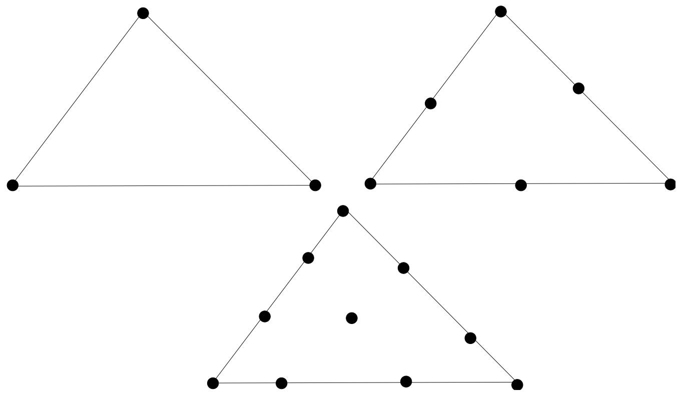


Figure A.15: Triangular elements of type $\mathbf{P}_1$, $\mathbf{P}_2$, and $\mathbf{P}_3$ from left to right. For these elements, we need 3, 6 and 10 degrees, respectively, as indicated by the number of nodes necessary to build the basis functions of necessary order.

As mentioned, our domain, Ω, does not necessarily need to be triangulated. Instead, we could subdivide it into a series of quadrilaterals as in figure A.16.

This is a very irregular division of our domain, but if we looked at a single node, N_i up close, we still define a basis function, ϕ_i, such that it is equal to 1 at N_i and 0 at all other nearby nodes such as in figure A.17. Elements of this type have their basis functions defined in function spaces defined by the notation $\mathbf{Q}_k$ where functions in $\mathbf{Q}_1$ are *bilinear* of the form $(a_0 + a_1 x)(b_0 + b_1 y)$, and functions in $\mathbf{Q}_2$ are *bivariate quadratic* of the form $(a_0 + a_1 x + a_2 x^2)(b_0 + b_1 y + b_2 y^2)$, etc.

The degree of freedom of each quadrilateral element in $\mathbf{Q}_1$ is 4 as can be seen. For $\mathbf{Q}_2$, the degrees of freedom increase to 9 as we need 9 points to adequately define a bivariate quadratic function over each quadrilateral element of this type. In figure A.18, we can see the location of points on such quadrilateral elements.

We can continue defining different types of elements with higher-order accuracy. As we add

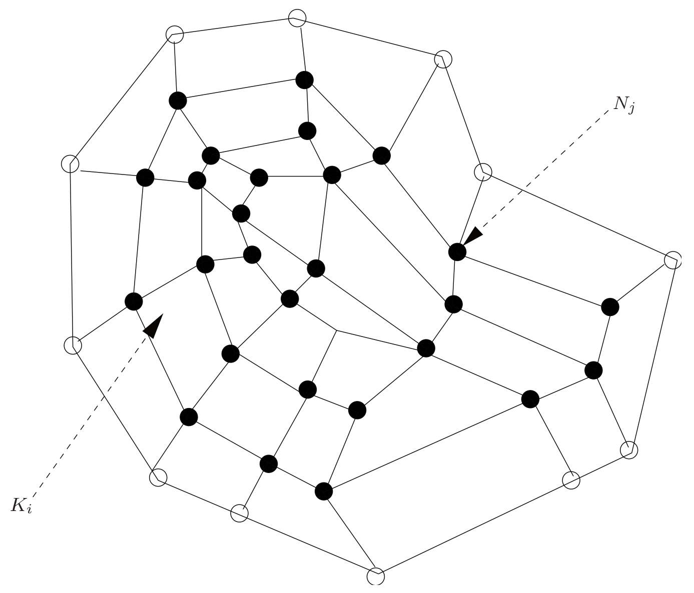


Figure A.16: A domain, Ω which has been subdivided into quadrilaterals of varying size. Black nodes indicate interior points while white nodes indicate points on the boundary, $\partial\Omega$.

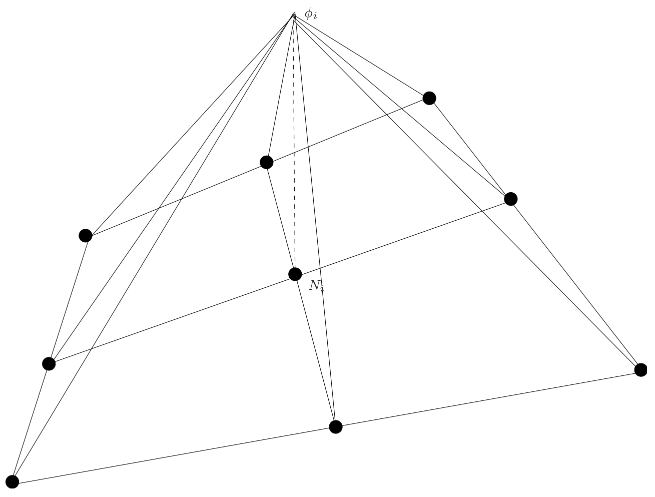


Figure A.17: Example of a basis function, $\phi_i \in \mathbf{Q}_1$, defined over quadrilateral elements.

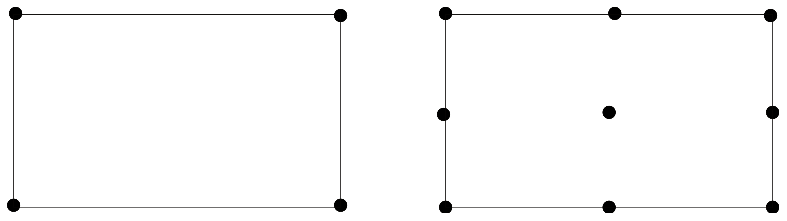


Figure A.18: Example of a basis function, $\phi_i \in \mathbf{Q}_1$, defined over quadrilateral elements.

nodes to our elements, the resulting stiffness matrix becomes more dense. Hence, considerations of accuracy versus computation time must be taken into account.

Additionally, all of the elements and function spaces shown thus far have had their functions, $v \in V_h$ (for $V_h = \mathbf{P}_1, \mathbf{P}_2, \mathbf{Q}_1, \mathbf{Q}_2 \subset H^1$), in C^0. However, at node points of our elements, we can require that not only function values but their first, second, etc. derivatives are defined, or that normal derivatives to the element are defined. Doing so can make functions from our function space be C^1 for example. Further details on these types of higher order elements are available in (Braess, 2001) and (Johnson, 1987). Some of the more famous elements also have different names, such as the *Argyris*, *Hsieh-Clough-Tocher*, and *Serendipity* elements, and details of these constructions are available in (Braess, 2001).

Finally, we could imagine how to extend this to three dimensions. For example, a *brick* element in three dimensions would be an extension of $\mathbf{Q}_1$ to 3D by creating three dimensional brick from six quadrilateral faces and allowing the basis functions to be trilinear of the form $(a_0 + a_1 x)(b_0 + b_1 y)(c_0 + c_1 y)$. For $\mathbf{P}_1$ and $\mathbf{P}_2$ we can create tetrahedral elements whose function spaces are linear or quadratic, respectively; more details are available in (Johnson, 1987; Braess, 2001; Elman et al., 2005).

A.4.5 Other Boundary Conditions for Finite Elements

So far, the only Poisson equation we have discussed is one with Dirichlet boundary conditions set to zero on $\partial\Omega$. If we have different Dirichlet boundary conditions, it is very easy to modify our finite element space to take them into consideration. For example, consider that we have the following Poisson equation:

$$-\Delta u = f \text{ in } \Omega$$

$$u = g \text{ on } \partial\Omega.$$

Galerkin's method dictates the following problem,

$$\text{Find } u \in V_g = \{v \mid v = g \text{ on } \partial\Omega, ||v||_H^1(\Omega) < \infty\} \text{ where}$$

$$\forall v \in V_0, (\nabla u, \nabla v) = (f, v)$$

Here, $V_g \subset H^1$ and $V_0 \subset H_0^1$. Given some sort of triangulation or subdivision, T_h of our domain, Ω, where the set of internal nodes is designated by N_Ω and boundary nodes by $N_{\partial\Omega}$, we approximate the solution u as u_h:

$$u_h = \sum_{N_i \in N_\Omega} u(N_i)\phi_i + \sum_{N_i \in N_{\partial\Omega}} u(N_i)\phi_j$$

Therefore, an approximation to the Dirichlet boundary value Poisson problem becomes (where $u_i = u(N_i)$, $g_i = g(N_i)$)

$$\sum_{N_j \in N_\Omega} u_j(\nabla\phi_i, \nabla\phi_j) = (f, \phi_i) - \sum_{N_j \in N_{\partial\Omega}} g_j(\nabla\phi_i, \nabla\phi_j).$$

Consider that the following Poisson equation with Neumann boundary conditions:

$$-\Delta u = f \text{ in } \Omega$$
$$\frac{\partial u}{\partial \mathbf{n}} = g \text{ on } \partial\Omega.$$

Galerkin's method dictates the following formulation:

$$\text{Find } u \in V_h = H^1(\Omega) \text{ where}$$
$$\forall v \in V_0, (\nabla u, \nabla v) - \left(\frac{\partial u}{\partial n}, v\right)_{\partial\Omega} = (f, v).$$

The notation $\left(\frac{\partial u}{\partial n}, v\right)_{\partial\Omega}$ indicates the scalar product over the boundary, or

$$\left(\frac{\partial u}{\partial n}, v\right)_{\partial\Omega} = \int\limits_{\partial\Omega} \frac{\partial u}{\partial n} v ds \text{ where } ds \text{ is an element piece of } \partial\Omega.$$

Substituting and rearranging, we are now solving the following variational problem for u:

$$\forall v \in H^1(\Omega), a(u, v) = (f, v) + (g, v)_{\partial\Omega}.$$

We are again solving a system of equations of the form $A\alpha = b$ where $A_{i,j} = a\phi_i, \phi_j$, $\alpha_i = u_i = u_h(N_i)$ and $b_i = (f, \phi_i) + (g, \phi_i)_{\partial\Omega}$.

Imagine now that we are solving a Poisson equation with Robin boundary conditions. That is, we seek a solution, u to the following problem:

$$-\Delta u = f \text{ in } \Omega$$
$$\gamma u + \frac{\partial u}{\partial \mathbf{n}} = g \text{ on } \partial\Omega$$

We put forth the variational formulation:

$$\text{Find } u \in V_h = H^1(\Omega) \text{ where}$$

$$\forall v \in V_0, (\nabla u, \nabla v) - \left(\frac{\partial u}{\partial n}, v\right)_{\partial\Omega} = (f, v).$$

Plugging in and rearranging the variational formulation such that we are solving the following problem for u:

$$\forall v \in H^1(\Omega), a(u, v) + (\gamma u, v)_{\partial_\Omega} = (f, v) + (g, v)_{\partial\Omega}.$$

Again, we can put this into matrix-vector form $A\alpha = b$ where $A_{i,j} = a(\phi_i, \phi_j) + (\gamma\phi_i, \phi_j)$, $\alpha_i = u_i = u_h(N_i)$, and $b_i = (f, \phi_i) + (g, \phi_i)_{\partial\Omega}$.

From this, we can also very easily extrapolate how to use Galerkin's method to find a weak solution to a Poisson equation with mixed boundary conditions. That is, if $\partial\Omega = \partial\Omega_1 \cap \partial\Omega_2$, we can dictate that Dirichlet boundary conditions on $\partial\Omega_1$ and Robin or Neumann boundary conditions on $\partial\Omega_2$, for example.

Understanding the basic ideas behind finite difference and finite element methods for constructing a system of linear equations for the Poisson equation, we will now look at various methods for solving these systems of equations.

A.5 Background for Solving Various Systems of Linear Equations

For a system of linear equations of the form $Au = b$ where we are solving for u, A being a matrix of coefficients and b being a vector of known values, there are many known ways of solving for u. Many software packages offer a variety of solvers for different situations. In fact, many of these packages are built on top of LAPACK routines. Examples include MATLAB and PETSC. Sometimes, any solver is adequate, but more often than not, it is important to exploit the nature of the matrix A. Some matrices have a very regular structure (such as in finite differences or finite elements on regular grids). Other times, the matrices have a less regular structure. We will first consider matrices with a regular structure that are most usually sparse and look at a couple of basic algorithms used for solving these systems. For basic linear algebra definitions, we refer to (Strang, 1988). Much of the discussion below is adapted from (Iserles, 1996; Demmel, 1997; Trefethen and Bau, 1997).

A.5.1 Direct Methods for Sparse and Banded Matrices

Gaussian Elimination

Gaussian elimination or *LU factorization* is one of the more commonly seen algorithms for solving systems of linear equations. The basic idea is to compute a factorization of the form $A = LU$ where L is a lower triangular matrix and U is an upper triangular matrix. That is, if A is an n by n matrix, then L and U will be of the form:

$$L = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ l_{2,1} & 1 & 0 & \dots & 0 \\ l_{3,1} & l_{3,2} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ l_{n,1} & l_{n,2} & \dots & l_{n,n-1} & 1 \end{bmatrix}, U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \dots & u_{1,n} \\ 0 & u_{2,2} & u_{2,3} & \dots & u_{2,n} \\ 0 & 0 & u_{3,3} & \dots & u_{3,n} \\ \vdots & \vdots & \vdots & \ddots & u_{n-1,n} \\ 0 & 0 & \dots & 0 & u_{n,n} \end{bmatrix}$$

One can easily see how to construct the L and U matrices. Basic pseudocode from (Trefethen and Bau, 1997) computes the LU factorization for a matrix A in Algorithm A.5.1.

Algorithm 7 Gaussian Elimination with matrix A as input (size $n \times n$)

```
$U = A, L = I$
for $i = 1$ to $n - 1$ do
    for $j = i + 1$ to $n$ do
        $l_{i,j} = \frac{u_{i,j}}{u_{j,j}}$
        for $k = i$ to $n$ do
            $u_{j,k} = u_{j,k} - l_{j,i}u_{i,k}$
        end for
    end for
end for
```

Using this type of Gaussian elimination algorithm can lead to instability issues, so *pivoting* is used to prevent this (Trefethen and Bau, 1997). Further, looking at this algorithm, we see it performs $O(n^3)$ floating point operations. This number of operations grows too rapidly; however, if A is *sparse* (i.e., many entries are zero), then many of the operations in the algorithm are unnecessary. To further explain, we introduce the idea of *bandwidth*. A has *upper* bandwidth, β_{up} if $A_{i,j} = 0$, $\forall j > i + \beta_{up}, i, j \leq n$, and A has *lower* bandwidth β_{low} if $A_{i,j} = 0$, $\forall i > j + \beta_{low}, i, j \leq n$. If $\beta_{low} = \beta_{up} = \beta$, A has bandwidth β. For example, the matrix resulting from an $O(h^2)$ finite difference approximation to the 1D Poisson Equation has $\beta = 1$

while in 2D, the matrix will have bandwidth based on the size of the problem. That is, if our domain is an $n \times n$ grid, then A is an $n^2 \times n^2$ matrix, so $\beta = n - 1$ for the 5-point finite difference method. On a 5 by 5 regular square grid, the resulting 25 by 25 matrix will have bandwidth of 4.

If we know that an $n \times n$ matrix A has bandwidth $\beta < n$, then many of the operations in our Gaussian Elimination algorithm are unnecessary. That is, if A has bandwidth β, then it is clear that L and U will also have bandwidth β. This can be proved inductively, as shown in (Iserles, 1996).

If L and U both have bandwidth β, then each $A_{i,j} = (LU)_{i,j}$ can be composed as (McLean, 2004)

$$
\begin{aligned}
(LU)_{i,j} &= \sum_{k=1}^{n} l_{i,k} u_{k,j} \\
&= \begin{cases} \sum\limits_{k=max(1,i-\beta,j-\beta)}^{j} l_{i,k} u_{k,j} & \text{if } 1 \leq j < i \leq n, i \leq j + \beta \\ u_{i,j} + \sum\limits_{k=max(1,i-\beta,j-\beta)}^{i-1} l_{i,k} u_{k,j} & \text{if } 1 \leq i < j \leq n, j \leq i + \beta \\ 0 & \text{else} \end{cases}
\end{aligned}
$$

We can avoid many operations in the Gaussian elimination algorithm; one can solve the above equation for $l_{i,j}$ to see how to compute each entry in L. We approximate the number of operations it costs to compute L as $\approx \sum\limits_{j=1}^{n} \sum_{i=j+1}^{j+\beta} (j - i + \beta)$. It is clear that the asymptotic running time for this is $O(n\beta^2)$. Computing U would be similar as it has the same bandwidth as L, as assumed. So, the cost of computing a banded Gaussian elimination is $O(n\beta^2)$ for bandwidth β.

We now indicate how the LU factorization of A aids in solving $Au = b$. Solving our system is equivalent to trying to construct $u = A^{-1}b$, or $u = (LU)^{-1}b = U^{-1}L^{-1}b$. We first solve $Lv = b$ for v and then solve $Uu = v$ for u. Initially, it looks like we have wasted our time because instead of solving one system of linear equations, we now have to solve two systems of linear equations of the same size. But, remember that L and U will have the same bandwidth as A, so if A has a low bandwidth, so do L and U. We solve $Ly = b$ in Algorithm 8.

Algorithm 8 Banded Solve with matrix L of size $n \times n$ and vector b of size n. Band size = β

```
$y(1...n, 1...n) = 0$
**for** $i = 1$ to $n$ **do**
    $y_i = b_i$
    **for** $j = i - \beta$ to $i - 1$ **do**
        $y_i = y_i - l_{i,j} y_j$
    **end for**
**end for**
```

Therefore, solving for each y_i involves approximately $O(\beta)$ operations, and we do this n times, so the total operation count to do the banded matrix-vector solve for $Ly = b$ is $O(\beta n)$. Solving $Uu = y$ is equivalent, and the algorithm is straightforwardly similar. Hence, solving a system $Au = b$ involves $O((\beta^2 + \beta)n)$ operations.

Cholesky Factorization

For a matrix such as those seen in 5-point and modified 9-point finite difference schemes, we can do slightly better, using *Cholesky* in lieu of Gaussian elimination. We formally define a *symmetric* matrix as one equal to its transpose (i.e., $A = A^T$) and a *positive definite* matrix as one for which $\mathbf{x}^T A^{\mathbf{x}} > 0$ for a vector, $\mathbf{x}$ of size n when A is of size $n \times n$ (we henceforth write $A \in \mathcal{R}^{n \times n}$. Let $A \in \mathcal{R}^{n \times n}$ be of the form

$$A = \begin{bmatrix} 1 & x^T \\ 0 & C \end{bmatrix}.$$

Here, $x \in \mathcal{R}^n$ and $C \in \mathcal{R}^{(n-1) \times (n-1)}$. Performing one step of Gaussian elimination in A results in

$$\begin{bmatrix} 1 & x^T \\ 0 & C \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ x & I \end{bmatrix} \begin{bmatrix} 1 & x^T \\ 0 & C - xx^T \end{bmatrix}.$$

If we perform another step of Gaussian elimination on the second matrix above, we get the following factorization result:

$$\begin{bmatrix} 1 & x^T \\ 0 & C - xx^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & C - xx^T \end{bmatrix} \begin{bmatrix} 1 & x^T \\ 0 & I \end{bmatrix}.$$

Putting these two results together,

$$A = D \begin{bmatrix} 1 & 0 \\ 0 & C - xx^T \end{bmatrix} D^T \text{ where } D = \begin{bmatrix} 1 & 0 \\ 0 & C - xx^T \end{bmatrix}.$$

We can continue to iterate on the middle matrix, involving C until we obtain a factorization of the form $A = D_1 D_2 \cdots D_n \cdot D_1^T D_2^T \cdots D_n^T = LL^T$. We can generalize this result to an example when the top-left entry, $A_{1,1}$, is not necessarily 1, the details of which are left to (Trefethen and Bau, 1997). We have computed an LU factorization where $U = L^T$. A straightforward algorithm, such as the one in (Trefethen and Bau, 1997) can be seen in Algorithm 9.

This factorization, just as the LU factorization, involves $O(n^3)$ operations, but the actual operation cost is about half as much or $O(\frac{n^3}{2})$. Therefore, for a banded Cholesky solver, we would perform about half as many overall operations, which is significant, or $O(\frac{\beta^2 n}{2} + \beta n)$ total operations to solve $Au = b$ via Cholesky factorization.

Discretization Considerations

These solvers work best on linear systems of equations for sparse matrices that are well-organized in a tightly banded-fashion. For example, for a 2D finite-difference Poisson equation solver on a unit square, we would enumerate the discretization points using a simple five or nine point

Algorithm 9 Cholesky Factorization of matrix A of size $n \times n$

```
$L = A$
for $i = 1$ to $n$ do
    for $j = i + 1$ to $n$ do
        for $k = i$ to $n$ do
            $l_{j,k} = l_{j,k} - \frac{l_{i,k} l_{i,j}}{l_{i,i}}$
        end for
    end for
    for $k = i$ to $n$ do
        $l_{i,k} = \frac{l_{i,j}}{\sqrt{l_{i,i}}}$
    end for
end for
```

differencing scheme. If we imagined that we were solving a Neumann boundary valued Poisson problem there, the resulting matrix of coefficients would be symmetric, with low bandwidth and positive-definite (for $-\Delta u = b$). Now, if we poorly enumerated our domain as in figure A.19, then the resulting matrix would be symmetric, but it would be inefficiently banded.

As one would expect, avoiding such a poor enumeration scheme on a regular domain would be quite easy. However, if one were using finite elements to create a stiffness matrix, it would be easy to accidentally number elements next to each other with numbers far from each other. For example, if we triangulated an irregular domain into 100 triangular elements, we could accidentally number a triangular element 1 and one next to it 100 in the interior of our domain. The resulting stiffness matrix would be poorly banded, and a Cholesky Solver would be inefficient. Often these problems can be fixed, and there are ways of rearranging matrices to make them more tightly banded to exploit the sparsity, but not paying attention to the sparse or dense nature of a matrix can create unnecessary bottlenecks.

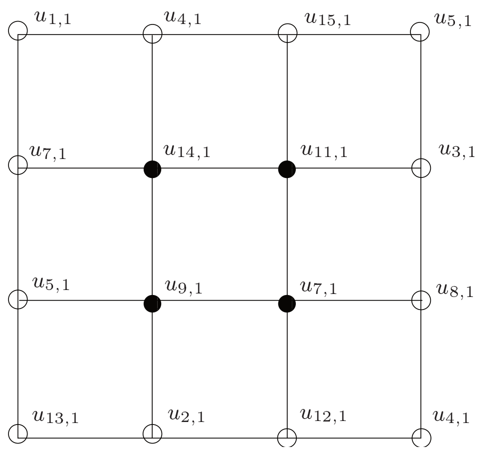


Figure A.19: A domain whose domain points have been poorly numbered for the Poisson equation. The resulting matrix of coefficients is inefficiently banded, resulting in an inefficient solver.

Other Direct Methods

Again we mention that a stable Gaussian or Cholesky algorithm would incorporate *pivoting* (Trefethen and Bau, 1997). Further, there are many other direct solver methods which involved various factorizations. Examples are *QR Factorization* (including Gram-Schmidt Orthogonalization, modified GS and Householder Triangulation) where we factor A as $A = QR$, R being upper-triangular, and Q being an orthonormal matrix (Trefethen and Bau, 1997), and the *Singular Value Decomposition* (or SVD) where A is factored as $A = USV^T$ where U and V are orthonormal, and S has positive real values along the main diagonal and zero everywhere else.

These factorizations methods, when done stably, provide nice accurate solutions to well-posed problems. Additionally, as imagined, they can often be performed with very little extra storage space, or can often be done *in place* which is important when the problem size is large. However, often times a very accurate solution is not as necessary, or often we would like to be able to cut off the accuracy of the solution at a certain point. For this, we turn to *iterative* *methods*.

A.5.2 Iterative Methods

Direct Methods such as those described in the last section work well for some problems, but for more dense matrices the amount of work for solving an $n \times n$ system often involves $O(n^3)$ work. When well-posed, these problems result in a high-level of accuracy. *Iterative methods* for solving systems of linear equations can reduce this operation count to $O(n^2)$ or even $O(cn)$ for some constant c; however, reducing the operation count may result in less accuracy. The basic idea behind iterative methods is to take a matrix, A and a vector, x and compute Ax as quickly as possible. Many of the more contemporary iterative methods refer to *Krylov subspaces*. For example, the Krylov subspaces generated by A and x would be defined as $K_{(1)}, K_{(2)}, ..., K_{(n)}$ where $K_{(k)}$ is the subspace spanned by the vectors, $x, Ax, A^2x, A^3x, ..., A^{k-1}x$, where A^3x is generated by successive multiplication, $A(A(Ab))$, etc. In general, we project an n-sized system into a Krylov subspace of lower dimension. There are several iterative methods of note, referred to by a variety of names. *Conjugate Gradient* and *Generalized Minimal Residuals (GMRES)*, *Lanczos*, and *Arnoldi Iteration* are methods based on Krylov subspaces. *Successive Over-Relaxation (SOR)*, *Gauss-Seidel*, and *Jacobi* are classic iterative methods, which are older than the Conjugate Gradient and GMRES methods, but often just as powerful.

We will begin with an introduction to some of the classic iterative schemes and then discuss some of the more modern Krylov subspace methods.

Classic Iterative Methods

The idea behind the classic iterative methods is to turn $Au = b$ into an iterative sequence of steps to reach a point of convergence. We set up a system of the form

$$u_{[i+1]} = Hu_{[i]} + v. \tag{A.24}$$

H is an *iteration matrix*, and v is a vector, both independent of the step number, i, but based on the original form of A and b. The sequence, $u_{[0]}, u_{[1]}, ...$ should converge to the solution, $\bar{u}$ of the system.

We show a theorem, yet do not prove it, which states that an iterative sequence of this type, for appropriate H and v does in fact converge. To do this, we define the *spectral radius*, $\rho(W)$, of a matrix $W \in \mathcal{R}^{n\times n}$ where $\Lambda = \{\lambda_1, \lambda_2, ..., \lambda_n\}$ is the set of the n eigenvalues of W:

$$\rho(B) = \max \Lambda.$$

Given any linear system $Au = b$, we have the following theorem from (Iserles, 1996):

Theorem A.7. *A scheme such as the one in equation A.24 for appropriate H and v converges to a unique limit $\bar{u}$ if and only if $\rho H < 1$, independent of the initial choice for the iteration, $u_{[0]}$. Further, $\bar{u}$ is the correct solution if and only if $v = (I - H)A^{-1}b$.*

The goal is to construct the appropriate H and v. This is, in fact, the main difference between the different classic iterative methods. Imagine we split some matrix A into a matrix, D, whose diagonal is equal to that of A, and all other entries are zero, and into another matrix, E whose diagonal entries are zero and all other entries are equal to those in A. Further split E into a strictly lower triangular matrix, L_0 whose diagonal entries are zero, and a strictly upper triangular matrix, U_0 whose diagonal entries are zero. L_0's and U_0's nonzero entries correspond to entries in A. Graphically, this can be seen in figure A.20, reproduced in part from (Iserles, 1996).

Define $L = D^{-1}L_0, U = D^{-1}U_0$. The *Jacobi* iteration method defines H and v as:

$$H = L + U = D^{-1}E$$

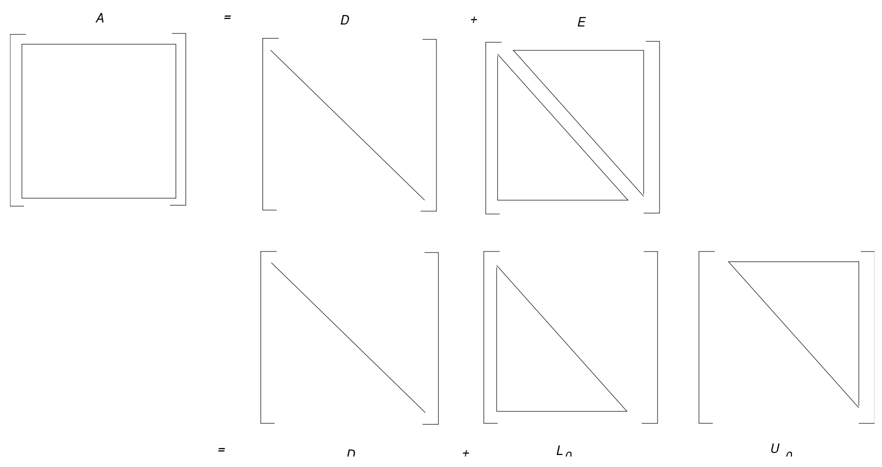


Figure A.20: Split A into matrices D and E as seen and described above. Further E is split into L_0 and U_0.

$$v = D^{-1}b.$$

The *Gauss-Seidel* iteration method defines H and v as

$$H = (I - L)^{-1}U$$

$$v = (I - L)^{-1}D^{-1}b.$$

The *Successive Over-Relaxation* or *SOR* iteration method defines H and v as

$$H = (I - \omega L)^{-1}[(1 - \omega)I + \omega U]$$

$$v = \omega(I - \omega L)^{-1}D^{-1}b.$$

The Gauss-Seidel method is simply the SOR method with $\omega = 1$; however, it is such a commonly seen method, it is often singled out.

All three methods conform to the necessary properties in the above theorem. The Jacobi method, in practice, converges more slowly than Gauss-Seidel and SOR (Demmel, 1997), and Jacobi actually requires more storage space (Iserles, 1996). For specific values of ω, SOR can outperform Gauss-Seidel quite substantially, and analysis of how to choose different values of ω for different problems and spectral radii are available in (Iserles, 1996).

We will revisit SOR when we discuss Multigrid Methods, but we now discuss Krylov-based iterative methods, specifically the Conjugate Gradient method, and we briefly introduce the GMRES method and Arnoldi and Lanczos iteration.

Krylov-Based Iterative Methods

We briefly introduced the idea of Krylov subspaces in section A.4. Different Krylov-based iterative methods are best used for different types of matrices, A and problems. (Trefethen and Bau, 1997) arranges several of the Krylov-based algorithms into a table to indicate when they should be used. This table is reproduced in figure A.21 for a real matrix, A.

The majority of systems we have seen are symmetric, so we will begin with a discussion of the basic Conjugate Gradient method.

Conjugate Gradient Method

The Conjugate Gradient (CG) method is best suited for symmetric positive definite matrices. As these are the type of matrices often encountered in the numerical approximation to the Laplacian, it makes the most sense to discuss this for the Poisson equation, $-\Delta u = b$ along with boundary conditions. This discussion of CG will be based largely on (Shewchuk, 1994)'s introduction and further details are available there.

Define the *quadratic form* for vectors u and b, matrix A and constant c as

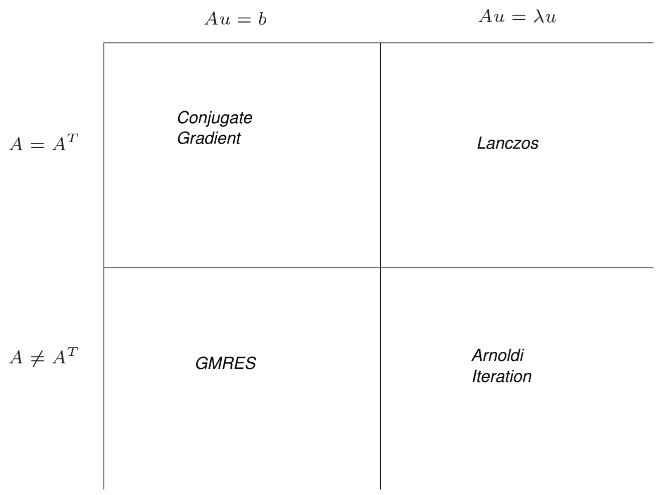


Figure A.21: For a system of equations, we can have $Au = b$ for $b \neq c * u$ for some constant, c. If $Au = \lambda u$, then u is an eigenvector with corresponding eigenvalue, λ. $A = A^T$ corresponds to a symmetric real matrix.

$$f(u) = \frac{1}{2}u^T Au - b^T u + c. \tag{A.25}$$

Take the gradient of equation A.25, $\nabla f(u)$, to obtain

$$\nabla f(u) = \frac{1}{2}\left(A^T u + Au\right) - b.$$

If A is a symmetric matrix, then this equation becomes

$$\nabla f(u) = Au - b.$$

The solution to $Au = b$ is a critical point of equation A.25. In fact, if A is positive-definite, then u represents a minimum vale of f. To see why, imagine that $\bar{u}$ solves $Au = b$ exactly. Then, for $d \neq 0$, $f(\bar{u}+d) = f(u) + \frac{1}{2}d^TAd$. But, A positive-definite $\Leftrightarrow d^TAd > 0$. Hence, $f(u)$ can be thought of as a paraboloid with a minimum saddle-point (Shewchuk, 1994).

Imagine we begin at some arbitrary value, $u_{[0]}$, and successively move down the *paraboloid* until we reach the critical point. We indicate our steps as a series, $u_{[i]}$, $i = 1, 2, ...$ At some step, k, we know that f increases the fastest when ∇f is maximized. So, we move in the direction, $\nabla f(u_{[k]}) = b - Au_{[k]}$.

Let $\bar{r}$ to be the negation of the residual so that $\bar{r}_{[i]} = -\nabla f(u_{[i]}) = b - Au_{[i]}$. Choose some value α such that we take a step in the direction of $\bar{r}_{[i]}$ and end up at a new point $u_{[i]} = u_{[i-1]} + \alpha\bar{r}_{[i-1]}$. In order to find α, though, we need to minimize the quadratic function, f. This is done when $\frac{df(u_{[i]})}{d\alpha} = 0$. Using the chain rule, this equation is the same as $\nabla f(u_{[i]})\bar{r}_{[i-1]} = 0$. But, $\nabla f(u_{[i]}) = \bar{r}_{[i]}$, so α minimizes f when $\bar{r}_{[i]}$ and $\bar{r}_{[i-1]}$ are orthogonal (Shewchuk, 1994). For example, if we set $\bar{r}_{[1]}^T\bar{r}_{[0]} = 0$ to enforce orthogonality, we could determine the initial value for α as follows from (Shewchuk, 1994):

$$\begin{aligned}
& \bar{r}_{[1]}^T\bar{r}_{[0]} = 0 \\
\Rightarrow\quad & (b - Au_{[1]})^T\bar{r}_{[0]} = 0 \\
\Rightarrow\quad & (b - A(u_{[0]} + \alpha\bar{r}_{[0]}))^T\bar{r}_{[0]} = 0 \\
\Rightarrow\quad & (b - A(u_{[0]}))^T\bar{r}_{[0]} = \alpha(A(\bar{r}_{[0]}))\bar{r}_{[0]} \\
\Rightarrow\quad & \alpha = \frac{\bar{r}_{[0]}^T\bar{r}_{[0]}}{\bar{r}_{[0]}^TA\bar{r}_{[0]}}
\end{aligned}$$

Generalizing this, imagine that we have an algorithm, which at the ith step computes the following:

$$r_{[i]} = b - Au_{[i]}$$

$$\alpha_{[i]} = \frac{\bar{r}_{[i]}^T \bar{r}_{[i]}}{\bar{r}_{[i]}^T A \bar{r}_{[i]}}$$

$$u_{[i+1]} = u_{[i]} + \alpha_{[i]} \bar{r}_{[i]}$$

Intuitively, this would generate the result of sliding down the paraboloid as described earlier. There are several problems, however. First, when do we stop? This can actually be fixed easily by stopping when the norm of the residual reaches a desired value or when the value of the norm of the difference $u_{[i+1]} - u_{[i]}$ becomes so small, that we are effectively no longer moving. These two parameters are obviously problem-dependent.

Another problem, however, is more complicated. We may be taking steps in the same direction as steps taken earlier. This is incredibly inefficient. The above method is accurately referred to as the method of *Steepest Descent*. We modify this to a similar method of *Conjugate Directions* (Shewchuk, 1994).

We could take a series of m orthogonal steps, $p_{[0]}, ..., p_{[m-1]}$ such that the ith *error*, vector $e_{[i]} = u_{[i]} - \bar{u}$ is orthogonal to $p_{[i]}$, where $\bar{u}$ is the exact solution to $Au = b$. Then, calculate $u_{[i+1]}$ as $u_{[i]} + \alpha_{[i]} p_{[i]}$, where $\alpha_{[i]}$ as calculated before. This would work better, but in order to calculate the α values, we need $e_{[i]}$ which needs the exact solution, $\bar{u}$, and if we had the exact solution,why would we do this at all? In fact, we *can* do this by realizing that $Ae_{[i]} = -\bar{r}_{[i]}$.

Now, we make the direction vectors *A-conjugate* or just *conjugate* (also known as *A-orthogonal*) to each other. That is, $p_{[i]}^T A p_{[j]} = 0$ for $i \neq j$, and we make the ith vector, $p_{[i]}$ conjugate to $Ae_{[i+1]}$. Using these definitions and what we now know about $Ae_{[i]} = -\bar{r}_{[i]}$, we derive $\alpha_{[i]}$:

$$
\begin{aligned}
& p_{[i]}^T A e_{[i+1]} = 0 \\
\Rightarrow\quad & p_{[i]}^T A (e_{[i]} + \alpha_{[i]} p_{[i]}) = 0 \\
\Rightarrow\quad & \alpha_{[i]} = -\frac{p_{[i]}^T A e_{[i]}}{p_{[i]}^T A p_{[i]}} \\
\Rightarrow\quad & \alpha_{[i]} = -\frac{p_{[i]}^T \bar{r}_{[i]}}{p_{[i]}^T A p_{[i]}}
\end{aligned}
$$

The proof that this converges can be seen in (Shewchuk, 1994). Now, all we need to be able to do is calculate the vectors, $p_{[i]}$. To do this, assume we have n linearly independent vectors, $\gamma_0, \gamma_1, ..., \gamma_{n-1}$. Then, we set $p_{[0]} = \gamma_0$ and derive the other $p_{[i]}$ using a Gram-Schmidt type process (Trefethen and Bau, 1997):

$$
p_{[i]} = \gamma_i + \sum_{j=0}^{i-1} \eta_{i,j} p_{[j]}.
$$

Define the $\eta_{i,j}$ values using previously calculated $p_{[i]}$ values, the fact that the vectors are conjugate, and the last equation to achieve:

$$
\begin{aligned}
& p_{[i]}^T = \gamma_i^T + \sum_{j=0}^{i-1} \eta_{i,j} p_{[j]}^T \\
\Rightarrow\quad & p_{[i]}^T A p_{[k]} = \gamma_i^T A p_{[k]} + \sum_{j=0}^{i-1} \eta_{i,j} p_{[j]}^T A p_{[k]} \\
\Rightarrow\quad & \gamma_i^T A p_{[k]} + \eta_{i,k} p_{[k]}^T A p_{[k]} = 0, i > k \\
\Rightarrow\quad & \eta_{i,k} = \frac{\gamma_i^T A p_{[k]}}{p_{[k]}^T A p_{[k]}}
\end{aligned}
$$

The basic idea as to how we construct each $p_{[i]}$ can be seen in figure A.22, reproduced from (Shewchuk, 1994).

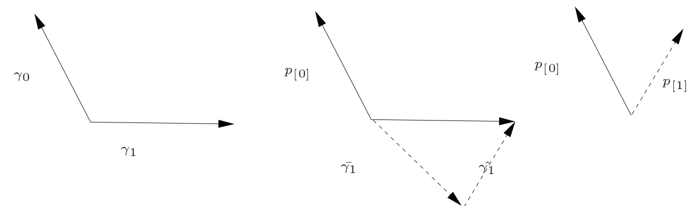


Figure A.22: Construction of $p_{[1]}$ begins with setting $p_{[0]} = \gamma_0$ where γ_0 and γ_1 are pre-known linearly independent vectors. We use the fact that γ_1 consists of a part, $\bar{\gamma_1}$ which is parallel to $p_{[0]}$ and $\hat{\gamma_1}$ which is conjugate to $p_{[0]}$. Following conjugation, via the process above, we set $p_{[0]} = \hat{\gamma_1}$ such that $p_{[0]}$ and $p_{[1]}$ are conjugate.

Unfortunately, the way we are constructing the $p_{[i]}$ vectors via Gram-Schmidt conjugation forces us to store old vectors. Storage is less of an issue than the fact that building all of the $p_{[i]}$ involves $O(n^3)$ operations. This can be improved with some tricks and knowledge of Krylov subspaces to build what is finally known as the method of Conjugate Gradients.

Building the search directions, $p_{[i]}$ is what is causing a bottleneck in the Conjugate Directions method above, but it was noticed by Hestens and Stiefel in the 1950s that the residual at some step j is orthogonal to the previous steps search direction. The residuals were shown to work for the Steepest Descent method, so using this and what we know about the Conjugate Directions method, we can build the Conjugate Gradients method.

Let K_i be the ith Krylov subspace, $K_i = span\{b, Ab, A^2b, ..., A^{i-1}b\}$. Then, the ith Krylov subspace for A and a search direction, $p_{[j]}$ is $K_i = span\{p_{[j]}, Ap_{[j]}, A^2p_{[j]}, ..., A^{i-1}p_{[j]}\}$. Also, let P_i be the subspace spanned by the first i search vectors, $P_i = span\{p_{[0]}, p_{[1]}, ..., p_{[i-1]}\}$.

The subspace, $span\{\bar{r}_{[0]}, \bar{r}_{[1]}, ..., \bar{r}_{[i-1]}\} = P_i$ since the residuals are simply built from the search vectors. Further, as the residual is, as designed, orthogonal to the previous search direction, the fact that the residuals span the same subspace as the search vectors implies that the residuals must all be orthogonal as well. That is,

$$p_{[i]}^T \bar{r}_{[j]} = 0 \text{ for } i \neq j \Rightarrow \bar{r}_{[i]}^T \bar{r}_{[j]} \text{ for } i \neq j.$$

However, each residual is a product of A and the corresponding error vector. So, each residual is a linear combination of the previous residual and the product of A and the previous search vector. That is,

$$\begin{aligned}
\bar{r}_{[i+1]} &= -Ae_{[i+1]} \\
&= -A\left(e_{[i]} + \alpha_i p_{[i]}\right) \\
&= -Ae_{[i]} - A\alpha_i p_{[i]} \\
&= \bar{r}_{[i]} - \alpha_i A p_{[i]}
\end{aligned}$$

Thus, since $p_{[i]} \in P_i$, and P_{i+1} is simply the union of P_i and the product AP_i (the product of A with the elements in P_i), then $P_i = span\{p_{[0]}, Ap_{[0]}, A^2 p_{[0]}, ..., A^{i-1} p_{[0]}\}$. This is simply the ith Krylov subspace for A and $p_{[0]}$. Hence, $P_i = K_i$. Further, this means that $K_i = span\{\bar{r}_{[0]}, A\bar{r}_{[0]}, A^2\bar{r}_{[0]}, ..., A^{i-1}\bar{r}_{[0]}\}$. The benefit of this becomes clear when we realize the power of the fact that $AK_i \subset K_{i+1}$. Coupled with the fact that we know $\bar{r}_{[i+1]}$ is orthogonal to K_{i+1}, we now know that $\bar{r}_{i+1}$ is conjugate to K_i. If we notice that $p_{[i]}$ is not in K_i, but all previous search vectors are, then $\bar{r}_{i+1}$ will be conjugate to all search vectors, $p_{[k]}$, $k < i$. This greatly simplifies the Gram-Schmidt Conjugation algorithm from before. From (Shewchuk, 1994), we can evaluate the $\eta_{i,j}$ values as:

$$\eta_{i,j} = \begin{cases} \frac{\bar{r}_{[i]}^T \bar{r}_{[i]}}{\alpha_{i-1} p_{[i-1]}^T A p_{[i-1]}} & \text{if } i = j+1 \\ 0 & \text{if } i > j+1 \end{cases}$$

For each $\eta_{i,j}$, we only need to compute a matrix-vector multiply, and if the number of nonzero entries in A is small, such as $m << n^2$, then we can do this in $O(m)$ work.

We simplify the notation for η as $\eta_{i,i-1} = \eta_i$ and note we can rewrite $\eta_{i,i-1}$ as $\frac{\bar{r}_{[i]}^T \bar{r}_{[i]}}{\bar{r}_{[i-1]}^T \bar{r}_{[i-1]}}$. In fact, as η_i represents a comparison between the new residual and the previous residual, this can be used to decide when to discontinue the iteration process. That is, as $\eta_i \to 1$, the residual for time step i will be close to that from $i-1$. We use this to set a threshold for η. Putting this all together, we present the Conjugate Gradient as seen in (Trefethen and Bau, 1997) in Algorithm A.5.2.

Algorithm 10 Conjugate Gradient Iteration (CG) on matrix A of size $n \times n$ and vector b of size n

```
$u_{[0]} = (0...n-1, 0...n-1) = 0$
$\bar{r}_{[0]} = b$ and $p_{[0]} = \bar{r}_{[0]}$
for $i = 1$ to ... do
    $\alpha_i = \frac{\bar{r}_{[i-1]}^T \bar{r}_{[i-1]}}{p_{[i-1]}^T A p_{[i-1]}}$ (step length)
    $u_{[i]} = u_{[i-1]} + \alpha_i p_{[i-1]}$ ( new updated approximation to solution)
    $\bar{r}_{[i]} = \bar{r}_{[i-1]} - \alpha_i A p_{[i-1]}$ (new residual)
    $\eta_i = \frac{\bar{r}_{[i]}^T \bar{r}_{[i]}}{\bar{r}_{[i-1]}^T \bar{r}_{[i-1]}}$ (measure of improvement)
    $p_{[i]} = \bar{r} + [i] + \eta_i p_{[i-1]}$ (new search direction)
end for
```

Arnoldi Iteration, GMRES, and Lanczos Iteration

Arnoldi Iteration and the *Generalized Minimal Residuals* methods are two Krylov subspace based methods, which we will mention briefly. *Lanczos Iteration* is a special case for Arnoldi Iteration. Much of the discussion is based on (Trefethen and Bau, 1997). Both of these methods are readily available in modern linear algebra packages and software, including LAPACK and MATLAB.

Arnoldi Iteration is a process, in which we iteratively project onto increasingly larger Krylov subspaces. In order to understand this, we step back, and first explain the idea of an *Hessenberg* matrix. This is a matrix where everything is zero below the first diagonal. For example, if $H \in \mathcal{R}^{n\times n}$ is a Hessenberg matrix, then it is of the form

$$H = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \dots & h_{1,n} \\ h_{2,1} & h_{2,2} & h_{2,3} & \dots & h_{2,n} \\ 0 & h_{3,2} & h_{3,3} & \dots & h_{3,n} \\ \vdots & \vdots & \vdots & \ddots & h_{n-1,n} \\ 0 & 0 & \dots & h_{n,n-1} & h_{n,n} \end{bmatrix}.$$

As mentioned earlier, direct methods such as Gram-Schmidt orthogonalization or Householder triangulation factor a matrix as $A = QR$ (also known as QR factorization), where Q is an orthogonal matrix and R is upper-triangular. Assuming we know how to do this and refer to (Demmel, 1997; Trefethen and Bau, 1997) for specifics and further assume we can factor a matrix A as $A = QHQ^T$ where H is a Hessenberg matrix and Q is orthogonal. We know we can do this if a QR factorization exists for $AQ = QH$. Now let $Q_m \in \mathcal{R}^{m\times m}$ be the matrix formed by the first m columns of $Q \in \mathcal{R}^{n\times n}$ $(m < n)$:

$$Q_m = \begin{bmatrix} q_{1,1} & q_{1,2} & q_{1,3} & \cdots & q_{1,m} \\ q_{2,1} & q_{2,2} & q_{2,3} & \cdots & q_{2,m} \\ q_{3,1} & q_{3,2} & q_{3,3} & \cdots & q_{3,m} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ q_{n,1} & q_{n,2} & \cdots & \cdots & q_{n,m} \end{bmatrix}. \tag{A.26}$$

Let $\hat{H}_m$ be the $(m+1) \times m$ upper section of H:

$$\hat{H}_m = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \dots & h_{1,m} \\ h_{2,1} & h_{2,2} & h_{2,3} & \dots & h_{2,m} \\ 0 & h_{3,2} & h_{3,3} & \dots & h_{3,m} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & h_{m,m-1} & h_{m,m} \\ 0 & 0 & \dots & 0 & h_{m+1,m} \end{bmatrix}.$$

Then, $AQ_m = Q_{m+1}\hat{H}_m$, or if q_m is the mth column of Q (the last column of Q_m, of length n),

$$Aq_m = h_{1,m}q_1 + h_{2,m}q_2 + ... + h_{m,m}q_m + h_{m+1,m}q_{m+1}.$$

The set of the orthonormal vectors, $\{q_i\}$ spans the same basis as the Krylov subspaces (Trefethen and Bau, 1997). That is, for our matrix, $A \in \mathcal{R}^{n\times n}$ and some vector $b \in \mathcal{R}^n$, for the mth Krylov subspace, we obtain

$$\begin{aligned} K_m &= span\{b, Ab, A^2b, ..., A^{m-1}\} \\ &= span\{q_1, q_2, ..., q_m\}. \end{aligned}$$

Therefore, if we let $\hat{K}_m \in \mathcal{R}^{m\times n}$ be the matrix whose columns are $b, Ab, A^2b, ..., A^{m-1}b$, then $\hat{K}_m$ has a QR factorization of the form $\hat{K}_m = Q_mR_m$ where Q_m is the same as in equation A.26.

We now project A orthogonally into the m^{th} Krylov subspace, K_m as $Q_m^TAQ_m$; however, if we notice that $Q_m^TQ_{m+1}$ is just a modified identity matrix of size $m \times (m+1)$ (there is an extra row of zeros on the $(m+1)$st row), $Q_m^TQ_{m+1}\hat{H}_m$ is just H_m, the first m rows and columns of H (Trefethen and Bau, 1997).

$$H_m = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \dots & h_{1,m} \\ h_{2,1} & h_{2,2} & h_{2,3} & \dots & h_{2,m} \\ 0 & h_{3,2} & h_{3,3} & \dots & h_{3,m} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & h_{m,m-1} & h_{m,m} \end{bmatrix}$$

Remember $A = QHQ^T$; therefore, $H_m = Q_m^T AQ_m$, implying that H_m is a way of representing the orthogonal projection of A into the mth Krylov subspace. (Trefethen and Bau, 1997).

This introduction into Arnoldi Iteration is very brief, but it gives a better understanding of how we can represent the Krylov subspaces, and how they can be used for factoring a matrix. This process can be used for locating eigenvalues very quickly, and it is a powerful tool for other Krylov-based algorithms such as GMRES. For now, we will quickly show the basic algorithm from (Trefethen and Bau, 1997) in Algorithm 11 for computing Q and H from A.

Algorithm 11 Arnoldi Iteration for matrix A of size $n \times n$

```
$q_1$ = random $(n, 1)$ vector
$q_1 = \frac{q_1}{||q_1||}$ ( normalize $q_1$)
for $i = 1$ to ... do
    $v = Aq_i$
    for $j = 1$ to $n$ do
        $h_{j,n} = q_j^T v$
        $v = v - h_{j,n} q_j$
    end for
    $h_{n+1,n} = ||v||$
    $q_{n+1} = \frac{v}{h_{n+1,n}}$
end for
```

Generalized Minimal Residuals (GMRES) is based on Arnoldi iteration, which as shown, can be used for transforming a matrix into Hessenberg form, using the Krylov subspaces. Additionally, as mentioned, Arnoldi can be used for locating eigenvalues. We can use Arnoldi iteration to solve for a system $Au = b$ where A is not necessarily symmetric (but it must be nonsingular).

Let the exact solution to this system be $\bar{u}$, and at some step, i, let $u_{[i]}$ be an approximation where $\bar{r}_{[i]} = b - Au_{[i]}$ is a residual. We further require that $u_{[i]} \in K_i$, the ith Krylov subspace. GMRES attempts to solve the problem,

$$\text{Find } v \in \mathcal{R}^n | \left|\left|A\hat{K}_n v - b\right|\right| \text{ is minimized.}$$

$\hat{K}_n$ is the matrix whose columns are $b, Ab, ..., A^n b$. Solving this problem can be unstable and slower than necessary, so we modify it slightly. We use the Arnoldi Iteration algorithm to build the matrices, Q_i for $i = 1...n$, whose columns, as described, are vectors which also span the ith Krylov subspace, and solve the following modified problem:

$$\text{Find } y \in \mathcal{R}^n | \, ||AQ_n y - b|| \text{ is minimized.}$$

This is equivalent to

$$\text{Find } y \in \mathcal{R}^n | \left|\left|Q_{n+1}\hat{H}_n y - b\right|\right| \text{ is minimized.}$$

Multiply this by Q_{n+1}. This does not change the norm, so the minimum value of the norm will not change either. Hence, $\left|\left|Q_{n+1}\hat{H}_n y - b\right|\right|$ is equivalent to $\left|\left|\hat{H}_n y - Q_{n+1}^T b\right|\right|$. Further, $Q_{n+1}^T b = ||b|| \, e_1$, where e_1 is the first column of the $n \times n$ identity matrix. So,

$$\text{Find } y \in \mathcal{R}^n | \left|\left|\hat{H}_n y - ||b|| \, e_1\right|\right| \text{ is minimized.}$$

Solving this problem amounts to solving $\hat{H}_n y = t$ where $\hat{H}_n$ is a Hessenberg matrix, so this can be solved very quickly using computational tricks, further explained in (Demmel, 1997; Trefethen and Bau, 1997). Once we have y, we quickly construct the iterative solution$u_{[i]} = Q_i y$. Putting this all together, we have the algorithm from (Trefethen and Bau, 1997) in Algorithm 12.

Algorithm 12 GMRES for matrix A of size $n \times n$ and vector b of size n

```
$q_1 = \frac{b}{||b||}$ (normalize $q_1$)
for $i = 1$ to ... do
    Run the $i$th step of the Arnoldi Iteration Algorithm
    Minimize $\left|\left|\hat{H}_n y - ||b||\, e_1\right|\right|$ for $y$
    $u_{[i]} = Q_i y$ ($Q_i$ from Arnoldi Iteration)
end for
```

Lanczos Iteration is another Krylov-based method for problems as seen in figure A.21. For real matrices, A, we require that A is symmetric (*hermitian* for complex matrices, which we are not concerned with here). Lanczos iteration can be seen as a restriction on Arnoldi iteration. If we remember from the section on Arnoldi Iteration, we can write the Hessenberg matrices, H_m as

$$H_m = Q_m^T A Q_m.$$

Since A is symmetric, $Q_m^T A Q_m$ is symmetric, but H_m symmetric means that it has to be tridiagonal since everything below the first diagonal has to be zero. (Trefethen and Bau, 1997) This greatly simplifies the Arnoldi iteration method. If we consider H_n as being composed of only two vectors along the diagonal, α and β we get the following:

$$H_m = \begin{bmatrix} h_{1,1} & h_{1,2} & 0 & 0 & \dots & 0 \\ h_{2,1} & h_{2,2} & h_{2,3} & 0 & \dots & 0 \\ 0 & h_{3,2} & h_{3,3} & h_{3,4} & \dots & 0 \\ 0 & 0 & h_{4,3} & h_{4,4} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & h_{m,m-1} & h_{m,m} \end{bmatrix} = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & 0 & \dots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & 0 & \dots & 0 \\ 0 & \beta_2 & \alpha_3 & \beta_3 & \dots & 0 \\ 0 & 0 & \beta_3 & \alpha_4 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \beta_{m-1} & \alpha_m \end{bmatrix}$$

So, the Arnoldi Iteration algorithm is simplified to the following (Trefethen and Bau, 1997) in Algorithm 13.

Algorithm 13 Lanczos Iteration matrix A of size $n \times n$

```
$\beta_0 = 0$, $q_0 =$ random $(n, 1)$ vector, and $q_1 = \frac{q_1}{||q_1||}$
for $i = 1$ to ... do
    $v = Aq_i$
    $\alpha_i = q_i^T v$
    $v = v - \beta_{i-1}q_{i-1} - \alpha_i q_i$
    $\beta_i = ||v||$
    $q_{i+1} = \frac{v}{\beta_i}$
end for
```

Lanczos Iteration can also be modified in order to construct orthogonal polynomials. Specifically, we can use a modified version to construct the *Legendre polynomials*, which can be used in the *Gauss-Legendre Quadrature* method for approximating an integral with a finite sum more stably than other methods such as *Newton-Cotes* (Iserles, 1996; Trefethen and Bau, 1997).

A.5.3 The Multigrid Method

The *Multigrid Method* is a technique which uses the classic iterative methods and a *divide and conquer* approach. From a high-level viewpoint it uses a hierarchy of fine and coarse grids to approximate solutions, which helps remove high-frequency error. This method has been covered heavily in several papers and texts, specifically (Hackbusch and Trottenberg, 1982; Hackbusch, 1985), which collects significant papers from Brandt, Hackbusch and Trottenberg on this subject.

If we want to solve a system of linear equations for the 2D Poisson equation of the form $Au = b$ on a grid of size $n \times n$ using an iterative method, we could use a solution on an $\frac{n}{2} \times \frac{n}{2}$ grid as an approximation to $u_{[0]}$. However, we will also solve our equation on the $\frac{n}{2} \times \frac{n}{2}$ grid iteratively, using a solution from an $\frac{n}{4} \times \frac{n}{4}$ as the initial solution vector. And the $\frac{n}{4} \times \frac{n}{4}$ solution is solved iteratively, using a solution from an $\frac{n}{8} \times \frac{n}{8}$, etc.(Demmel, 1997). The coarsest grid is the stopping point for the approximations.

The listing of finer and coarser grids is known as a *hierarchy* of grids. We can move up and down this hierarchy; *coarsening* refers to moving from a higher to finer grid, and *refining* means moving from a coarser to a finer grid (Iserles, 1996). An example of a hierarchy of grids can be seen in figure A.23, including the directions for grid refining and coarsening.

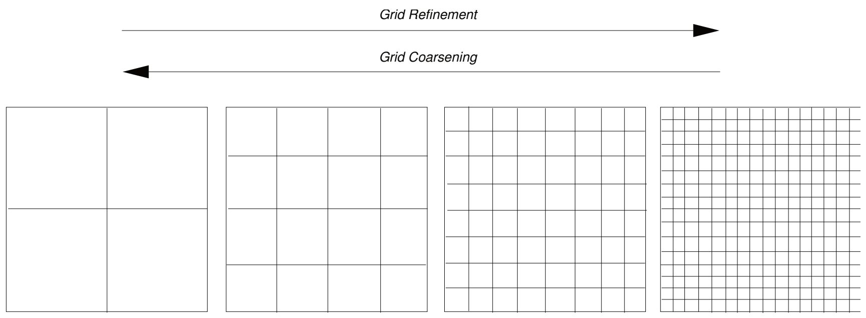


Figure A.23: A series of grids of increasing refinement. For Multigrid, we maintain a *hierarchy* of grids where the finest grid is of size $n \times n$, the next grid in the hierarchy in the coarsening direction is of size $\frac{n}{2} \times \frac{n}{2}$, the next of size $\frac{n}{4} \times \frac{n}{4}$ and so for the number of grids in our hierarchy. The example above shows a hierarchy of 4 grids, where the coarsening and refinement directions are indicated.

To explain Multigrid, begin with a simple example, assuming we already have an approximation for the starting point on the finest grid, and we wish to solve a system $Au = b$, representing the Poisson equation using the Multigrid method, but we will use only two grids: a coarse and a fine one. Further, we will use Gauss-Seidel as our iterative method for now. Indicate $G_{[c]}$ as our coarse grid and $G_{[f]}$ as our fine grid as in figure A.24. That is, we are solving

$$A_f u_f = b_f \text{ on } G_{[f]}.$$

Further assume that we already have $u_{f_{[0]}}$ for $G_{[f]}$. We perform a few iterations with an iterative method on $G_{[f]}$, and we have a residual,

$$r_f = A_f u_f - b_f.$$

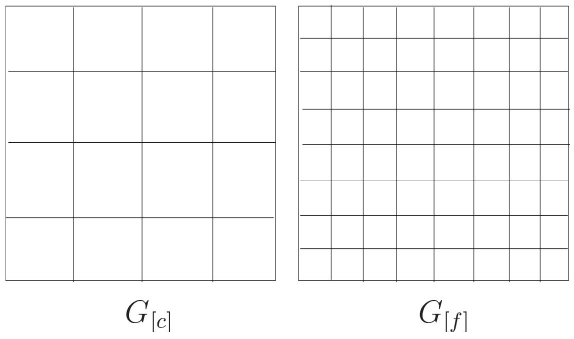


Figure A.24: A hierarchy of two grids: a coarse one, $G_{[c]}$ and a fine one, $G_{[f]}$.

Running an iterative method on $G_{[f]}$ has helped to smooth the higher-frequency components of the residual error. We now move the residual onto the coarse grid, so we use a *restriction* function. That is, if r_f is the residual on $G_{[f]}$, we compute what this residual looks like on $G_{[c]}$. Before we restrict ourselves to the coarse grid, we save our current solution, u_f and call it $u_{f_{old}}$. Then let the restriction matrix be R where

$$r_c = Rr_f.$$

We now smooth the residual on the coarse grid by solving

$$A_c u_c = -r_c.$$

A_c is a matrix composed by approximating our system on the finer grid. For example, if A_f consists of the coefficients of the five-point or modified nine-point finite difference method for approximating the Poisson equation on $G_{[f]}$, A_c consists of coefficients for the same method on $G_{[c]}$. Assume we run a few iterations of an iterative method (such as Gauss-Seidel) on $G_{[c]}$ and wish to now project back to $G_{[f]}$. To this end, we will use a *prolongation* function in the form of a matrix, P where

$$\hat{u}_f = Pu_c.$$

Update the solution on $G_{[f]}$ as

$$u_{f_{new}} = u_{f_{old}} + \hat{u}_f.$$

Evaluate $r_{f_{new}}$ as:

$$\begin{aligned}
r_{f_{new}} &= A_f u_{f_{new}} - b_f \\
&= A_f(u_{f_{old}} + \hat{u}_f) - b_f \\
&= A_f u_{f_{old}} + A_f \hat{u}_f - b_f \\
&= r_{f_{old}} + b_f + A_f P u_c - b_f \\
&= r_{f_{old}} - A_f P A_c^{-1} r_c + (b_f - b_f) \\
&= r_{f_{old}} - A_f P A_c^{-1} R r_{f_{old}} \\
&= (I - A_f P A_c^{-1} R) r_{f_{old}}.
\end{aligned}$$

The new residual is the old residual plus a contribution from going from $G_{[f]}$ to $G_{[c]}$ and back (Iserles, 1996). The operators, R and P can be represented as matrices, but this is an ineffective use of space since the resulting matrix-vector products only involve a few operations per row. An incredibly simple way to build u_c from u_f on a 2D grid would be (assuming $G_c \in \mathcal{R}^{n\times n}$, $G_f \in \mathcal{R}^{2n\times 2n}$):

$$u_{c_{i,j}} = u_{f_{2i,2j}} \text{ for } i, j = 1, 2, ...n$$

This method proves too naïve and can present problems when trying to construct P. Instead, (Iserles, 1996) suggests using the *full weighting* scheme:

$$\begin{aligned}
u_{c_{i,j}} &= \frac{1}{4} u_{f_{2i,2j}} + \frac{1}{8}\left(u_{f_{2i-1,2j}} + u_{f_{2i,2j-1}} + u_{f_{2i+1,2j}} + u_{f_{2i,2j+1}}\right) + \\
&\quad \frac{1}{16}\left(u_{f_{2i-1,2j-1}} + u_{f_{2i+1,2j-1}} + u_{f_{2i-1,2j+1}} + u_{f_{2i+1,2j+1}}\right) \\
&\quad \text{for } i, j = 1, 2, ...n
\end{aligned}$$

Similarly, P is represented as a very straightforward *linear interpolation* scheme from (Iserles, 1996). Obviously, higher-order schemes are available, but we present this one for the general idea:

$$\begin{array}{lll} u_{f_{2i-1,2j-1}} = u_{c_{i,j}} & i = 1, ..., n & j = 1, ...n \\ u_{f_{2i-1,2j}} = \frac{1}{2}\left(u_{c_{i,j}} + u_{c_{i,j+1}}\right) & i = 1, ..., n-1 & j = 1, ...n \\ u_{f_{2i,2j-1}} = \frac{1}{2}\left(u_{c_{i,j}} + u_{c_{i+1,j}}\right) & i = 1, ..., n & j = 1, ...n-1 \\ u_{f_{2i,2j}} = \frac{1}{4}\left(u_{c_{i,j}} + u_{c_{i+1,j}} + u_{c_{i,j+1}} + u_{c_{i+1,j+1}}\right) & i = 1, ..., n-1 & j = 1, ...n-1 \end{array}$$

Knowing how to move between a coarse and fine grid, we can travel between a hierarchy of several grids, but it remains to be seen how we use all of this to solve a system of linear equations from beginning to end. Assume we have a series of four grids from coarse to fine as in figure A.23, labeling the coarsest as $G_{[1]}$ and the finest as $G_{[4]}$. The basic idea is to use what is known as a *V-cycle* as seen in figure A.25 to form a scheme for solving our system. Moving down the $G_{[i]}$ axis indicates a coarsening process, and moving up the $G_{[i]}$ axis indicates a refinement.

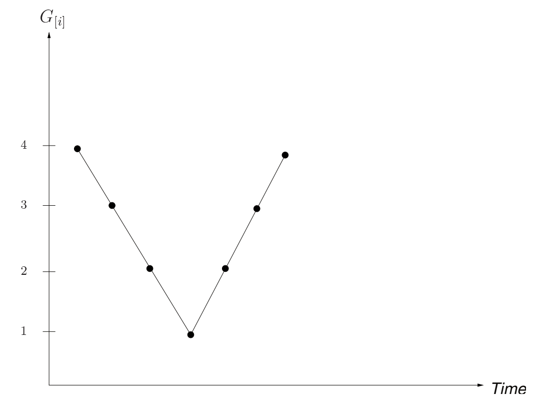


Figure A.25: A basic *V-cycle*, indicating how we can move between our hierarchy of grids in coarsening direction (down the $G_{[i]}$ axis), and the refinement direction (down the $G_{[i]}$ axis).

As discussed earlier, one of the greatest possible benefits of this method is that we can use the solution at a coarse grid as the initial solution for an iterative method on a finer grid. We incorporate this idea by starting at the coarsest grid, $G_{[1]}$, to construct a solution for the starting vector on $G_{[2]}$. That is, if at the end of a few iterations on $G_{[1]}$, we have an approximation, $u1_{new}$, we use this for the initial vector, $u2_{[0]}$ for the iterative method on $G_{[2]}$. We run a few iterations on $G_{[2]}$ to construct $u2_{[new]}$, which can be used as the initial vector, $u3_{[0]}$, on $G_{[3]}$, etc. Once we get to our finest grid ($G_{[4]}$ in our example), we run a few iterations to remove the highest frequency components of our residual as described above. Then, we travel back down to the coarser grid, and so on.

The *full* Multigrid Method or *Full Multigrid V-Cycle* involves using a series of V-cycles as seen in figure A.26. This idea can be described as *nested iteration* (Hackbusch and Trottenberg, 1982; Hackbusch, 1985; Iserles, 1996; Demmel, 1997).

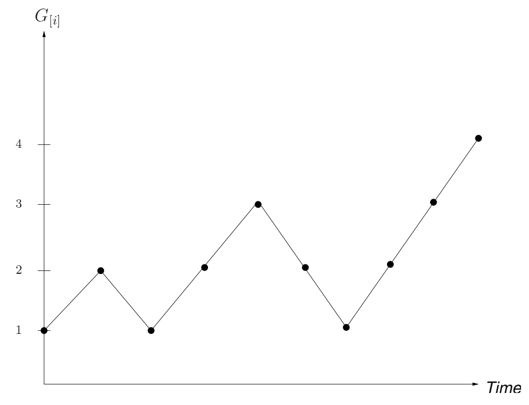


Figure A.26: A full Multigrid V-cycle often seen for constructing a solution to a system of linear equations. We begin at the coarsest grid, $G_{[1]}$ to construct an initial solution for the iterative method used on $G_{[2]}$ and so on. We move up and down the hierarchy of grids to smooth the high frequencies from the residuals.

A high-level pseudocode algorithm for the full multigrid method is provided in (Demmel, 1997). As would be apparent, solving a problem such as the Poisson equation, using a basic system of equations from a finite-difference approximation and a full Multigrid Method with a classic iterative method such as Gauss-Seidel would be relatively straightforward. (Iserles, 1996; Demmel, 1997) show example errors for V-cycles of various sizes.

Additionally, parallelizing the Multigrid technique is what has made it such a powerful and useful tool in numerical analysis. In this case, we can break each grid into a series of sub-grids, where a single processor maintains control over that sub-grid in each of the grids in the hierarchy (Hackbusch, 1985). For a sparse system such as one derived from the five-point approximation to the Poisson equation on a fine grid of size $n \times n$, Multigrid has been shown to run in $O(n)$ time on a single processor, and $O(log^2(n))$ time in parallel on n processors (Demmel, 1997).

A.5.4 Fourier Transforms

We review how Fourier Transforms work and how they can be quickly computed for the purposes of solving systems of linear equations as generated for solving problems such as the Poisson equation.

Fourier Transforms and Their Discrete Representations

There are many sources for discussions of Fourier transforms and inverses, specifically their purposes in Computer Science, including (Aho et al., 1974; Iserles, 1996; Demmel, 1997), and we give a brief introduction to discrete Fourier transforms based on these sources.

We will be using i to indicate $\sqrt{-1}$ over the complex numbers, $\Im$. For some element, $\omega \in \mathcal{R}$, we say that ω is an n *principal root of unity* if (Ahlfors, 1979):

$$\omega \neq 1$$

$$\omega^n = 1$$

$$\sum_{j=0}^{n-1} \omega^{jp} = 0, 1 \leq p \leq n.$$

The general n roots of unity are $\{1, \omega, \omega^2, ..., \omega^{n-1}\}$. Based on Euler's identity $\exp^{ix} = \cos x + i \sin x$, a classic example of a principal nth root of unity over the complex field, $\mathcal{C}$, is $\exp^{\frac{2\pi i}{n}}$.

If a vector, $\mathbf{x} \in \mathcal{R}^n$, $d \in Z$ is of the form, $\mathbf{x}\,[x_0, x_1, ..., x_{n-1}]^T$, its DFT, or discrete Fourier transform, is the vector, $\mathbf{y} = [y_0, y_1, ..., y_{n-1}]^T$, formed by the following formula:

$$y_k = \sum_{l=0}^{n-1} x_l \omega^{-kl}, 0 \leq k \leq n-1, \omega = \exp^{-\frac{2\pi i}{n}}. \tag{A.27}$$

Construct the original sequence by using the *inverse discrete Fourier transform* or *IDFT*:

$$x_l = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega^{kl}, 0 \leq l \leq n-1, \omega = \exp^{-\frac{2\pi i}{n}}. \tag{A.28}$$

The power of the DFT and IDFT become obvious when we consider the ubiquity of *Fourier transforms* in mathematical analysis. For a continuous function, f, periodic in the interval, $[-L, L]$, its expansion into an infinite series of since and cosines is its *Fourier series* representation (Guenther and Lee, 1988):

$$f(x) = \sum_{j=-\infty}^{\infty} \gamma_j \exp^{\frac{i\pi jx}{L}}$$

$$\gamma_j = \int_{-L}^{L} f(x) exp^{-\frac{i\pi jx}{L}} dx, j = 0, 1, 2, ...$$

Take the limit $L \to \infty$ to get the Fourier transform (Demmel, 1997),

$$F_j(x) = \int_{-\infty}^{\infty} f(x) exp^{-\frac{i\pi jx}{L}} dx$$

and its inverse,

$$f_j(x) = \int_{-\infty}^{\infty} F(x) exp^{\frac{i\pi jx}{L}} dx.$$

The DFT and IDFT in equations A.27 and A.28 are exactly that: discrete representations of the continuous Fourier transforms, allowing us to numerically calculate them via quadrature. Returning to DFTs, define the *discrete sine transform*:

$$y_k = 2\sum_{l=1}^{n-1} x_l \sin\frac{\pi kl}{n}, 0 \le k \le n-1. \tag{A.29}$$

Using basic complex analysis techniques (Ahlfors, 1979), rewrite equation A.29 as seen in (McLean, 2004):

$$\begin{aligned} \sin\frac{\pi kl}{n} &= \frac{\hat{\omega}^{\frac{kl}{2}} - \hat{\omega}^{-\frac{kl}{2}}}{2i}, \hat{\omega} = \exp^{\frac{2\pi i}{n'}}, n' = 2n \\ \Rightarrow \quad -iy_k &= 2\sum_{l=1}^{n-1} x_l \left(\hat{\omega^{-kl}} - \hat{\omega^{kl}}\right), 0 \le k \le n-1. \end{aligned}$$

Let $j = 2n - l \Rightarrow \hat{\omega^{kl}} = \omega^{2n\hat{k}-kj} = \hat{\omega^{-kj}}$. Therefore,

$$\begin{aligned} \sum_{l=1}^{n-1} x_l \hat{\omega^{kl}} &= \sum_{j=1}^{2n-1} x_{2n-j} \hat{\omega^{-kl}} \\ \Rightarrow \quad y_k &= \sum_{l=0}^{2n-1} z_l \hat{\omega^{-kl}} \text{ where} \\ z_l &= \begin{cases} 0 & \text{if } l = 0 \text{ or } l = n \\ ix_l & \text{if } 1 \le l \le n-1 \\ -ix_{2n-1} & \text{if } n+1 \le l \le 2n-1 \end{cases} \end{aligned}$$

After all of this, we still haven't addressed the question as to why someone would want to use a DFT for anything. To see why, we introduce the idea of *convolutions*. In the typical Fourier transform sense, for two smooth continuous functions, f and g, the convolution $(f * g)$ is (Demmel, 1997):

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x-y)g(y)dy \tag{A.30}$$

We can state the discrete form of equation A.30 as:

$$\text{If } \mathbf{a}, \mathbf{b} \in \mathcal{R}^{2n} \text{ then } \mathbf{a} * \mathbf{b} = \mathbf{c} \text{ where } c_k = \sum_{j=0}^{k} a_j b_{k-j} \tag{A.31}$$

The Convolution Theorem provides a powerful result. The proof of this is straightforward and left to (Aho et al., 1974; Demmel, 1997).

Theorem A.8. *(**Convolution Theorem**) Let* $\mathbf{a} = [a_0, ..., a_{n-1}, 0, ..., 0]^T$*,* $\mathbf{b} = [b_0, ..., b_{n-1}, 0, ..., 0]^T$ $\in \mathcal{R}^{2n}$*. Further, let* $F(\mathbf{a})$ *and* $F(\mathbf{b})$ *be the DFTs of* $\mathbf{a}$ *and* $\mathbf{b}$*. Then, for* $\mathbf{c} = \mathbf{a}*\mathbf{b} = [c_0, ..., c_{2n-1}]^T$*,* $F(\mathbf{c}) = F(\mathbf{a}) * F(\mathbf{b})$*. Further, if* $F^{-1}(\cdot)$ *is the IDFT of* $F(\cdot)$*, then,* $\mathbf{c} = F^{-1}(F(\mathbf{a}) \cdot F(\mathbf{b}))$

We can now compute the product of two vectors using DFTs and IDFTs. This may seem to be a waste of time since it would seem that computing the DFT of a vector of size n would involve n summations of n terms, implying $O(n^2)$ total operations. But, there is a fast way of computing the DFT, known as the FFT or *fast Fourier transform* which lowers this time to $O(n \log n)$.

Fast Fourier Transforms and Various Implementations

There are a variety of algorithms for constructing the Fast Fourier Transform (FFT) (Press et al., 1992). We look at a high-level overview from (Demmel, 1997), showing how speed-up is possible using recursion even though recursion is not used for the better FFT implementations.

From (Demmel, 1997), imagine we want to calculate $\sum_{j=0}^{n-1} \alpha_j x^j$ at the discrete points, ω^k, $k = 0, ..., n$. Write $\alpha(x)$ as:

$$\begin{aligned}\alpha(x) &= a\alpha_0 + \alpha_1 x + \alpha_2 x^2 + ... + \alpha_{n-1}x^{n-1} \\ &= (\alpha_0 + \alpha_2 x^2 + ...) + x(\alpha_1 + \alpha_3 x^2 + ...) \\ &= \alpha_e(x^2) + x\alpha_o(x^2)\end{aligned}$$

Here, α_e represents all even coefficients of α and α_o all odd coefficients. So, we are evaluating two polynomials of size $\frac{n}{2} - 1$ at the discrete points ω^{2k}. Continue doing this recursively in a *divide and conquer* approach, using Algorithm 14 from (Demmel, 1997) (assume for simplicity that $\alpha(x)$ is of length 2^m):

Algorithm 14 (RFPE) Recursive FFT Polynomial Evaluation for input vector α of size n

```
$b = Vector(\frac{n}{2} - 1)$
**if** $n == 1$ **then**
    RETURN $\alpha$
**else**
    $\hat{\alpha}_e = RFPE(\alpha_e(\frac{n}{2}))$
    $\hat{\alpha}_o = RFPE(\alpha_o(\frac{n}{2}))$
    $\omega = \exp^{-\frac{2\pi i}{n}}$
    **for** $j = 0$ to $\frac{n}{2} - 1$ **do**
        $b[j] = \omega^j$
    **end for**
    **for** $j = 0$ to $\frac{n}{2} - 1$ **do**
        $\hat{\alpha}[j] = \hat{a}_e[j] + b[j]\hat{\alpha}_o[j]$
        $\hat{\alpha}\left[\frac{n}{2} + j\right] = \hat{a}_e\left[\frac{n}{2} + j\right] + b\left[\frac{n}{2} - j\right]\hat{\alpha}_o\left[\frac{n}{2} + j\right]$
    **end for**
    RETURN $\hat{\alpha}$
**end if**
```

The recursion equation for the number of operations for a problem of size n can be written as $T(n) = 2T\left(\frac{n}{2}\right) + \frac{3n}{2}, T(1) = 1$. Using domain and range transformations (Siegel and Cole, 1999), $T(n) = \frac{3n}{2}lg(n)$.

Most FFT implementations do not use recursion; instead, they use nested loops to achieve speed on the order of $O(nlg(n))$ operations. Further, the fastest implementations place the input into bit-reversed order (Press et al., 1992; Demmel, 1997). The most popular and adapted of these algorithms seems to be the *Cooley-Tukey* algorithm (Cooley and J.W.Tukey, 1998) and its many variations. Going into the details of the bitwise manipulations of these fast implementations can be seen in these and other publications.

A variety of code versions of the FFT are available in (Press et al., 1992) and its online versions. Further, Netlib's FFTPACK, based on (Swartztrauber, 1982), contain a variety of Fortran FFT codes. FFTPACK appears to have been popular for a while, used in software such as MATLAB. However, recent years have seen the introduction of the Fastest Fourier Transform in the West (FFTW) (Frigo and Johnson, 1997). FFTW is based on the Cookey-Turkey algorithm and has been shown via benchmarks to provide the fastest implementation on a variety of platforms; these benchmarks can be seen in the papers (Frigo and Johnson, 1998) and most recently in (Frigo and Johnson, 2005), showing recent upgrades to the FFTW. The FFTW has become the de facto fast implementation, the code being publicly available, and MATLAB now bases its FFT code on the FFTW.

FFTs Applied to the Poisson Equation

Here we discuss a specific example for a two dimensional Poisson problem, discretized using a five point approximation scheme on a regular square grid. Much of this discussion is an expansion on section 1.1. We begin by assuming we are solving the 2D Dirichlet Poisson problem on a easily discretized grid (rectangular or circular for example):

$$-\Delta u = f \text{ in } \Omega \subset \mathcal{R}^2$$

$$u = 0 \text{ on } \partial\Omega.$$

Specifically, we are solving this problem on a space which is easily discretized in the x and y directions such that $dx = dy$. For example, let $\Omega \cup \partial\Omega$ represent a unit square. If we discretize our space into an $n \times n$ grid, we can evaluate $\frac{\partial^2 u}{\partial x^2}$ at the (i, j)th point as

$$\frac{\partial^2 u_{i,j}}{\partial x^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}$$

Let U be the $n \times n$ matrix whose entries are the variables, $u_{i,j}$, and T be the symmetric matrix of coefficients used to discretize $\frac{\partial^2 u_{i,j}}{\partial x^2}$ above. Then, T is just the 1D matrix of coefficients for the 1D Poisson Equation:

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

T is a *Toeplitz* matrix. A Toeplitz matrix is one whose entries are constant along each diagonal. Further, T is symmetric and tridiagonal. A Toeplitz symmetric tridiagonal matrix is referred to as *TST* (Iserles, 1996). TST matrices, as we will see, have many benefits, derived from their convenient form. Rewrite $\frac{\partial^2 u}{\partial x^2}$ as the product

$$-\frac{1}{h^2}(TU)$$

Perform a similar derivation for $\frac{\partial^2 u}{\partial y^2}$ and let B be the matrix where the $(i, j)$the entry of B is $b_{i,j} = -h^2 f_{i,j}$ and rewrite the Poisson equation.

$$TU + UT = B \tag{A.32}$$

Returning to TST matrices, we use the following Lemma, proved in (Iserles, 1996):

Theorem A.9. *(**TST Theorem**) If T is a TST matrix of size $n \times n$, with entries α along the main diagonal and β along both diagonals one away from the main diagonal, then T has eigenvalues, $\lambda_j = \alpha + 2\beta \cos\left(\frac{\pi j}{n+1}\right)$. Further, each eigenvalue has a corresponding eigenvector, $\mathbf{q}_j$ where $q_{j,k} = \left(\sqrt{\frac{2}{n+1}}\right) \sin\left(\frac{\pi jk}{n+1}\right)$. Here, $j = 1, 2, ..., n$.*

The power of this lemma is that we can write T as $Q\Lambda Q^T$ where Λ is the diagonal matrix of eigenvalues, λ_j and Q is an orthonormal matrix of eigenvectors, $\mathbf{q}_j$, where $Q^{-1} = Q^T = Q$ ($\subset \mathcal{R}$ for our example). Rewrite the discretized equation A.32 as

$$\begin{aligned}
& (Q\Lambda Q^T)U + U(Q\Lambda Q^T) = B \\
\Rightarrow \quad & Q^T(Q\Lambda Q^T)UQ + Q^T U(Q\Lambda Q^T)Q = Q^T BQ \\
\Rightarrow \quad & Q(Q\Lambda Q)UQ + QU(Q\Lambda Q)Q = QBQ \\
\Rightarrow \quad & \Lambda\hat{U} + \hat{U}\Lambda = \hat{B} \text{ where } \hat{(\cdot)} = Q(\cdot)Q \\
\Rightarrow \quad & (\Lambda\hat{U} + \hat{U}\Lambda)_{j,k} = \lambda_j \hat{u}_{j,k} + \hat{u}_{j,k}\lambda_j = \hat{b}_{j.k} \\
\Rightarrow \quad & \hat{u}_{j,k} = \frac{\hat{b}_{j,k}}{\lambda_j + \lambda_k}
\end{aligned}$$

Therefore, a straightforward algorithm for solving the 2D Dirichlet Poisson problem would be (Demmel, 1997):

1. Compute QBQ
2. Compute $\hat{u}_{j,k} = \dfrac{\hat{b}_{j,k}}{\lambda_j + \lambda_k} \; \forall \; j, k$
3. Compute $U = Q\hat{U}Q$

The second step involves $O(n^2)$ operations, and the first and third steps involve matrix multiplications, which would seem slow at $O(n^3)$ operations for a straightforward method. There are several algorithms which speed up dense matrix-matrix multiplications (Aho et al., 1974), but remember Λ and Q have the special structure derived from T being a TST matrix. Using fast Fourier transforms, in fact, Q times a vector involves $O(nlogn)$ operations, and Q times a matrix takes $O(n^2logn)$ operations. So, notice

$$\exp\left(\frac{-\pi ijk}{n+1}\right) = \cos\frac{\pi jk}{n+1} - i\sin\frac{\pi jk}{n+1}.$$

This implies the entries of Q are just the imaginary parts of a complex exponential times $-\sqrt{\frac{2}{n+1}}$. However, the left-hand side of the above equation looks similar to the DFT. In fact, if Φ is a $(2n+2)\times(2n+2)$ matrix whose (j,k)th entry is $\exp\left(\frac{-2\pi ijk}{2n+2}\right) = \exp\left(\frac{-\pi ijk}{n+1}\right)$, then if multiplication by Φ can be sped up using fast versions of the DFT, then we can also use the speed-up to multiply by Q. To see this further, notice that using *MATLAB*-like notation, $Q = Imaginary\left(\Phi(1:n, 1:n)\right)$. That is, Q is the first n rows and columns of the imaginary part of Φ. Therefore, for some vector, $v \in \mathcal{R}^n$ and $\bar{v} = [0, v^T, 0 * v^T]^T \in \mathcal{R}^{2n+1}$,

$$Qv = Imaginary\left((\Phi * \bar{v})(1:n)\right).$$

Assuming that we have an algorithm called FFT which can quickly compute the DFT of a matrix, D, we can now write the following pseudocode to solve a simple Dirichlet Poisson problem on a regular grid, first showing Algorithm 15 for the Fast Sine Transform, and the full solver in Algorithm 16 (Demmel, 1997).

The pseudocode algorithm above takes some unnecessary steps in order to be more explicit, such as computing the matrix LL; other parts of the code can be sped up using memory storage exploitation (this is straightforward in MATLAB). Additionally, in order to use the complex part of the Fourier transform, we can compute the use FFTs to compute the discrete sine transform.

Algorithm 15 (FST) Fast Sine Transform on Matrix D of size $n \times m$

```
$z =$ zero-vector of length $m$
$D_1 = [z; D]$ (A new matrix whose first row is z and 2nd to $m+1$st rows are $D$)
for $j = 1$ to $n+1$ do
    $D_1 = [D_1; z]$ (Append $n+1$ rows of zeros - can be done in $O(1)$)
end for
$D_2 = Im(FFT(D_1))$ (Imaginary part of the FFT of $D_1$)
RETURN $D_2(2 : n+1, :)$ (The 2nd to $n+1$st rows of $D_2$)
```

Algorithm 16 FFT Poisson Solver

```
$L = Vector(n)$ (Vector of eigenvalues - diagonal($\Lambda$))
$LL = Matrix(n, n)$ (Will store reciprocal sums of L)
$U = Vector(n)$
$temp = \frac{2}{n+1}$
for $j = 1$ to $n$ do
    $L[j] = 2 - 2\left(\frac{\cos jn}{n+1}\right)$
end for
for $j = 1$ to $n$ do
    for $k = 1$ to $n$ do
        $LL(j,k) = \frac{temp}{L(j)+L(k)}$
    end for
end for
$U = FST(B^T)$
$U = FST(X^T)$
for $j = 1$ to $n$ do
    for $k = 1$ to $n$ do
        $U(j,k) = LL[j,k]U[j,k]$
    end for
end for
$U = FST(U^T)$
$U = FST(U^T)$
```

Despite the inefficiencies of the above algorithm, we can see how the FFT allows us to more quickly compute the multiplications.

(Iserles, 1996) explains a similar yet slightly different approach to an FFT-based method, known as the *Hockney* method. In the Hockney method, we assume that for the system of linear equations $Au = b$, A is a *block-TST* matrix as discussed in section 1.1. As a summary of the discussion there, the Hockney method involves the following steps:

$\mathbf{1}$. Form $\hat{B} = QB$, or $\hat{b}_j = Qb_j$ for each $j = 1, ...n$

$\mathbf{2}$. Change to $\hat{B}^T$ whose columns are $\hat{b}^t_j$, the rows of $\hat{B}$

$\mathbf{3}$. Solve $\Gamma_j \hat{u}^t_j = \hat{b}^t_j$ via band Cholesky for $j = 1, ..., n$

$\mathbf{4}$. Change back to $\hat{U}$ from $\hat{U}^T$

$\mathbf{5}$. Set $U = Q\hat{U}$ or $u_j = Q\hat{u}_j$ for $j = 1, ..., n$

Cyclic Reduction

We mention another method, known as *cyclic reduction*, *cyclic odd-even reduction* and *block cyclic reduction*. Cyclic reduction is a generalization of the Hockney method for the FFT Poisson solver. Many sources exist for discussion of cyclic reduction including (Buzbee et al., 1969; Buzbee et al., 1970; Swartztrauber, 1974), the latter more explicitly applied to the Poisson equation. A general overview of cyclic reduction compared to FFTs can be seen in (Swartztrauber and Sweet, 1996).

For an $n \times n$ system, $Au = b$, pick the jth row and multiply it by $-A$ and then add it to the $j-1$st and $j+1$st row to get the following result:

$$\begin{array}{lllllll} & +u_{j-2} & +Au_{j-1} & +u_j & & & = b_{j-1} \\ +(-A)* & (& +u_{j-1} & +Au_j & +u_{j+1} & & = b_{j-2}) \\ + & & & +u_j & +Au_{j+1} & +u_{j+2} & = b_{j+1} \end{array}$$

$$\Rightarrow \quad u_{j-2} + (2I - A^2)u_j + u_{j-2} = b_{j-1} - Ab_j + b_{j+2}$$

Repeat the above process for every third set of equations. Setting $\tilde{A} = 2I - A^2$ and $\tilde{b}_j = b_{j-1} - Ab_j + b_{j+1}$, we get a new system in which

$$\begin{bmatrix} \tilde{A} & I & & & & & \\ I & \tilde{A} & I & & & & \\ & I & \tilde{A} & I & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & I & \tilde{A} & I & \\ & & & & I & \tilde{A} & I \\ & & & & & I & \tilde{A} \end{bmatrix} \begin{bmatrix} u_2 \\ u_4 \\ \vdots \\ u_{n-3} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} \tilde{b}_2 \\ \tilde{b}_4 \\ \vdots \\ \tilde{b}_{n-3} \\ \tilde{b}_{n-1} \end{bmatrix}.$$

We have reduced the number of unknowns by half (Swartztrauber and Sweet, 1996). Repeat this recursively until we have reduced the number of unknowns to 1 (we assume that $n = 2^m - 1$). Once the base equation is solved, work backwards to solve a series of equations to build the final result. Algorithms for this process can be seen in (Press et al., 1992; Demmel, 1997). We reproduce the pseudocode here from (Demmel, 1997) in Algorithm 17. Let $n_0 = 2^{j+1} - 1$, $n_r = 2^{j+1-r} - 1$, $A^{(0)} = A$, $A^{(r)} = 2I - (A^{(r-1)})^2$, $b_j^{(0)} = b_j$, $b_j^{(r)} = b_{2j-1}^{(r-1)}) - A^{(r)}b_{2j}^{(r-1)}) + b_{2j+1}^{(r-1)})$. Further γ is considered a stopping point for the iteration.

Unfortunately, this algorithm is numerically unstable as the matrices, $A^{(r)}$ grow exponentially as r grows (Swartztrauber and Sweet, 1996). Hence, $b_j^{(r)}$ loses all of its numerical sig-

Algorithm 17 Block Cyclic Odd-Even Reduction for matrix A of size $n \times n$, vector b of size n, γ

```
for $r = 0$ to $\gamma - 1$ do
    Compute $A^{(r+1)}$
    for $k = 1$ to $n_{r+1}$ do
        Compute $b_k^{(r+1)}$
    end for
end for
Solve $A^{(\gamma)} u^{(\gamma)} = b^{(\gamma)}$ using direct or iterative method
for $r = \gamma - 1$ to 0 do
    for $k = 1$ to $n_{r+1}$ do
        $u_{2k}^{(r)} = u_k^{(r+1)}$
    end for
    for $k = 1$ to $n_r$ do
        Solve $A^{(r)} u_k^{(r)} = b_k^{(r)} - u_{k-1}^{(r)} - u_{k+1}^{(r)}$ using direct or iterative method
        $u_{k+1}^{(r)}$ and $u_{k-1}^{(r)}$ are either known from the previous step or are known boundary conditions
    end for
end for
RETURN $u^{(0)}$
```

nificance. Fortunately, (Buneman, 1969) provides a way of stabilizing the system by assuming $b_j^{(r)} = A^{(r)} p_j^{(r)} + q_j^{(r)}$, where the p's and q's are developed via additional recurrence relations. (Buzbee et al., 1970; Buzbee et al., 1971) use this variation applied directly to the Poisson equation.

If we begin with $n = 2^m - 1$, we solve $n(m+1) = nlg(n) + n$ tridiagonal systems (Swartztrauber and Sweet, 1996), which using band Cholesky can be solved in $O(m)$ operations for $m < n$. Therefore, cyclic reduction, in theory, is quite fast. However, in practice, using the stabilization method of (Buneman, 1969; Swartztrauber and Sweet, 1996) claims that cyclic reduction is slower than FFTs for a Poisson problem.

BIBLIOGRAPHY

Adams, M. (1998). *Multigrid Equation Solvers for Large-Sale Nonlinear Finite Element Simulations*. Ph.d. thesis, University of California, Berkeley, Berkeley, CA.

Adams, M. F. (2004). Algebraic multrigrid methods for constrained linear systems with applications to contact problems in solid mechanics. *Numerical Linear Algebra with Applications*, 11(2-3):141–153.

Adams, M. F., Bayraktar, H., Keaveny, T., and Papadopoulos, P. (2004). Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom. In *ACM/IEEE Proceedings of SC 2004: High Performance Networking and Computing*.

Adams, M. F., Brezina, M., Hu, J. J., and Tuminaro, R. S. (2003). Parallel multigrid smoothing: polynomial versus Gauss-Seidel. *Journal of Computational Physics*, 188(2):593–610.

Adams, M. F. and Demmel, J. (1999). Parallel multigrid solver algorithms and implementations for 3D unstructured finite element problem. In *ACM/IEEE Proceedings of SC 1999: High Performance Networking and Computing*, Portland, Oregon.

Aftosmis, M., Berger, M., and Melton, J. (1998). Adaptive Cartesian mesh generation. In Thompson, J., editor, *The Handbook of Grid Generation*, pages 22–1–22–26. CRC Press.

Ahlfors, L. (1979). *Complex Analysis*. McGraw-Hill, 3rd edition.

Aho, A., Hopcroft, J., and Ullman, J. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Anderson, C. (1986). A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *Journal of Computational Physics*, 62(1):111–123.

Aslam, T. (2003). A partial differential equation approach to multidimensional extrapolation. *Journal of Computational Physics*, 193:349–355.

Atkinson, K. (1997). *The Numerical Solution of Integral Equations of the Second Kind: Volume 4 of Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press.

Baker, C. (1977). *The Numerical Treatment of Integral Equations: Monographs on Numerical Analysis*. Clarendon Press, Oxford.

Balls, G. and Colella, P. (2002). A finite difference domain decomposition method using local corrections for the solution of Poisson's equation. *Journal of Computational Physics*, 180(1):25–53.

Barnes, J. and Hut, P. (1986). A hierarchical O(N log N) force calculation algorithm. *Nature*, 324:446–449. Technical report.

Beatson, R. and Greengard, L. (1997). A short course on fast multipole methods. In Ainsworth, M. et al., editors, *Wavelets, multilevel methods and elliptic PDEs*, pages 1–37, Walton Street, Oxford OX2 6DP, UK. Oxford University Press.

Berger, M., Aftosmis, M., and Melton, J. (1996). Accuracy, adaptive methods and complex geometry. In *1st AFOSR Conference on Dynamic Motion CFD*.

Berger, M. and Colella, P. (1989). Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82:64–84.

Berger, M. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512.

Berman, C. and Greengard, L. (1994). A renormalization method for the evaluation of lattice sums. *Journal of Mathematical Physics*, 35(6036–6049).

Berntsen, J., Espelid, T. O., and Genz, A. (1991). Algorithm 698; dcuhre: an adaptive multidemensional integration routine for a vector of integrals. *ACM Trans. Math. Softw.*, 17(4):452–456.

Bloomenthal, J. (1994). An implicit surface polygonizer. *Graphics Gems*, 4:324–349.

Börm, S. (2006). H2-matrix arithmetics in linear complexity. *Computing*, 77(1):1–28.

Börm, S. and Hackbusch, W. (2005). Hierarchical quadrature for singular integrals. *Computing*, 74(2):75–100.

Boschitsch, A., Fenley, M., and Olson, W. (1999). A fast adaptive multipole algorithm for calculating screened coulomb (yukawa) interactions. *Journal of Computational Physics*, 151:212–241.

Braess, D. (2001). *Finite Elements: Theory, Fast Solvers and Applications in Solid Mechanics*. Cambridge University Press, 2nd edition.

Brandt, A. (1977). Multilevel adaptive solutions to boundary value problems. *Math. Comp.*, 31:333–390.

Brebbia, C., Telles, J., and Wrobel, L. (1984). *Boundary element techniques*. Springer-Verlag,, Berlin.

Brezina, M., Falgout, R., MacLachlan, S., Manteuffel, T., McCormicj, S., and Ruge, J. (Submitted September 7, 2004a). Adaptive algebraic multigrid. *SIAM Journal on Scientific Computing*.

Brezina, M., Tong, C., and Becker, R. (Submitted for Publication. Also available as LLNL Technical Report UCRL-JRNL-204167, 2004b). Parallel algebraic multigrids for structural mechanics. *SIAM Journal on Scientific Computing*.

Briggs, L., Emden Henson, V., and McCormick, S. F. (2000). *A Multigrid Tutorial*. SIAM, Philadelphia.

Bruno, O. P. and Kunyansky, L. A. (2001). A fast, high-order algorithm for the solution of surface scattering problems: Basic implementation, tests, and applications. *Journal of Computational Physics*, 169:80–110.

Buneman, O. (1969). A compact non-iterative Poisson solver. Technical Report 294, Institute for Plasma Research, Stanford University, Stanford, CA.

Buzbee, B., Golub, G., and Nielson, C. (1969). The method of odd/even reduction and factorization with application to Poisson's equation. Technical Report CS-TR-69-128, Stanford University, Department of Computer Science.

Buzbee, B., Golub, G., and Nielson, C. (1970). On direct methods for solving Poisson's equation. *SIAM Journal on Numerical Analysis*, 7:627–656.

Buzbee, B., Golub, G., and Nielson, C. (1971). The direct solution of the discrete Poisson equation on irregular regions. *SIAM Journal on Numerical Analysis*, 8:722–736.

Canuto, C., Hussaini, M. Y., Quarteroni, A., and Zang, T. A. (1987). *Spectral Methods in Fluid Dynamics*. Springer–Verlag, New York.

Chan, T., Glowinski, R., Périaux, J., , and Widlund, O. (1989). *Domain decomposition Methods*. SIAM, Philadelphia.

Chan, T. and Smith, B. (1994). Domain decomposition and multigrid algorithms for elliptic problems on unstructured meshes. *Electron. Trans. Numer. Anal.*, 1994:171–182.

Chapman, B., Jost, G., and Pas, R. (2007). *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press.

Chen, G. and Zhou, J. (1992). *Boundary Element Methods*. Academic Press Limited, London.

Cheng, H., Crutchfield, W., Gimbutas, Z., Greengard, L., Ethridge, J., Huang, J., Rokhlin, V., Yarvin, N., and Zhao, J. (2006a). A wideband fast multipole method for the Helmholtz equation in three dimensions. *Journal of Computational Physics*, 216:300–325.

Cheng, H., Greengard, L., and Rokhlin, V. (1999). A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*, 155:468–498.

Cheng, H., Huang, J., and Leiterman, T. J. (2006b). An adaptive fast solver for the modified Helmholtz equation in two dimensions. *Journal of Computational Physics*, 211(2):616–637.

Cheng, L., Fedkiw, R., Gibou, F., and Kang, M. (2001). A second order accurate symmetric discretization of the Poisson equation on irregular domains. *Journal of Computational Physics*, 176:205–227.

Chesshire, G. and Henshaw, W. (1991). Composite overlapping meshes for the solution of partial differential equations. *Journal of Computational Physics*, 90:1–64.

Chorin, A. and Marsden, J. (1993). *A Mathematical Introduction to Fluid Dynamics*. Springer-Verlag, 3rd edition.

Cooley, J. and J.W.Tukey (1998). An algorithm for the machine computation of the complex fourier series. *Mathematics of Computation*, 19:297–301.

Cottet, G.-H. and Koumoutsakos, P. (2000). *Vortex Methods, Theory and Practice*. Cambridge University Press, 1st edition.

Demmel, J. (1996). Fast hierarchical methods for the n-body problem, part 1: http://www.cs.berkeley.edu/ demmel/cs267/lecture26/lecture26.html. http://www.cs.berkeley.edu/ demmel/cs267/lecture25/lecture25.html.

Demmel, J. (1997). *Applied Numerical Linear Algebra*. SIAM.

Elman, H., Silvester, D., and Wathen, A. (2005). *Finite Elements and Fast Iterative Solvers*. Oxford University Press, 1st edition.

Ethridge, F. (2000). *Fast Algorithms for Volume Integrals in Potential Theory*. Ph.d. thesis, New York University, New York, NY.

Ethridge, F. and Greengard, L. (2001). A new fast-multipole accelerated Poisson solver in two dimensions. *SIAM Journal on Scientific Computing*, 23(3):741–760.

Fedkiw, R., Aslam, T., Merriman, B., and Osher, S. (1999). A non-oscillatory eulerian approach to interfaces in multimaterial flows. *Journal of Computational Physics*, 152:457–492.

Fournier, A. and Montuno, D. (1984). Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics*, 3:153–174.

Frigo, M. and Johnson, S. (1997). The fastest fourier transform in the west. http://theory.lcs.mit.edu/ fftw/fftw-paper.ps.gz.

Frigo, M. and Johnson, S. (1998). Fftw: An adaptive software architecture for the fft. *ICASSP Conference Proceedings*, 3:1381–1384.

Frigo, M. and Johnson, S. (2005). The design and implementation of fftw3. *Proceedings of the IEEE*, 93(2):216–231.

Gibbon, P. and G.Sutmann (2002). *Long-Range Interactions in Many-Particle Simulations in Quantum Simulatinos of Complex Many-Body Systems From Theory to Algorithms NIC Series Volume 10*, pages 467–506. John von Neumann Institute for Comuting, Jülich.

Gingold, R. and Monaghan, J. (1977). Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375–389.

Greengard, L. (1988). *The Rapid Evaluation of Potential Fields in Particle Systems: ACM Distringuished Dissertation*. MIT Press.

Greengard, L. (1994). Fast algorithms for classical physics. *Science*, 265(5174):909–914.

Greengard, L. and Huang, J. (1999). A fast direct solver for elliptic partial differential equations on adaptively refined meshes. *SIAM Journal on Scientific Computing*, 21:1551–1566.

Greengard, L. and Huang, J. (2002). A new version of the fast multipole method for screened coulomb interactions in three dimensions. *Journal of Computational Physics*, 180:642–658.

Greengard, L., Kropinski, M., and Mayo, A. (1996). Integral equation methods for Stokes flow and isotropic elasticity in the plane. *Journal of Computational Physics*, 125:403–414.

Greengard, L. and Lee, J. (1996). A direct adaptive Poisson solver of arbitrary order accuracy. *Journal of Computational Physics*, 125:415–424.

Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348. This paper is credited as the origin of the fast multipole method, with an $O(N)$ algorithm. It was reprinted in the same journal, vol. 135, pp. 280–292, August 1997.

Greengard, L. and Rokhlin, V. (1988). The rapid evaluation of potential fields in three dimensions. In Anderson, C. and Greengard, C., editors, *Vortex Methods*, Lecture Notes in Mathematics. Springer Verlag, N.Y.

Greengard, L. and Rokhlin, V. (1997). A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6:229–269.

Guenther, R. and Lee, J. (1988). *Partial Differential Equations of Mathematical Physics and Integral Equations*. Dover.

Gumerov, N. and Duraiswami, R. (2006). Fast multipole method for the biharmonic equation in three dimensions. *Journal of Computational Physics*, 215:363–383.

Hackbusch, W. (1985). *Multigrid Methods and Applications*. Springer, 1st edition.

Hackbusch, W. (1995). *Integral Equations: Volume 120 of the International Series of Numerical Mathematics*. Birkhauser Verlag, translated and revised from 1989 german original edition.

Hackbusch, W. (1999). A sparse matrix arithmetic based on h-matrices. part i: introduction to h-matrices. *Computing*, 62(2):89–108.

Hackbusch, W. and Börm, S. (2002). H2-matrix approximation of integral operators by interpolation. *Appl. Numer. Math.*, 43(1-2):129–143.

Hackbusch, W. and Nowak, Z. (1989). On the fast matrix multiplication in the boundary element method by panel clustering. *Numerische Mathematik*, 54(4):463–491.

Hackbusch, W. and Trottenberg, U. (1982). *Multigrid Methods, Lecture Notes in Mathematics Volume 960*. Springer-Verlag, 1st edition.

Helsing, J. (1994). Fast and accurate calculations of structural parameters for suspensions. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 445:127–140.

Hockney, R. and Eastwood, J. (1981). *Computer Simulation Using Particles*. McGraw-Hill, NY.

Hunter, P. and Pullan, A. (2002). Fem/bem notes. http://www3.esc.auckland.ac.nz/people/staff/apul001/publications.html.

Iserles, A. (1996). *Numerical Analysis of Differential Equations*. Cambridge University Press, 1st edition.

Johansen, H. and Colella, P. (1998). A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *Journal of Computational Physics*, 147:60–85.

Johnson, C. (1987). *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Cambridge University Press.

Kellogg, O. (1967). *Foundations of Potential Theory*. Springer-Verlag.

Kress, R. (1989). *Linear Integral Equations: Applied Mathematical Sciences 82*. Springer-Verlag.

Kress, R. (1999). *Linear Integral Equations: Applied Mathematical Sciences 82*. Springer-Verlag, 2nd edition.

Ladyzhenskaya, O. (1964). *The Mathematical Theory of Viscous Incompressible Flow*. Gordon and Breach, 2nd edition.

Langston, M., Greengard, L., and Zorin, D. (2011). A free-space adaptive fmm-based pde solver in three dimensions. *Communications in Applied Mathematics and Computational Science*, 6(1):79–122.

Lashuk, I., Chandramowlishwaran, A., Langston, M., Nguyen, T., Sampath, R., Shringarpure, A., Vuduc, R., Ying, L., Zorin, D., and Biros, G. (2009). A massively parallel adaptive fast-multipole method on heterogeneous architectures. In *Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, November 14-20, 2009, Portland, Oregon, USA*, Portland, OR. IEEE/ACM SIGARCH.

Lischinski, D. (1994). Incremental delauney triangulation. *Graphics Gems*, 4:47–59.

Liu, X.-D., Fedkiw, R. P., and Kang, M. (2000). A boundary condition capturing method for Poisson's equation on irregular domains. *Journal of Computational Physics*, 160:151–178.

Lucy, L. (1977). A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024.

Majda, A. and Bertozzi, A. (2002). *Vorticity and Incompressible Flow*. Cambridge University Press.

Martin, D. (1998). *An Adaptive Cell-Centered Projection Method for the Incompressible Euler Equations*. Ph.d. thesis, University of California, Berkeley, Berkeley, CA.

Martin, D. and Cartwright, K. (1996a). Amr Poisson code. http://seesar.lbl.gov/anag/staff/martin/AMRPoisson.html.

Martin, D. and Cartwright, K. (1996b). Solving Poisson's equations using adaptive mesh refinement. Technical Report M96/66, University of California, Berkeley Electronic Research Laboratory.

Mavripilis, D. (1997). Unstructured grid techniques. *Annual Review of Fluid Mechanics*, 29:473–514.

Mayo, A. (1984). The fast solution of Poisson's and the biharmonic equations on irregular regions. *SIAM Journal on Numerical Analysis*, 21(2):285–299.

Mayo, A. (1985). Fast high order accurate solution of Laplace's equation on irregular regions. *SIAM Journal on Scientific Computing*, 6(1):144–157.

Mayo, A. and Greenbaum, A. (1992). Fast parallel iterative solution of Poisson's and the biharmonic equations on irregular regions. *SIAM Journal on Scientific Computing*, 13(1):101–118.

McCorquodale, P., Colella, P., Balls, G., and Baden, S. (2007). A local corrections algorithm for solving Poisson's equation in three dimensions. *Communications in Applied Mathematics and Computational Science*, 2(1):57–81.

McKenney, A., Greengard, L., and Mayo, A. (1995). A fast Poisson solver for complex geometries. *Journal of Computational Physics*, 118:348–355.

McLean, W. (2004). Poisson solvers. http://www.cs.northwestern.edu/ jet/.

Mikhlin, S. (1964). *Integral Equations and Their Applications to Certain Problems in Mechanics, Mathematical Physics and Technology*. Pergamon Press, MacMillian Company, NY, NY, 2nd: translated from russian by a.h. armstrong edition.

Minion, M. (1999). A projection method for locally refined grids. *Journal of Computational Physics*, 148(1):81–124.

Monaghan, J. (1992). Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30(1):543–574.

Osher, S. and Fedkiw, R. (2002). *Level Set Methods and Dynamic Implicit Surfaces*. Springer, Berlin.

Peskin, C. S. (1977). Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, pages 200–252.

Peskin, C. S. and McQueen, D. M. (1989). A three-dimensional computational method for blood flow in the heart. I: Immersed elastic fibers in a viscous incompressible fluid. *Journal of Computational Physics*, 81:372–405.

Peskin, C. S. and Prinz, B. (1993). Improved volume conservation in the computation of flows with immersed elastic boundaries. *Journal of Computational Physics*, 105:113–132.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes in C, 2nd. edition*. Cambridge University Press.

Rayleigh, L. (1892). On the influence of obstacles arranged in rectangular order upon the properties of a medium5a. *Philosophical Magazine*, 34:481–502.

Rodin, G. and Overfelt, J. (2004). Periodic conduction problems: The fast multipole method and convergence of integral equations and lattice sums. *The Royal Society Proceedings: Mathematical, Physical and Engineering Sciences*, 460(2050):2883–2902.

Rokhlin, V. (1985). Rapid solution of integral equations of classical potential theory. *Journal of Computational Physics*, 60:187–207.

Rokhlin, V. (1990). Rapid solution of integral equations of scattering theory in two dimensions. *Journal of Computational Physics*, 86:414–439.

Rosner, R., Calder, A., Dursi, J., Fryxell, B., Lamb, D., Niemeyer, J., Olson, K., Ricker, P., Timmes, F., Truran, J., Tueo, H., Young, Y.-N., Zingale, M., Lusk, E., and Stevens, R. (2000). Flash code: studying astrophysical thermonuclear flashes. *Computing in Science and Engineering*, 2(2):33–41.

Ruge, J. W. and Stüben, K. (1987). Algebraic multigrid. In McCormick, S. F., editor, *SIAM Frontiers on Applied Mathematics, Volume 3, Multigrid Methods*, pages 73–130. SIAM.

Russo, G., Strain, J., and Vetoio, V. (1994). Fast triangulated vortex methods for the 2-D euler equations. *Journal of Computational Physics*, 111:291–323.

Salmon, J. K. and Warren, M. S. (1994). Skeletons from the treecode closet. *Journal of Computational Physics*, 111:136–155.

Seidel, J. (1991). A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygones. *Computational Geometry: Theory and Applications*, 1:51–64.

Shewchuk, J. (1994). An introduction to the conjugate gradient method without the agonizing pain. http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf.

Siegel, A. and Cole, R. (1999). *An Inside Guide to Algorithms: Their Application, Design and Analysis*. Not Yet Published.

Sloan, I. (1992). Error analysis of boundary integral methods. *Acta Numerica*, pages 287–339.

Strain, J. (1997). Fast adaptive 2D vortex methods. *Journal of Computational Physics*, 132:108–122.

Strain, M., Scuseria, G., and Frisch, M. (1996). Achieving linear scaling for the electronic quantum Coulomb problem. *Science*, 271:51–53.

Strang, G. (1988). *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, 3rd edition.

Sundar, H., Sampath, R., and Biros, G. (2008). Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30:2675–2708.

Swartztrauber, P. (1974). A direct method for the discrete solution of separable elliptic equations. *SIAM Journal on Numerical Analysis*, 11:1136–1150.

Swartztrauber, P. (1982). *Vectorizing the FFTs*, pages 51–83. Academic Press.

Swartztrauber, P. (1984). *Studies in Numerical Analysis: Fast Poisson Solvers*, volume 24, pages 319–370. Mathematical Association of America.

Swartztrauber, P. and Sweet, R. (1979). Algorithm 541: Efficient FORTRAN subprograms for the solution of separable elliptic partial differential equations. *ACM Transactions on Mathematical Software*, 5(3):352–364.

Swartztrauber, P. and Sweet, R. (1996). *Handbook of Fluid Dynamics and Fluid Machinery: The Fourier and Cyclic Reduction Methods for Solving Poisson's Equation*. John Wiley and Sons.

Tor, S. and Middleditch, A. (1984). Convex decomposition of simple polygons. *ACM Transactions on Graphics*, 3:244–265.

Tornberg, A.-. and Greengard, L. (2008). A fast multipole method for the three-dimensional Stokes equations. *Journal of Computational Physics*, 227(3):1613–1619.

Trefethen, L. and Bau, D. (1997). *Numerical Linear Algebra*. SIAM.

Wang, H., Lei, T., Li, J., Huang, J., and Yao, Z. (2007). A parallel fast multipole accelerated integral equation scheme for the 3d stokes equations. *Journal of Numerical Methods in Engineering*, 70:812–839.

Weir, A. (1973). *Lebesgue Integration and Measure*. Cambridge University Press.

Wendland, W. (1985). *On some Mathematical Aspects of Boundary Element Methods for Elliptic Problems: In The Mathematics of Finite Elements and Applications, V*, pages 193–227. Academic Press, London.

White, C., Johnson, B., Gill, P., and Head-Gordon, M. (1994). The continuous fast multipole method. *Chemical Physics Letters*, 230:8–16.

Y. Fu, G. R. (2000). Fast solution method for three-dimensional Stokesian many-particle problems. *Communications in Numerical Methods in Engineering*, 16(2):145–149.

Ying, L. (2004). *An Efficient and High-Order Accurate Boundary Integral Solver for the Stokes Equations in Three Dimensional Complex Geometries*. PhD thesis, New York University.

Ying, L., Biros, G., and Zorin, D. (2004a). A fast solver for the Stokes equations with distributed forces in complex geometries. *Journal of Computational Physics*, 194(1):317–348.

Ying, L., Biros, G., and Zorin, D. (2004b). A kernel-independent adaptive fast multipole method in two and three dimensions. *Journal of Computational Physics*, 196(2):596–626.

Ying, L., Biros, G., and Zorin, D. (2006). A high-order boundary integral equation solver for elliptic pdes in smooth domains. *Journal of Computational Physics*, 219(1):247–275.

Ying, L., Biros, G., Zorin, D., and Langston, M. (2003). A new parallel kernel-independent fast multipole method. In *SC 2003 Conference CD*, Phoenix, AZ. IEEE/ACM SIGARCH.

Zorin, D., Schröder, P., DeRose, A., Kobbelt, L., Levin, A., and Sweldens, W. (2000). Subdivision for modeling and animation, siggraph 2000 course notes. http://www.cs.nyu.edu/ dzorin/sig00course/.