ABSTRACT

Title of dissertation: FAST AND ACCURATE BOUNDARY ELEMENT METHODS IN THREE DIMENSIONS

Ross Adelman, Doctor of Philosophy, 2016

Dissertation directed by: Professor Ramani Duraiswami
Department of Computer Science

The Laplace and Helmholtz equations are two of the most important partial differential equations (PDEs) in science, and govern problems in electromagnetism, acoustics, astrophysics, and aerodynamics. The boundary element method (BEM) is a powerful method for solving these PDEs. The BEM reduces the dimensionality of the problem by one, and treats complex boundary shapes and multi-domain problems well. The BEM also suffers from a few problems. The entries in the system matrices require computing boundary integrals, which can be difficult to do accurately, especially in the Galerkin formulation. These matrices are also dense, requiring $\mathrm{O}\left(N^2\right)$ to store and $\mathrm{O}\left(N^3\right)$ to solve using direct matrix decompositions, where N is the number of unknowns. This can effectively restrict the size of a problem.

Methods are presented for computing the boundary integrals that arise in the Galerkin formulation to any accuracy. Integrals involving geometrically separated triangles are non-singular, and are computed using a technique based on spherical harmonics and multipole expansions and translations. Integrals involving triangles that have common vertices,

edges, or are coincident are treated via scaling and symmetry arguments, combined with recursive geometric decomposition of the integrals.

The fast multipole method (FMM) is used to accelerate the BEM. The FMM is usually designed around point sources, not the integral expressions in the BEM. To apply the FMM to these expressions, the internal logic of the FMM must be changed, but this can be difficult. The correction factor matrix method is presented, which approximates the integrals using a quadrature. The quadrature points are treated as point sources, which are plugged directly into current FMM codes. Any inaccuracies are corrected during a correction factor step. This method reduces the quadratic and cubic scalings of the BEM to linear.

Software is developed for computing the solutions to acoustic scattering problems involving spheroids and disks. This software uses spheroidal wave functions to analytically build the solutions to these problems. This software is used to verify the accuracy of the BEM for the Helmholtz equation.

The product of these contributions is a fast and accurate BEM solver for the Laplace and Helmholtz equations.

FAST AND ACCURATE BOUNDARY ELEMENT METHODS IN THREE DIMENSIONS

by

Ross Adelman

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

Advisory Committee:
Professor Ramani Duraiswami, Chair/Advisor
Dr. Nail A. Gumerov
Professor Howard Elman
Professor Amitabh Varshney
Professor Thomas Antonsen, Dean's Representative

Acknowledgments

This dissertation is the result of a lot of time and hard work spent studying, reading, programming, tinkering, testing, experimenting, and writing. Even with all of the effort I put in, it would never have been possible without the help and support of colleagues, friends, family, and all of the scientists that came before me.

First, I would like to thank the pioneers of mathematical physics, Pierre-Simon Laplace, George Green, Carl Friedrich Gauss, Vladimir Rokhlin, Leslie Greengard, and many others, as well as the tens of thousands of mathematicians, scientists, and engineers that built upon their work. When reading through the scientific literature, I was always amazed at the sheer volume of great work that had been done on the topic. Without this prior work, my research and this dissertation would never have been possible.

Second, I would like to thank my advisor, Ramani Duraiswami, for guiding my research and keeping me on track. I could always rely on him for solid advice on not just my research, but also my career in general. Nail Gumerov for helping me understand some of the more nitty gritty mathematics that I sometimes encountered in my research. His knowledge of mathematics and physics is unmatched by anyone else I know. And my other committee members, Howard Elman, Amitabh Varshney, and Thomas Antonsen, for taking the time to read this dissertation and proving great feedback.

Third, I would like to thank David Hull at the Army Research Laboratory (ARL) for hiring me as an engineering intern back in high school (thus starting my career in engineering and science) and keeping me around through college, graduate school, and soon full-time. I would also like to thank the other engineers at ARL: Stephen Vinci,

Simon Ghionea, Sean Heintzelman, Brandon Parks, David Bartlett, and Kevin Claytor.

Fourth, I would like to thank all my fellow students at school and ARL that I have worked with over the years: Qi Hu, Yuancheng Luo, Sam Potter, Darren Smith, Phil Sandborn, and Jack Zhu. None of this would have been nearly as fun without them.

Lastly, but most importantly, I would like to thank my mom and dad for their unwavering love and support throughout my entire life. I could always count on them for help whenever I most needed it. And the rest of my family, Brad, Evan, Colleen, and Jack, for their moral support.

Table of Contents

List of Figures

Chapter 1: Introduction

The Laplace equation in three dimensions is one of the most important, perhaps the most important, partial differential equations (PDEs) in science, and governs problems in a large number of disciplines, including electrostatics [2, 3], magnetostatics, plasma physics [4], astrophysics [5], molecular dynamics, aerodynamics [6], and even in computer graphics for character animation [7] and collision detection [8]. The Helmholtz equation in three dimensions is another very important PDE, and is found in problems involving acoustics [9], electromagnetism [10], heat and other forms of diffusion, and even in the study of gravitational waves, which were recently detected experimentally [11]. In fact, many other equations in mathematical physics are highly related to these two PDEs, and in many cases, can be reduced to them. These include, for example, the polyharmonic, elasticity, and Stokes equations. Therefore, having fast and accurate numerical solvers for these problems is important. Of course, accuracy is required for all numerical solvers, as they should mimic reality as closely as possible. Speed is also essential. A solver that can solve large problems on a local deskside machine in minutes (or even seconds) is incredibly powerful. During their research and development process, a scientist or engineer can interactively experiment with a problem to improve their design, or simply to better understand the underlying physics.

The Laplace and Helmholtz equations are given by, respectively,

$$\nabla^2\phi = 0, \quad \nabla^2\phi + k^2 = 0, \tag{1.1}$$

where, in the Helmholtz equation, k is the wavenumber. The Laplace equation is actually a special case of the Helmholtz equation (i.e., $k = 0$), so they share some of the same properties. The boundary element method (BEM), also known as the method of moments, is a powerful method for solving these PDEs in three dimensions [12–14]. The BEM works by transforming the PDE from a differential equation into an integral equation using Green's identity:

$$\pm\phi = L\left[q\right] - M\left[\phi\right], \tag{1.2}$$

where L and M are the single- and double-layer potentials [15]. Green's identity is powerful: we can compute the potential at any point as long as we know the potential, ϕ, and normal derivative, q, on the boundary. In the differential equation, we seek a solution to the potential throughout the entire domain. However, in the integral equation, we seek a solution to the potential and normal derivative only on the boundary. The dimensionality of the problem is, therefore, reduced by one. The BEM has a number of other advantages. The method allows for the treatment of complex boundary shapes, handles the boundary conditions at infinity accurately, and treats thin objects and multi-domain problems well.

The equations are discretized via boundary elements (e.g., triangles, quadrilaterals, or their curved equivalents), and when combined with boundary conditions, result in linear systems. The entries in these matrices are based on the quadrature of the product of local

basis functions and Green's functions or their derivatives over the boundary elements. In the collocation method, these entries are obtained by a quadrature of the Green's function integrand over one boundary element, while in the Galerkin method, they are obtained by a double quadrature over two boundary elements. There are advantages to Galerkin formulations for integral equations, as they treat problems associated with kernel singularity, and lead to symmetric and better conditioned matrices [16, 17]. However, the Galerkin method requires the computation of double surface integrals over pairs of triangles. There are many semi-analytical methods to treat these integrals, which all have some issues and are discussed in this dissertation. Some of these include, to name a few, singularity subtraction and "to the boundary" methods [18], singularity cancellation methods [19], and specialized quadrature methods that are designed for the singularities involved [20].

This dissertation presents novel methods inspired by the treatment of these kernels in the fast multipole method for computing all the integrals that arise in the Galerkin formulation to any accuracy. Integrals involving completely geometrically separated triangles are non-singular, and are computed using a technique based on spherical harmonics and multipole expansions and translations, which require the integration of polynomial functions over the triangles. Integrals involving cases where the triangles have common vertices, edges, or are coincident are treated via scaling and symmetry arguments, combined with automatic recursive geometric decomposition of the integrals.

The computation of these integrals is highly parallelizable because any two integrals are independent of each other. The parallel nature of multi-core CPUs and GPUs, therefore, offers the opportunity for incredibly large speedups. We have parallelized our code using

C++ and OpenMP, and also ported the C++ code to run on the GPU using CUDA.

The linear systems arising in the BEM are conventionally solved via direct matrix decompositions. However, the system matrices are dense, requiring $\mathrm{O}\left(N^2\right)$ storage, where N is the number of discretization unknowns. This can effectively restrict the size of a problem that can be solved on a given machine. For example, on a machine with 8 GB of main memory, problem sizes are restricted to around 30,000 boundary elements. This restriction can be avoided by not explicitly storing the matrix coefficients and recomputing them from scratch whenever necessary, or by using out-of-core methods. However, these alternatives are computationally inefficient. Moreover, solving the system by direct means (e.g., LU decomposition) requires $\mathrm{O}\left(N^3\right)$ operations. Using an iterative solver based on Krylov subspace methods, such as GMRES, alleviates this issue somewhat, providing an $\mathrm{O}\left(N_{\text{iter}}N^2\right)$ method, where N_{iter} is the number of iterations and $\mathrm{O}\left(N^2\right)$ is the cost of the matrix-vector product that is computed every iteration. However, for large N, this quadratic scaling in both time and memory can still be prohibitive.

The fast multipole method (FMM) allows for the acceleration of many types of matrix-vector products [21, 22], including the ones found in the BEM. The FMM accelerates sums of the following form:

$$v(\mathbf{y}_i) = \sum_{j=1}^{N} u_j \Phi(\mathbf{y}_i - \mathbf{x}_j), \quad i = 1, 2, \ldots, M. \tag{1.3}$$

Computing this sum by direct means requires $\mathrm{O}\left(MN\right)$ operations. The FMM computes the sum to any specified error, ε, in $\mathrm{O}\left(M + N\right)$ operations and storage, where the asymptotic constant depends on the desired accuracy. This linear scaling allows for very large problem

sizes. The FMM works by separating the matrix-vector product into two pieces, one for far-field interactions and one for near-field interactions. The FMM accelerates the far-field piece using spatial data structures (e.g., an octree), spherical harmonics, and multipole and local expansions and translations. Many authors have used the FMM to accelerate the BEM [8, 23–26].

However, the FMM is usually designed around monopole and dipole sources, not the integral expressions that appear in the BEM. To apply the FMM to these expressions, the internal data structures and logic of the FMM must be changed, but this can be difficult to do. For example, computing the multipole expansions due to the boundary elements requires computing single and double surface integrals over them. Moreover, FMM codes for monopole and dipole sources are widely available and highly optimized, including some that are free [27, 28]. This dissertation describes a method for applying the FMM unchanged to the integral expressions in the BEM. This method, called the correction factor matrix method, works by approximating the integrals using a quadrature. The quadrature points are treated as monopole and dipole sources, which can be plugged directly into current FMM codes. Any inaccuracies from the quadrature are corrected during a correction factor step. The method is further accelerated in the case of the Laplace equation by using a heterogeneous CPU/GPU FMM.

The accuracy of the FMM/GPU-accelerated BEM must be validated. In the case of the Laplace equation, there are many problems that have analytical solutions and can be used to verify the accuracy of the methods. For the Helmholtz equation, there are fewer. For example, the indirect BEM is capable of treating open, infinitely thin surfaces. These surfaces are good approximations to those often encountered in practice – those that

are much smaller than a wavelength in one dimension, but span several in the other two. However, there is only one analytically tractable problem posed on such a surface that has a solution and could be used to validate the indirect BEM: an acoustic wave scattering off a disk. The disk is actually the degenerate form of the oblate spheroid, so methods for solving scattering problems involving oblate spheroids can also be applied to the disk. We have developed computational software for calculating the solutions to acoustic scattering problems involving spheroids and disks.

This software uses spheroidal wave functions to analytically build the solutions to these problems. However, the spheroidal wave functions, which are the solutions to the Helmholtz equation in spheroidal coordinates, are notoriously difficult to compute. Because of this, practically no programming language comes equipped with the means to compute them. This makes problems that require their use hard to tackle. We have developed computational software for calculating these special functions. This software has a number of features, including the use of arbitrary precision arithmetic to provide greater accuracy in the computations.

The remainder of this dissertation is organized as follows. Chap. 2 states the problem and derives the BEM, the two different formulations (direct and indirect), and the two different methods for enforcing the boundary conditions (collocation and Galerkin). Chap. 3 describes methods for computing the boundary integrals needed by the BEM for the Laplace equation. Chap. 4 describes methods for computing the boundary integrals needed by the BEM for the Helmholtz equation. Chap. 5 introduces the correction factor matrix method for accelerating the BEM using the FMM, and characterizes the accuracy and performance of the method through several example problems. Chap. 6 describes our soft-

ware for computing the solutions to acoustic scattering problems involving spheroids and disks Chap. 7 explores the spheroidal wave functions, derives expressions for computing them, and documents our software for computing them accurately. Chap. 8 concludes and summarizes this dissertation, and discusses possible future work. Appx. A derives the contour integration method, which is used for computing the collocation integrals in the BEM for the Laplace equation. Finally, Appx. B gives an overview of the spherical harmonics, which are used in Chap. 3 for computing the Galerkin integrals in the BEM for the Laplace equation.

Chapter 2: Problem Statement

2.1 Direct Boundary Element Method

Consider the following boundary value problem (BVP):

$$\nabla^2\phi(\mathbf{x}) = 0 \quad \text{or} \quad \nabla^2\phi(\mathbf{x}) + k^2\phi(\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega, \tag{2.1}$$

where, in the case of the Helmholtz equation, k is the wavenumber. General boundary conditions are enforced on the boundary:

$$\alpha(\mathbf{x})\,\phi(\mathbf{x}) + \beta(\mathbf{x})\,q(\mathbf{x}) = \gamma(\mathbf{x}), \quad \mathbf{x} \in \Gamma, \tag{2.2}$$

where

$$q(\mathbf{x}) = \frac{\partial\phi}{\partial\mathbf{n}}(\mathbf{x}) = (\mathbf{n}\cdot\nabla_{\mathbf{x}})\,\phi(\mathbf{x}) \tag{2.3}$$

is the normal derivative. The boundary can be arbitrarily shaped, even made of several disconnected boundaries, but they must all be closed (see Fig. 2.1). For external problems, the potential should decay to zero at large distances:

$$\lim_{|\mathbf{x}|\to\infty} \phi(\mathbf{x}) = 0. \tag{2.4}$$

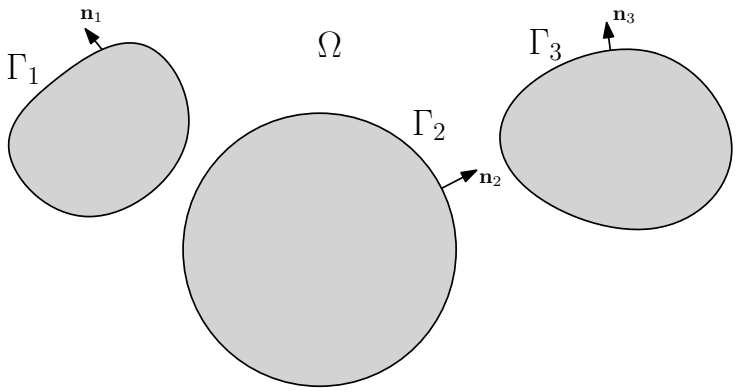

Figure 2.1: Several different types of boundaries in the direct BEM.

In the case of the Helmholtz equation, the potential should be composed of outgoing waves only. The Sommerfeld radiation condition provides such a constraint:

$$\lim_{|\mathbf{x}|\to\infty} |\mathbf{x}| \left(\frac{d\phi}{d\,|\mathbf{x}|} (\mathbf{x}) - ik\phi (\mathbf{x}) \right) = 0. \tag{2.5}$$

To solve the BVP, we use a direct boundary integral formulation called the Green's identity formulation. Using Green's identity, the partial differential equation (PDE) is transformed from a differential equation into an integral equation:

$$\pm\phi (\mathbf{x}) = L [q] (\mathbf{x}) - M [\phi] (\mathbf{x}), \quad \mathbf{x} \notin \Gamma, \tag{2.6}$$

where

$$L [q] (\mathbf{x}) = \int_{\mathbf{x}'\in\Gamma} q (\mathbf{x}') G (\mathbf{x} - \mathbf{x}')\, dS (\mathbf{x}'), \tag{2.7}$$

$$M [\phi] (\mathbf{x}) = \int_{\mathbf{x}'\in\Gamma} \phi (\mathbf{x}') (\mathbf{n}' \cdot \nabla_{\mathbf{x}'} G (\mathbf{x} - \mathbf{x}'))\, dS (\mathbf{x}') \tag{2.8}$$

are the single- and double-layer potentials [15], respectively, and $G (\mathbf{r})$ is the Green's

function for the PDE, either $1/\left(4\pi\left|\mathbf{r}\right|\right)$ for the Laplace equation or $\exp\left(ik\left|\mathbf{r}\right|\right)/\left(4\pi\left|\mathbf{r}\right|\right)$ for the Helmholtz equation. The $\pm$ in Eq. (2.6) should be a $+$ for internal problems and a $-$ for external problems. Green's identity is incredibly powerful: we can compute the potential at any point not on the boundary as long as we know the potential and normal derivative on the boundary. Due to the behavior of the double-layer potential, for points on the boundary, we have

$$\pm\frac{1}{2}\phi\left(\mathbf{x}\right)=L\left[q\right]\left(\mathbf{x}\right)-M\left[\phi\right]\left(\mathbf{x}\right),\quad\mathbf{x}\in\Gamma. \tag{2.9}$$

In the differential equation, we seek a solution to the potential governed by the PDE throughout the entire domain. However, in the integral equation, we seek a solution to the potential and normal derivative only on the boundary. The dimensionality of the problem is, therefore, reduced by one. Another advantage of the BEM is that the expression in Eq. (2.6) automatically satisfies the original partial differential equation. Moreover, the Green's function satisfies the boundary conditions at infinity, so as long as the potential and normal derivative on the boundary are bounded and finite, the single- and double-layer potentials will satisfy them as well. Thus, we need only concern ourselves with searching for the potential and normal derivative on the boundary that satisfy the remaining boundary conditions.

In order to make the problem computationally tractable, the boundary is discretized using planar triangular elements. By doing so, the integrals over the original boundary become sums of integrals over these triangles, and instead of searching for a solution to the potential and normal derivative on the original boundary, we now seek a solution to

these on the triangles. In addition, the potential, $\phi(\mathbf{x})$, and normal derivative, $q(\mathbf{x})$, on the triangles are each written as a linear combination of N basis functions:

$$\phi(\mathbf{x}) = \sum_{j=1}^{N} \phi_j f_j(\mathbf{x}), \quad q(\mathbf{x}) = \sum_{j=1}^{N} q_j f_j(\mathbf{x}). \tag{2.10}$$

The discretization process and the selection of basis functions are discussed further in Sec. 2.3. We need to compute the coefficients of these basis functions so that the boundary conditions are satisfied. In other words, we seek $\phi_1, \phi_2, \ldots, \phi_N, q_1, q_2, \ldots$, and q_N such that

$$\sum_{j=1}^{N} \phi_j \left(M[f_j] \pm \frac{1}{2} f_j \right) - \sum_{j=1}^{N} q_j L[f_j] = 0, \tag{2.11}$$

$$\sum_{j=1}^{N} \phi_j \alpha f_j + \sum_{j=1}^{N} q_j \beta f_j = \gamma. \tag{2.12}$$

We have dropped the argument, $\mathbf{x}$, for clarity (i.e., $L[q](\mathbf{x})$ becomes $L[q]$). In effect, instead of searching for a solution to the original problem in an infinite-dimensional function space, we now seek a solution to the discretized problem in a finite-dimensional vector space. The discretization process introduces geometric and approximation errors, which are discussed further in Chap. 5.

In the case of the Helmholtz equation, when solving external problems using the direct formulation, the solutions are affected by so-called spurious modes. These spurious modes are the solutions to the internal problem for the same geometry whose potential or normal derivative is zero on the boundary. Because they are zero, they do not interfere with

the boundary conditions, but they nevertheless infect the solution away from the boundary. There are many methods for suppressing these spurious modes. One popular method is to add additional constraints to the system by placing points inside the boundary, and forcing the potential at these points to be zero [29]. Another method uses Maue's identity, which is given by

$$\pm\frac{1}{2}q\left(\mathbf{x}\right)=L'\left[q\right]\left(\mathbf{x}\right)-M'\left[\phi\right]\left(\mathbf{x}\right), \tag{2.13}$$

where

$$L'\left[\sigma\right]\left(\mathbf{x}\right)=\left(\mathbf{n}\cdot\nabla_{\mathbf{x}}\right)\int_{\mathbf{x}'\in\Gamma}\sigma\left(\mathbf{x}'\right)G\left(\mathbf{x}-\mathbf{x}'\right)dS\left(\mathbf{x}'\right), \tag{2.14}$$

$$M'\left[\mu\right]\left(\mathbf{x}\right)=\left(\mathbf{n}\cdot\nabla_{\mathbf{x}}\right)\int_{\mathbf{x}'\in\Gamma}\mu\left(\mathbf{x}'\right)\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}\right)\times G\left(\mathbf{x}-\mathbf{x}'\right)dS\left(\mathbf{x}'\right). \tag{2.15}$$

Linearly combining Green's identity and Maue's identity yields

$$\pm\frac{1}{2}\left(\phi\left(\mathbf{x}\right)+\lambda q\left(\mathbf{x}\right)\right)=\left(L+\lambda L'\right)\left[q\right]\left(\mathbf{x}\right)-\left(M+\lambda M'\right)\left[\phi\right]\left(\mathbf{x}\right). \tag{2.16}$$

Burton and Miller showed in [15] that as long as $\mathrm{Im}\left\{\lambda\right\}\neq 0$, the solution will contain no spurious modes. Due to their work, this formulation is usually called the Burton-Miller formulation. A third approach for suppressing the spurious modes is the indirect formulation, which is described in Sec. 2.2, and is ultimately what we used in the case of the Helmholtz equation.

2.1.1 Collocation Method

The collocation method [30, 31] works by enforcing the boundary conditions at N matching points:

$$\sum_{j=1}^{N} \phi_j \left(M\left[f_j\right](\mathbf{x}_i) \pm \frac{1}{2} f_j(\mathbf{x}_i) \right) - \sum_{j=1}^{N} q_j L\left[f_j\right](\mathbf{x}_i) = 0, \tag{2.17}$$

$$\sum_{j=1}^{N} \phi_j \alpha(\mathbf{x}_i) f_j(\mathbf{x}_i) + \sum_{j=1}^{N} q_j \beta(\mathbf{x}_i) f_j(\mathbf{x}_i) = \gamma(\mathbf{x}_i), \tag{2.18}$$

where $i = 1, 2, \ldots, N$. In this system, there are $2N$ equations in $2N$ unknowns. The integral expressions necessary for implementing the collocation method have been derived for piecewise constant and linear basis functions on triangular elements by many authors [32–36]. They have also been derived for higher-order basis functions on curvilinear elements [37]. However, many of the boundary integrals are hypersingular, which make them difficult to compute, especially for points on the corners or edges of the boundary. Furthermore, the resulting system matrices are non-symmetric.

The system matrix takes the following form:

$$\mathrm{S} = \begin{bmatrix} \mathrm{U} + \mathrm{D} & \mathrm{V} \\ \mathrm{A} & \mathrm{B} \end{bmatrix}, \tag{2.19}$$

where U, D, V, A, and B are $N \times N$ matrices whose (i,j)th entries are given by

$$U_{i,j} = M\left[f_j\right](\mathbf{x}_i), \quad D_{i,j} = \pm\frac{1}{2} f_j(\mathbf{x}_i), \quad V_{i,j} = L\left[f_j\right](\mathbf{x}_i), \tag{2.20}$$

$$A_{i,j} = \alpha(\mathbf{x}_i) f_j(\mathbf{x}_i), \quad B_{i,j} = \beta(\mathbf{x}_i) f_j(\mathbf{x}_i). \tag{2.21}$$

Let $\boldsymbol{\phi}$ be the vector of coefficients, $\phi_1, \phi_2, \ldots, \phi_N$, and $\mathbf{q}$ be the vector of coefficients, $q_1, q_2, \ldots, q_N$. The matrix-vector product, U$\boldsymbol{\phi}$, computes the double-layer potential at the N matching points due to the potential, $\phi(\mathbf{x})$, on the boundary. Likewise, the matrix-vector product, V$\mathbf{q}$, computes the single-layer potential at the N matching points due to the normal derivative, $q(\mathbf{x})$, on the boundary. For piecewise linear basis functions, the matrices D, A, and B, are highly sparse, while the matrices, U and V, are dense.

The right-hand side takes the following form:

$$\mathbf{b} = \begin{bmatrix} \mathbf{0} \\ \\ \mathbf{b}_2 \end{bmatrix}, \tag{2.22}$$

where $\mathbf{b}_2$ is an $N \times 1$ column vector whose ith entry is given by $b_{2,i} = \gamma(\mathbf{x}_i)$.

In some cases, this $2N \times 2N$ linear system can be reduced to an $N \times N$ one by solving for either $\boldsymbol{\phi}$ or $\mathbf{q}$, and then solving for the other. For example, in the Dirichlet case, the matrix, B, is zero, so $\boldsymbol{\phi}$ can be computed by solving A$\boldsymbol{\phi} = \mathbf{c}$. Since A is highly sparse for piecewise linear basis functions, this can usually be done relatively quickly. Once $\boldsymbol{\phi}$ is known, $\mathbf{q}$ can be computed by solving $\mathrm{V}\mathbf{q} = -(\mathrm{U} + \mathrm{D})\boldsymbol{\phi}$.

2.1.2 Galerkin Method

The Galerkin method enforces the boundary conditions in an integral sense. The boundary integral equations are multiplied by each of the N basis functions and integrated over the boundary a second time:

$$\sum_{j=1}^{N} \phi_j \int_\Gamma f_i \left(M\left[f_j\right] \pm \frac{1}{2} f_j \right) dS - \sum_{j=1}^{N} q_j \int_\Gamma f_i L\left[f_j\right] dS = 0, \tag{2.23}$$

$$\sum_{j=1}^{N} \phi_j \int_\Gamma \alpha f_i f_j dS + \sum_{j=1}^{N} q_j \int_\Gamma \beta f_i f_j dS = \int_\Gamma f_i \gamma dS, \tag{2.24}$$

where $i = 1, 2, \ldots, N$. In this system, there are $2N$ equations in $2N$ unknowns. By integrating over the boundary a second time, all of the hypersingular integrals become weakly singular. Moreover, the system matrices arising in the Galerkin method are typically symmetric, better conditioned, and have better convergence properties [16, 17]. However, the extra integral over the boundary complicates the computation of the entries in the system matrix.

The system matrix takes the same form as in the collocation method, but the entries in the $N \times N$ matrices, U, D, V, A, and B, are given by

$$U_{i,j} = \int_\Gamma f_i M\left[f_j\right] dS, \quad D_{i,j} = \pm\frac{1}{2} \int_\Gamma f_i f_j dS, \quad V_{i,j} = \int_\Gamma f_i L\left[f_j\right] dS, \tag{2.25}$$

$$A_{i,j} = \int_\Gamma \alpha f_i f_j dS, \quad B_{i,k} = \int_\Gamma \beta f_i f_j dS. \tag{2.26}$$

The matrix-vector product, U$\boldsymbol{\phi}$, integrates the product of the double-layer potential and

each of the N basis functions over the boundary. Likewise, the matrix-vector product, $\mathrm{V}\mathbf{q}$, integrates the product of the single-layer potential and each of the N basis functions over the boundary. For piecewise linear basis functions, the matrices D, A, and B, are highly sparse, while the matrices, U and V, are dense.

The right-hand side takes the same form as in the collocation method, but the entries in the $N \times 1$ column vector, $\mathbf{b}_2$, are given by

$$b_{2,i} = \int_\Gamma f_i \gamma dS. \tag{2.27}$$

Like in the collocation method, in some cases, this $2N \times 2N$ linear system can be reduced to an $N \times N$ one by solving for either $\boldsymbol{\phi}$ or $\mathbf{q}$, and then solving for the other. For example, in the Dirichlet case, the matrix, B, is zero, so $\boldsymbol{\phi}$ can be computed by solving $\mathrm{A}\boldsymbol{\phi} = \mathbf{c}$. Since A is highly sparse for piecewise linear basis functions, this can usually be done relatively quickly. Once $\boldsymbol{\phi}$ is known, $\mathbf{q}$ can be computed by solving $\mathrm{V}\mathbf{q} = -\left(\mathrm{U} + \mathrm{D}\right)\boldsymbol{\phi}$.

In the Galerkin method, the boundary integral equations are multiplied by each of the N basis functions and integrated over the boundary a second time. By choosing the test functions to be the same as the basis functions, the resulting system matrices are symmetric, which have a number of nice properties. In general, the test functions can be chosen more or less arbitrarily, so long as they are orthogonal to each other and lead to a nonsingular system matrix. In particular, choosing the test functions to be the Dirac delta functions centered at the N matching points from the collocation method causes the method to reduce to the collocation method.

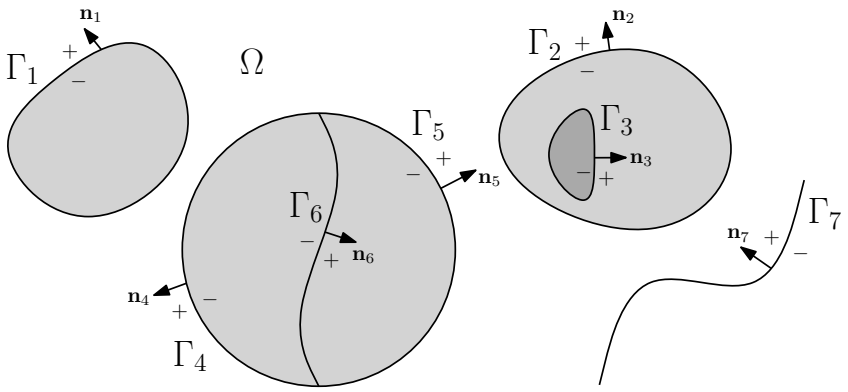

Figure 2.2: Several different types of boundaries in the indirect BEM: Γ_1 is closed; Γ_2 is closed, but contains another boundary, Γ_3, also closed; Γ_4 and Γ_5 form a closed region, but this region is divided into two by Γ_6; and Γ_7 is open.

2.2 Indirect Boundary Element Method for Thin Surfaces

The direct BEM requires that all boundaries be closed. This makes modeling very thin surfaces difficult, as the boundary elements must be made very small to account for the reduced thickness. Making the elements very small requires using many more of them, leading to larger meshes, which require more time to solve. Also, the direct BEM can typically only solve either the internal problem or the external problem, but not both at the same time. Coupling an internal problem and external problem is possible, but requires linking two separate solutions together. The indirect BEM can be used to overcome these issues, allowing for double-sided and open boundaries.

Consider the same BVP from before:

$$\nabla^2\phi\left(\mathbf{x}\right) = 0 \quad \text{or} \quad \nabla^2\phi\left(\mathbf{x}\right) + k^2\phi\left(\mathbf{x}\right) = 0, \quad \mathbf{x} \in \Omega. \tag{2.28}$$

General boundary conditions are enforced on both sides of the boundary:

$$\alpha_k^+(\mathbf{x})\,\phi^+(\mathbf{x}) + \beta_k^+(\mathbf{x})\,q^+(\mathbf{x}) + \alpha_k^-(\mathbf{x})\,\phi^-(\mathbf{x}) + \beta_k^-(\mathbf{x})\,q^-(\mathbf{x}) = \gamma_k(\mathbf{x}), \quad \mathbf{x} \in \Gamma, \tag{2.29}$$

where $k = 1, 2$. The boundaries can be closed or open (see Fig. 2.2). For closed boundaries, the exterior side of the boundary is the "+" side and the interior side is the "−" side. For open boundaries, since there is no inside or outside, designating each side of the boundary as "+" or "−" can be done arbitrarily. The values, ϕ^+ and ϕ^-, are the potentials on the "+" and "−" sides of the surface, respectively. Likewise, the values, q^+ and q^-, are the normal derivatives of the potential on the "+" and "−" sides of the surface, respectively, and are given by

$$q^\pm = \frac{\partial \phi^\pm}{\partial \mathbf{n}^\pm} = \left(\mathbf{n}^\pm \cdot \nabla_{\mathbf{x}}\right)\phi^\pm, \tag{2.30}$$

where $\mathbf{n}^\pm = \mp\mathbf{n}$ (i.e., $\mathbf{n}^+$ goes from the "+" side to the "−" side, and vice versa). The potential should decay to zero at large distances:

$$\lim_{|\mathbf{x}|\to\infty} \phi(\mathbf{x}) = 0. \tag{2.31}$$

In the case of the Helmholtz equation, the potential should be composed of outgoing waves only. The Sommerfeld radiation condition provides such a constraint:

$$\lim_{|\mathbf{x}|\to\infty} |\mathbf{x}| \left(\frac{d\phi}{d\,|\mathbf{x}|}(\mathbf{x}) - ik\phi(\mathbf{x})\right) = 0. \tag{2.32}$$

To solve the BVP, we use an indirect boundary integral formulation called the layer

potential formuation. The PDE is transformed from a differential equation into an integral equation:

$$\phi(\mathbf{x}) = L[\sigma](\mathbf{x}) + M[\mu](\mathbf{x}). \tag{2.33}$$

The single-layer potential, $L[\sigma](\mathbf{x})$, is the potential due to the monopole source density distribution, $\sigma(\mathbf{x})$, over the boundary. Likewise, the double-layer potential, $M[\mu](\mathbf{x})$, is the potential due to the dipole source density distribution, $\mu(\mathbf{x})$, over the boundary. In the direct formulation, we seek the potential and normal derivative on the boundary, which we can use to compute the potential everywhere else. In the indirect formulation, we seek the source density distributions, $\sigma(\mathbf{x})$ and $\mu(\mathbf{x})$, over the boundary that give rise to that potential. Thus, we must search for the source density distributions that satisfy the boundary conditions. To do this, we express the potential and normal derivative on either side of the boundary in terms of the source density distributions. Jump conditions provide such a relationship:

$$\phi^{\pm}(\mathbf{x}) = L[\sigma](\mathbf{x}) + M[\mu](\mathbf{x}) \pm \frac{1}{2}\mu(\mathbf{x}), \tag{2.34}$$

$$q^{\pm}(\mathbf{x}) = \mp L'[\sigma](\mathbf{x}) \mp M'[\mu](\mathbf{x}) + \frac{1}{2}\sigma(\mathbf{x}). \tag{2.35}$$

Plugging the expressions for $\phi^{\pm}$ and $q^{\pm}$ into the boundary conditions and rearranging,

$$a_k(L[\sigma] + M[\mu]) + b_k(L'[\sigma] + M'[\mu]) + c_k\sigma + d_k\mu = \gamma_k, \tag{2.36}$$

where

$$a_k = \alpha_k^+ + \alpha_k^-, \quad b_k = -\beta_k^+ + \beta_k^-, \quad c_k = \frac{1}{2}\left(\beta_k^+ + \beta_k^-\right), \quad d_k = \frac{1}{2}\left(\alpha_k^+ - \alpha_k^-\right), \tag{2.37}$$

and $k = 1, 2$. We have dropped the argument, $\mathbf{x}$, for clarity (i.e., $L\left[\sigma\right]\left(\mathbf{x}\right)$ becomes $L\left[\sigma\right]$).

Like in the direct formulation, in order to make the problem computationally tractable, the boundary is discretized using planar triangular elements. By doing so, the integrals over the original boundary become sums of integrals over these triangles, and instead of searching for a solution to the source density distributions over the original boundary, we now seek a solution to these over the triangles. In addition, the source density distributions, $\sigma\left(\mathbf{x}\right)$ and $\mu\left(\mathbf{x}\right)$, over the triangles are each written as a linear combination of N basis functions:

$$\sigma\left(\mathbf{x}\right) = \sum_{j=1}^{N} \sigma_j f_j\left(\mathbf{x}\right), \quad \mu\left(\mathbf{x}\right) = \sum_{j=1}^{N} \mu_j f_j\left(\mathbf{x}\right). \tag{2.38}$$

The discretization process and the selection of basis functions are discussed further in Sec. 2.3. We need to compute the coefficients of these basis functions so that the boundary conditions are satisfied. In other words, we seek $\sigma_1, \sigma_2, \ldots, \sigma_N, \mu_1, \mu_2, \ldots,$ and μ_N such that

$$\sum_{j=1}^{N} \sigma_j A_k\left[f_j\right] + \sum_{j=1}^{N} \mu_j B_k\left[f_j\right] = \gamma_k, \tag{2.39}$$

where

$$A_k\left[f_j\right] = a_k L\left[f_j\right] + b_k L'\left[f_j\right] + c_k f_j, \quad B_k\left[f_j\right] = a_k M\left[f_j\right] + b_k M'\left[f_j\right] + d_k f_j, \tag{2.40}$$

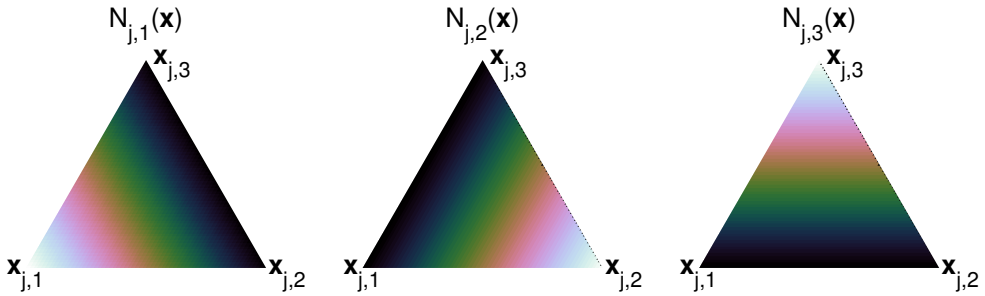

Figure 2.3: Each triangle has three linear basis functions, one per vertex.

and $k = 1, 2$. In effect, instead of searching for a solution to the original problem in an infinite-dimensional function space, we now seek a solution to the discretized problem in a finite-dimensional vector space. The discretization process introduces geometric and approximation errors, which are discussed further in Chap. 5.

Like in the direct BEM, the collocation method or the Galerkin method can be used to enforce the boundary conditions. The same procedures from Secs. 2.1.1 and 2.1.2 can be used to derive the resulting linear systems. In particular, in some cases, the resulting $2N \times 2N$ linear systems can be reduced to $N \times N$ ones by solving for one of the source density distributions, and then solving for the other. For example, in the Dirichlet case, where the potential is specified on both sides of the boundary, the dipole source density distribution, $\mu(\mathbf{x})$, can be computed by calculating the difference in potential from the "$-$" side to the "$+$" side of the boundary.

2.3 Selection of Basis Functions

Discretize the boundary into N triangles: $\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_N$. Let $\mathbf{x}_{j,n}$ be the nth vertex of the jth triangle, T_j. The potential and normal derivative on these triangles are each

written as a linear combination of $3N$ basis functions:

$$\phi(\mathbf{x}) = \sum_{j=1}^{N} \sum_{n=1}^{3} \phi_{j,n} N_{j,n}(\mathbf{x}), \quad q(\mathbf{x}) = \sum_{j=1}^{N} \sum_{n=1}^{3} q_{j,n} N_{j,n}(\mathbf{x}), \tag{2.41}$$

where $\phi_{j,n}$ and $q_{j,n}$ are the potential and normal derivative at $\mathbf{x}_{j,n}$, and $N_{j,n}(\mathbf{x})$ is the corresponding linear nodal basis function (see Fig. 2.3). For $\mathbf{x} \notin \mathrm{T}_j$, $N_{j,n}(\mathbf{x}) = 0$. Otherwise,

$$N_{j,n}(\mathbf{x}_{j,n'}) = \begin{cases} 1 & , \quad n' = n \\ 0 & , \quad n' \neq n \end{cases} \tag{2.42}$$

for $n, n' = 1, 2, 3$. This set of basis functions allows for piecewise linear (possibly discontinuous) potentials and normal derivatives on the triangles.

The same set of basis functions can be used in the indirect formation for the source density distributions.

2.4 Solving the Systems of Linear Equations

2.4.1 Direct Matrix Decompositions

The linear systems arising in the BEM are conventionally solved via direct matrix decompositions [38]. One such decomposition is the LU decomposition, where the system matrix, S, is decomposed into two matrices:

$$\mathbf{S}\mathbf{u} = \mathbf{L}\mathbf{U}\mathbf{u} = \mathbf{b}, \tag{2.43}$$

where L is a lower triangular matrix and U is an upper triangular matrix. Having this decomposition, an intermediate solution, $\mathbf{v}$, is computed by solving $\mathrm{L}\mathbf{v} = \mathbf{b}$. The solution, $\mathbf{u}$, is then computed by solving $\mathrm{U}\mathbf{u} = \mathbf{v}$. Solving the linear systems by direct means is straight foward. There are many efficient libraries available for doing so, including LAPACK, which is accessible through MATLAB and many other software packages. Also, once the LU decomposition is available, multiple right-hand sides can be solved simultaneously.

However, in the case of the BEM, the system matrices are dense, requiring $\mathrm{O}\left(N^2\right)$ storage, where N is the number of discretization unknowns. This can effectively restrict the size of a problem that can be solved on a given machine. For example, on a machine with 8 GB of main memory, problem sizes are restricted to around 30,000 boundary elements. This restriction can be avoided by not explicitly storing the matrix coefficients and recomputing them from scratch whenever necessary, or by using out-of-core methods. However, these alternatives are computationally inefficient. Moreover, computing the LU decomposition requires $\mathrm{O}\left(N^3\right)$ operations, plus $\mathrm{O}\left(N^2\right)$ operations to solve for $\mathbf{v}$ and then $\mathbf{u}$. These steep scalings prevent the consideration of even moderately sized problems. Therefore, we seek a solution method that scales better than quadratically and cubically, preferably one that scales linearly. Iterative solvers, when combined with fast matrix-vector product algorithms, such as the FMM, provide such a method.

2.4.2 Iterative Solvers

Most iterative solvers work in roughly the same manner [39–41]. Given a starting guess, $\mathbf{u}_0$, they compute a sequence of approximations, $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k, \ldots$, such that the approximations get better and better (i.e., they converge to the exact solution). Once the approximation is close enough to the exact solution, the solver stops and returns the most recent approximation as the solution. Deciding when to stop is usually done by computing the residual of the approximation, $\mathbf{r}_k = \mathrm{S}\mathbf{u}_k - \mathbf{b}$, and stopping when the norm of this drops below some threshold. How quickly the solution converges depends on the condition number of the system matrix, which gives a measure of how well-posed the system matrix is, as well as which iterative method is being used. There are several types of iterative solvers, including stationary methods (such as Jacobi and Gauss-Seidel) and Krylov subspace methods (such as GMRES).

Krylov subspace methods are among the best performing iterative methods available for solving linear systems. These solvers work by constructing a kth-order Krylov subspace for a matrix, S, and vector, $\mathbf{b}$:

$$\mathcal{K}_k\left(\mathrm{S}, \mathbf{b}\right) = \left\{\mathbf{b}, \mathrm{S}\mathbf{b}, \mathrm{S}^2\mathbf{b}, \ldots, \mathrm{S}^{k-1}\mathbf{b}\right\}. \tag{2.44}$$

Once constructed, the solvers then search for a vector inside $\mathcal{K}_k$ that approximates the solution.

However, constructing the Krylov subspace by repeatedly premultiplying $\mathbf{b}$ by S is numerically unstable. This is because the sequence of vectors approaches the eigenvector

corresponding to the largest eigenvalue of the matrix. In fact, this sequence of vectors appears in the power method, which is designed to compute this dominant eigenvector. As a result, the sequence of vectors, especially those near the end of the sequence, will be nearly linearly dependent.

The Arnoldi method avoids this numerical instability by orthonormalizing the sequence of vectors as they are computed. The modified Gram-Schmidt process is typically used to do this. The method normalizes the input vector: $\mathbf{q}_1 = \mathbf{b}/\left|\mathbf{b}\right|$. The method then computes $\mathrm{S}\mathbf{q}_1$, orthonormalizes the result against $\mathbf{q}_1$ to compute $\mathbf{q}_2$, and appends $\mathbf{q}_2$ to the sequence of vectors. This is repeated $n-2$ more times: compute $\mathrm{S}\mathbf{q}_{k-1}$, orthonormalize the result against the previous $k-2$ vectors to compute $\mathbf{q}_k$, and then append $\mathbf{q}_k$ to the sequence of vectors. The sequence of vectors are organized into a orthogonal matrix, Q_n. The iterative procedure of the Arnoldi method can be described in symbols by

$$\mathrm{S}\mathrm{Q}_k = \mathrm{Q}_{k+1}\mathrm{H}_k, \tag{2.45}$$

where H_n is an upper Hessenberg matrix that contains the coefficients computed during the Gram-Schmidt orthonormalization process. The kth step in the Arnolid method requires $\mathrm{O}\left(N^2\right)$ operations to perform the matrix-vector product (or $\mathrm{O}\left(N\right)$ when a fast algorithm, such as the FMM, is used), and $\mathrm{O}\left(kN\right)$ operations to orthonormalize the result.

GMRES uses the Arnoldi method to solve a system of linear equations. Consider again the linear system, $\mathrm{S}\mathbf{u} = \mathbf{b}$. The solution is approximated by $\mathbf{u}_k \in \mathcal{K}_k$, i.e., a linear combination of the vectors computed during the Arnolid method: $\mathbf{u}_k = \mathrm{Q}_k\mathbf{v}_k$. Using this

and Eq. (2.45), we have

$$\mathbf{Q}_{k+1}\mathbf{H}_k\mathbf{v}_k = \mathbf{b}. \tag{2.46}$$

Because the leading column of $\mathbf{Q}_{k+1}$ is $\mathbf{b}/|\mathbf{b}|$ and all of the other columns are orthogonal to $\mathbf{b}$, we have

$$\mathbf{H}_k\mathbf{v}_k = |\mathbf{b}|\,\mathbf{e}_1, \tag{2.47}$$

where $\mathbf{e}_1 = (1, 0, 0, \ldots, 0)$. This new linear system is overdetermined, and can be solved via least squares.

The Arnoldi method and GMRES provide an $\mathrm{O}\left(N_{\text{iter}}N^{\alpha}\right)$ method for solving the original linear system, where N_{iter} is the number of iterations and $\mathrm{O}\left(N^{\alpha}\right)$ is the cost of the matrix-vector product that is computed during each iteration ($\alpha = 2$ for a direct matrix-vector product algorithm, and $\alpha = 1$ for a fast matrix-vector product algorithm, such as the FMM). The number of iterations largely depends on the condition number of the system matrix, which gives a measure of how well-posed the system matrix is. When the system matrix is positive real definite, the norm of the residual can be bounded as a function of the iteration number by [42]

$$|\mathbf{r}_k| \leq \left(1 - \frac{\lambda_{\min}\left(\left(\mathrm{S} + \mathrm{S}^{\mathrm{T}}\right)/2\right)^2}{\lambda_{\max}\left(\mathrm{S}^{\mathrm{T}}\mathrm{S}\right)}\right)^{k/2} |\mathbf{r}_0|\,. \tag{2.48}$$

In the general case when the system matrix is not positive real definite, this bound no longer holds, but similar bounds can be derived. The convergence of GMRES can be accelerated by using a preconditioner. The use of a preconditioner can help reduce the condition number of the system matrix, reducing the number of iterations required. However, in our

computational experiments, for the problems we consider, the iteration counts are usually no more than 100, so we do not use a preconditioner. For large N, the O (N^2) behavior of the matrix-vector product that is computed during each iteration can be prohibitive. We use the FMM to reduce this cost to O (N) (see Chap. 5). This leaves us with an O $(N_{\text{iter}}N)$ method for solving the original linear system.

Chapter 3: Computation of Boundary Integrals for the Laplace Equation[1]

3.1 Introduction

The Laplace equation in three dimensions is one of the most important, perhaps the most important, partial differential equations (PDEs) in science, and governs problems in a large number of disciplines, including electrostatics [2, 3], magnetostatics, plasma physics [4], astrophysics [5], molecular dynamics, aerodynamics [6], and even in computer graphics for character animation [7] and collision detection [8]. The Galerkin boundary element method (BEM), also known as the method of moments, is a powerful method for solving the Laplace equation in three dimensions [12–14]. When the boundary is discretized using triangular elements, constructing the system matrix requires computing double surface integrals over pairs of these triangles. Because the kernels being integrated are singular, these integrals can be difficult to compute, especially when the two triangles are proximate, share a vertex, an edge, or are the same. Depending on the relative geometry of the two triangles, there are many different methods for computing them. For example, when the two triangles do not touch, the integral is completely regular and can be computed via numerical means, e.g., Gaussian quadrature. Semi-analytical methods, where the inside integral is computed analytically and the outside integral is computed numerically, have

[1] This chapter is based on our original work in [43].

also been proposed [44, 45].

However, in the cases when the two triangles share a vertex, an edge, or are the same, things become more complicated. There are analytical expressions for the case when the two triangles are the same [1], but not for when they share only a vertex or an edge. In these cases, the semi-analytical methods do not always work. This is because, depending on the kernel being integrated, the inside integral can be hypersingular. While there are analytical expressions available for the inside integral, these expressions are singular along the corners and edges of the corresponding triangle. When the two triangles share a vertex or an edge, these singularities are included in the outside integral. The usual semi-analytical methods will not work in these cases because they are not designed to properly handle the singularities.

The double integrals are weakly singular, so while the inside integrals may be hypersingular and the expressions for them may be singular in some places, they are completely integrable. Nevertheless, actually integrating them in practice can be hard. Therefore, more sophisticated semi-analytical methods have been developed over the years. These include: singularity subtraction and "to the boundary" techniques [18, 46–51]; singularity cancellation techniques [19, 52]; specialized quadrature methods that are designed for the singularities involved, such as those based on the double exponential formula [20, 53]; and other regularization methods, such as the Duffy transformation [54–56]. Many of these methods work very well, but because they all attempt to tackle the singularity issue directly, their analysis is very involved.

In this chapter, we present a method for computing the integrals that completely avoids the computation of singular integrals. The approach relies on several scaling

properties of the integrals and the kernels being integrated. When two triangles share a vertex, an edge, or are the same, the integral is decomposed into several smaller integrals, some of which are related back to the original integral via scaling and symmetry arguments. This is done in such a way so that only regular integrals need to be computed explicitly. Any integrals involving singularities are computed implicitly during the procedure. The regular integrals can be computed using standard semi-analytical methods, but in this chapter, we also present an arbitrarily accurate approximate analytical method for doing so. This method uses spherical harmonics and multipole and local expansions and translations. The only source of error in the method is from truncating these expansions. However, this error is precisely controlled by choosing the appropriate truncation number or recursively subdividing the problem. We implemented these methods in MATLAB and C++. We verify their accuracy, and use them to solve some example problems.

In addition, the computation of these integrals is highly parallelizable because any two integrals are independent of each other. The highly parallel nature of GPU hardware, therefore, offers the opportunity for incredibly large speedups. Indeed, using the GPU to accelerate the BEM has been studied by many authors. For example, the GPU was used to speed up monostatic scattering problems in [57], elasticity problems in [58], the Helmholtz equation in [59], and vortex problems in [60]. We have adapted our C++ code to run on the GPU using CUDA.

3.2 Collocation Integrals

Enforcing the boundary conditions in the BEM using the collocation method requires computing single- and double-layer potentials and gradients due to linear source density distributions over triangular elements:

$$L(\mathbf{x}) = \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}') \, G(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}') \,, \tag{3.1}$$

$$M(\mathbf{x}) = \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}') \, (\mathbf{n}' \cdot \nabla_{\mathbf{x}'} G(\mathbf{x} - \mathbf{x}')) \, dS(\mathbf{x}') \,, \tag{3.2}$$

$$\nabla_{\mathbf{x}} L(\mathbf{x}) = \nabla_{\mathbf{x}} \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}') \, G(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}') \,, \tag{3.3}$$

$$\nabla_{\mathbf{x}} M(\mathbf{x}) = \nabla_{\mathbf{x}} \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}') \, (\mathbf{n}' \cdot \nabla_{\mathbf{x}'} G(\mathbf{x} - \mathbf{x}')) \, dS(\mathbf{x}') \,. \tag{3.4}$$

A method for computing these integrals, called the contour integration method, is described in Appx. A.

3.3 Galerkin Integrals

When the boundary is discretized using linear triangular elements, the double surface integrals are performed over pairs of these triangles. In each pair, one is called the "source" triangle, and the other the "receiver" triangle. The inside integral is over the source triangle, S, and the outside integral is over the receiver triangle, R (see Fig. 3.1). Thus, when

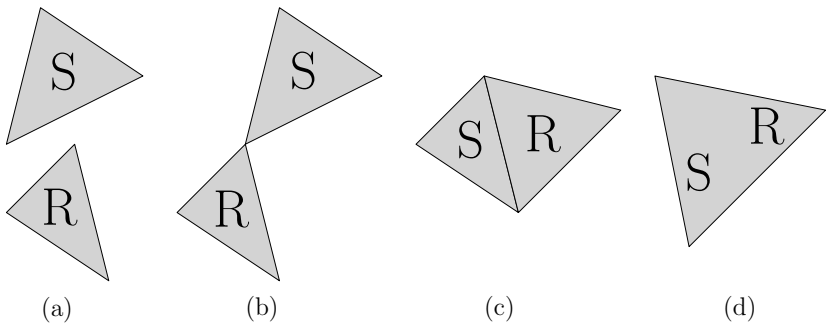

Figure 3.1: In practice, the relative geometry of two triangles is one of the following: (a) the two triangles do not touch; (b) they share a vertex; (c) they share an edge; or (d) they are the same.

populating the system matrix, we need to compute integrals of the following form:

$$I = \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) \int_{\mathbf{x}'\in\mathrm{S}} (\sigma_0' + \mathbf{p}'\cdot\mathbf{x}')\, F(\mathbf{x}-\mathbf{x}')\, dS(\mathbf{x}')\, dS(\mathbf{x}), \tag{3.5}$$

where $\sigma_0' + \mathbf{p}'\cdot\mathbf{x}'$ is the source density distribution over the source triangle, $\sigma_0 + \mathbf{p}\cdot\mathbf{x}$ is the weight function over the receiver triangle, and $F(\mathbf{r})$ is the kernel being integrated. To implement the Galerkin BEM described in Sec. 2.1.2, we need to compute this integral for the following two kernels:

$$F_1(\mathbf{x}-\mathbf{x}') = G(\mathbf{x}-\mathbf{x}'), \tag{3.6}$$

$$F_2(\mathbf{x}-\mathbf{x}') = \mathbf{n}'\cdot\nabla_{\mathbf{x}'} G(\mathbf{x}-\mathbf{x}'), \tag{3.7}$$

where $F_1(\mathbf{r})$ and $F_2(\mathbf{r})$ correspond to the single- and double-layer potentials, respectively.

Computing the integral for these two kernels for all commonly encountered geometries is the focus of this chapter. In practice, the relative geometry of the two triangles is one of the following: (a) the two triangles do not touch; (b) they share a vertex; (c) they

share an edge; or (d) they are the same (see Fig. 3.1). These are called the zero-, one-, two-, and three-touch cases, respectively. In this naming scheme, the number represents how many vertices the two triangles share.

3.4 Computation of Galerkin Integrals Using Spherical Harmonics

In the zero-touch case, because the two triangles do not touch, the double surface integral is regular and can be computed via standard numerical or semi-analytical means. However, in this section, we present an analytical method for computing the integral in this case. This method was inspired by the fast multipole method, and uses spherical harmonics and multipole and local expansions and translations. Similar methods were presented in [25, 61] as part of fast multipole-accelerated solvers for problems in elastostatics. However, we build on the methods described in these references by: (1) adapting them to the kernels considered in Secs. 2.1 and 3.3; (2) computing the multipole expansion coefficients for a triangle exactly using Gaussian quadrature; and (3) controlling the error by adaptively truncating the multipole expansions and subdividing the problem when necessary.

3.4.1 Derivation of Method

We want to compute the following double surface integral over a source triangle, S, and a receiver triangle, R:

$$I = \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) \int_{\mathbf{x}'\in\mathrm{S}} (\sigma_0' + \mathbf{p}'\cdot\mathbf{x}')\, G(\mathbf{x}-\mathbf{x}')\, dS(\mathbf{x}')\, dS(\mathbf{x})\,. \tag{3.8}$$

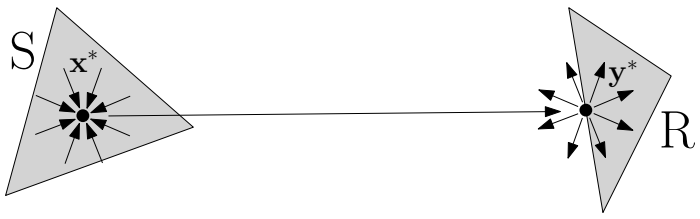

Figure 3.2: A diagram showing how to compute the double surface integral for two triangles that do not touch.

This section gives an analytical method for computing this integral based on spherical harmonics expansions and translations (see Appx. B). A visual representation of the steps in this method can be seen in Fig. 3.2.

First, expand the Green's function as a multipole expansion:

$$\begin{aligned} I &= \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) \int_{\mathbf{x}'\in\mathrm{S}} (\sigma_0' + \mathbf{p}'\cdot\mathbf{x}') \\ &\times \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} R_{n'}^{m'*}(\mathbf{x}'-\mathbf{x}^*)\, S_{n'}^{m'}(\mathbf{x}-\mathbf{x}^*) \\ &\times dS(\mathbf{x}')\, dS(\mathbf{x})\,. \end{aligned} \tag{3.9}$$

where the expansion center, $\mathbf{x}^*$, is near the source triangle. Ideally, $\mathbf{x}^*$ should be chosen so that the sphere centered around $\mathbf{x}^*$ and completely containing the source triangle can be made as small as possible. There are actually only four possible choices of $\mathbf{x}^*$: the midpoints of the three edges of the source triangle and the center of the source triangle's circumsphere. The one corresponding to the smallest sphere that completely contains the source triangle is chosen.

Second, rearrange Eq. (3.9) by moving the double sum and the $S_{n'}^{m'}$ outside the inside

integral:

$$\begin{aligned} I &= \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} \\ &\times \int_{\mathbf{x}'\in\mathrm{S}} (\sigma_0' + \mathbf{p}'\cdot\mathbf{x}') \, R_{n'}^{m'*} (\mathbf{x}' - \mathbf{x}^*) \, dS(\mathbf{x}') \\ &\times S_{n'}^{m'} (\mathbf{x} - \mathbf{x}^*) \, dS(\mathbf{x}) . \end{aligned} \tag{3.10}$$

The integral over the source triangle computes the expansion coefficients for the multipole expansion that represents the potential due to the linear source distribution over the source triangle:

$$I = \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) \left(\sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} a_{n'}^{m'} S_{n'}^{m'} (\mathbf{x} - \mathbf{x}^*) \right) dS(\mathbf{x}) , \tag{3.11}$$

where

$$a_{n'}^{m'} = \int_{\mathbf{x}'\in\mathrm{S}} (\sigma_0' + \mathbf{p}'\cdot\mathbf{x}') \, R_{n'}^{m'*} (\mathbf{x}' - \mathbf{x}^*) \, dS(\mathbf{x}') . \tag{3.12}$$

Analytical expressions for computing the integral in Eq. (3.12) are available, e.g. in [62,63]. In [64], a recursive algorithm for computing the expansion coefficients was presented: only $a_{n'}^{m'}$ for lower order and degree need to be computed explicitly; the others can be computed recursively from them. However, we make the following observation: $R_{n'}^{m'}$ is polynomial, so the integrand in Eq. (3.12) is polynomial. Thus, the integral can be computed exactly via Gaussian quadrature. A similar approach was used in [6, 65], although the integration domains in these references were lines and boxes, not triangles. We use the techniques given in [66] for performing Gaussian quadrature over the triangles.

Third, translate the multipole expansion centered around $\mathbf{x}^*$ to a local expansion

centered around $\mathbf{y}^*$, where $\mathbf{y}^*$ is near the receiver triangle:

$$I = \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) \left(\sum_{n=0}^{\infty} \sum_{m=-n}^{n} b_n^m R_n^m (\mathbf{x} - \mathbf{y}^*) \right) dS(\mathbf{x}). \tag{3.13}$$

Ideally, $\mathbf{y}^*$ should be chosen so that the sphere centered around $\mathbf{y}^*$ and completely containing the receiver triangle can be made as small as possible. The procedure for choosing $\mathbf{y}^*$ is the same as for choosing $\mathbf{x}^*$. The local expansion coefficients, b_n^m, are computed from the multipole expansion coefficients, $a_{n'}^{m'}$, via a multipole-to-local translation (see Appx. B). There are several translation methods available, and a good overview of them can be found online in [67]. For example, one popular method is the point and shoot method, whose computational cost scales as p^3.

Fourth, rearrange Eq. (3.13) by moving the double sum and the expansion coefficients outside the integral:

$$I = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} b_n^m c_n^m, \tag{3.14}$$

where

$$c_n^m = \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) R_n^m (\mathbf{x} - \mathbf{y}^*) dS(\mathbf{x}). \tag{3.15}$$

Like before, since R_n^m is polynomial, the integral in Eq. (3.15) can be computed exactly via Gaussian quadrature.

The above analysis assumes the source distribution over the source triangle and the weight function over the receiver triangle are linear. Higher-order basis functions, such as quadratic or cubic functions, could be used with very few changes. Indeed, the only change that must be made is to increase the order of the Gaussian quadrature to

account for the increase in the degree of the integrand. The above analysis also assumes the source and receiver surfaces are planar triangles. Arbitrary surfaces, e.g., curved elements, would be possible provided there was an integration method for that surface. One possible method for doing this would be to transform the surfaces into triangles via a change of variables, and use the same Gaussian quadrature as before. Of course, the integrand of the transformed integral would now include the Jacobian associated with the change of variables. In this dissertation, however, only planar linear triangular elements are considered.

3.4.2 Error Control

The expressions derived in Secs. 3.4.1 involving multipole and local expansions must be truncated so that they can be implemented in code. For example, the last expression given in Sec. 3.4.1 becomes

$$I \approx I_p = \sum_{n=0}^{p-1} \sum_{m=-n}^{n} b_n^m c_n^m, \tag{3.16}$$

where only p^2 terms have been kept. Obviously, the expression is exact as $p \to \infty$, but there are truncation errors when $p < \infty$. These errors come from two sources: (1) the construction of the multipole expansion at the source triangle; and (2) the translation of the multipole expansion to a local expansion at the receiver triangle. Luckily, these expansions converge geometrically, so these errors can be precisely controlled by picking an appropriate value of p.

Theoretical bounds for these errors as a function of p have been derived by many

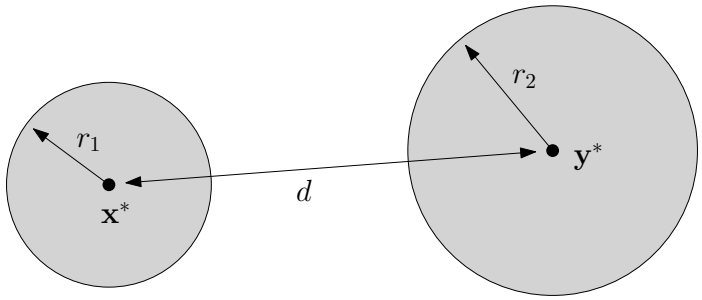

Figure 3.3: The radii and distances used to compute the geometric factor, η, given in Eq. (3.18).

authors over the years. A good overview is given in [68]. The relative error is bounded by:

$$\varepsilon = \left| \frac{I_p - I}{I} \right| \leq A\eta^p, \tag{3.17}$$

where the error constant, A, depends on the problem being solved, and

$$\eta = \frac{\max(r_1, r_2)}{d - \min(r_1, r_2)}, \tag{3.18}$$

where r_1 is the radius of the multipole expansion's bounding sphere, r_2 is the radius of the local expansion's bounding sphere, and $d = |\mathbf{y}^* - \mathbf{x}^*|$ (see Fig. 3.3). Using Eq. (3.17), we can easily pick a truncation number that gives us a desired accuracy:

$$p = \left\lceil \log\left(\frac{\varepsilon}{A}\right) \Big/ \log(\eta) \right\rceil, \tag{3.19}$$

where $\lceil x \rceil$ is the ceiling of x.

There are two issues. The number of terms in Eq. (3.16) grows as p^2, so the memory costs grow as p^2 as well. In addition, the computational costs grow as p^3 due to the

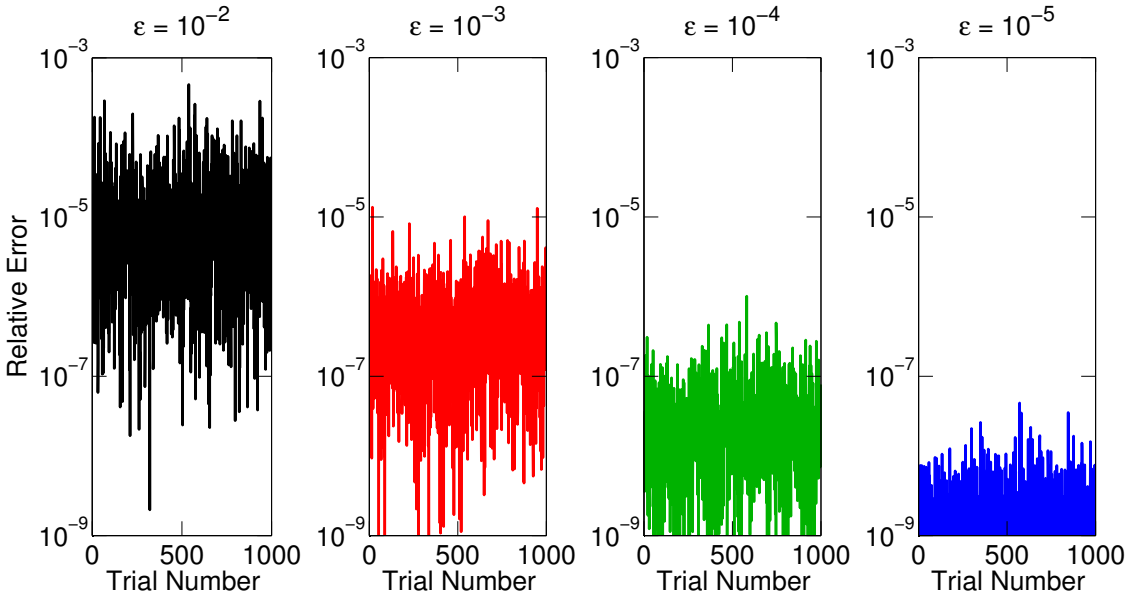

Figure 3.4: These graphs shows the actual relative errors when computing the double surface integrals over 1,000 randomly generated pairs of triangles using the analytical method for several different error bounds. In all cases, these errors were below the error bounds.

multipole-to-local translation (assuming the point and shoot method is used). In the event that p becomes too large, we divide the larger triangle into four smaller triangles and recurse. We do the same in the event that the two bounding spheres overlap.

Second, while values of A have been derived for special cases, such as point sources, they have not for the case of triangles. Instead of attempting to derive such a value analytically, we computed one experimentally. The experiment worked as follows. We generated 10,000 pairs of randomly placed triangles, where, in each pair, the two triangles did not touch. For each pair, we computed I using a semi-analytical method (using a high-order Gaussian quadrature for the outside integral) and I_p for $p = 1, 2, \ldots, 10$. We chose a value for A so that the experimental data satisfied Eq. (3.17). We ran this experiment several times, and found A to be between 0.3 and 0.4, so we conservatively set $A = 0.4$.

We ran a second experiment to verify that $A = 0.4$ works well. The experiment

worked as follows. We generated 1,000 pairs of triangles like before. For each pair, we computed I using the analytical method for several different choices of ε: 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}. Again, we used the value of I returned by the semi-analytical method as the reference value. In Fig. 3.4, the actual errors (along the y axis) for each pair of triangles (along the x axis) are plotted for each choice of ε (the different colored curves). In all cases, the realized errors are below the desired error bounds.

The speed of the analytical method depends strongly on the truncation number. The number of expansion coefficients grows as p^2, and the computational cost of the analytical method grows as p^3. For pairs of triangles that are close to each other and require a large p, the analytical method can be slower than semi-analytical methods. However, for pairs of triangles that are far away from each other, p will be small, so the analytical method will be faster. In addition, the method can be used as part of a fast multipole-accelerated BEM. The method is used to compute the interactions between large groups of triangles instead of between pairs of individual triangles. In this case, the analytical method has great value and can provide tremendous speedups, as was seen in the references given at the beginning of Sec. 3.4.

3.5 Computation of Galerkin Integrals Using Subdivision and Scaling Method

In the one-, two-, and three-touch cases, because the two triangles touch, the double surface integrals are not regular, so they cannot be computed using standard numerical or semi-analytical means, or even the analytical method described in Sec. 3.4.1. In this

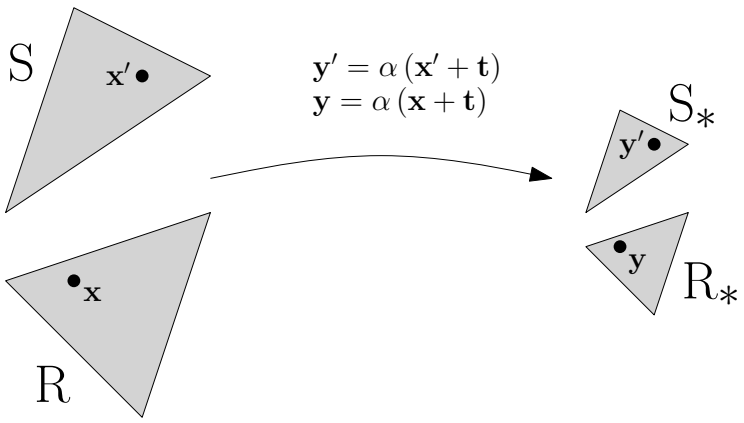

Figure 3.5: A diagram showing the process for transforming the pair of triangles, S and R, into the pair of triangles, S_* and R_*.

section, we present a novel method for dealing with these integrals. The approach relies on several scaling properties of the integrals and the kernels being integrated. The method works in the following way: (1) the integral is broken up into several smaller integrals; (2) some of these integrals are related back to the original integral via scaling and symmetry arguments; and (3) the terms are rearranged to yield an expression for the original integral that only requires computing regular integrals explicitly (all other integrals are computed implicitly).

3.5.1 Preliminaries: Scaling Results

We want to compute the following double surface integral over a source triangle, S, and a receiver triangle, R:

$$I = \int_{\mathbf{x}\in\mathrm{R}} (\sigma_0 + \mathbf{p}\cdot\mathbf{x}) \int_{\mathbf{x}'\in\mathrm{S}} (\sigma_0' + \mathbf{p}'\cdot\mathbf{x}')\, F(\mathbf{x}-\mathbf{x}')\, dS(\mathbf{x}')\, dS(\mathbf{x}), \tag{3.20}$$

where $F(\mathbf{r})$ is any kernel that has the following scaling property:

$$F(\alpha \mathbf{r}) = s(\alpha) F(\mathbf{r}). \tag{3.21}$$

To begin, let us break the integral into four smaller integrals by expanding the product of the two linear functions:

$$\begin{aligned} I &= \int_{\mathbf{x} \in \mathrm{R}} \int_{\mathbf{x}' \in \mathrm{S}} \\ &\times \left(\sigma_0 \sigma_0' + \sigma_0 (\mathbf{p}' \cdot \mathbf{x}') + (\mathbf{p} \cdot \mathbf{x}) \sigma_0' + (\mathbf{p} \cdot \mathbf{x})(\mathbf{p}' \cdot \mathbf{x}')\right) \\ &\times F(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}') \, dS(\mathbf{x}), \end{aligned} \tag{3.22}$$

$$I = \sigma_0 \sigma_0' I^1 + \sigma_0 I^{\mathbf{p}'} + \sigma_0' I^{\mathbf{p}} + I^{\mathbf{p}'\mathbf{p}}, \tag{3.23}$$

where

$$I^1 = \int_{\mathbf{x} \in \mathrm{R}} \int_{\mathbf{x}' \in \mathrm{S}} F(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}') \, dS(\mathbf{x}), \tag{3.24}$$

$$I^{\mathbf{p}'} = \int_{\mathbf{x} \in \mathrm{R}} \int_{\mathbf{x}' \in \mathrm{S}} (\mathbf{p}' \cdot \mathbf{x}') F(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}') \, dS(\mathbf{x}), \tag{3.25}$$

$$I^{\mathbf{p}} = \int_{\mathbf{x} \in \mathrm{R}} \int_{\mathbf{x}' \in \mathrm{S}} (\mathbf{p} \cdot \mathbf{x}) F(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}') \, dS(\mathbf{x}), \tag{3.26}$$

$$I^{\mathbf{p}'\mathbf{p}} = \int_{\mathbf{x} \in \mathrm{R}} \int_{\mathbf{x}' \in \mathrm{S}} (\mathbf{p}' \cdot \mathbf{x}')(\mathbf{p} \cdot \mathbf{x}) F(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}') \, dS(\mathbf{x}). \tag{3.27}$$

Consider two different triangles, S_* and R_*, which, taken together, are scaled and translated versions of S and R, also taken together (see Fig. 3.5). In other words, given a

pair of points, $\mathbf{x}', \mathbf{x} \in \mathrm{S} \cup \mathrm{R}$, there is a corresponding pair of points, $\mathbf{y}', \mathbf{y} \in \mathrm{S}_* \cup \mathrm{R}_*$, given by

$$\mathbf{y}' = \alpha\left(\mathbf{x}' + \mathbf{t}\right), \quad \mathbf{y} = \alpha\left(\mathbf{x} + \mathbf{t}\right). \tag{3.28}$$

Suppose we want to compute the same integral as before, except over S_* and R_*:

$$I_* = \int_{\mathbf{y} \in \mathrm{R}_*} \left(\sigma_0 + \mathbf{p} \cdot \mathbf{y}\right) \int_{\mathbf{y}' \in \mathrm{S}_*} \left(\sigma_0' + \mathbf{p}' \cdot \mathbf{y}'\right) F\left(\mathbf{y} - \mathbf{y}'\right) dS\left(\mathbf{y}'\right) dS\left(\mathbf{y}\right). \tag{3.29}$$

The integrand is exactly the same as before. The only thing we have changed is the integration domain from $\mathrm{S} \times \mathrm{R}$ to $\mathrm{S}_* \times \mathrm{R}_*$. Like before, let us break the integral into four smaller integrals by expanding the product of the two linear functions:

$$I_* = \sigma_0 \sigma_0' I_*^1 + \sigma_0 I_*^{\mathbf{p}'} + \sigma_0' I_*^{\mathbf{p}} + I_*^{\mathbf{p}'\mathbf{p}}, \tag{3.30}$$

where I_*^1, $I_*^{\mathbf{p}'}$, $I_*^{\mathbf{p}}$, and $I_*^{\mathbf{p}'\mathbf{p}}$ are like their non-starred versions from before, except for the different integration domain.

Theorem 1. The four integrals over S_* and R_*, I_*^1, $I_*^{\mathbf{p}'}$, $I_*^{\mathbf{p}}$, and $I_*^{\mathbf{p}'\mathbf{p}}$, can be expressed in terms of the four integrals over S and R, I^1, $I^{\mathbf{p}'}$, $I^{\mathbf{p}}$, and $I^{\mathbf{p}'\mathbf{p}}$:

$$I_*^1 = s\left(\alpha\right) \alpha^4 I^1, \tag{3.31}$$

$$I_*^{\mathbf{p}'} = s\left(\alpha\right) \alpha^5 \left(I^{\mathbf{p}'} + \left(\mathbf{p}' \cdot \mathbf{t}\right) I^1\right), \tag{3.32}$$

$$I_*^{\mathbf{p}} = s(\alpha)\,\alpha^5\left(I^{\mathbf{p}} + (\mathbf{p}\cdot\mathbf{t})\,I^1\right), \tag{3.33}$$

$$I_*^{\mathbf{p}'\mathbf{p}} = s(\alpha)\,\alpha^6\left(I^{\mathbf{p}'\mathbf{p}} + (\mathbf{p}\cdot\mathbf{t})\,I^{\mathbf{p}'} + (\mathbf{p}'\cdot\mathbf{t})\,I^{\mathbf{p}} + (\mathbf{p}\cdot\mathbf{t})\,(\mathbf{p}'\cdot\mathbf{t})\,I^1\right). \tag{3.34}$$

Proof. To prove Eqs. (3.31) - (3.34) in Theorem 1: (1) make the change of variables in Eq. (3.28); (2) use the scaling property of $F(\mathbf{r})$ in Eq. (3.21); and (3) break the resulting integral into one or more integrals that are equal to the original four integrals, I^1, $I^{\mathbf{p}'}$, $I^{\mathbf{p}}$, and $I^{\mathbf{p}'\mathbf{p}}$.

To prove Eq. (3.31):

$$I_*^1 = \int_{\mathbf{y}\in \mathrm{R}_*}\int_{\mathbf{y}'\in \mathrm{S}_*} F(\mathbf{y}-\mathbf{y}')\,dS(\mathbf{y}')\,dS(\mathbf{y}), \tag{3.35}$$

$$I_*^1 = \int_{\mathbf{x}\in \mathrm{R}}\int_{\mathbf{x}'\in \mathrm{S}} F(\alpha(\mathbf{x}+\mathbf{t}) - \alpha(\mathbf{x}'+\mathbf{t}))\,dS(\alpha(\mathbf{x}'+\mathbf{t}))\,dS(\alpha(\mathbf{x}+\mathbf{t})), \tag{3.36}$$

$$I_*^1 = s(\alpha)\,\alpha^4\int_{\mathbf{x}\in \mathrm{R}}\int_{\mathbf{x}'\in \mathrm{S}} F(\mathbf{x}-\mathbf{x}')\,dS(\mathbf{x}')\,dS(\mathbf{x}), \tag{3.37}$$

$$I_*^1 = s(\alpha)\,\alpha^4 I^1. \tag{3.38}$$

To prove Eq. (3.32):

$$I_*^{\mathbf{p}'} = \int_{\mathbf{y}\in \mathrm{R}_*}\int_{\mathbf{y}'\in \mathrm{S}_*} (\mathbf{p}'\cdot\mathbf{y}')\,F(\mathbf{y}-\mathbf{y}')\,dS(\mathbf{y}')\,dS(\mathbf{y}), \tag{3.39}$$

$$I_*^{\mathbf{p}'} = s(\alpha)\,\alpha^5\int_{\mathbf{x}\in \mathrm{R}}\int_{\mathbf{x}'\in \mathrm{S}} (\mathbf{p}'\cdot(\mathbf{x}'+\mathbf{t}))\,F(\mathbf{x}-\mathbf{x}')\,dS(\mathbf{x}')\,dS(\mathbf{x}), \tag{3.40}$$

$$I_*^{\mathbf{p}'} = s(\alpha)\,\alpha^5\left(I^{\mathbf{p}'} + (\mathbf{p}'\cdot\mathbf{t})\,I^1\right). \tag{3.41}$$

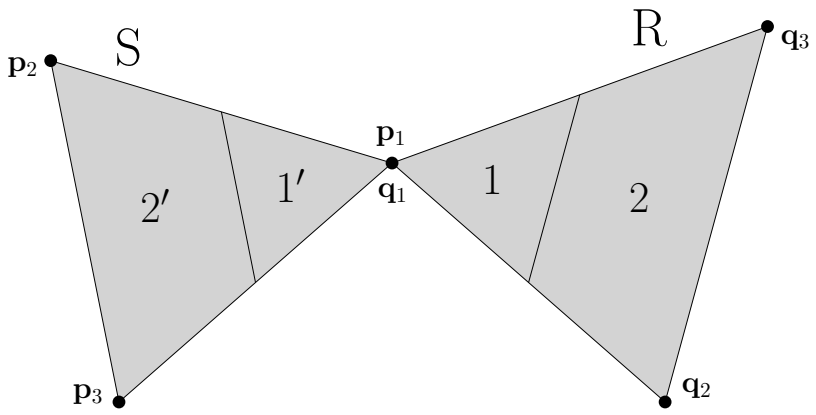

Figure 3.6: One-Touch Case. A diagram showing how to compute the double surface integral for two triangles that share a vertex.

The analysis to prove Eq. (3.33) is the same as for Eq. (3.32).

To prove Eq. (3.34):

$$I_*^{\mathbf{p}'\mathbf{p}} = \int_{\mathbf{y}\in \mathrm{R}_*} \int_{\mathbf{y}'\in \mathrm{S}_*} (\mathbf{p}\cdot\mathbf{y})\,(\mathbf{p}'\cdot\mathbf{y}')\,F(\mathbf{y}-\mathbf{y}')\,dS(\mathbf{y}')\,dS(\mathbf{y})\,, \tag{3.42}$$

$$I_*^{\mathbf{p}'\mathbf{p}} = s(\alpha)\,\alpha^6 \int_{\mathbf{x}\in \mathrm{R}} \int_{\mathbf{x}'\in \mathrm{S}} (\mathbf{p}\cdot(\mathbf{x}+\mathbf{t}))\,(\mathbf{p}'\cdot(\mathbf{x}'+\mathbf{t}))\,F(\mathbf{x}-\mathbf{x}')\,dS(\mathbf{x}')\,dS(\mathbf{x})\,, \tag{3.43}$$

$$I_*^{\mathbf{p}'\mathbf{p}} = s(\alpha)\,\alpha^6 \left(I^{\mathbf{p}'\mathbf{p}} + (\mathbf{p}\cdot\mathbf{t})\,I^{\mathbf{p}'} + (\mathbf{p}'\cdot\mathbf{t})\,I^{\mathbf{p}} + (\mathbf{p}\cdot\mathbf{t})\,(\mathbf{p}'\cdot\mathbf{t})\,I^1 \right). \tag{3.44}$$

□

3.5.2 One-Touch Case

Consider the one-touch case in Fig. 3.6. Without loss of generality, assume the vertex that the two triangles share is located at the origin (i.e., $\mathbf{p}_1 = \mathbf{0}$). We divide each triangle into a triangle and a quadrilateral: the source triangle, S, is divided into $1'$ and $2'$,

and the receiver triangle, R, is divided into 1 and 2. Note that $1'$ and 1 are similar to S and R ($1/2$ the size), respectively. We break the integral into three smaller integrals over these shapes.

Let us look at I^1:

$$I^1 = I^1_{1'1} + I^1_{1'2} + I^1_{2'\mathrm{R}}. \tag{3.45}$$

The subscripts denote the surfaces of integration. For example, $I^1_{1'1}$ is the integral over $1'$ and 1. Similarly, $I^1_{2'\mathrm{R}}$ is the integral over $2'$ and R. The I^1 without a subscript is the original integral over S and R. In Eq. (3.45), the two integrals, $I^1_{1'2}$ and $I^1_{2'\mathrm{R}}$ are zero-touch integrals. Because we know how to compute them, let us combine them into a single integral:

$$I^1 = I^1_{1'1} + I^1_{\text{remainder}}. \tag{3.46}$$

The integral, $I^1_{1'1}$, however, has the same problem as the original integral: $1'$ and 1 share a vertex. Fortunately, $1'$ and 1 are scaled and translated versions of S and R. We derived in Theorem 1 that $I^1_* = s\left(\alpha\right)\alpha^4 I^1$. Since the pair of triangles, $1'$ and 1, is $1/2$ the size of the original pair, S and R, $\alpha = 1/2$, so

$$I^1_{1'1} = s\left(\frac{1}{2}\right)\frac{1}{16}I^1. \tag{3.47}$$

Inserting this into Eq. (3.46) and rearranging,

$$I^1 = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{16}\right)^{-1} I^1_{\text{remainder}}. \tag{3.48}$$

The integral, $I^1_{1'1}$, is computed implicitly in this expression.

Now, let us look at $I^{\mathbf{p}'}$:

$$I^{\mathbf{p}'} = I^{\mathbf{p}'}_{1'1} + I^{\mathbf{p}'}_{\text{remainder}}, \tag{3.49}$$

where $I^{\mathbf{p}'}_{\text{remainder}} = I^{\mathbf{p}'}_{1'2} + I^{\mathbf{p}'}_{2'\text{R}}$. We derived in Theorem 1 that

$$I^{\mathbf{p}'}_* = s(\alpha)\,\alpha^5\left(I^{\mathbf{p}'} + (\mathbf{p}'\cdot\mathbf{t})\,I^1\right). \tag{3.50}$$

Like before, $\alpha = 1/2$, but we still need to determine $\mathbf{t}$. When $\alpha = 1/2$, $\mathbf{t}$ is the point that does not change during the scaling and translation. This is simply the vertex that the two triangles share, so $\mathbf{t} = \mathbf{p}_1$. Because we assumed that $\mathbf{p}_1 = \mathbf{0}$, $\mathbf{t} = \mathbf{0}$ as well, so

$$I^{\mathbf{p}'}_{11} = s\left(\frac{1}{2}\right)\frac{1}{32}I^{\mathbf{p}'}. \tag{3.51}$$

Plugging this into Eq. (3.49) and rearranging,

$$I^{\mathbf{p}'} = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{32}\right)^{-1} I^{\mathbf{p}'}_{\text{remainder}}. \tag{3.52}$$

These same procedures can be used to compute $I^{\mathbf{p}}$ and $I^{\mathbf{p}'\mathbf{p}}$:

$$I^{\mathbf{p}} = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{32}\right)^{-1} I^{\mathbf{p}}_{\text{remainder}}, \tag{3.53}$$

$$I^{\mathbf{p}'\mathbf{p}} = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{64}\right)^{-1} I^{\mathbf{p}'\mathbf{p}}_{\text{remainder}}. \tag{3.54}$$

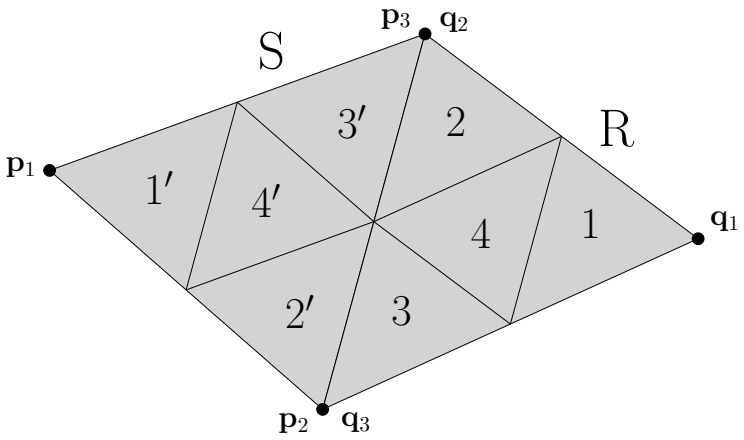

Figure 3.7: Two-Touch Case. A diagram showing how to compute the double surface integral for two triangles that share an edge.

3.5.3 Two-Touch Case

Consider the two-touch case in Fig. 3.7. Without loss of generality, assume the midpoint of the edge that the two triangles share is located at the origin (i.e., $\mathbf{p}_2 + \mathbf{p}_3 = \mathbf{0}$). Like in the one-touch case, we solve this problem by breaking the integral into several smaller integrals.

Let us look at I^1:

$$I^1 = I^1_{1'\mathrm{R}} + I^1_{2'2} + I^1_{2'3} + I^1_{2'4} + I^1_{3'2} + I^1_{3'3} + I^1_{3'4} + I^1_{4'2} + I^1_{4'3} + I^1_{4'4} + I^1_{\mathrm{S}1} - I^1_{1'1}. \quad (3.55)$$

The integrals, $I^1_{1'\mathrm{R}}$, $I^1_{\mathrm{S}1}$, and $I^1_{1'1}$ are zero-touch integrals, and the integrals, $I^1_{2'2}$, $I^1_{2'4}$, $I^1_{3'3}$, $I^1_{3'4}$, $I^1_{4'2}$, $I^1_{4'3}$, and $I^1_{4'4}$, are one-touch integrals. Since we know how to compute them, let us combine them into a single integral:

$$I^1 = I^1_{2'3} + I^1_{3'2} + I^1_{\text{remainder}}. \quad (3.56)$$

That leaves $I^1_{2'3}$ and $I^1_{3'2}$, which have the same problem as the original integral: they correspond to pairs of triangles that share an edge. Fortunately, the pair, $2'$ and 3, and the pair, $3'$ and 2, are each scaled and translated versions of the original pair, S and R. Using Eq. (3.31) from Theorem 1, we have

$$I^1_{2'3} = s\left(\frac{1}{2}\right)\frac{1}{16}I^1, \quad I^1_{3'2} = s\left(\frac{1}{2}\right)\frac{1}{16}I^1. \tag{3.57}$$

Substituting these into Eq. (3.56) and rearranging,

$$I^1 = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{8}\right)^{-1} I^1_{\text{remainder}}. \tag{3.58}$$

The integrals, $I^1_{2'3}$ and $I^1_{3'2}$, are computed implicitly in this expression.

Now, let us look at $I^{\mathbf{p}'}$:

$$I^{\mathbf{p}'} = I^{\mathbf{p}'}_{2'3} + I^{\mathbf{p}'}_{3'2} + I^{\mathbf{p}'}_{\text{remainder}}. \tag{3.59}$$

where, again, the ten integrals that we know how to compute have been combined into $I^{\mathbf{p}'}_{\text{remainder}}$. Using Eq. (3.32) from Theorem 1, we have

$$I^{\mathbf{p}'}_{2'3} + I^{\mathbf{p}'}_{3'2} = s\left(\frac{1}{2}\right)\frac{1}{16}I^{\mathbf{p}'} + s\left(\frac{1}{2}\right)\frac{1}{32}\left(\mathbf{p}' \cdot (\mathbf{t}_{2'3} + \mathbf{t}_{3'2})\right) I^1. \tag{3.60}$$

We need to determine $\mathbf{t}_{2'3}$ and $\mathbf{t}_{3'2}$. As discussed earlier, when $\alpha = 1/2$, the translation vector, $\mathbf{t}$, corresponds to the point that does not change during the transformation. Thus, for the pair, $2'$ and 3, that is $\mathbf{p}_2$, and for the pair, 2 and $3'$, that is $\mathbf{p}_3$. The edge that S and R

share is centered around the origin, so $\mathbf{t}_{2'3} + \mathbf{t}_{3'2} = \mathbf{0}$, which means that

$$I^{\mathbf{p}'}_{2'3} + I^{\mathbf{p}'}_{3'2} = s\left(\frac{1}{2}\right)\frac{1}{16}I^{\mathbf{p}'}. \tag{3.61}$$

Inserting this into Eq. (3.59) and rearranging,

$$I^{\mathbf{p}'} = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{16}\right)^{-1} I^{\mathbf{p}'}_{\text{remainder}}. \tag{3.62}$$

The same analysis can be used for computing $I^{\mathbf{p}}$:

$$I^{\mathbf{p}} = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{16}\right)^{-1} I^{\mathbf{p}}_{\text{remainder}}. \tag{3.63}$$

Finally, let us look at $I^{\mathbf{p}'\mathbf{p}}$:

$$I^{\mathbf{p}'\mathbf{p}} = I^{\mathbf{p}'\mathbf{p}}_{2'3} + I^{\mathbf{p}'\mathbf{p}}_{3'2} + I^{\mathbf{p}'\mathbf{p}}_{\text{remainder}}. \tag{3.64}$$

Using Eq. (3.34) from Theorem 1, we have

$$I^{\mathbf{p}'\mathbf{p}}_{2'3} + I^{\mathbf{p}'\mathbf{p}}_{3'2} = s\left(\frac{1}{2}\right)\frac{1}{32}I^{\mathbf{p}'\mathbf{p}} + s\left(\frac{1}{2}\right)\frac{1}{64}\left(\left(\mathbf{p}\cdot\mathbf{t}_{2'3}\right)\left(\mathbf{p}'\cdot\mathbf{t}_{2'3}\right) + \left(\mathbf{p}\cdot\mathbf{t}_{3'2}\right)\left(\mathbf{p}'\cdot\mathbf{t}_{3'2}\right)\right)I^{1}. \tag{3.65}$$

The terms linear in $\mathbf{p}'$ and $\mathbf{p}$ disappear because $\mathbf{t}_{2'3} + \mathbf{t}_{3'2} = \mathbf{0}$, but the other terms remain.

Plugging Eq. (3.65) into Eq. (3.64) and rearranging,

$$I^{\mathbf{p}'\mathbf{p}} = \left(1 - s\left(\frac{1}{2}\right)\frac{1}{32}\right)^{-1}\left(I^{\mathbf{p}'\mathbf{p}}_{\text{remainder}} + a^{\mathbf{p}'\mathbf{p}}\right), \tag{3.66}$$

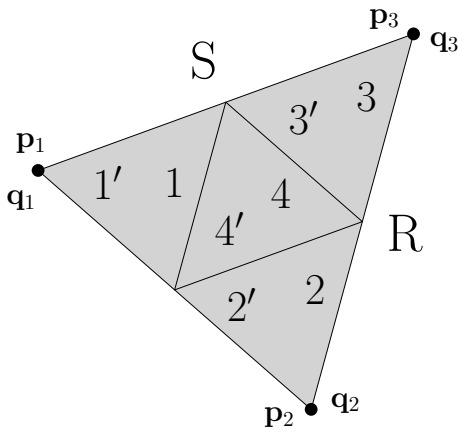

Figure 3.8: Three-Touch Case. A diagram showing how to compute the double surface integral for two triangles that are the same.

where

$$a^{\mathbf{p}'\mathbf{p}} = s\left(\frac{1}{2}\right)\frac{1}{64}\left(\left(\mathbf{p}\cdot\mathbf{t}_{2'3}\right)\left(\mathbf{p}'\cdot\mathbf{t}_{2'3}\right)+\left(\mathbf{p}\cdot\mathbf{t}_{3'2}\right)\left(\mathbf{p}'\cdot\mathbf{t}_{3'2}\right)\right)I^1. \tag{3.67}$$

3.5.4 Three-Touch Case

Consider the three-touch case in Fig. 3.8. In this case, the two triangles are the same. Without loss of generality, assume that the centroids of the two triangles are located at the origin (i.e., $\mathbf{p}_1+\mathbf{p}_2+\mathbf{p}_3=\mathbf{0}$). Like in the one- and two-touch cases, we solve this problem by breaking the integral into several smaller integrals.

Let us look at I^1:

$$I^1 = \sum_{i=1}^{4}\sum_{j=1}^{4} I^1_{j'i}. \tag{3.68}$$

The integrals, $I^1_{1'2}$, $I^1_{1'3}$, $I^1_{2'1}$, $I^1_{2'3}$, $I^1_{3'1}$, and $I^1_{3'2}$, are one-touch integrals, and the integrals, $I^1_{1'4}$, $I^1_{2'4}$, $I^1_{3'4}$, $I^1_{4'1}$, $I^1_{4'2}$, and $I^1_{4'3}$, are two-touch integrals. Like before, because we know how to compute them, let us combine these 12 integrals into a single integral:

$$I^1 = I^1_{1'1}+I^1_{2'2}+I^1_{3'3}+I^1_{4'4}+I^1_{\text{remainder}}. \tag{3.69}$$

That leaves the four integrals, $I^1_{1'1}$, $I^1_{2'2}$, $I^1_{3'3}$, and $I^1_{4'4}$, which have the same problem as the original: they correspond to pairs of triangles that are the same. Using Eq. (3.31) from Theorem 1, we have

$$I^1_{1'1} + I^1_{2'2} + I^1_{3'3} = s\left(\frac{1}{2}\right)\frac{3}{16}I^1. \tag{3.70}$$

The pair of triangles, $4'$ and 4, are not only scaled by a factor of $1/2$, but also rotated by 180 degrees. Because they are centered around the origin, we can achieve this rotation by setting $\alpha = -1/2$:

$$I^1_{4'4} = s\left(-\frac{1}{2}\right)\frac{1}{16}I^1. \tag{3.71}$$

Substituting these into Eq. (3.69) and rearranging,

$$I^1 = \left(1 - s\left(\frac{1}{2}\right)\frac{3}{16} - s\left(-\frac{1}{2}\right)\frac{1}{16}\right)^{-1} I^1_{\text{remainder}}. \tag{3.72}$$

The same analysis from here and before can be used to compute expressions for $I^{\mathbf{p}'}$, $I^{\mathbf{p}}$, and $I^{\mathbf{p}'\mathbf{p}}$:

$$I^{\mathbf{p}'} = \left(1 - s\left(\frac{1}{2}\right)\frac{3}{32} + s\left(-\frac{1}{2}\right)\frac{1}{32}\right)^{-1} I^{\mathbf{p}'}_{\text{remainder}}, \tag{3.73}$$

$$I^{\mathbf{p}} = \left(1 - s\left(\frac{1}{2}\right)\frac{3}{32} + s\left(-\frac{1}{2}\right)\frac{1}{32}\right)^{-1} I^{\mathbf{p}}_{\text{remainder}}, \tag{3.74}$$

$$I^{\mathbf{p}'\mathbf{p}} = \left(1 - s\left(\frac{1}{2}\right)\frac{3}{64} - s\left(-\frac{1}{2}\right)\frac{1}{64}\right)^{-1} \left(I^{\mathbf{p}'\mathbf{p}}_{\text{remainder}} + a^{\mathbf{p}'\mathbf{p}}\right), \tag{3.75}$$

where

$$a^{\mathbf{p}'\mathbf{p}} = s\left(\frac{1}{2}\right)\frac{1}{64}\left(\left(\mathbf{p}\cdot\mathbf{t}_{1'1}\right)\left(\mathbf{p}'\cdot\mathbf{t}_{1'1}\right)+\left(\mathbf{p}\cdot\mathbf{t}_{2'2}\right)\left(\mathbf{p}'\cdot\mathbf{t}_{2'2}\right)+\left(\mathbf{p}\cdot\mathbf{t}_{3'3}\right)\left(\mathbf{p}'\cdot\mathbf{t}_{3'3}\right)\right)I^{1}, \tag{3.76}$$

and $\mathbf{t}_{1'1} = \mathbf{p}_1$, $\mathbf{t}_{2'2} = \mathbf{p}_2$, and $\mathbf{t}_{3'3} = \mathbf{p}_3$.

3.5.5 Extension to Higher-Order or Curved Elements

The analysis in Secs. 3.5.1, 3.5.2, 3.5.3, and 3.5.4 assumes the source distribution over the source triangle and the weight function over the receiver triangle are linear. Higher-order basis functions are possible, but would require additional work to realize. For example, constant elements require only the computation of I^1. Increasing the order to linear requires also $I^{\mathbf{p}'}$, $I^{\mathbf{p}}$, and $I^{\mathbf{p}'\mathbf{p}}$. Moving to quadratic or cubic elements would require more such terms. The analysis also assumes the source and receiver surfaces are planar triangles. This assumption is crucial for the subdivision scheme to work properly. As discussed in Sec. 3.4.1, curved elements could be realized by transforming them to planar triangles via a change of variables. Extending the methods presented in this chapter to higher-order or curved elements is likely possible, but we leave that as future work.

3.5.6 Implementation Details

The one-, two-, and three-touch integral computation methods described in Secs. 3.5.2, 3.5.3, and 3.5.4 are recursive in nature. A one-touch integral is decomposed into two zero-touch integrals. A two-touch integral is decomposed into three zero-touch integrals and seven one-touch integrals. A three-touch integral is decomposed into six one-touch

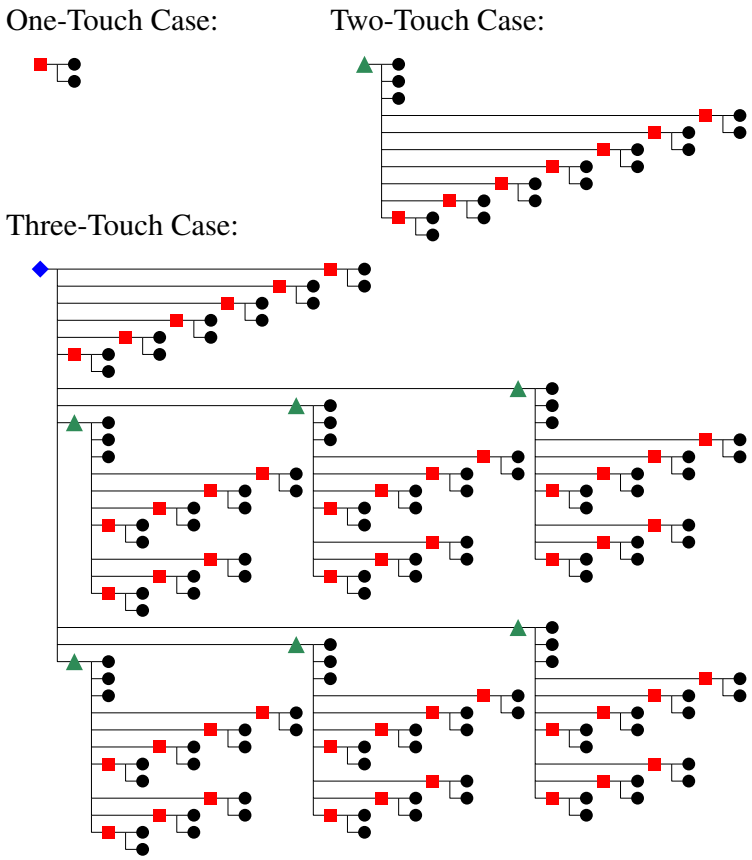

Figure 3.9: Recursion trees for the one-touch (top left), two-touch (top right), and three-touch (bottom) integral computation methods described in Secs. 3.5.2, 3.5.3, and 3.5.4. The black circles, red squares, green triangles, and blue diamonds correspond to zero-, one-, two-, and three-touch integrals, respectively.

integrals and six two-touch integrals. Recursion trees for these three cases can be seen in Fig. 3.9. The leaves at the bottom of these recursion trees are all zero-touch integrals. All combined, a one-touch integral requires two zero-touch integrals, a two-touch integral requires 17, and a three-touch integral requires 114. Indeed, to compute a one-, two-, or three-touch integral for a given kernel, we need only two things: (1) the scaling function, $s\left(\alpha\right)$, associated with that kernel; and (2) a method to compute a zero-touch integral for that kernel. For the two kernels, $F_1\left(\mathbf{r}\right)$ (the Green's function) and $F_2\left(\mathbf{r}\right)$ (the normal derivative of the Green's function), needed by the Galerkin BEM and given in Sec. 3.3, we

have both of these things. The scaling functions for the two kernels are: $s_1(\alpha) = |\alpha|^{-1}$ and $s_2(\alpha) = \alpha|\alpha|^{-3}$.

In Sec. 3.4.1, we presented an analytical method for computing a zero-touch integral. However, the analytical method is better for pairs of triangles that are relatively far away from each other. In the present case, because the pairs of triangles are typically much closer together, we use a semi-analytical method. In this method, the inside integral is computed analytically, and the outside integral is computed numerically via Gaussian quadrature. The use of the semi-analytical method introduces error, but this error can be precisely controlled by choosing the number of quadrature points. Let Q be this number. When computing one-, two-, and three-touch integrals, the only source of error comes from the zero-touch integrals, so Q can be used to control the error in these cases as well. Each zero-touch integral requires a Q-point quadrature, so the computational complexity of the one-, two-, and three-touch integral methods scale as $\mathrm{O}(ZQ)$, where Z is the number of zero-touch integrals that need to be computed in each method.

3.5.7 Numerical Examples

We ran a series of computational experiments to validate the one-, two-, and three-touch integral computation methods described in Secs. 3.5.2, 3.5.3, and 3.5.4. There are two kernels that need to be integrated: (1) $F_1(\mathbf{r})$, the Green's function; and (2) $F_2(\mathbf{r})$, the normal derivative of the Green's function.

To validate that the methods work for $F_1(\mathbf{r})$, we compared them to an exact method that was developed in [1] for computing the double surface integral in the three-touch case.

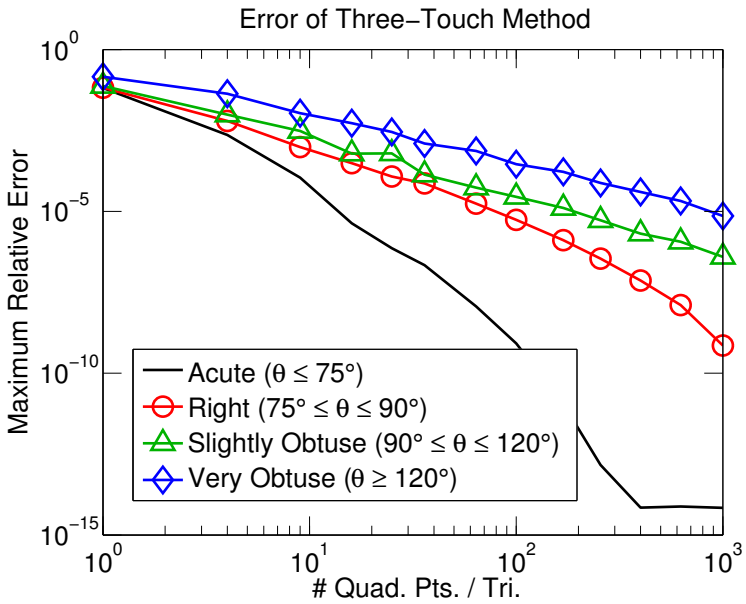

Figure 3.10: Comparison between the three-touch method described in Sec. 3.5.4 and the Eibert method described in [1].

We call this method the Eibert method. Because our three-touch method exercises our zero-, one-, and two-touch methods, this comparison provides a validation for all four methods.

The experiment worked as follows. We generated 10,000 randomly shaped triangles, and computed the three-touch integral for each using the Eibert method. Then, we computed the same integrals using our method with varying degrees of accuracy. As discussed in the previous section, the accuracy is governed by Q, which is the number of quadrature points used during the Gaussian quadrature in the zero-touch case. We varied the accuracy from low ($Q = 1$) to high ($Q = 1024$). After doing so, the triangles were divided up into four groups based on the largest interior angle, θ: (1) acute (nearly equilateral) triangles ($\theta \leq 75°$); (2) right triangles ($75° \leq \theta \leq 90°$); (3) slightly obtuse triangles ($90° \leq \theta \leq 120°$); and (4) very obtuse triangles ($\theta \geq 120°$). For each group, we plotted the maximum relative error in that group as a function of Q (see Fig. 3.10). In

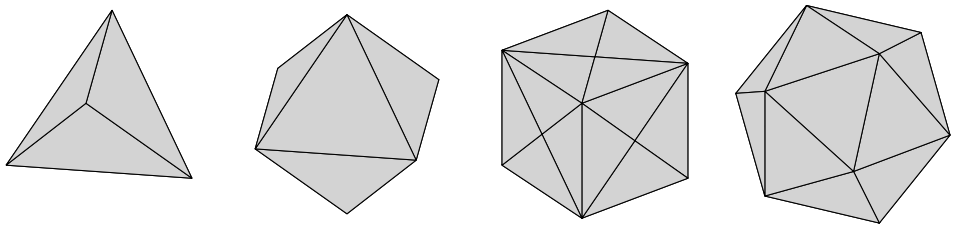

Figure 3.11: The tetrahedron, octahedron, cube, and icosahedron meshes used in the example indirect BEM problem. The tetrahedron mesh has 4 elements, the octahedron mesh has 8 elements, the cube mesh has 24 elements, and the icosahedron mesh has 20 elements.

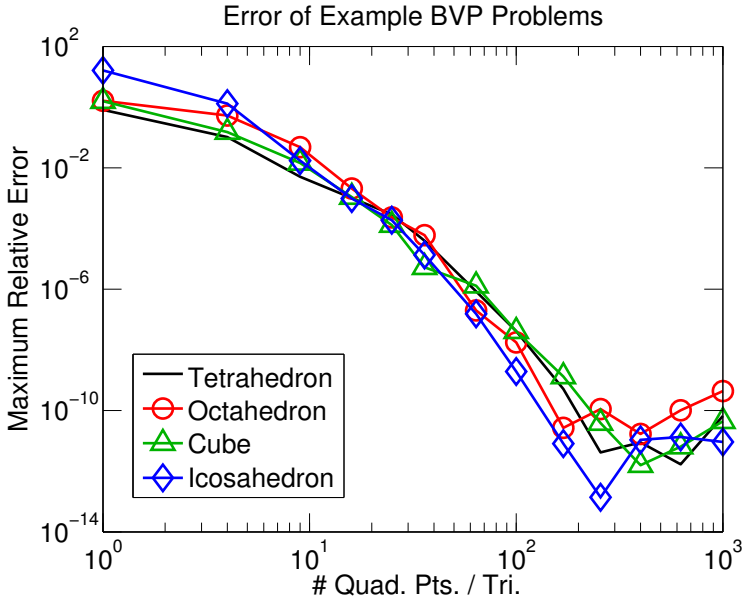

Figure 3.12: Error of the four example BVP problems for different levels of accuracy for the double surface integrals.

general, all four types of triangles become more accurate as Q increases. However, acute triangles performed the best, requiring only 144 quadrature points at the lowest level to achieve a maximum relative error of less than 10^{-10}. For a good BEM mesh, we expect most triangles to be acute.

Next, let us validate that the methods work for $F_2(\mathbf{r})$. To do so, we implemented the Galerkin BEM described in Sec. 2.2 using linear elements, and solved four example BVP problems. The four problems were exactly the same, except for the boundaries used.

The boundaries were: a tetrahedron, an octahedron, a cube, and an icosahedron (see Fig. 3.11). In each problem, the potential, ϕ, was set to zero outside the boundary. Inside the boundary, the potential was set to $\phi(\mathbf{x}) = a + \mathbf{b} \cdot \mathbf{x}$, where, in these examples, $a = 1$ and $\mathbf{b} = (0.5, 0.8, -0.7)$. The exact solution to this problem can be derived using the jump conditions given in Sec. 2.2. In fact, the single- and double-layer source distributions are piecewise linear along the boundaries, meaning that linear elements can provide an exact solution.

The experiment worked as follows. We computed the solution to each of the four problems using the indirect BEM. The integrals for when the kernel was $F_1(\mathbf{r})$ were computed as accurately as possible (i.e., $Q = 1024$). This left the integrals for when the kernel was $F_2(\mathbf{r})$ as the primary source of error. We varied the accuracy at which these integrals were computed from low ($Q = 1$) to high ($Q = 1024$), and computed the maximum relative error of the solution for each choice of Q. Fig. 3.12 shows the maximum relative error of the solution as a function of Q for each of the four different boundaries. As the accuracy of the integrals was increased, the total accuracy of the problem increased as well. For $Q > 200$, the maximum relative error drops to around 10^{-10} for all four geometries.

3.6 GPU Acceleration

3.6.1 Background

The graphics processing unit (GPU) is an auxiliary chip or card found on most computers that is dedicated to the graphics of the computer. This includes not only

rendering the desktop in Windows, but also performing advanced 3-D graphics in video games and movies. In fact, it's the latter which has driven the amazing advancements in GPU technology over the past three decades. As video game makers have pushed the hardware limits of current generation GPUs, GPU manufacturers, such as AMD and NVIDIA, have responded by continuing to release better and better GPUs.

Originally, the graphics pipelines on these cards was relatively fixed. The behavior of the vertex and pixel/fragment shaders were hard-wired, and software interfaces such as DirectX and OpenGL allowed video game and other software developers to access this functionality. Eventually, AMD and NVIDIA allowed every part of the rendering pipeline to be customized. This could be done via DirectX's High-Level Shading Language (HLSL) or OpenGL's GL Shading Language (GLSL). The shaders could be modified to produce special graphics effects, such as more realistic textures, lighting, and so on. Using clever programming tricks, these shading languages could also be used to perform non-graphics-related tasks. However, in order to do so, the problem at hand had to be reformulated as a rendering problem.

Perhaps seeing an opportunity to capture a new market, the GPU manufacturers introduced the capability for their GPUs to run non-graphics-related code. Software developers could now run custom code on the GPU. NVIDIA released their Compute Unified Device Architecture (CUDA) and AMD uses the OpenCL programming interface. Both APIs use a single instruction/multiple data computing architecture. The program transfers a large amount of data to the GPU, and then launches dozens or hundreds of threads to operate on that data. The one catch is that all of these threads must run the same exact code at the same exact time. The only difference from thread to thread is the data

being operated on. This is in stark contrast to the way threads operate on the CPU, where every thread can run its own program independent of the others. In the way, the CPU is more flexible, but with only up to eight cores on most CPUs, the amount of parallelization is limited.

In addition to developing and selling graphics-oriented GPUs, such as the GeForce series, NVIDIA has also started selling scientific-computing-targeted cards. This includes the Tesla-branded lines of cards. These cards have no graphics capabilities (the graphics pipeline has been removed entirely), but provide immense computing power. For example, one of the most recent Tesla cards, the K80, boasts over 8 TFLOPS of computer power.

3.6.2 Implementation Details

There are two versions of the code for computing the integrals needed by the BEM. The CPU version, written in C++ and parallelized using OpenMP, ran on a Quad Intel Xeon Dual E5-2690 (32 cores). The GPU version, written in CUDA, ran on an NVIDIA Tesla K20c. Because C++ and CUDA are largely the same programming language, the two versions are almost the same. However, because of the unique nature of GPU programming, several changes had to be made so that the CUDA code could take full advantage of the GPU hardware.

In the collocation method, each integral corresponds to a triangle/collocation point pair, while in the Galerkin method, each integral corresponds to a triangle/triangle pair. In the CPU version, there were 32 threads, each running independently from the others on one of the 32 available cores. The N_{pair} pairs were divided up into 32 equally sized chunks.

Vertex List

x_1	x_2	$\ldots$	$x_{N_{\text{vert}}}$
y_1	y_2	$\ldots$	$y_{N_{\text{vert}}}$
z_1	z_2	$\ldots$	$z_{N_{\text{vert}}}$

Connectivity List

$v_{1,1}$	$v_{2,1}$	$\ldots$	$v_{N_{\text{tri}},1}$
$v_{1,2}$	$v_{2,2}$	$\ldots$	$v_{N_{\text{tri}},2}$
$v_{1,3}$	$v_{2,3}$	$\ldots$	$v_{N_{\text{tri}},3}$

Triangle Pairs

i_1	i_2	$\ldots$	$i_{N_{\text{pair}}}$
j_1	j_2	$\ldots$	$j_{N_{\text{pair}}}$

Figure 3.13: On the CPU, the pairs of triangles are stored as pairs of indices, which reference values stored in the list of vertices and the connectivity list.

Each thread received one chunk, and computed all the integrals corresponding to the pairs in that chunk. In the GPU version, there was one thread per pair, meaning there were N_{pair} threads. Each thread was responsible for computing one integral. The block size was 256, so there were $N_{\text{pair}}/256$ blocks.

In the CPU version, the data structures are compact (see Fig. 3.13). The mesh is stored as a list of vertices and a connectivity list. In the collocation method, there is also a list of collocation points. The CPU code accepts a list of index pairs. In the collocation method, one index in the pair corresponds to a triangle, and the other index in the pair corresponds to a collocation point. In the Galerkin method, the two indices in the pair correspond to two triangles. Because global memory access on the CPU is fast due to the many layers of cache, this combination of data structures works well. However, global memory access on the GPU is slow, typically costing a thread hundreds of cycles in latency.

Triangle Pairs

$x_{1,1,1}$	$x_{2,1,1}$	$\cdots$	$x_{N_{\text{pair}},1,1}$
$y_{1,1,1}$	$y_{2,1,1}$	$\cdots$	$y_{N_{\text{pair}},1,1}$
$z_{1,1,1}$	$z_{2,1,1}$	$\cdots$	$z_{N_{\text{pair}},1,1}$
$x_{1,1,2}$	$x_{2,1,2}$	$\cdots$	$x_{N_{\text{pair}},1,2}$
$y_{1,1,2}$	$y_{2,1,2}$	$\cdots$	$y_{N_{\text{pair}},1,2}$
$z_{1,1,2}$	$z_{2,1,2}$	$\cdots$	$z_{N_{\text{pair}},1,2}$
$x_{1,1,3}$	$x_{2,1,3}$	$\cdots$	$x_{N_{\text{pair}},1,3}$
$y_{1,1,3}$	$y_{2,1,3}$	$\cdots$	$y_{N_{\text{pair}},1,3}$
$z_{1,1,3}$	$z_{2,1,3}$	$\cdots$	$z_{N_{\text{pair}},1,3}$
$x_{1,2,1}$	$x_{2,2,1}$	$\cdots$	$x_{N_{\text{pair}},2,1}$
$y_{1,2,1}$	$y_{2,2,1}$	$\cdots$	$y_{N_{\text{pair}},2,1}$
$z_{1,2,1}$	$z_{2,2,1}$	$\cdots$	$z_{N_{\text{pair}},2,1}$
$x_{1,2,2}$	$x_{2,2,2}$	$\cdots$	$x_{N_{\text{pair}},2,2}$
$y_{1,2,2}$	$y_{2,2,2}$	$\cdots$	$y_{N_{\text{pair}},2,2}$
$z_{1,2,2}$	$z_{2,2,2}$	$\cdots$	$z_{N_{\text{pair}},2,2}$
$x_{1,2,3}$	$x_{2,2,3}$	$\cdots$	$x_{N_{\text{pair}},2,3}$
$y_{1,2,3}$	$y_{2,2,3}$	$\cdots$	$y_{N_{\text{pair}},2,3}$
$z_{1,2,3}$	$z_{2,2,3}$	$\cdots$	$z_{N_{\text{pair}},2,3}$

Figure 3.14: On the GPU, the coordinates of the two triangles in each pair are stored explicitly.

To overcome this problem, instead of passing in a list of index pairs, the data structures are unraveled to avoid the use of indices, and a matrix of coordinates is sent to the GPU (see Fig. 3.14). Each column of the matrix corresponds to a pair, and each row is a coordinate of the vertices of the triangles and/or collocation point in that pair.

In the Galerkin method, the exact method for computing an integral over a pair of triangles depends on their relative geometry (see Sec. 3.5). There are four cases: (1) the two triangles do not touch; (2) they share a vertex; (3) they share an edge; or (4) they are

```
__global__ void do_something(int *outgoing, int *incoming)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx % 2 == 0)
        do_this(outgoing, incoming, idx);
    else
        do_that(outgoing, incoming, idx);
}
```

Figure 3.15: A snippet of CUDA code that will have branch divergence. All of the even threads will run one function, and all of the odd threads will run a different function. The odd threads will have to pause while the even threads run their code, and vice versa.

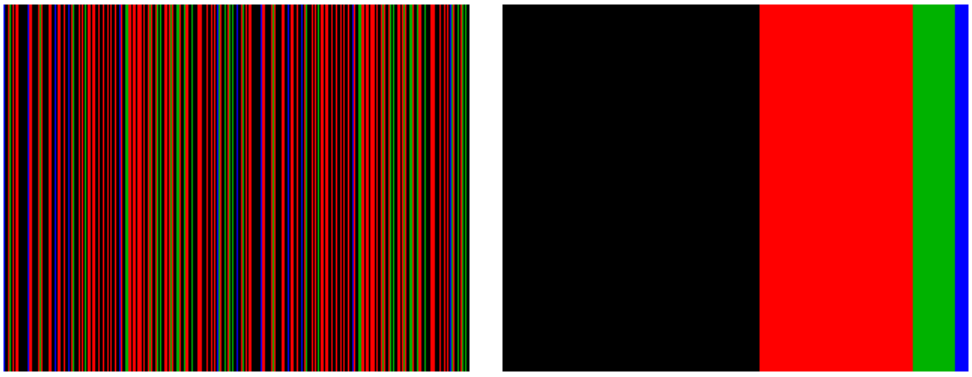

Figure 3.16: The double surface integrals in the Galerkin method are sorted by case (i.e., zero-, one-, two-, and three-touch) before sending them to the GPU to avoid branch divergence. On the left is the list of integrals before sorting, and on the right is the list after sorting. Black, red, green, and blue correspond to zero-, one-, two-, and three-touch integrals, respectively.

the same. These are called the zero-, one-, two-, and three-touch cases, respectively. In this naming scheme, the number represents how many vertices the two triangles share. The CPU version is not affected by the order of the index pairs with respect to these cases. That is, the four cases can be distributed among the index pairs in any way. This is because the 32 threads in the CPU version are working independently on separate cores. Which instructions are being executed in one thread have no effect on those in another. However, in the GPU version, this is not case. In fact, all the threads in a particular block must execute the same instructions at the same time. When a branch does occur because two or

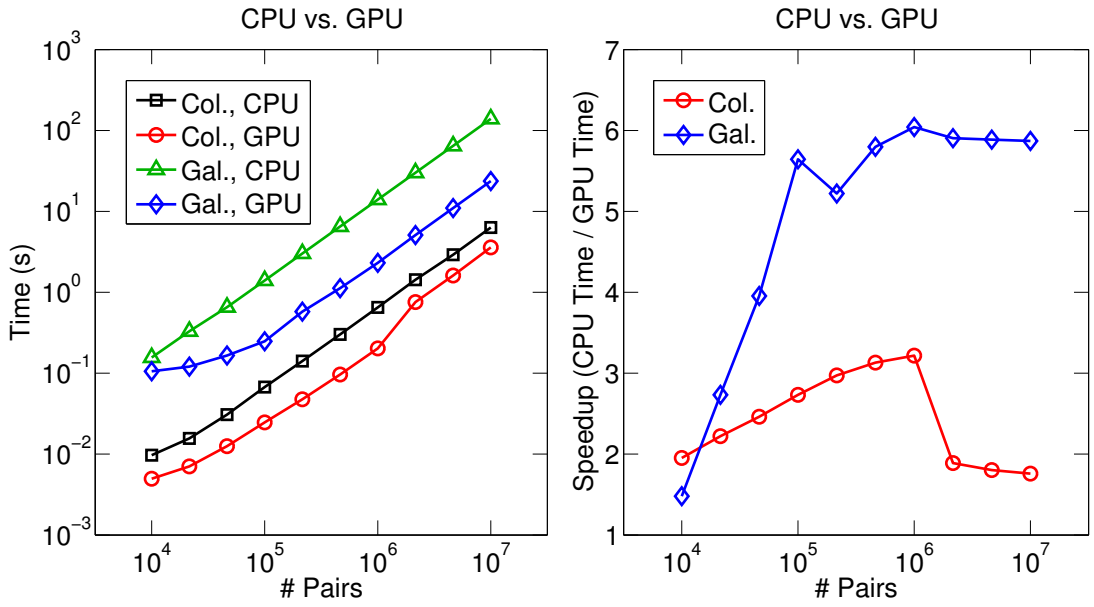

Figure 3.17: The GPU computes the integrals needed by the collocation method around $2\times$ faster than the CPU for large numbers of triangle/collocation point pairs. Likewise, the GPU computes the integrals needed by the Galerkin method around $6\times$ faster than the CPU for large numbers of triangle/triangle pairs. In these experiments, the CPU was a Quad Intel Xeon Dual E5-2690 (32 cores), and the GPU was an NVIDIA Tesla K20c.

more threads in a block correspond to different cases, each thread must pause for the other to complete their work before doing their own, and vice versa. This thread divergence can lead to slowdowns on the GPU, eliminating any advantage the GPU offers (see Fig. 3.15). To overcome this problem, before passing any data to the GPU, the list of index pairs is sorted by the case. That is, the zero-touch cases are at the beginning of the list, followed by the one-, two-, and three-touch cases, in that order (see Fig. 3.16). In this manner, only three blocks (out of hundreds of thousands) suffer from thread divergence.

Times were measured for each version, and then plotted against N_{pair} (see Fig. 3.17). The GPU version was faster than the CPU version for all values of N_{pair}. The GPU computed the integrals needed by the collocation method around $2\times$ faster than the CPU for large N_{pair}. Likewise, the GPU computed those needed by the Galerkin method around

$6\times$ faster. The speedup for the Galerkin integrals is much higher because computing these integrals is much more computational intensive, and the I/O overhead was not as much of an issue. On the other hand, the collocation integrals are fast to compute, requiring relatively few operations, so the I/O overhead had a much bigger effect. However, in both cases, the GPU versions provided large speedups, especially considering that the CPU had 32 cores.

3.7 Conclusion

We have presented a method for computing the double surface integrals encountered in the Galerkin BEM. When the boundary is discretized using triangular elements, these integrals are performed over pairs of these triangles. They can be extremely tough to compute, especially when the two triangles share a vertex, an edge, or are the same. This is because the kernels being integrated are often singular along the corners and edges of these triangles. We have solved this problem by using several scaling properties of the integrals and the kernels being integrated. The integral is broken up into several smaller ones, some of which are written in terms of the original. This is done in such a way that only completely regular integrals have to be computed explicitly.

We have also presented an analytical method for computing the integrals when the two triangles do not touch. The method uses spherical harmonics and multipole and local expansions and translations. The only source of error in this method is how soon to truncate these expansions. However, the truncation number is adaptively selected to achieve a desired error bound. We have validated both of these methods, and shown that they are

accurate.

Finally, we have parallelized our code using C++ and OpenMP, and also ported the C++ code to run on the GPU using CUDA. The GPU version of the code, which ran on a NVIDIA Tesla K20c, ran $6\times$ faster than the CPU version of the code, which ran on a Quad Intel Xeon Dual E5-2690 (32 cores).

Chapter 4: Computation of Boundary Integrals for the Helmholtz Equation

4.1 Introduction

The Helmholtz equation is one of the most important PDEs in science, and governs problems in a large number of disciplines, including acoustics [9], electromagnetism [10], heat and other forms of diffusion, and even gravitational waves, which were recently detected experimentally [11]. The BEM is a powerful method for solving the Helmholtz equation in three dimensions [12–14].

As discussed in Chap. 2, in the case of the Helmholtz equation, when solving external problems using the Green's identity formulation, the solution is affected by so-called spurious modes. These spurious modes are the solutions to the internal problem for the same geometry whose potential or normal derivative is zero on the boundary. Because they are zero, they do not interfere with the boundary conditions, but they nevertheless infect the solution away from the boundary. There are many methods for suppressing these spurious modes. One popular method is to add additional constraints to the system by placing points inside the boundary, and forcing the potential at these points to be zero [29]. Another method is the Burton-Miller formulation [15]. A third approach for suppressing

the spurious modes is the indirect formulation, which is described in Sec. 2.2 and is what we use.

Implementing the indirect BEM requires a method for computing single- and double-layer potentials and gradients due to linear source density distributions over triangular elements. In this chapter, we describe a method for doing so based on the singularity subtraction method [49]. This method works by splitting the Green's function for the Helmholtz equation into two pieces, a singular part and a regular part. The singular part happens to be the Green's function for the Laplace kernel, so methods for computing that, including those from Sec. 3.2 and Appx. A, can be used. The regular part can then be integrated using standard numerical means, e.g., Gaussian quadrature.

4.2 Collocation Integrals

Enforcing the boundary conditions in the indirect BEM using the collocation method requires computing single- and double-layer potentials and gradients due to linear source density distributions over triangular elements:

$$L(\mathbf{x}) = \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}')\, G(\mathbf{x} - \mathbf{x}')\, dS(\mathbf{x}'), \tag{4.1}$$

$$M(\mathbf{x}) = \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}')\, (\mathbf{n}' \cdot \nabla_{\mathbf{x}'} G(\mathbf{x} - \mathbf{x}'))\, dS(\mathbf{x}'), \tag{4.2}$$

$$\nabla_{\mathbf{x}} L(\mathbf{x}) = \nabla_{\mathbf{x}} \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}')\, G(\mathbf{x} - \mathbf{x}')\, dS(\mathbf{x}'), \tag{4.3}$$

$$\nabla_{\mathbf{x}} M(\mathbf{x}) = \nabla_{\mathbf{x}} \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}')\, (\mathbf{n}' \cdot \nabla_{\mathbf{x}'} G(\mathbf{x} - \mathbf{x}'))\, dS(\mathbf{x}'). \tag{4.4}$$

Methods for computing these integrals for the Green's function for the Laplace equation were discussed in Sec. 3.2 and Appx. A. These methods can be extended to the Green's function for the Helmholtz equation using the singularity subtraction method [49].

The Green's function for the Helmholtz equation is

$$G(r) = \frac{\exp(ikr)}{4\pi r}, \tag{4.5}$$

$$\frac{dG}{dr}(r) = \frac{ik\exp(ikr)}{4\pi r} - \frac{\exp(ikr)}{4\pi r^2}, \tag{4.6}$$

$$\frac{d^2G}{dr^2}(r) = \frac{(ik)^2\exp(ikr)}{4\pi r} - \frac{ik\exp(ikr)}{2\pi r^2} + \frac{\exp(ikr)}{2\pi r^3}. \tag{4.7}$$

The Helmholtz kernel is singular, but this singularity comes from the Laplace equation. We can remove this singularity by subtracting the Laplace kernel:

$$\widetilde{G}(r) = \frac{\exp(ikr) - 1}{4\pi r}, \tag{4.8}$$

$$\frac{d\widetilde{G}}{dr}(r) = \frac{ik\exp(ikr)}{4\pi r} - \frac{\exp(ikr) - 1}{4\pi r^2}, \tag{4.9}$$

$$\frac{d^2\widetilde{G}}{dr^2}(r) = \frac{(ik)^2\exp(ikr)}{4\pi r} - \frac{ik\exp(ikr)}{2\pi r^2} + \frac{\exp(ikr) - 1}{2\pi r^3}. \tag{4.10}$$

This new kernel is completely regular. To demonstrate this, let's construct a Taylor series expansion around $r = 0$:

$$\exp(ikr) = \sum_{n=0}^{\infty} \frac{(ikr)^n}{n!}. \tag{4.11}$$

Rearranging,

$$\widetilde{G}(r) = \frac{\exp(ikr) - 1}{4\pi r} = \sum_{n=1}^{\infty} \frac{(ik)^n r^{n-1}}{4\pi n!}. \tag{4.12}$$

Let's take the derivative of $\widetilde{G}$:

$$\frac{d\widetilde{G}}{dr}(r) = \frac{d}{dr}\left(\sum_{n=1}^{\infty} \frac{(ik)^n r^{n-1}}{4\pi n!}\right) = \sum_{n=2}^{\infty} \frac{(ik)^n (n-1) r^{n-2}}{4\pi n!}. \tag{4.13}$$

Let's take the second derivative of $\widetilde{G}$:

$$\frac{d^2\widetilde{G}}{dr^2}(r) = \frac{d}{dr}\left(\sum_{n=2}^{\infty} \frac{(ik)^n (n-1) r^{n-2}}{4\pi n!}\right) = \sum_{n=3}^{\infty} \frac{(ik)^n (n-1)(n-2) r^{n-3}}{4\pi n!}. \tag{4.14}$$

Even though they are completely regular, Eqs. (4.8), (4.9), and (4.10) can still "blow up" or return NaNs when evaluated by a computer. For example, consider the expression for $\widetilde{G}$ in Eq. (4.8). When $r = 0$, the computer will evaluate the numerator and the denominator as zero, and zero divided by zero yields a NaN. However, as $r \to 0$, the expression approaches $(ik) / (4\pi)$, but the computer does not know that. Therefore, when $kr < \alpha_0$, Eqs. (4.12), (4.13), and (4.14) are used instead, where only up to the $(n = p - 1)$th term is kept. Good values for these two parameters are $\alpha_0 = 10^{-3}$ and $p = 6$.

The single- and double-layer potentials and their derivatives in Eqs. (4.1), (4.2), (4.3), and (4.4) are computed by splitting them into two pieces, one for the Laplace kernel and one for $\widetilde{G}$. For example, consider Eq. (4.1):

$$L(\mathbf{x}) = \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}') \, G(\mathbf{x} - \mathbf{x}') \, dS(\mathbf{x}'), \tag{4.15}$$

$$L(\mathbf{x}) = \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}') \, \frac{1}{4\pi r} dS(\mathbf{x}') + \int_{\mathbf{x}' \in \mathrm{T}} (\sigma_0' + \mathbf{p}' \cdot \mathbf{x}') \, \widetilde{G}(r) \, dS(\mathbf{x}'), \tag{4.16}$$

where $r = |\mathbf{r}| = |\mathbf{x} - \mathbf{x}'|$. The left integral is computed using the methods discussed in

Sec. 3.2 and Appx. A, and the right integral is computed via Gaussian quadrature. We use the techniques given in [66] for performing the Gaussian quadrature over the triangles.

The integrals in Eqs. (4.1), (4.2), and (4.3) are completely regular. However, a special case appears in the integral in Eq. (4.4) when computing

$$\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}\widetilde{G}\right)=\frac{d^2\widetilde{G}}{dr^2}\left(\nabla_{\mathbf{x}}r\right)\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)+\frac{d\widetilde{G}}{dr}\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right), \tag{4.17}$$

where

$$\nabla_{\mathbf{x}}r=\frac{\mathbf{r}}{r},\quad \mathbf{n}'\cdot\nabla_{\mathbf{x}'}r=-\frac{\mathbf{n}'\cdot\mathbf{r}}{r},\quad \nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=-\frac{\mathbf{n}'}{r}+\frac{\left(\mathbf{n}'\cdot\mathbf{r}\right)\mathbf{r}}{r^3}. \tag{4.18}$$

This is because the gradient of the normal derivative of the displacement vector,

$$\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=-\frac{\mathbf{n}'}{r}+\frac{\left(\mathbf{n}'\cdot\mathbf{r}\right)\mathbf{r}}{r^3}, \tag{4.19}$$

is singular at $r=0$. This occurs when the evaluation point is on the triangle, which means that $\mathbf{n}'\cdot\mathbf{r}=0$, and

$$\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=-\frac{\mathbf{n}'}{r}. \tag{4.20}$$

This expression is still singular, but is a little simpler to manage. What we're going to do is move the singularity from this quantity to $d\widetilde{G}/dr$. We'll then break the troublesome term

off from the rest, and treat that separately. We have

$$\frac{d\widetilde{G}}{dr}\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=\left(\sum_{n=2}^{\infty}\frac{(ik)^n\,(n-1)\,r^{n-2}}{4\pi n!}\right)\left(-\frac{\mathbf{n}'}{r}\right), \tag{4.21}$$

$$\frac{d\widetilde{G}}{dr}\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=-\mathbf{n}'\left(\sum_{n=2}^{\infty}\frac{(ik)^n\,(n-1)\,r^{n-3}}{4\pi n!}\right). \tag{4.22}$$

Separate the $n=2$ term from the others:

$$\frac{d\widetilde{G}}{dr}\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=-\mathbf{n}'\left(\frac{(ik)^2}{8\pi r}+\sum_{n=3}^{\infty}\frac{(ik)^n\,(n-1)\,r^{n-3}}{4\pi n!}\right), \tag{4.23}$$

$$\frac{d\widetilde{G}}{dr}\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=-\frac{(ik)^2\,\mathbf{n}'}{2}\frac{1}{4\pi r}-\mathbf{n}'\left(\sum_{n=3}^{\infty}\frac{(ik)^n\,(n-1)\,r^{n-3}}{4\pi n!}\right), \tag{4.24}$$

$$\frac{d\widetilde{G}}{dr}\nabla_{\mathbf{x}}\left(\mathbf{n}'\cdot\nabla_{\mathbf{x}'}r\right)=-\frac{(ik)^2\,\mathbf{n}'}{2}\frac{1}{4\pi r}-\mathbf{n}'\widetilde{H}, \tag{4.25}$$

where

$$\widetilde{H}=\sum_{n=3}^{\infty}\frac{(ik)^n\,(n-1)\,r^{n-3}}{4\pi n!}. \tag{4.26}$$

The singular term can be removed from the expression, and treated separately using the Laplace code. The remainder is now completely regular again, and can be integrated numerically like before.

4.3 Conclusion

Implementing the indirect BEM requires a method for computing single- and double-layer potentials and gradients due to linear source density distributions over triangular elements. In this chapter, we described a method for doing so based on the singularity

subtraction method. This method works by splitting the Green's function for the Helmholtz equation into two pieces, a singular part and a regular part. The singular part can be computed using methods previously developed for the Laplace equation, and the regular part can be computed using Gaussian quadrature. A special case appeared when computing the gradient of the double-layer potential. When the evaluation point was in the plane containing the triangular element, the expression became singular. To solve this problem, the expression was written as a Taylor series, and the troublesome term was broken off and treated separately.

Chapter 5: Correction Factor Matrix Method[1]

5.1 Introduction

Integral equation methods for solving partial differential equations, such as the boundary element method (BEM) or the method of moments [12–14], have a number of advantages, including that they reduce the space dimensionality of the discretization by one, allow for the treatment of complex boundary shapes, handle the boundary conditions at infinity accurately, and treat thin objects and multi-domain problems well. Because of these advantages, they are used extensively in electrostatics, antenna theory, acoustics, and for simulation of free surface flows, among others. The equations are discretized via boundary elements (e.g., linear or curvilinear elements in two dimensions, and triangles, quadrilaterals, or their curved equivalents in three dimensions), and when combined with boundary conditions, result in linear systems. These systems, while smaller than those from volumetric discretization approaches (such as the finite difference and finite element methods), are dense. The entries in these matrices are based on the quadrature of the product of local basis functions and Green's functions or their derivatives over the boundary elements. In the collocation method, these entries are obtained by a quadrature of the Green's function integrand over one boundary element, while in the Galerkin method, they

[1] This chapter is based on our original work in [69].

are obtained by a double quadrature over two boundary elements.

The argument of the Green's functions (or their derivatives) in these integrands involves points from the triangles over which the quadrature is performed, or the triangle and the collocation point. A crucial ingredient of integral equation methods is the accurate computation of these integrals, especially for the nearly singular, singular, and hypersingular cases. These cases arise in the collocation method when the triangle and collocation point are close to each other or lie on top of each other, and in the Galerkin method when the two triangles are proximal, share a vertex, an edge, or are the same. In these cases, careful treatment of the integrals is needed for accurate quadrature. We studied these integrals for the Laplace equation in Chap. 3, where we developed novel techniques for computing them based on subdivision and scaling arguments. We studied them for the Helmholtz equation in Chap. 4.

The linear systems arising in the BEM are conventionally solved via direct matrix decompositions. However, the system matrices are dense, requiring $\mathrm{O}\left(N^2\right)$ storage, where N is the number of discretization unknowns. This can effectively restrict the size of a problem that can be solved on a given machine. Moreover, solving the system by direct means (e.g., LU decomposition) requires $\mathrm{O}\left(N^3\right)$ operations. Using an iterative solver based on Krylov subspace methods, such as GMRES, alleviates this issue somewhat, providing an $\mathrm{O}\left(N_{\text{iter}}N^2\right)$ method, where N_{iter} is the number of iterations and $\mathrm{O}\left(N^2\right)$ is the cost of the matrix-vector product that is computed every iteration. However, for large N, this quadratic scaling in both time and memory can still be prohibitive.

The fast multipole method (FMM) allows for the acceleration of many matrix-vector products [21, 22], and was originally developed to address the N-body problem arising in

stellar and molecular dynamics. The FMM accelerates sums of the following form:

$$v(\mathbf{y}_i) = \sum_{j=1}^{N} u_j \Phi(\mathbf{y}_i - \mathbf{x}_j), \quad i = 1, 2, \ldots, M. \tag{5.1}$$

Computing this sum by direct means requires O (MN) operations. The FMM computes the sum to any specified error, ε, in O $(M + N)$ operations and storage, where the asymptotic constant depends on the desired accuracy. This linear scaling allows for very large problem sizes. The FMM works by separating the matrix-vector product into two pieces, one for far-field interactions and one for near-field interactions. The FMM accelerates the far-field piece using spatial data structures (e.g., an octree), spherical harmonics, and multipole and local expansions and translations. There has been significant effort to speed up the FMM on parallel architectures, including shared memory, distributed, GPU, and heterogeneous architectures, and these algorithms have also been used to show benchmark high-performance computing calculations. For example, the FMM can be accelerated by parallelizing across many cores using OpenMP or the GPU [68]. The entire FMM can also be parallelized across an entire cluster [70, 71]. Some open source FMM codes are available [27, 28]. For the Laplace equation, we used our own highly optimized in-house GPU-accelerated FMM. For the Helmholtz equation, we used a third-party FMM [27].

The FMM is usually designed around point and dipole sources, not the integral expressions that appear in the BEM. Extending the FMM to handle these integral expressions is possible, but requires modifying the internal data structures and logic. Many authors have done so. For example, the FMM was used to solve problems in elastostatics [25, 61], formulations involving volume integrals [24], problems in acoustics and the computation

of head-related transfer functions [23, 72, 73], electromagnetic scattering problems [26], and even in computer graphics to accelerate collision detection [8]. As was seen in these references, the issues associated with adding the integral expressions directly to the FMM are manageable, and the results are often very good, but these changes complicate the inner workings. Moreover, FMM codes for point and dipole sources are widely available and highly optimized. Newly developed FMM codes for integral-based methods may not be as highly optimized. Thus, we seek a procedure for applying the FMM unchanged to the integral expressions in the BEM.

This approach, which we call the correction factor matrix method, is described in this chapter. The method works by approximating the integrals using a quadrature, and then treating the quadrature points as point and dipole sources, which can be plugged directly into the FMM. Any inaccuracies from the quadrature are corrected during a correction factor step, which runs in $\mathrm{O}(N)$. The FMM is effectively treated as a black box, and the correction factor matrix method is simply a series pre- and post-processing steps, which also run in $\mathrm{O}(N)$ and achieve the prescribed error tolerance. The method is derived, and example problems are presented showing accuracy and performance. The method is shown to scale linearly and be able to solve problems with a million elements in only minutes on a single workstation.

5.2 Basic Approach

The linear systems given in Chap. 2 are solved using an iterative solver, such as GMRES, or one of their preconditioned variants. The matrix-vector product that is

computed during each iteration is accelerated using the FMM. Most FMM codes are designed around point and dipole sources, not the integral expressions present in the BEM. Thus, we have developed the correction factor matrix method, which allows for the FMM to be applied unchanged to these integral expressions. The method works by approximating the integrals using a quadrature, and then treating the quadrature points as point and dipole sources, which can be plugged directly into the FMM. Any inaccuracies from the quadrature, especially those from the approximation of singular and nearly singular integrals, are corrected by computing and adding correction factors.

The matrix-vector product that is computed during each iteration is given by

$$\mathbf{Su} = \begin{bmatrix} \mathrm{U}+\mathrm{D} & \mathrm{V} \\ \mathrm{A} & \mathrm{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi}_{123} \\ \mathbf{q}_{123} \end{bmatrix} = \begin{bmatrix} \mathrm{U}\boldsymbol{\phi}_{123} + \mathrm{D}\boldsymbol{\phi}_{123} + \mathrm{V}\mathbf{q}_{123} \\ \mathrm{A}\boldsymbol{\phi}_{123} + \mathrm{B}\mathbf{q}_{123} \end{bmatrix}. \tag{5.2}$$

Consider

$$\mathbf{y} = \mathrm{A}\mathbf{x}, \tag{5.3}$$

where the matrix, A, is one of those from above, U or V. Suppose we have another matrix, $\widetilde{\mathrm{A}}$, that approximates the original matrix, A:

$$\mathbf{y} = \widetilde{\mathrm{A}}\mathbf{x} + \left(\mathrm{A} - \widetilde{\mathrm{A}}\right)\mathbf{x} = \widetilde{\mathrm{A}}\mathbf{x} + \mathrm{C}\mathbf{x}. \tag{5.4}$$

The exact structure of the matrix, $\widetilde{\mathrm{A}}$, and how to compute the matrix-vector product, $\widetilde{\mathrm{A}}\mathbf{x}$, for the collocation and Galerkin methods are given in Secs. 5.3 and 5.4, respectively. Those sections give methods for computing $\widetilde{\mathrm{A}}\mathbf{x}$ in $\mathrm{O}(N)$ using the FMM. The matrix-vector

product, $\mathbf{Cx}$, corrects the errors from using $\widetilde{\mathrm{A}}\mathbf{x}$. Therefore, the matrix, C, is called the correction factor matrix. Computing and storing the entire correction factor matrix is possible and would lead to zero error. However, this is expensive, requiring $\mathrm{O}\left(N^2\right)$ not only to construct C, but also to compute $\mathbf{Cx}$. This would undermine the desired $\mathrm{O}\left(N\right)$ behavior. In general, $\widetilde{\mathrm{A}}$ approximates A very well, so almost all the entries in C are nearly equal to zero. In fact, only $\mathrm{O}\left(N\right)$ entries in C need to be computed to achieve the $\mathrm{O}\left(\varepsilon\right)$ tolerance, and the rest of the entries can be set to zero. Letting $\widetilde{\mathrm{C}}$ be the partially computed correction factor matrix, we have

$$\mathbf{y} \approx \widetilde{\mathbf{y}} = \widetilde{\mathrm{A}}\mathbf{x} + \widetilde{\mathrm{C}}\mathbf{x}. \tag{5.5}$$

This expression becomes more accurate (i.e., $\widetilde{\mathbf{y}} \to \mathbf{y}$) as more entries in the correction factor matrix are computed and stored (i.e., $\widetilde{\mathrm{C}} \to \mathrm{C}$).

Which entries to compute depends largely on the geometry of the problem. In the collocation method, only the entries corresponding to triangle/matching point pairs that are close to each other are computed, while all the others are set to zero. A collocation point, $\mathbf{y}$, is close to a triangle when $|\mathbf{y} - \mathbf{x}| < Cr$, where $\mathbf{x}$ and r are the center and radius of the triangle, and C is the close ratio. In the Galerkin method, only the entries corresponding to triangle/triangle pairs that are close to each other are computed. Two triangles are close when $|\mathbf{x}_2 - \mathbf{x}_1| < C\left((r_1 + r_2)/2\right)$, where $\mathbf{x}_1$, $\mathbf{x}_2$, r_1, and r_2 are the centers and radii of the two triangles. Fig. 5.1 shows how the close ratio affects which pairs of triangles are close to each other in the Galerkin method. A larger close ratio leads to more entries in the correction factor matrix being computed, while a smaller close ratio leads to fewer. A

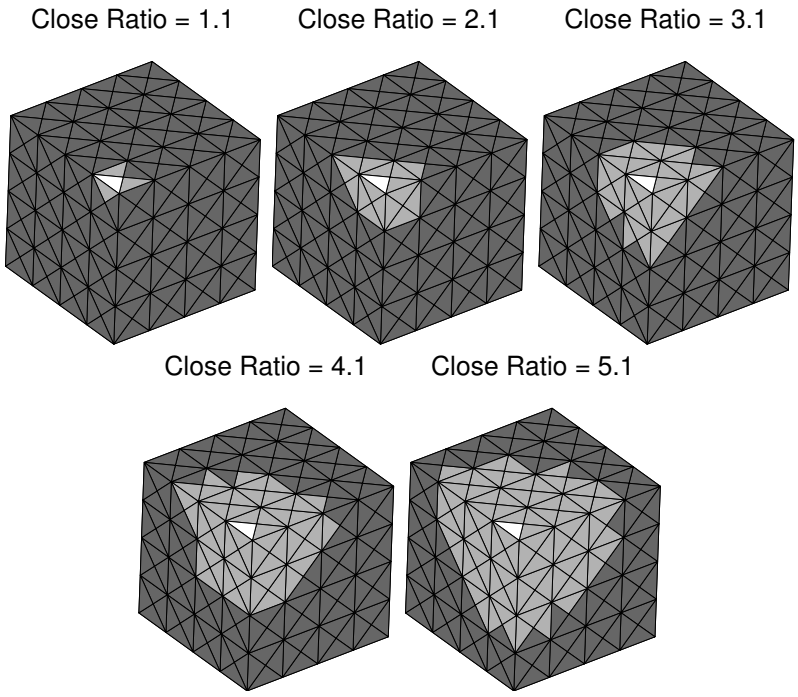

Figure 5.1: In the Galerkin method, two triangles are close (and correction factors are computed) when $|\mathbf{x}_2 - \mathbf{x}_1| < C\left((r_1 + r_2)/2\right)$, where $\mathbf{x}_1$, $\mathbf{x}_2$, r_1, and r_2 are the centers and radii of the two triangles, and C is the close ratio. The light gray triangles are close to the white triangles. Five different close ratios are shown.

denser correction factor matrix leads to more accurate results, so making the close ratio larger increases the accuracy of the method, but at the expense of speed. In our numerical experiments, we establish a way to get a consistent approximation whose error is lower than the errors in the other parts of the FMM/GPU-accelerated BEM.

5.3 Collocation Method/Evaluation of Solution

Consider M evaluation points, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$. We want to compute the single- or double-layer potential at these points due to a potential or normal derivative on the N triangles. These M evaluation points could be the N matching points in the collocation method, and we are computing one of the matrix-vector products, $\mathrm{U}\boldsymbol{\phi}$ or $\mathrm{V}\mathbf{q}$, required by

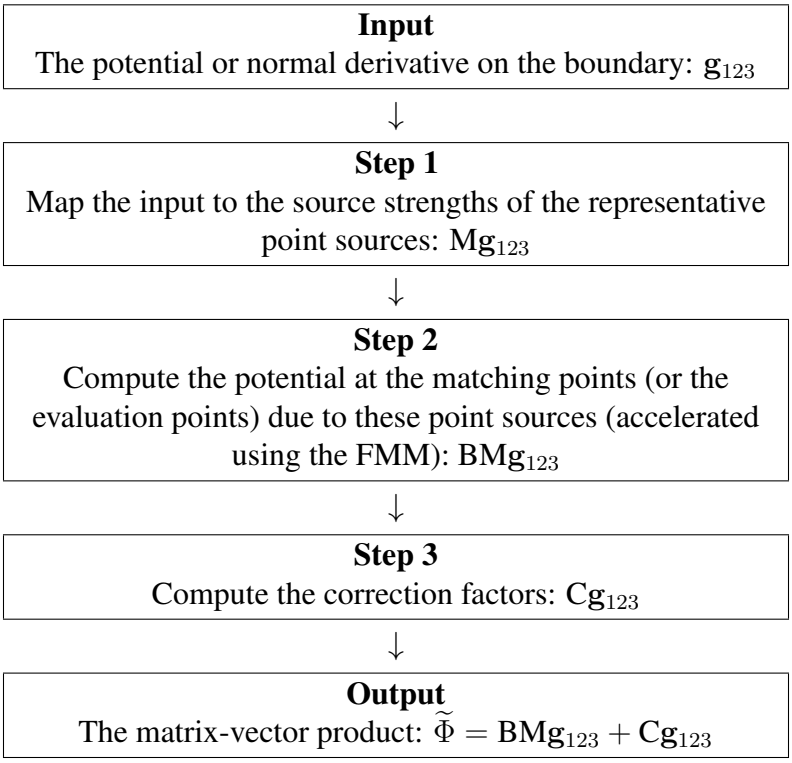

Figure 5.2: The procedure of the correction factor matrix method for accelerating the matrix-vector product in the collocation method/evaluation of solution.

that method. Alternatively, we have already solved the problem, and we want to evaluate the solution at M different evaluation points. The analysis is the same for the two cases. A diagram showing the procedure of the correction factor matrix method for these cases is shown in Fig. 5.2.

Consider the following matrix-vector product:

$$\Phi = \mathbf{Ag}_{123}, \tag{5.6}$$

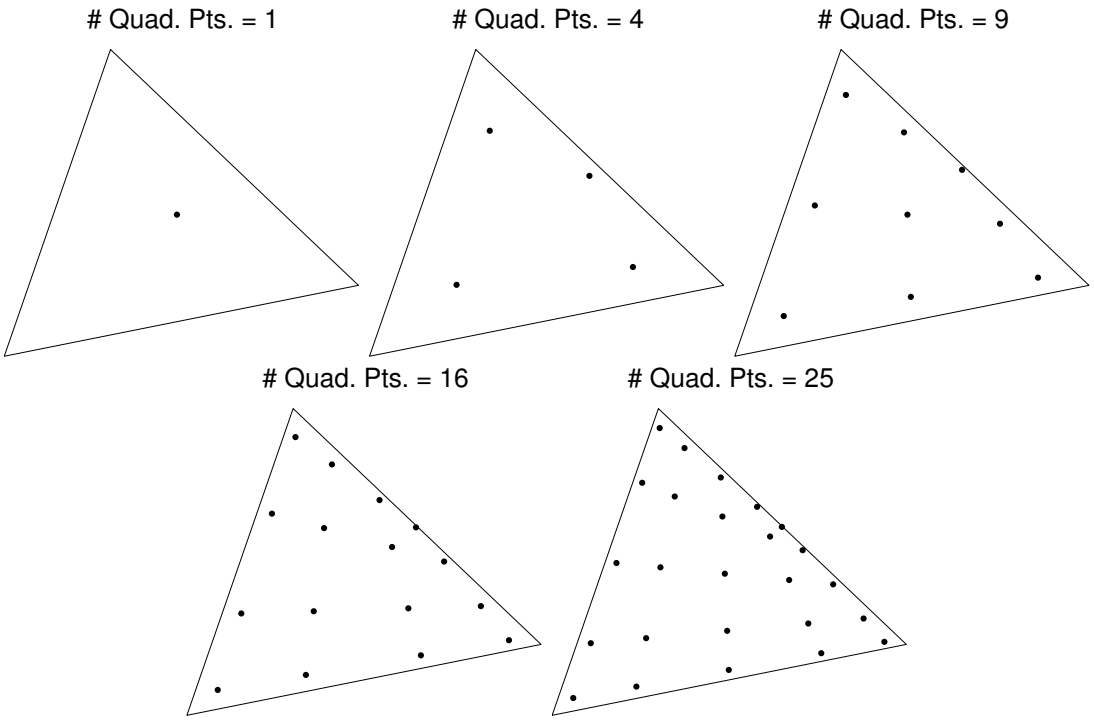

Figure 5.3: The quadrature points for an example triangle. The quadrature points are based on a two-dimensional Gauss-Legendre quadrature.

where

$$\mathrm{A} = \begin{bmatrix} \mathrm{A}_{1,1} & \dots & \mathrm{A}_{1,N} \\ \vdots & \ddots & \vdots \\ \mathrm{A}_{M,1} & \dots & \mathrm{A}_{M,N} \end{bmatrix}, \quad \mathrm{A}_{i,j} = \begin{bmatrix} A_{i,j,1} & A_{i,j,2} & A_{i,j,3} \end{bmatrix}, \tag{5.7}$$

and $\mathbf{g}_{123}$ is one of the vectors of coefficients, $\boldsymbol{\phi}_{123}$ or $\mathbf{q}_{123}$. The entries in A are given by

$$A_{i,j,n} = \int_{\mathbf{x}' \in \mathrm{T}_j} N_{j,n}\left(\mathbf{x}'\right) F\left(\mathbf{x}_i - \mathbf{x}'\right) dS\left(\mathbf{x}'\right), \tag{5.8}$$

where $F\left(\mathbf{r}\right)$ is the kernel being integrated, such as the Green's function or a derivative of the Green's function.

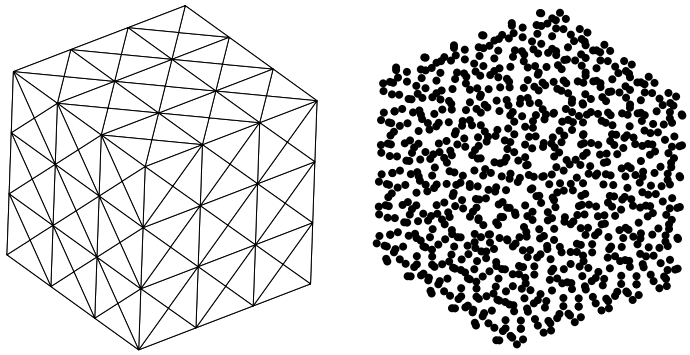

Figure 5.4: On the left is a mesh of a cube, and on the right is the set of representative point sources for the mesh. In this example, there are four point sources per triangle. In the correction factor matrix method, the potential or normal derivative on the triangles is mapped to the source strengths of the point sources. These point sources approximate the single- or double-layer potentials due to the potential or normal derivative on the mesh. Correction factors are computed and added to correct any inaccuracies from this approximation.

The entries in A are approximated using a quadrature:

$$A_{i,j,n} \approx \widetilde{A}_{i,j,n} = \sum_{l=1}^{Q_j} w_{j,l} N_{j,n} \left(\mathbf{p}_{j,l}\right) F\left(\mathbf{x}_i - \mathbf{p}_{j,l}\right), \tag{5.9}$$

where $\mathbf{p}_{j,l}$ and $w_{j,l}$ are the Q_j quadrature points and weights for T_j. The quadrature points we used are based on a two-dimensional Gauss-Legendre quadrature (the triangle is mapped to a quadrilateral [66]). Fig. 5.3 shows different numbers of quadrature points for an example triangle. More compactly,

$$\widetilde{A}_{i,j,n} = \mathrm{B}_{i,j}\mathrm{M}_{j,n}, \tag{5.10}$$

where

$$\mathrm{B}_{i,j} = \begin{bmatrix} F\left(\mathbf{x}_i - \mathbf{p}_{j,1}\right) & \dots & F\left(\mathbf{x}_i - \mathbf{p}_{j,Q_j}\right) \end{bmatrix}, \tag{5.11}$$

$$\mathbf{M}_{j,n} = \begin{bmatrix} w_{j,1} N_{j,n}\left(\mathbf{p}_{j,1}\right) \\ w_{j,2} N_{j,n}\left(\mathbf{p}_{j,2}\right) \\ \vdots \\ w_{j,Q_j} N_{j,n}\left(\mathbf{p}_{j,Q_j}\right) \end{bmatrix}. \tag{5.12}$$

Using this, we approximate the matrix-vector product, Ag_{123}, using a quadrature:

$$\Phi \approx \Psi = \widetilde{\mathrm{A}}\mathbf{g}_{123}, \tag{5.13}$$

where

$$\widetilde{\mathrm{A}} = \begin{bmatrix} \widetilde{\mathrm{A}}_{1,1} & \dots & \widetilde{\mathrm{A}}_{1,N} \\ \vdots & \ddots & \vdots \\ \widetilde{\mathrm{A}}_{M,1} & \dots & \widetilde{\mathrm{A}}_{M,N} \end{bmatrix}, \quad \widetilde{\mathrm{A}}_{i,j} = \begin{bmatrix} \widetilde{A}_{i,j,1} & \widetilde{A}_{i,j,2} & \widetilde{A}_{i,j,3} \end{bmatrix}. \tag{5.14}$$

Plugging in Eq. (5.10) and rearranging, we have

$$\Psi = \mathrm{BM}\mathbf{g}_{123}, \tag{5.15}$$

where

$$\mathrm{B} = \begin{bmatrix} \mathrm{B}_{1,1} & \dots & \mathrm{B}_{1,N} \\ \vdots & \ddots & \vdots \\ \mathrm{B}_{M,1} & \dots & \mathrm{B}_{M,N} \end{bmatrix}, \tag{5.16}$$

$$\mathrm{M} = \begin{bmatrix} \mathrm{M}_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathrm{M}_N \end{bmatrix}, \quad \mathrm{M}_j = \begin{bmatrix} \mathrm{M}_{j,1} & \mathrm{M}_{j,2} & \mathrm{M}_{j,3} \end{bmatrix}. \tag{5.17}$$

The mapping matrix, M, maps the potential or normal derivative on the triangles to the source strengths of a set of representative point sources (see Fig. 5.4). This set is the union of the quadrature points for all the triangles in the mesh. The mapping matrix is a block diagonal matrix, and so is highly sparse: there are only three nonzero entries per row. The matrix, B, computes the potential at the evaluation points due to these point sources. This step is accelerated using the FMM, and runs in O (N).

5.4 Galerkin Method

In the collocation method, the matrix-vector products, V$\mathbf{q}$ and U$\boldsymbol{\phi}$, compute the single- and double-layer potentials at the N matching points. In the Galerkin method, these matrix-vector products compute the integral over the boundary of these potentials when weighted by the basis functions. A diagram showing the procedure of the correction factor matrix method for the Galerkin method is shown in Fig. 5.5.

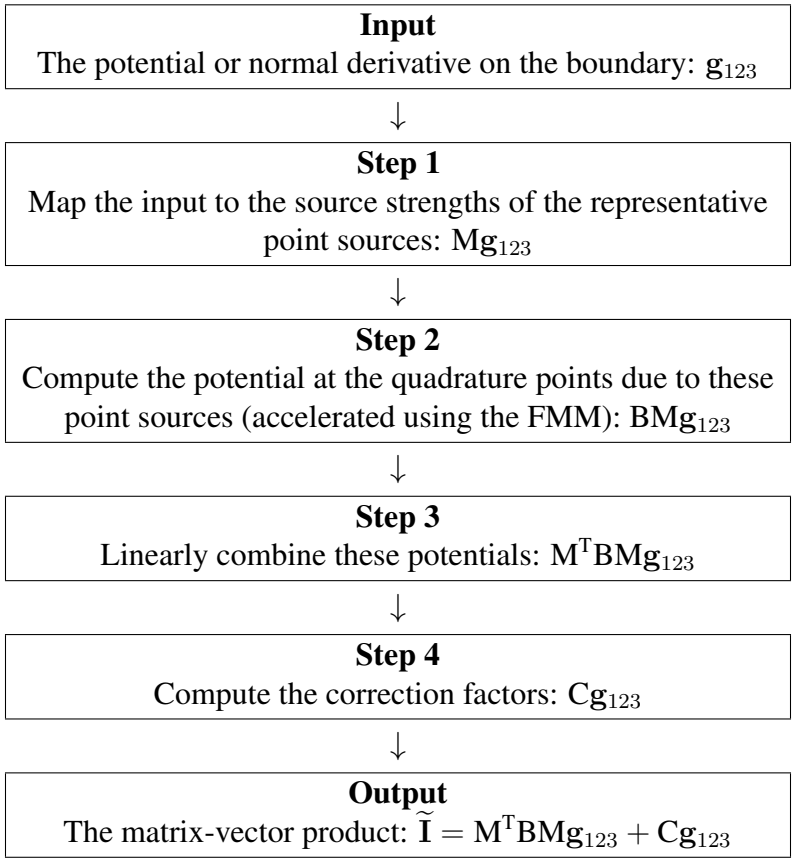

Figure 5.5: The procedure of the correction factor matrix method for accelerating the matrix-vector product in the Galerkin method.

Consider the following matrix-vector product:

$$\mathbf{I} = \mathrm{A}\mathbf{g}_{123}, \tag{5.18}$$

where

$$\mathrm{A} = \begin{bmatrix} \mathrm{A}_{1,1} & \dots & \mathrm{A}_{1,N} \\ \vdots & \ddots & \vdots \\ \mathrm{A}_{M,1} & \dots & \mathrm{A}_{M,N} \end{bmatrix}, \quad \mathrm{A}_{i,j} = \begin{bmatrix} A_{i,j,1,1} & A_{i,j,1,2} & A_{i,j,1,3} \\ A_{i,j,2,1} & A_{i,j,2,2} & A_{i,j,2,3} \\ A_{i,j,3,1} & A_{i,j,3,2} & A_{i,j,3,3} \end{bmatrix}. \tag{5.19}$$

The entries in A are given by

$$A_{i,j,m,n} = \int_{\mathbf{x}\in \mathrm{T}_i} N_{i,m}\left(\mathbf{x}\right) \int_{\mathbf{x}'\in \mathrm{T}_j} N_{j,n}\left(\mathbf{x}'\right) F\left(\mathbf{x}-\mathbf{x}'\right) dS\left(\mathbf{x}'\right) dS\left(\mathbf{x}\right), \tag{5.20}$$

where $F\left(\mathbf{r}\right)$ is the kernel being integrated, such as the Green's function or a derivative of the Green's function. The inside integral computes the potential due to the potential or normal derivative, $N_{j,n}\left(\mathbf{x}'\right)$, on T_j. The outside integral multiplies this potential by $N_{i,m}\left(\mathbf{x}\right)$, and integrates this product over T_i.

The entries in A are approximated using a quadrature:

$$A_{i,j,m,n} \approx \widetilde{A}_{i,j,m,n} = \sum_{k=1}^{Q_i} w_{i,k} N_{i,m}\left(\mathbf{p}_{i,k}\right) \sum_{l=1}^{Q_j} w_{j,l} N_{j,n}\left(\mathbf{p}_{j,l}\right) F\left(\mathbf{p}_{i,k}-\mathbf{p}_{j,l}\right), \tag{5.21}$$

where $\mathbf{p}_{j,l}$ and $w_{j,l}$ are the Q_j quadrature points and weights for T_j, and $\mathbf{p}_{i,k}$ and $w_{i,k}$ are the Q_i quadrature points and weights for T_i. More compactly,

$$\widetilde{A}_{i,j,m,n} = \mathbf{M}_{i,m}^{\mathrm{T}} \mathbf{B}_{i,j} \mathbf{M}_{j,n}, \tag{5.22}$$

where the matrices, $\mathbf{M}_{i,m}$ and $\mathbf{M}_{j,n}$, are given by Eq. (5.12), and

$$\mathbf{B}_{i,j} = \begin{bmatrix} F\left(\mathbf{p}_{i,1}-\mathbf{p}_{j,1}\right) & \dots & F\left(\mathbf{p}_{i,1}-\mathbf{p}_{j,Q_j}\right) \\ F\left(\mathbf{p}_{i,2}-\mathbf{p}_{j,1}\right) & \dots & F\left(\mathbf{p}_{i,2}-\mathbf{p}_{j,Q_j}\right) \\ \vdots & \ddots & \vdots \\ F\left(\mathbf{p}_{i,Q_i}-\mathbf{p}_{j,1}\right) & \dots & F\left(\mathbf{p}_{i,Q_i}-\mathbf{p}_{j,Q_j}\right) \end{bmatrix}. \tag{5.23}$$

Using this, we approximate the matrix-vector product, Ag_{123}, using a quadrature:

$$\mathbf{I} \approx \mathbf{J} = \widetilde{\mathrm{A}}\mathbf{g}_{123}, \tag{5.24}$$

where

$$\widetilde{\mathrm{A}} = \begin{bmatrix} \widetilde{\mathrm{A}}_{1,1} & \dots & \widetilde{\mathrm{A}}_{1,N} \\ \vdots & \ddots & \vdots \\ \widetilde{\mathrm{A}}_{M,1} & \dots & \widetilde{\mathrm{A}}_{M,N} \end{bmatrix}, \quad \widetilde{\mathrm{A}}_{i,j} = \begin{bmatrix} \widetilde{A}_{i,j,1,1} & \widetilde{A}_{i,j,1,2} & \widetilde{A}_{i,j,1,3} \\ \widetilde{A}_{i,j,2,1} & \widetilde{A}_{i,j,2,2} & \widetilde{A}_{i,j,2,3} \\ \widetilde{A}_{i,j,3,1} & \widetilde{A}_{i,j,3,2} & \widetilde{A}_{i,j,3,3} \end{bmatrix}. \tag{5.25}$$

Plugging in Eq. (5.22) and rearranging, we have

$$\mathbf{J} = \mathrm{M}^{\mathrm{T}}\mathrm{BM}\mathbf{g}_{123}, \tag{5.26}$$

where the matrices, M and B, are given by Eqs. (5.17) and (5.16), respectively. In this expression, the quadrature points for all N triangles are treated a point sources and assigned source strengths (Mg_{123}). Next, the potential due to these point sources are evaluated at the same set of quadrature points (BMg_{123}). So far, this process is identical to the one for the collocation method/evaluation of solution (the evaluation points are simply selected to be the quadrature points). Like in that case, the second step is accelerated using the FMM, and runs in $\mathrm{O}\left(N\right)$. Finally, the potentials are weighted and summed to obtain the integrals over the triangles ($\mathrm{M}^{\mathrm{T}}\mathrm{BMg}_{123}$). The mapping matrix, M, is used twice in this expression. When premultiplying $\mathbf{g}_{123}$, M maps the coefficients in $\mathbf{g}_{123}$ to the source strengths of the point sources. When premultiplying BMg_{123}, M is linearly combining the potentials at the

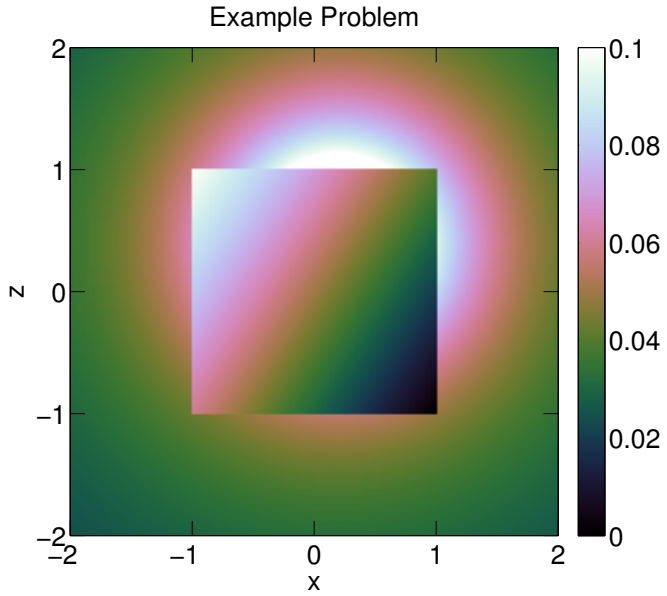

Figure 5.6: The analytical solution to the example problem. The potential outside the cube is due to a virtual point source inside the cube: $\phi^+(\mathbf{x}) = 1/(4\pi\,|\mathbf{x}-\mathbf{x}_0|)$, where $\mathbf{x}_0 = (0.2, 0.3, 0.4)$. The potential inside the cube is a linear gradient field: $\phi^-(\mathbf{x}) = \phi_0 - \mathbf{E}_0^{\mathrm{T}}\mathbf{x}$, where $\phi_0 = 0.05$ and $\mathbf{E}_0 = (0.03, 0.0, -0.02)$.

quadrature points to approximate the outside integral. The same mapping matrix is used in both cases because the quadrature points are the same for both the inside and outside integrals.

5.5 Numerical Examples for the Laplace Equation

5.5.1 Example Problem

We ran a series of computational experiments to verify the accuracy and characterize the performance of the correction factor matrix method for the Laplace equation. The example problem we used is shown in Fig. 5.6. An axis-aligned cube with a side length of two is centered at the origin. Dirichlet boundary conditions are enforced on both sides of the boundary. The potential on the outside surface of the cube is due to a virtual point

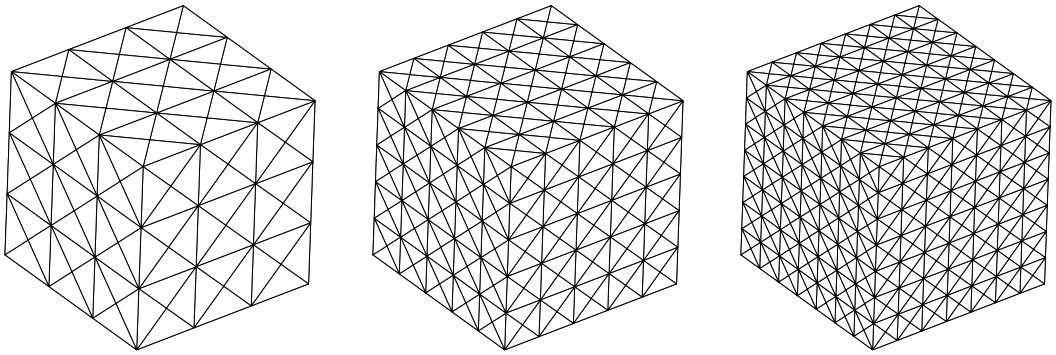

Figure 5.7: The cube mesh for three different mesh sizes.

source inside the cube:

$$\phi^{+}(\mathbf{x}) = \frac{1}{4\pi\,|\mathbf{x}-\mathbf{x}_0|}, \tag{5.27}$$

where $\mathbf{x}_0 = (0.2, 0.3, 0.4)$. The potential on the inside surface of the cube is a linear gradient field:

$$\phi^{-}(\mathbf{x}) = \phi_0 - \mathbf{E}_0^{\mathrm{T}}\mathbf{x}, \tag{5.28}$$

where $\phi_0 = 0.05$ and $\mathbf{E}_0 = (0.03, 0.0, -0.02)$. We want to compute the source density distributions, $\sigma(\mathbf{x})$ and $\mu(\mathbf{x})$, over the boundary that give rise to a potential that matches these boundary conditions. Of course, the analytical solution to the problem is given by the expressions in Eqs. (5.27) and (5.28). Thus, we can compute the potential at a set of validation points using the solution returned by the BEM, and then compare that to the analytical solution at the same set of points to verify that the method is accurate.

5.5.2 Accuracy

Our software has many components, each of which adds some error to the solution. Our experiments, therefore, sought to explore these sources of error, and to determine

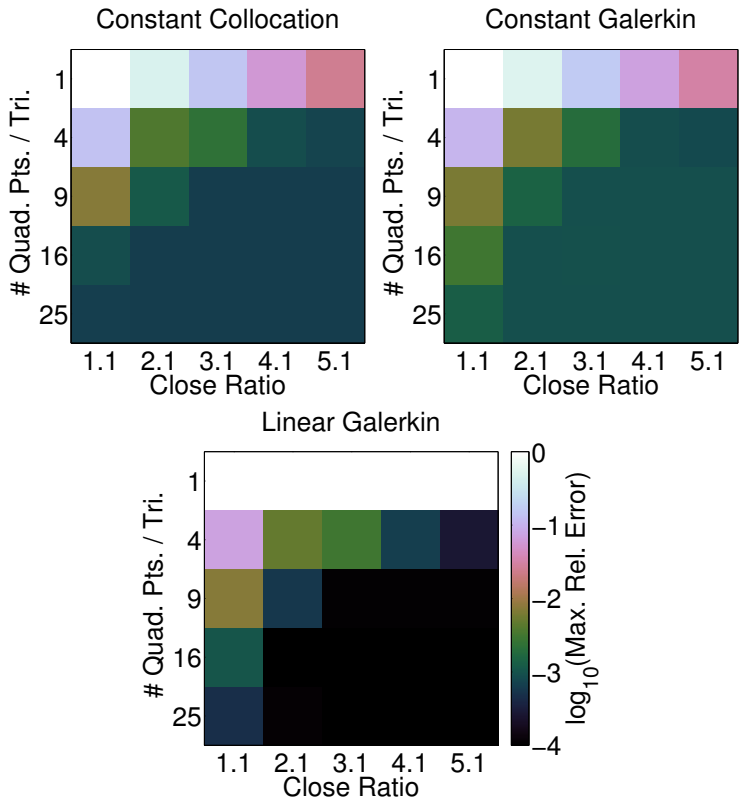

Figure 5.8: The maximum relative error of the example problem as a function of the number of quadrature points per triangle and the close ratio.

how best to set the available parameters to achieve the desired accuracy. There are several sources of error, namely:

1. The modeling error from using real-world measurements to construct the models.

2. The geometric error from discretizing the boundary using planar triangular elements.

3. The error from approximating the source density distributions, $\sigma(\mathbf{x})$ and $\mu(\mathbf{x})$, over the boundary using piecewise constant and linear basis functions.

4. The accuracy of the boundary integrals.

5. The error from using the correction factor matrix method, which is controlled by:

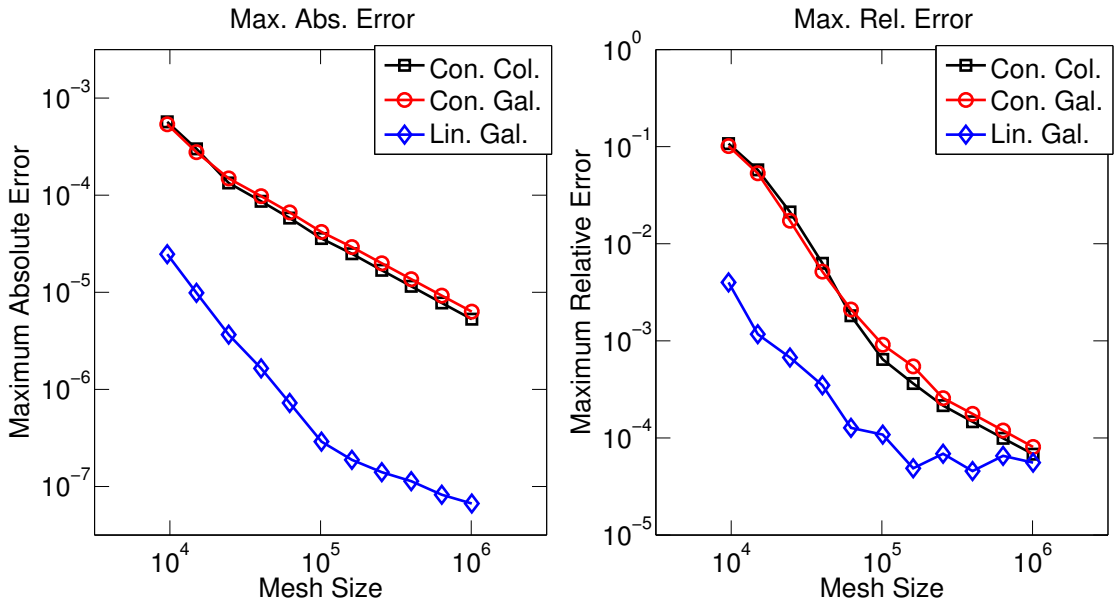

Figure 5.9: The maximum relative error of the example problem as a function of mesh size for the three solution methods. Constant collocation and constant Galerkin have similar error curves, while linear Galerkin gives the lowest errors. All three solution methods appear to saturate at around 0.01% error.

(a) the number of quadrature points per triangle; and

(b) the close ratio, which determines the density of the correction factor matrix.

6. The error from the multipole and local expansions and translations in the FMM. This error is controlled by the truncation number and the translation schemes used.

7. The error from the iterative solver, which is controlled by which solver is used (e.g., GMRES) and the chosen tolerance.

The modeling and geometric errors do not apply to the example problem. This is because the geometry of the problem is not based on real-world measurements, and also because the cube is a piecewise planar surface, which can be modeled exactly using planar triangles. Since these errors do not affect the problem at hand, this allows us to focus on the other sources of error. The approximation error is governed by the type of basis functions used

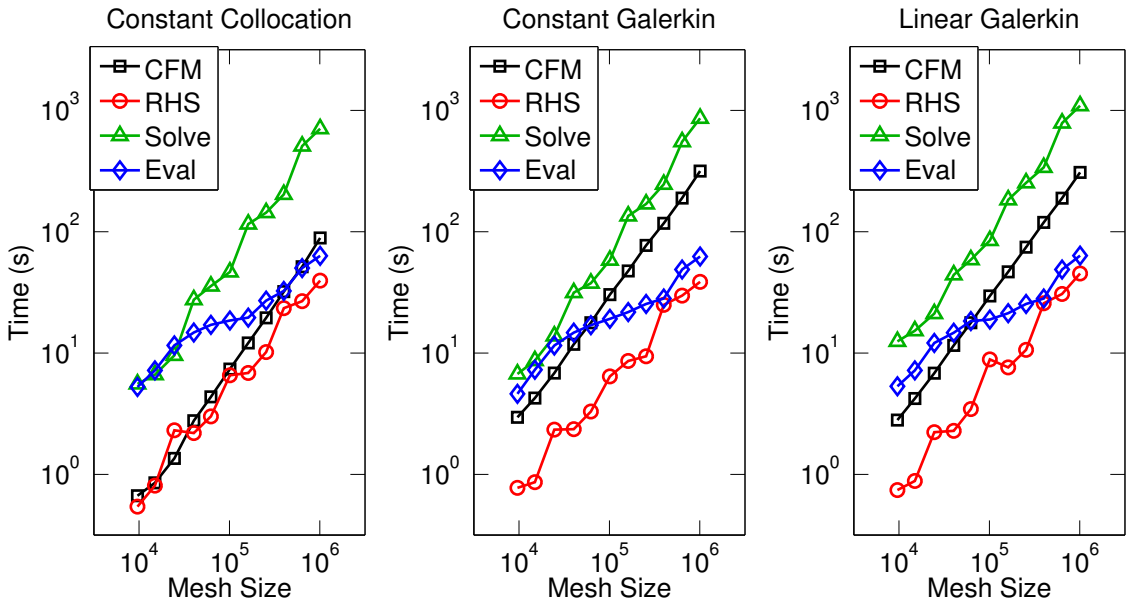

Figure 5.10: The performance of the four components of our code (correction factor matrix, right-hand side, solve, and evaluation) for the example problem as a function of mesh size for the three solution methods.

(i.e., piecewise constant and linear basis functions) and the mesh size (i.e., how many basis functions are used). We explored the behavior of the approximation error by comparing the errors from using either piecewise constant or piecewise linear basis functions, and also by varying the mesh size. Fig. 5.7 shows the meshes we used. The mesh size ranged from 10,000 triangles to over 1,000,000. The entries in the correction factor matrix should be computed accurately. For the collocation method, the required single surface integrals can be computed exactly (see Sec. 3.2 and Appx. A), so there is no error. For the Galerkin method, we used the subdivision and scaling method developed in Chap. 3 to compute the required double surface integrals accurately. The error from the FMM was controlled by selecting an appropriate truncation number, p. In our experiments, we chose $p = 20$. We used the unpreconditioned GMRES with a relative tolerance of 10^{-6}.

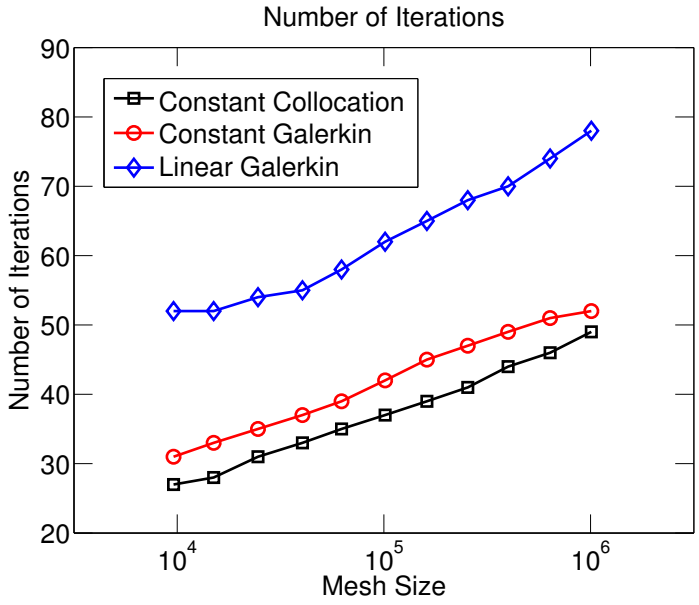

Figure 5.11: The number of iterations required by the iterative solver to converge as a function of mesh size for the three solution methods. Convergence graphs for each of the data points in this graph can be seen in Fig. 5.12.

This leaves two sources of error: the number of quadrature points per triangle, Q, and the close ratio, C. These two sources of error are highly intertwined, and must be set together to achieve the desired accuracy. In general, for fewer quadrature points per triangle, larger close ratios are needed. Likewise, for more quadrature points per triangle, smaller close ratios are needed. A balance must be struck between these two parameters. To study the effect of these two parameters on the accuracy, we solved the example problem several times, holding all parameters constant, but varying the number of quadrature points per triangle and the close ratio. The mesh had 101,400 triangles, and we varied $Q = 1, 4, 9, 16, 25$ and $C = 1.1, 2.1, 3.1, 4.1, 5.1$. For each combination of Q and C, we computed the maximum relative error at a set of validation points. Fig. 5.8 shows this error as a function of the two parameters. As expected, increasing Q or C improves the accuracy. Also, when one parameter is decreased, the other can be increased to offset

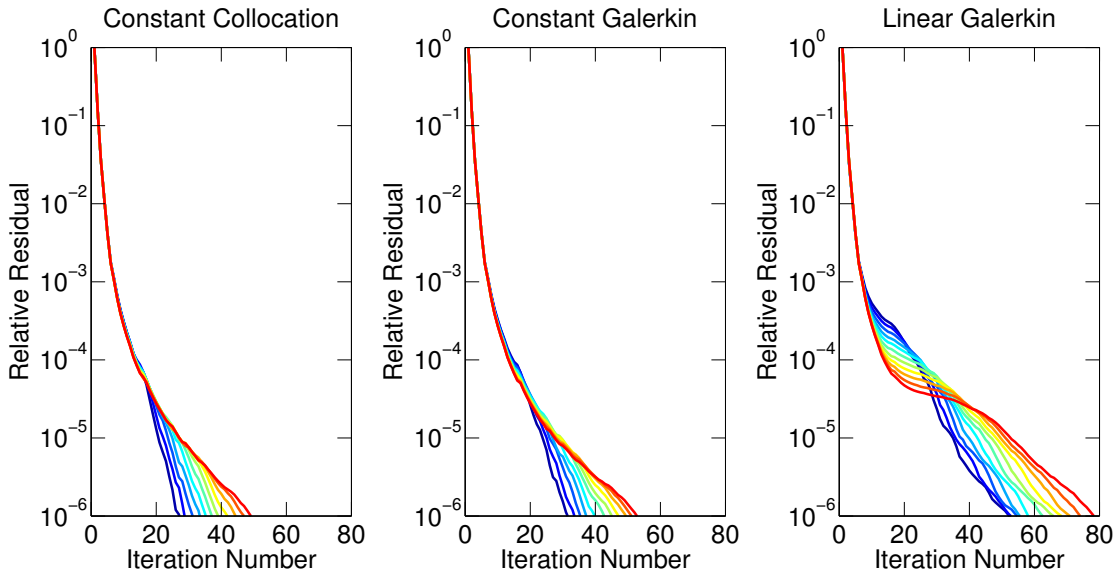

Figure 5.12: Convergence graphs for the example problem for the three solution methods showing the relative residual as a function of the iteration number. There are 11 curves in each plot, one for each data point in Figs. 5.9, 5.10, and 5.11. In general, the larger meshes took longer to converge.

any drop in accuracy. One interesting thing is that, for linear Galerkin, using $Q = 1$ is bad no matter the choice of C. This is likely due to the fact that only one quadrature point is unable to account for the linear variation across a triangle. For constant collocation and constant Galerkin, $Q = 1$ is also bad, but becomes slightly better for larger values of C. Based on this experiment, we chose $Q = 9$ and $C = 3.1$ as the best combination in terms of accuracy and performance. This yielded 0.1% error for constant collocation and constant Galerkin, and 0.01% error for linear Galerkin. In an engineering context, where only 1% errors are needed, lower values of Q and C could be used, which would lead to better performance. However, since we are interested in the best kinds of accuracy, we have kept $Q = 9$ and $C = 3.1$. Using these values, we computed the maximum absolute and relative errors as a function of mesh size, N (see Fig. 5.9). As expected, all errors decrease as N increases. Constant collocation and constant Galerkin yield similar errors,

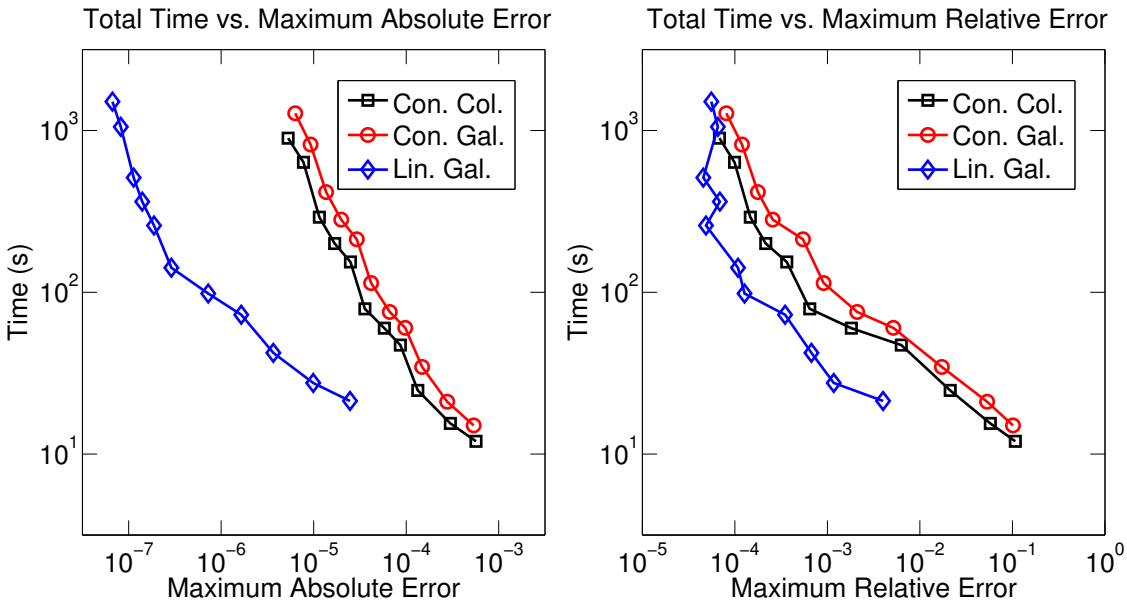

Figure 5.13: The total time (sum of the times for all four components of our code) as a function of the maximum absolute (left) and relative (right) errors for the three solution methods. Linear Galerkin gives the biggest bang for the buck, providing the best accuracy in the least amount of time.

while linear Galerkin yields much lower errors. All three solution methods reach 0.01% error, although linear Galerkin reaches this level much faster than the other two.

5.5.3 Performance

The code is divided into four pieces: (1) determining which entries in the correction factor matrix to compute, and then computing them; (2) computing the right-hand side; (3) solving the system; and (4) evaluating the solution at the evaluation points. To characterize the performance of each piece, we solved the example problem for different mesh sizes for the three solution methods (constant collocation, constant Galerkin, and linear Galerkin). Times were measured for each run, and then plotted as a function of mesh size (see Fig. 5.10). Information about the performance of the iterative solver was also logged. Fig. 5.11 shows the number of iterations required by the iterative solver to converge as a function of

mesh size for the three solution methods. Corresponding convergence graphs can be seen in Fig. 5.12.

The runs were done on a machine that had a Quad Intel Xeon Dual E5-2690 ($4 \times 8 = 32$ cores), 128 GM of RAM, and a Tesla K20c GPU. The code was written in a combination of Fortran, C, and C++. MATLAB was used as a glue to connect the different pieces of code together, but otherwise did very little work. The most computationally intensive code was parallelized using OpenMP to take advantage of all 32 available cores on the machine. The computation of the boundary integrals and correction factors was accelerated on the GPU (see Sec. 3.6). The FMM was also accelerated by computing and summing the local interactions on the GPU. The GPU code was written in CUDA.

Based on the analysis from Sec. 5.5.2, we used $Q = 9$ and $C = 3.1$. As discussed in that section, these parameters could be decreased (e.g., $Q = 4$ and $C = 2.1$) to achieve better performance at the expense of accuracy. For example, this could be done during the early stages of engineering design, where new ideas need to be tested rapidly, and answers do not need to be perfect. On the other hand, they could be increased (e.g., $Q = 16$ and $C = 4.1$) when higher accuracy is required, e.g., during the end stages of design.

The performance was very good. Timing scaled as $\mathrm{O}(N)$ or $\mathrm{O}(N \log(N))$ for all four pieces of the code. In particular, for $N = 1,008,600$, constant collocation ran in 14.9 minutes, constant Galerkin ran in 21.3 minutes, and linear Galerkin ran in 25.1 minutes. The difference in performance between the three solution methods is largely due to the computation of the correction factors. This makes sense: computing the double surface integrals in the two Galerkin methods is much more computationally expensive than computing the single surface integrals in the collocation method.

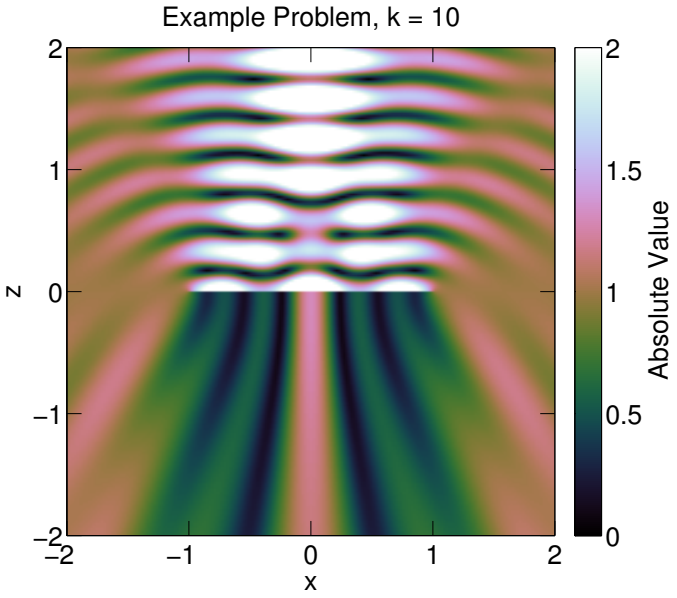

Figure 5.14: The example problem is a plane wave striking a sound-hard disk of radius one from directly above. Here, the absolute value of the analytical solution to the example problem is shown for $k = 10$.

For the same mesh size, constant Galerkin takes longer than constant collocation, and linear Galerkin takes longer than both of these. However, linear Galerkin is much more accurate than the other two. To determine which method is fastest for the same level of accuracy and should be used, we plotted the total time (sum of the times for all four components of our code) as a function of the maximum absolute and relative errors for the three solution methods (see Fig. 5.13). Linear Galerkin gives the biggest bang for the buck, providing the best accuracy in the least amount of time. This is because linear Galerkin provides lower errors at smaller mesh sizes, and these smaller mesh sizes take shorter amounts of time to solve.

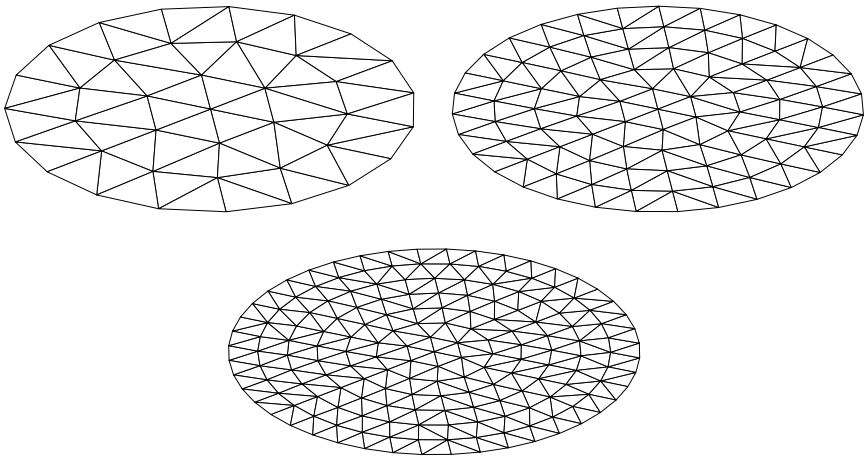

Figure 5.15: The disk mesh for three different mesh sizes.

5.6 Numerical Examples for the Helmholtz Equation

5.6.1 Example Problem

We ran a series of computational experiments to verify the accuracy and characterize the performance of the correction factor matrix method for the Helmholtz equation. The example problem we used was a plane wave striking a sound-hard disk from directly above, and is shown in Fig. 5.14. Scattering problems involving spheroids and disks have been studied for well over a century, and analytical expressions for computing their solutions have been documented extensively by many authors. However, even with the immense amount of work that has been done on the topic, there are currently no publicly available libraries that implement these expressions. We developed computational software for calculating the solutions to these problems, and a description of this software can be found in Chap. 6.

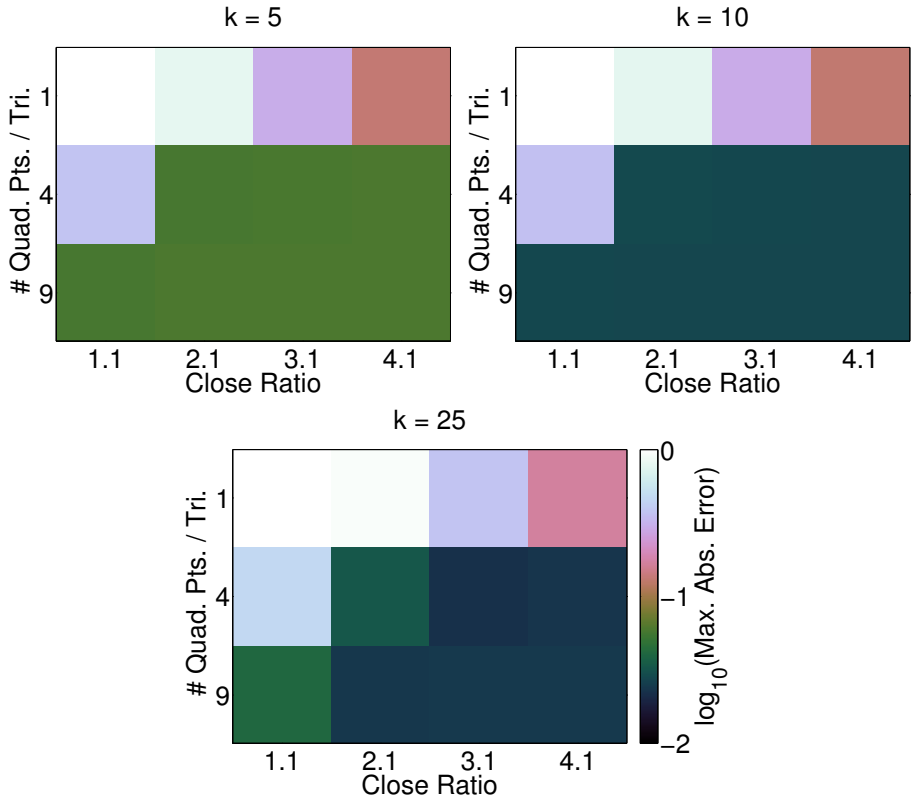

Figure 5.16: The maximum absolute error for the example problem as a function of the number of quadrature points per triangle and the close ratio for $k = 5$ (top left), 10 (top right), and 25 (bottom).

5.6.2 Accuracy

Like in the case of the Laplace equation, our software for solving the Helmholtz equation has many components, each of which adds some error to the solution. They are the same in this case: (1) the modeling error; (2) the geometric error; (3) the approximation error; (4) the error from computing the boundary integrals; (5) the error from the correction factor matrix method; (6) the error from the FMM; and (7) the error from the iterative solver. Our experiments, therefore, sought to explore these sources of error, and to determine how best to set the available parameters to achieve the desired accuracy. While the Laplace equation behaves reasonably well, the Helmholtz equation is not so forgiving in some

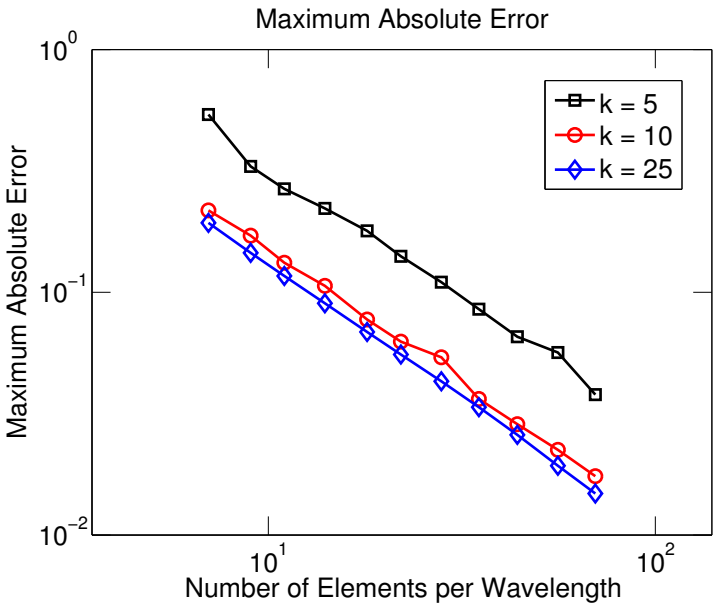

Figure 5.17: The maximum absolute error for the example problem as a function of the number of elements per wavelength for $k = 5, 10,$ and 25.

ways. As a result, for some of these sources of error, we had to take extra care to ensure the accuracy of our methods was good.

The modeling error does not apply to the example problem because the geometry of the problem is not based on real-world measurements. There is geometric error in the example problem because triangular elements are unable to model the smooth edge of the disk. The approximation error is governed by the type of basis functions used (i.e., piecewise constant and linear basis functions) and the mesh size (i.e., how many basis functions are used). We only considered piecewise constant basis functions in our experiments, but we otherwise explored the approximation error by varying the mesh size. Fig. 5.15 shows the meshes we used. For the Laplace equation, there is no intrinsic scale, so the only driving force behind the number of elements in a mesh is that the geometry is sufficiently resolved. However, for the Helmholtz equation, the wavelength, $\lambda = 2\pi/k$, introduces a scale to the problem. The Nyquist-Shannon sampling theorem requires that

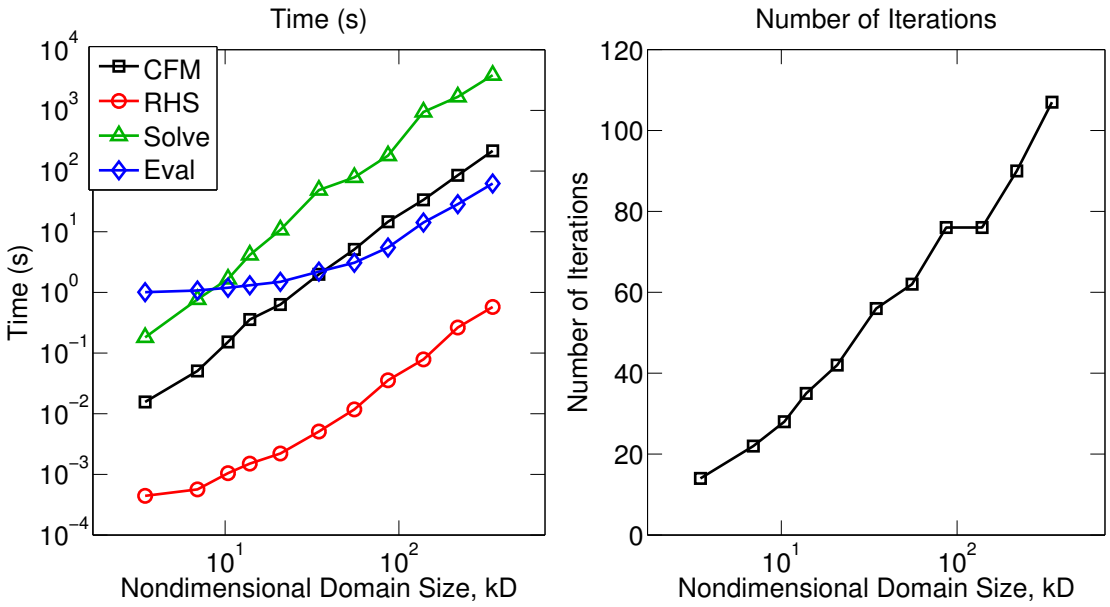

Figure 5.18: On the left: the performance of the four components of our code (computing the correction factor matrix, computing the right-hand side, solving the system, and evaluating the solution) for the example problem as a function of the nondimensional domain size, kD. For each value of kD, the mesh size was chosen so that there were 20 elements per wavelength. On the right: the number of iterations required by the iterative solver to converge as a function of kD.

there be at least two elements per wavelength, although at least 6 to 10 elements per wavelength are usually recommended [23]. In our experiments, the number of elements per wavelength ranged from 7 to 70, and depending on the wavenumber, the mesh size ranged from around 200 to over 600,000. The entries in the correction factor matrix should be computed accurately. As discussed in Sec. 4.2, the singularity subtraction method is used to separate the boundary integrals into two pieces, one for the Laplace kernel and one for $\widetilde{G}$ (i.e., the Helmholtz kernel minus the Laplace kernel). The former can be computed exactly (see Sec. 3.2 and Appx. A), so there is no error from that. The latter was computed using a 100-point Gaussian quadrature, and in our experiments, the error from this computation was always well below the other sources of error. We used a third-party FMM [27]. This FMM provided a single parameter for controlling the error, which we set so that the error

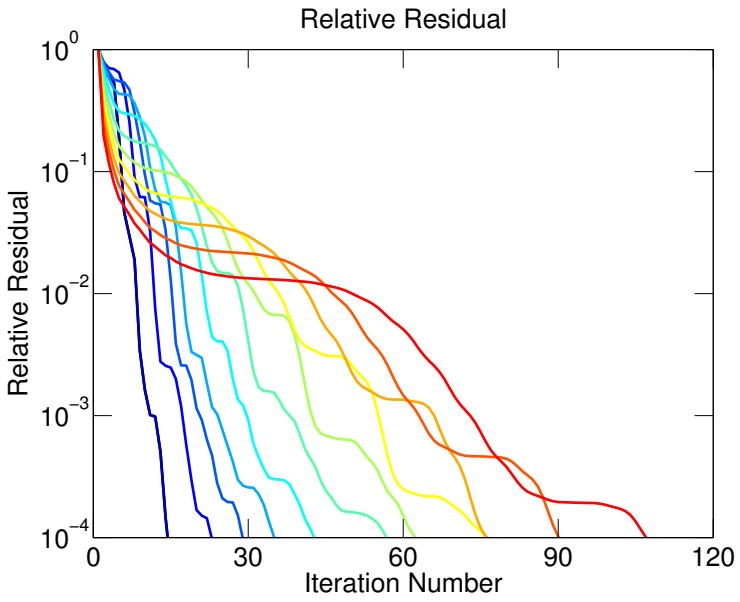

Figure 5.19: Convergence graphs showing the relative residual as a function of the iteration number. There is one curve for each data point in Fig. 5.18.

was below 10^{-3}. We used the unpreconditioned GMRES with a relative tolerance of 10^{-4}.

This leaves two sources of error: the number of quadrature points per triangle, Q, and the close ratio, C. Like in the case of the Laplace equation, these two sources of error are highly intertwined, and must be set together to achieve a desired accuracy. To study the effect of these two parameters on the accuracy, we solved the example problem several times, holding all parameters constant, but varying the number of quadrature points per triangle and the close ratio. Three wavenumbers were tested: $k = 5, 10$, and 25. For each value of k, the number of elements in the mesh was chosen so that there were 50 elements per wavelength. This corresponds to 10,053, 40,213, and 248,822 elements, respectively. We varied $Q = 1, 4, 9$ and $C = 1.1, 2.1, 3.1, 4.1$. For each combination of Q and C, we computed the maximum absolute error at a set of validation points. Fig. 5.16 shows this error as a function of the two parameters. As expected, increasing Q or C improves the accuracy. Also, when one parameter is decreased, the other can be increased to offset any

drop in accuracy. Based on this experiment, we chose $Q = 4$ and $C = 3.1$ as the best combination in terms of accuracy and performance. Using these values, we computed the maximum absolute error as a function of the number of elements per wavelength, ranging this value from 7 to 70 (see Fig. 5.17). As expected, all errors decrease as this value increases. In fact, the error drops by a factor of two for every doubling of the number of elements per wavelength.

5.6.3 Performance

Like in the case of the Laplace equation, the code is divided into four pieces: (1) determining which entries in the correction factor matrix to compute, and then computing them; (2) computing the right-hand side; (3) solving the system; and (4) evaluating the solution at the evaluation points. To characterize the performance of each piece, we solved the example problem for different values of kD. The wavenumber, k, varied from 1 to 100, and for the example problem, $D = 2 \times 3^{1/2}$, which is equal to the length of the diagonal of the smallest cube that contains a disk of radius one. Therefore, kD varied from 3.46 to 346. For each kD, the mesh size was chosen so that there were 20 elements per wavelength. Fig. 5.18 shows the times of the four components of our code as a function of kD. The performance was very good. The time required for computing the correction factor matrix, computing the right-hand side, and evaluating the solution all scaled as $\mathrm{O}\left((kD)^2\right)$ or better. The time required to solve the system scaled as $\mathrm{O}\left((kD)^{2.21}\right)$. In particular, solving the system for $kD = 346$ took 63.2 minutes. Fig. 5.11 also shows the number of iterations required by the iterative solver to converge as a function of kD.

Corresponding convergence graphs can be seen in Fig. 5.19.

The runs were done on a machine that had a Quad Intel Xeon Dual E5-2690 ($4 \times 8 = 32$ cores), 128 GM of RAM, and a Tesla K20c GPU. The code was written in a combination of Fortran, C, and C++. MATLAB was used as a glue to connect the different pieces of code together, but otherwise did very little work. The most computationally intensive code was parallelized using OpenMP to take advantage of all 32 available cores on the machine. Unlike in the case of the Laplace equation, the computation of the boundary integrals and correction factors was not accelerated using the GPU. We used a third-party FMM [27]. This FMM was written in Fortran, was multi-threaded, and ran across all 32 available cores on the machine. In general, the FMM performed very well, and was accurate in all of our experiments.

5.7 Conclusion

An FMM/GPU-accelerated BEM for the Laplace and Helmholtz equations in three dimensions was presented. The unaccelerated BEM suffers from two problems. The system matrix is dense, requiring $\mathrm{O}(N^2)$ storage, where N is the number of boundary elements, and solving the system by direct means (e.g., LU decomposition) requires $\mathrm{O}(N^3)$ operations. These quadratic and cubic costs can effectively restrict the size of a problem. The FMM can be used to reduce these scalings to linear, and allow for very large problem sizes. However, the FMM is usually designed around point and dipole sources, not the integral expressions that appear in the BEM. Moreover, the FMM for point and dipole sources has been studied extensively by previous authors, and implementations are widely

available and highly optimized. Therefore, instead of modifying the FMM to account for these expressions, we have developed a technique that allows for the FMM to be used unchanged to accelerate the BEM. This technique is called the correction factor matrix method. The method works by mapping the source density distributions over the boundary to a set of representative point and dipole sources, and plugging these into the FMM. Any inaccuracies from this step are corrected by computing and adding correction factors. In the case of the Laplace equation, the method was further accelerated by computing these correction factors on the GPU, and by using a heterogeneous CPU/GPU FMM. This chapter stated the problem, derived and described the correction factor matrix method, and presented example problems showing accuracy and performance.

Chapter 6: Acoustic Scattering by Spheroids and Disks[1]

6.1 Introduction

Even with the advent of efficient numerical solvers for many physical problems, analytical methods remain valuable. Many problems can be simplified to a point where analytical methods can be applied. These methods can provide insight into the problem, and can help researchers when moving to a numerical solver. Numerical methods, on the other hand, have the advantage of being able to treat problems with arbitrary geometries. In order to do so, they return numerical approximations. The numerical error can usually be reduced by increasing the number of modeling elements. Without some kind of validation, though, there is no way to know whether the numerical solution converges to the correct one. Analytical methods can provide this validation. Indeed, the FMM-accelerated indirect BEM for the Helmholtz equation requires this validation.

The indirect BEM is capable of treating open, infinitely thin surfaces. These surfaces are good approximations to those often encountered in practice – those that are much smaller than a wavelength in one dimension, but span several in the other two. However, there is only one analytically tractable problem posed on such a surface that has a solution and could be used to validate the indirect BEM: an acoustic wave scattering off a disk.

[1] This section is based on our original work in [74].

The disk is actually the degenerate form of the oblate spheroid, so methods for solving scattering problems involving oblate spheroids can also be applied to the disk. Scattering problems involving oblate spheroids, as well as the closely related prolate spheroids, have been studied for well over a century, and analytical expressions for computing their solutions have been documented extensively by many authors.

One of the earliest papers on the topic was by Lord Rayleigh in 1897 [75]. As discussed in [76], prior to the discovery and use of the spheroidal wave functions, the best solutions to these problems were approximations, either in frequency or distance. When separated in spheroidal coordinates, solutions to the Helmholtz equation can be written in terms of the spheroidal wave functions [77]. Because spheroids can be represented as isosurfaces in these coordinate systems, solutions to acoustic waves scattering off them can be written in terms of these special functions. The resulting expressions are accurate over a wide range of frequencies and distances. These expressions were applied to the disk in [78, 79], and they were validated experimentally in [80]. Spheroids of different sizes were studied in the following decades [81, 82], and much of this work was organized into an encyclopedic book [83], which also includes an extensive bibliography on the topic. More recently, these expressions were implemented and used to compute the solutions over a wide range of frequencies and spheroid sizes [84, 85]. However, even with the immense amount of work that has been done on the topic, there are currently no publicly available libraries that implement these expressions. Also, in all of these references, the spheroids and disks were assumed to be entirely sound soft or sound hard, but the more general case of Robin boundary conditions was never considered in detail.

We have developed computational software for calculating the solutions to acoustic

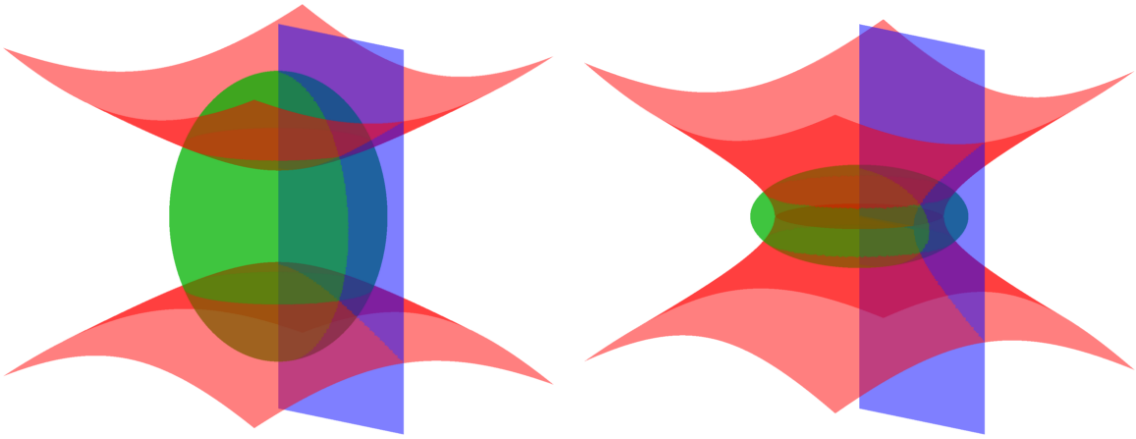

Figure 6.1: The prolate (left) and oblate (right) spheroidal coordinate systems. The three surfaces are isosurfaces for $\eta = \pm 1/2$ (the hyperboloids), $\xi = 3/2$ for the prolate case and $\xi = 1/2$ for the oblate case (the spheroids), and $\phi = 0$ (the planes).

scattering problems involving prolate spheroids, oblate spheroids, and disks. This software is called `scattering` and runs in MATLAB. We have also developed software for computing the spheroidal wave functions required by `scattering`. This software is called `spheroidal` and is described in Chap. 7. Using `spheroidal`, we have precomputed many values of the spheroidal wave functions, which, along with `scattering`, are freely available for download from our webpage [86]. Together, they can be used to recreate the examples seen here. Using this software, we were able to validate the accuracy of the FMM-accelerated indirect BEM for the Helmholtz equation.

6.2 Spheroidal Coordinates

The prolate spheroidal coordinate system, shown in Figure 6.1, is related to the Cartesian coordinate system by [77]

$$x = a\left(1-\eta^2\right)^{1/2}\left(\xi^2-1\right)^{1/2}\cos\left(\phi\right), \tag{6.1}$$

$$y = a\left(1-\eta^2\right)^{1/2}\left(\xi^2-1\right)^{1/2}\sin\left(\phi\right), \tag{6.2}$$

$$z = a\eta\xi, \tag{6.3}$$

where $2a$ is the interfocal distance. The Helmholtz equation can be written in prolate spheroidal coordinates as

$$\left(\frac{\partial}{\partial\eta}\left(1-\eta^2\right)\frac{\partial}{\partial\eta}+\frac{\partial}{\partial\xi}\left(\xi^2-1\right)\frac{\partial}{\partial\xi}+\frac{\xi^2-\eta^2}{\left(1-\eta^2\right)\left(\xi^2-1\right)}\frac{\partial^2}{\partial\phi^2}+c^2\left(\xi^2-\eta^2\right)\right)V=0, \tag{6.4}$$

where $c = ka$. Applying the method of separation of variables yields three uncoupled ordinary differential equations, one for each coordinate:

$$\frac{\partial}{\partial\eta}\left(\left(1-\eta^2\right)\frac{\partial}{\partial\eta}S_{mn}\left(c,\eta\right)\right)+\left(\lambda_{mn}-c^2\eta^2-\frac{m^2}{1-\eta^2}\right)S_{mn}\left(c,\eta\right)=0, \tag{6.5}$$

$$\frac{\partial}{\partial\xi}\left(\left(\xi^2-1\right)\frac{\partial}{\partial\xi}R_{mn}\left(c,\xi\right)\right)-\left(\lambda_{mn}-c^2\xi^2+\frac{m^2}{\xi^2-1}\right)R_{mn}\left(c,\xi\right)=0, \tag{6.6}$$

$$\frac{\partial^2}{\partial\phi^2}\Phi_m\left(\phi\right)+m^2\Phi_m\left(\phi\right)=0, \tag{6.7}$$

where $m = 0, 1, \ldots$ and $n = m, m+1, \ldots$. The solutions to Eq. (6.5) are called the prolate spheroidal angle functions, and the solutions to Eq. (6.6) are called the prolate spheroidal radial functions. Collectively, they are called the prolate spheroidal wave functions. Any solution to Eq. (6.4) can be written as

$$V = \sum_{m=0}^{\infty} \sum_{n=m}^{\infty} S_{mn}(c, \eta) \left(A_{mn} R_{mn}^{(1)}(c, \xi) + B_{mn} R_{mn}^{(3)}(c, \xi) \right) \cos(m\phi), \tag{6.8}$$

where the expansion coefficients, A_{mn} and B_{mn}, depend on the problem being solved.

The expression for an acoustic wave due to a point source is $V^{\text{ps}} = \exp(ikr) / (4\pi r)$, where r is the distance between the point source and evaluation point. When expanded in terms of the prolate spheroidal wave functions,

$$V^{\text{ps}} = \frac{ik}{2\pi} \sum_{m=0}^{\infty} \sum_{n=m}^{\infty} \frac{\epsilon_m}{N_{mn}(c)} S_{mn}^{(1)}(c, \eta_0) S_{mn}^{(1)}(c, \eta) R_{mn}^{(1)}(c, \xi_<) R_{mn}^{(3)}(c, \xi_>) \cos(m\phi), \tag{6.9}$$

where $(\eta_0, \xi_0, 0)$ and (η, ξ, ϕ) are the positions of the point source and evaluation point in prolate spheroidal coordinates, respectively (because of symmetry, $\phi_0 \neq 0$ can be achieved by rotating the problem around the z axis), $\xi_< = \min(\xi_0, \xi)$, and $\xi_> = \max(\xi_0, \xi)$. Likewise, the expression for a plane wave is $V^{\text{pw}} = \exp(i\mathbf{k} \cdot \mathbf{r})$, where $\mathbf{k}$ is the wavevector and $\mathbf{r}$ is the position vector of the evaluation point. When expanded in terms of the prolate spheroidal wave functions,

$$V^{\text{pw}} = 2 \sum_{m=0}^{\infty} \sum_{n=m}^{\infty} \frac{\epsilon_m i^n}{N_{mn}(c)} S_{mn}^{(1)}(c, \cos(\theta_0)) S_{mn}^{(1)}(c, \eta) R_{mn}^{(1)}(c, \xi) \cos(m\phi), \tag{6.10}$$

where, in this expression, $\mathbf{k}$ has been restricted to $\mathbf{k} = k\left(\sin\left(\theta_0\right), 0, \cos\left(\theta_0\right)\right)$ (unrestricted values of $\mathbf{k}$ can be achieved by rotating the problem around the z axis).

Consider a prolate spheroid, which is described by the isosurface, $\xi = \xi_1$. The prolate spheroid can be sound soft ($[V]_{\xi=\xi_1} = 0$), sound hard ($[dV/d\mathbf{n}]_{\xi=\xi_1} = 0$), or Robin boundary conditions can be used ($[V + \alpha dV/d\mathbf{n}]_{\xi=\xi_1} = 0$ and $\alpha =$ constant). The Robin case lies somewhere between the other two: the prolate spheroid is sound soft when $\alpha = 0$ and sound hard when $\alpha \to \infty$. Suppose an incident potential, V^{i}, due to either a point source or plane wave is generated, which strikes the prolate spheroid. We wish to compute the scattered potential, V^{s}, from the incident potential bouncing off the prolate spheroid. Depending on the boundary conditions and whether the incident potential is due to a point source or plane wave, the exact method of solution is slightly different, but they all follow the same procedure: (1) the incident potential is expanded in terms of the prolate spheroidal wave functions using either Eq. (6.9) or (6.10); (2) the same is done for the scattered potential using Eq. (6.8); (3) the two expressions are added together to form an expression for the total potential in terms of the prolate spheroidal wave functions; (4) the total potential and normal derivative are evaluated at the boundary; and (5) by using the orthogonality of the prolate spheroidal wave functions, the resulting expression is used to determine the expansion coefficients, A_{mn} and B_{mn}, so that the boundary conditions are satisfied. For the Robin case and a point source,

$$A_{mn} = 0, \quad B_{mn} = -\frac{ik}{2\pi}\frac{\epsilon_m}{N_{mn}\left(c\right)} S_{mn}^{(1)}\left(c, \eta_0\right) R_{mn}^{(3)}\left(c, \xi_0\right) \frac{R_{mn}^{(1)}\left(c, \xi_1\right) + \alpha R_{mn}^{(1)\prime}\left(c, \xi_1\right)}{R_{mn}^{(3)}\left(c, \xi_1\right) + \alpha R_{mn}^{(3)\prime}\left(c, \xi_1\right)}. \tag{6.11}$$

For the Robin case and a plane wave,

$$A_{mn} = 0, \quad B_{mn} = -2\frac{\epsilon_m i^n}{N_{mn}(c)} S_{mn}(c, \cos(\theta_0)) \frac{R^{(1)}_{mn}(c, \xi_1) + \alpha R^{(1)\prime}_{mn}(c, \xi_1)}{R^{(3)}_{mn}(c, \xi_1) + \alpha R^{(3)\prime}_{mn}(c, \xi_1)}. \quad (6.12)$$

For a prolate spheroid that is sound soft ($\alpha = 0$) or sound hard ($\alpha \to \infty$), these expressions reduce to those in [83].

The expressions arising in the oblate case are very similar to (and sometimes exactly the same as) those arising in the prolate case. In many cases, simply letting $c, \xi \to -ic, i\xi$ provides a transformation from the prolate case to the oblate case [77]. Indeed, the preceding paragraphs and equations for the prolate case can be transformed into those for the oblate case by using this transformation. The oblate spheroidal coordinate system is shown in Figure 6.1.

6.3 Scattering Routines

We have developed computational software for calculating the solutions to acoustic scattering problems involving prolate spheroids, oblate spheroids, and disks. This software is called `scattering` and runs in MATLAB. The spheroids and disks can be sound soft, sound hard, or Robin boundary conditions can be used, and the incident potential can be due to either a point source or plane wave. Internally, `scattering` needs to compute the spheroidal wave functions of different order and degree for different values of their argument. We have developed software for doing so. This software is called `spheroidal` and is described in Chap. 7.

There are 18 routines in `scattering`, each one solving a different scattering

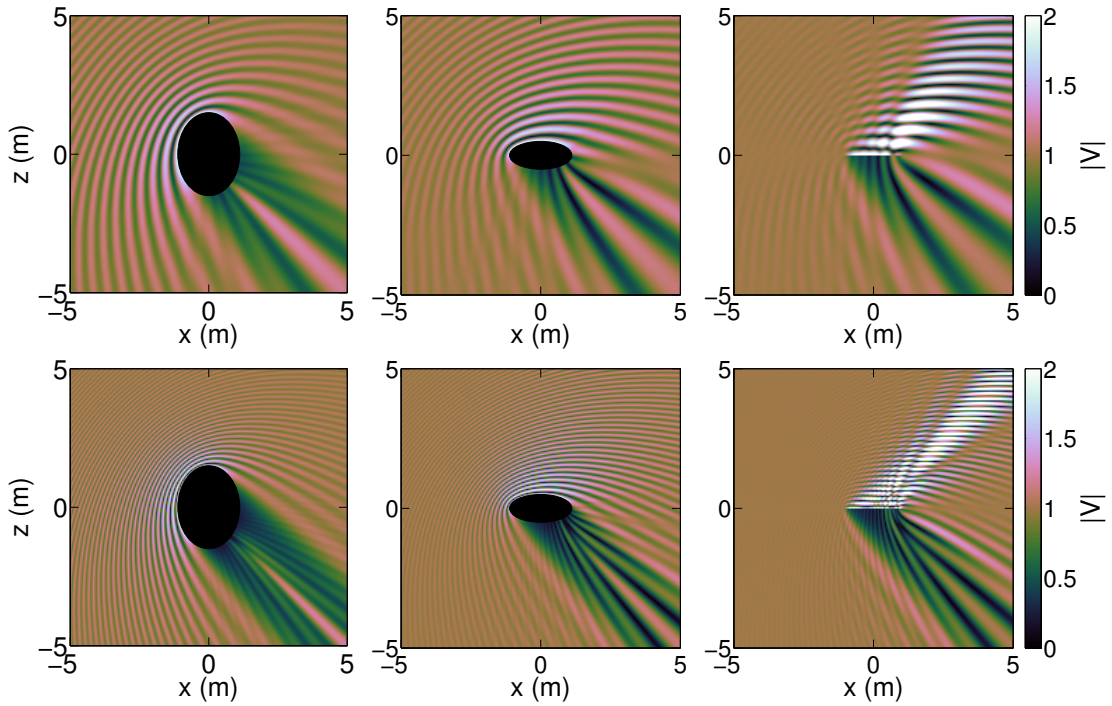

Figure 6.2: A plane wave scattering off a sound-hard prolate spheroid (left), oblate spheroid (center), and disk (right) for $k = 10$ (top) and $k = 25$ (bottom).

problem. For example, one is called `pro_plane_wave_scat_hard`, which computes the scattered potential from a plane wave striking a sound-hard prolate spheroid. They are organized into 18 separate MATLAB M-Files, one for each routine. At the top of each M-File is a detailed explanation of the calling convention, return values, and so on.

Each routine has a slightly different calling convention depending on what is being computed, but they all follow the same pattern. They all require k and a, as well as information about the incident potential, either η_0 and ξ_0 for a point source or θ_0 for a plane wave. Routines that compute a scattered potential also require ξ_1. For example, to compute the scattered potential from a plane wave striking a sound-hard prolate spheroid, the following MATLAB code fragment can be used:

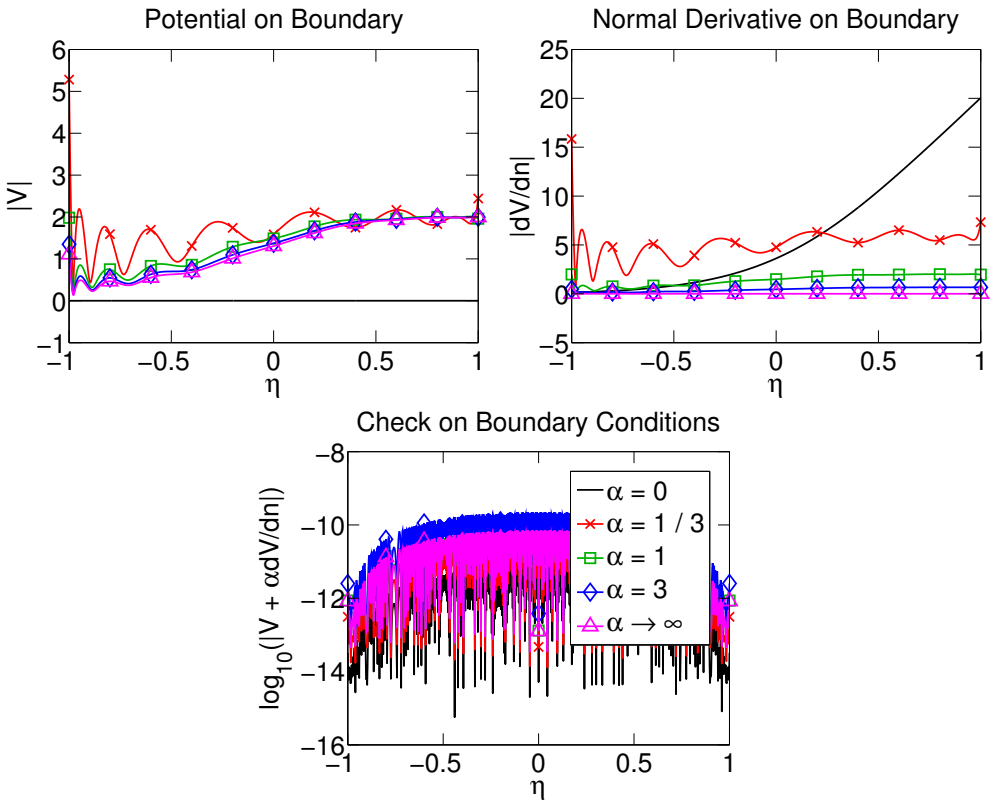

Figure 6.3: The two graphs on the top show the potential and normal derivative on the boundary of an oblate spheroid from a plane wave striking the oblate spheroid from directly above for five different choices of the boundary conditions. The one graph on the bottom shows that the boundary conditions have been satisfied in all five cases.

```
v_scat = pro_plane_wave_scat_hard( ...
                  10.0, 1.0, pi, 'saved', 1.5, x, y, z);
```

where, in this case, $k = 10$, $a = 1$, $\theta_0 = \pi$, `'saved'` is the directory in which the spheroidal wave functions have been precomputed and saved, and $\xi_1 = 3/2$. The variables, `x`, `y`, and `z`, are row vectors, which contain the positions of the evaluation points in Cartesian coordinates, and `v_scat` is a row vector, which will contain V^s.

Figs. 6.2 and 6.3 show some examples of running the scattering software for different wavenumbers, spheroid shapes and sizes, and boundary conditions.

6.4 Conclusion

We have developed computational software for calculating the solutions to acoustic scattering problems involving prolate spheroids, oblate spheroids, and disks. While these problems have been studied for well over a century and analytical expressions for computing their solutions have been documented by many authors, there are currently no publicly available libraries that implement them. This chapter gave an overview of these problems, derived their analytical solutions, described some theory behind the special functions required by them, and included several examples of running our software. Using this software, we were able to validate the accuracy of the FMM-accelerated indirect BEM for the Helmholtz equation.

Chapter 7: Spheroidal Wave Functions[1]

7.1 Introduction

The spheroidal wave functions are the solutions to the ordinary differential equations obtained by applying the method of separation of variables to the Helmholtz equation in spheroidal coordinates. There are two cases: the prolate spheroidal wave functions when prolate spheroidal coordinates are used; and the oblate spheroidal wave functions when oblate spheroidal coordinates are used. Unfortunately, there are no simple expressions for computing them. Instead, they must be written as infinite series expansions in terms of other special functions. For example, the spheroidal angle functions can be written in terms of the associated Legendre polynomials, and the spheroidal radial functions can be written in terms of the spherical Bessel and Neumann functions. Depending on the method used, there are three or four different sets of expansion coefficients that need to be computed. The spheroidal wave functions have applications in many disciplines. Our primary motivation for studying them was for computing the solutions to acoustic scattering problems involving prolate spheroids, oblate spheroids, and disks, which we did in Chap. 6. However, they are also encountered in signal processing [88].

We have developed computational software for calculating the spheroidal wave

[1]This section is based on our original work in [87].

functions using C++ and MATLAB. Our software is called `spheroidal` and has many features, including:

1. GNU MPFR, a library for performing arbitrary precision arithmetic [89], is used to achieve greater accuracy in many of the computations performed, especially for higher wavenumbers and modes.

2. The software allows the user to specify in two different ways how many expansion coefficients should be computed. The user can specify the number of expansion coefficients exactly. Alternatively, the user can allow the software to choose the number of expansion coefficients adaptively. All of the expansion coefficients decay exponentially in the long run, so in this method, the user specifies the minimum magnitude that the expansion coefficients should reach (e.g., keep computing expansion coefficients until the next one drops below 10^{-200}).

3. There are several methods for computing the spheroidal radial functions. The actual value of the Wronskian of these functions is easy to compute, so the combination of methods for computing the spheroidal radial functions is chosen so that the computed Wronskian has the smallest error.

4. We have made our software freely available for download from our webpage [86].

The spheroidal wave functions have been studied for over six decades. The most complete description of the spheroidal wave functions is given by Flammer [77]. Another good description of these functions can be found in [90, 91]. Actually implementing the spheroidal wave functions in code is very involved. In [92], they were implemented in

Fortran, and in [93, 94], they were implemented in C. These implementations used double precision. Due to round-off errors, double precision can lead to large errors, especially for higher frequencies and modes. In [95, 96], they were implemented in Fortran using quad precision and expressions that converge faster and more accurately in some cases to achieve better accuracy over a wide range of frequencies, modes, and argument values. These implementations, as well as our own, are only for real frequencies and integer modes. Other authors have investigated complex frequencies and non-integer modes [97, 98], and in these particular references, the respective authors used Mathematica, which can work in arbitrary precision. Some have also looked at numerical techniques, such as finite difference approximations and relaxation methods [99–101]. Many of these authors have released their code free to use.

7.2 Prolate Spheroidal Wave Functions

7.2.1 Angle Functions

The prolate spheroidal angle functions of the first and second kinds can be written in terms of the associated Legendre polynomials of the first and second kinds, respectively:

$$S_{mn}^{(1)}(c,\eta) = \sum_{r=0,1}^{\infty\prime} d_r^{mn}(c)\, P_{m+r}^{m}(\eta)\,, \quad S_{mn}^{(2)}(c,\eta) = \sum_{r=-\infty}^{\infty\prime} d_r^{mn}(c)\, Q_{m+r}^{m}(\eta)\,, \tag{7.1}$$

where the prime over the sum means that only the even terms are included when $n-m=$ even and only the odd terms are included when $n-m=$ odd. The angle functions of the

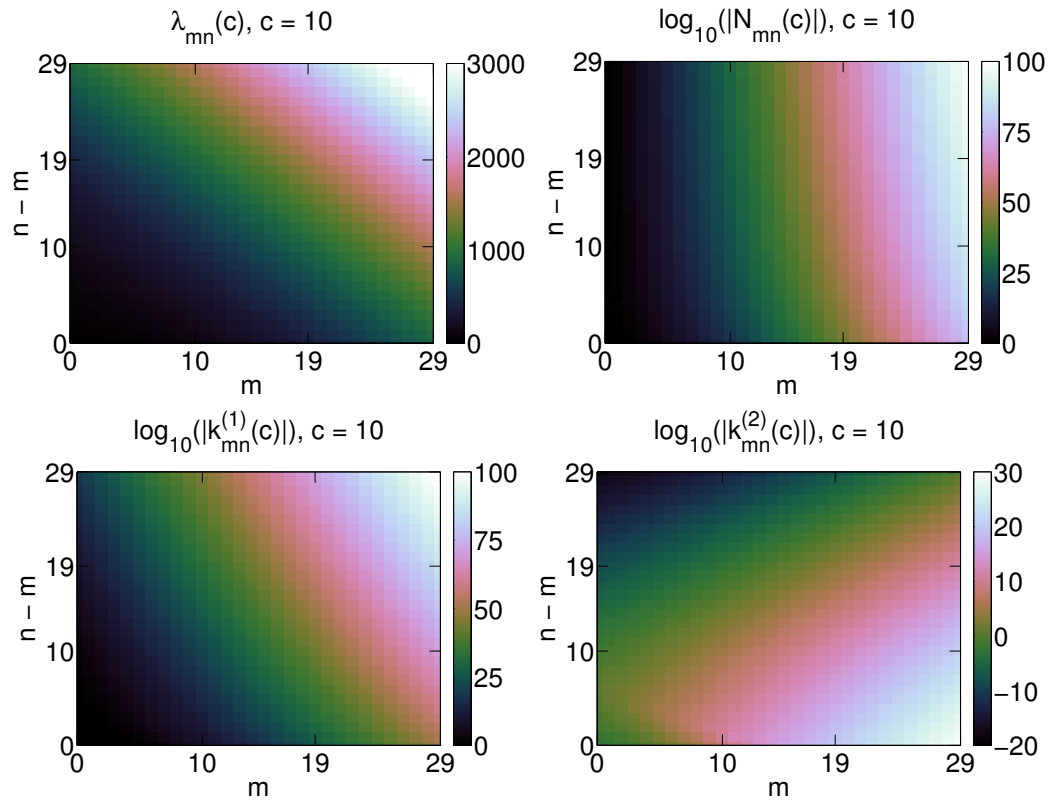

Figure 7.1: Characteristic and other special values for the prolate spheroidal wave functions for $c = 10$, $m = 0, 1, \ldots, 29$, and $n = m, m+1, \ldots, m+29$.

first kind are orthogonal over $[-1, 1]$:

$$\int_{-1}^{1} S_{mn}^{(1)}(c,\eta)\, S_{mn'}^{(1)}(c,\eta)\, d\eta = \delta_{nn'} N_{mn}(c)\,, \tag{7.2}$$

where

$$\delta_{nn'} = \begin{cases} 1, & n = n' \\ 0, & n \neq n' \end{cases}, \quad N_{mn}(c) = 2 \sum_{r=0,1}^{\infty}{}' d_r^{mn}(c)^2 \frac{(2m+r)!}{(2m+2r+1)\, r!}. \tag{7.3}$$

The angle functions of the first kind can also be written as a power series. This can

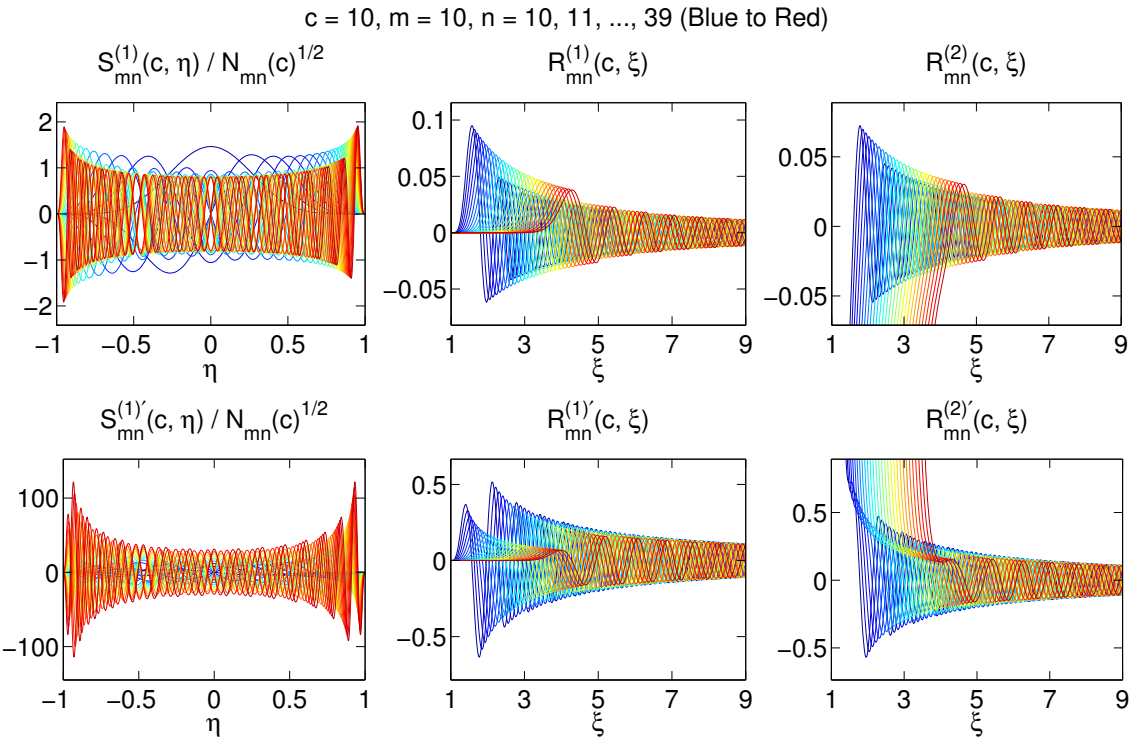

Figure 7.2: The prolate spheroidal wave functions and their derivatives for $c = 10$, $m = 10$, and $n = 10, 11, \ldots, 39$.

be done with the help of the hypergeometric function. When $n - m =$ even,

$$S_{mn}^{(1)}(c,\eta) = \sum_{r=0}^{\infty} d_{2r}^{mn}(c)\, P_{m+2r}^{m}(\eta)\,. \tag{7.4}$$

The associated Legendre polynomials of the first kind can be written as

$$P_n^m(x) = \frac{(-1)^m (n+m)!}{2^m m!\,(n-m)!}\left(1-x^2\right)^{m/2} F\left(m-n, n+m+1; m+1; \frac{1-x}{2}\right), \tag{7.5}$$

where

$$F(\alpha,\beta;\gamma;x) = \sum_{k=0}^{\infty} \frac{(\alpha)_k (\beta)_k}{k!\,(\gamma)_k} x^k \tag{7.6}$$

is the hypergeometric function and $(a)_0 = 1, (a)_k = a(a+1)\ldots(a+k-1)$ is the rising

Pochhammer symbol. Using this, the associated Legendre polynomials of the first kind in Eq. (7.4) can be written as

$$P_{m+2r}^{m}(x)=\frac{(-1)^{m}(2m+2r)!}{2^{m}m!(2r)!}\left(1-x^{2}\right)^{m/2}F\left(-2r,2m+2r+1;m+1;\frac{1-x}{2}\right). \tag{7.7}$$

Recall the following identity for the hypergeometric function:

$$F\left(\alpha,\beta;\frac{\alpha+\beta+1}{2};x\right)=F\left(\frac{\alpha}{2},\frac{\beta}{2};\frac{\alpha+\beta+1}{2};4x(1-x)\right). \tag{7.8}$$

Using this, we can rearrange the previous expression slightly:

$$P_{m+2r}^{m}(x)=\frac{(-1)^{m}(2m+2r)!}{2^{m}m!(2r)!}\left(1-x^{2}\right)^{m/2}F\left(-r,m+r+\frac{1}{2};m+1;1-x^{2}\right), \tag{7.9}$$

$$P_{m+2r}^{m}(x)=\frac{(-1)^{m}(2m+2r)!}{2^{m}m!(2r)!}\left(1-x^{2}\right)^{m/2}\sum_{k=0}^{\infty}\frac{(-r)_{k}(m+r+1/2)_{k}}{k!(m+1)_{k}}\left(1-x^{2}\right)^{k}. \tag{7.10}$$

Plugging this into Eq. (7.4) and rearranging, we have

$$S_{mn}^{(1)}(c,\eta)=(-1)^{m}\left(1-\eta^{2}\right)^{m/2}\sum_{k=0}^{\infty}c_{2k}^{mn}(c)\left(1-\eta^{2}\right)^{k}, \tag{7.11}$$

where

$$c_{2k}^{mn}(c)=\frac{1}{2^{m}(m+k)!k!}\sum_{r=2k}^{\infty}{}'\,d_{r}^{mn}(c)\frac{(2m+r)!}{r!}\left(-\frac{r}{2}\right)_{k}\left(m+\frac{r}{2}+\frac{1}{2}\right)_{k}. \tag{7.12}$$

When $n - m =$ odd,

$$S_{mn}^{(1)}(c,\eta) = \sum_{r=0}^{\infty} d_{2r+1}^{mn}(c) P_{m+2r+1}^{m}(\eta). \tag{7.13}$$

Recall the following identity for the associated Legendre polynomials of the first kind:

$$(n-m+1) P_{n+1}^{m}(x) = (n+1) x P_{n}^{m}(x) - \left(1-x^2\right) P_{n}^{m\prime}(x). \tag{7.14}$$

Setting $n = m + 2r$, plugging in Eq. (7.10), and rearranging,

$$\begin{aligned} P_{m+2r+1}^{m}(x) = \frac{(-1)^m (2m+2r+1)!}{2^m m! (2r+1)!} x \left(1-x^2\right)^{m/2} \times \\ \sum_{k=0}^{\infty} \frac{(-r)_k (m+r+3/2)_k}{k! (m+1)_k} \left(1-x^2\right)^k. \end{aligned} \tag{7.15}$$

Finally, plugging this into Eq. (7.13) and rearranging,

$$S_{mn}^{(1)}(c,\eta) = (-1)^m \eta \left(1-\eta^2\right)^{m/2} \sum_{k=0}^{\infty} c_{2k}^{mn}(c) \left(1-\eta^2\right)^k, \tag{7.16}$$

where

$$c_{2k}^{mn}(c) = \frac{1}{2^m (m+k)! k!} \sum_{r=2k+1}^{\infty}{}' d_r^{mn}(c) \frac{(2m+r)!}{r!} \left(-\frac{r-1}{2}\right)_k \left(m+\frac{r}{2}+1\right)_k. \tag{7.17}$$

7.2.2 Radial Functions

The prolate spheroidal radial functions of the first and second kinds can be written in terms of the spherical Bessel and Neumann functions, respectively:

$$R_{mn}^{(1)}(c,\xi) = \frac{1}{F_{mn}(c)}\left(1-\frac{1}{\xi^2}\right)^{m/2} \times \sum_{r=0,1}^{\infty}{}' (-1)^{(r-(n-m))/2} d_r^{mn}(c) \frac{(2m+r)!}{r!} j_{m+r}(c\xi), \tag{7.18}$$

$$R_{mn}^{(2)}(c,\xi) = \frac{1}{F_{mn}(c)}\left(1-\frac{1}{\xi^2}\right)^{m/2} \times \sum_{r=0,1}^{\infty}{}' (-1)^{(r-(n-m))/2} d_r^{mn}(c) \frac{(2m+r)!}{r!} y_{m+r}(c\xi), \tag{7.19}$$

where

$$F_{mn}(c) = \sum_{r=0,1}^{\infty}{}' d_r^{mn}(c) \frac{(2m+r)!}{r!}. \tag{7.20}$$

The radial functions of the third and fourth kinds are linear combinations of those of the first and second kinds:

$$R_{mn}^{(3)}(c,\xi) = R_{mn}^{(1)}(c,\xi) + iR_{mn}^{(2)}(c,\xi), \quad R_{mn}^{(4)}(c,\xi) = R_{mn}^{(1)}(c,\xi) - iR_{mn}^{(2)}(c,\xi). \tag{7.21}$$

The Wronskian of the radial functions of the first and second kinds is given by

$$W_{mn}(c,\xi) = R_{mn}^{(1)}(c,\xi)\frac{\partial}{\partial\xi}R_{mn}^{(2)}(c,\xi) - \frac{\partial}{\partial\xi}R_{mn}^{(1)}(c,\xi)R_{mn}^{(2)}(c,\xi) = \frac{1}{c(\xi^2-1)} \tag{7.22}$$

and is useful for validating computed values of these functions.

The radial functions are related to the angle functions by

$$S_{mn}^{(1)}(c,z) = k_{mn}^{(1)}(c)\, R_{mn}^{(1)}(c,z), \quad S_{mn}^{(2)}(c,z) = k_{mn}^{(2)}(c)\, R_{mn}^{(2)}(c,z), \tag{7.23}$$

where $k_{mn}^{(1)}(c)$ is given by

$$k_{mn}^{(1)}(c) = \frac{(2m+1)(m+n)!F_{mn}(c)}{2^{m+n}d_0^{mn}(c)\, c^m m! \left(\dfrac{n-m}{2}\right)! \left(\dfrac{m+n}{2}\right)!}, \quad n-m = \text{even}, \tag{7.24}$$

$$k_{mn}^{(1)}(c) = \frac{(2m+3)(m+n+1)!F_{mn}(c)}{2^{m+n}d_1^{mn}(c)\, c^{m+1} m! \left(\dfrac{n-m-1}{2}\right)! \left(\dfrac{m+n+1}{2}\right)!}, \quad n-m = \text{odd}, \tag{7.25}$$

and $k_{mn}^{(2)}(c)$ is given by

$$k_{mn}^{(2)}(c) = \frac{2^{n-m}(2m)! \left(\dfrac{n-m}{2}\right)! \left(\dfrac{m+n}{2}\right)! d_{-2m}^{mn}(c)\, F_{mn}(c)}{(2m-1)\, m!\, (m+n)! c^{m-1}}, \quad n-m = \text{even}, \tag{7.26}$$

$$k_{mn}^{(2)}(c) = -\frac{2^{n-m}(2m)!\left(\frac{n-m-1}{2}\right)!\left(\frac{m+n+1}{2}\right)!d_{-2m+1}^{mn}(c)F_{mn}(c)}{(2m-3)(2m-1)m!(m+n+1)!c^{m-2}},$$

$$n-m = \text{odd}. \tag{7.27}$$

The expression for the radial functions of the second kind using the spherical Neumann functions converges very slowly for values of ξ near 1 and is, therefore, inaccurate in these cases. While the expression for the radial functions of the first kind is accurate for all values of ξ, having a second method can be used as a check on the first method. Thus, these relationships can be used to construct secondary methods for computing these functions. For the radial functions of the first kind,

$$R_{mn}^{(1)}(c,\xi) = k_{mn}^{(1)}(c)^{-1} S_{mn}^{(1)}(c,\xi) = k_{mn}^{(1)}(c)^{-1} \sum_{r=0,1}^{\infty\prime} d_r^{mn}(c) P_{m+r}^{m}(\xi). \tag{7.28}$$

Similar to the angle functions of the first kind, this expression can be written as a power series with the help of the hypergeometric function. Because the argument is $\xi \geq 1$ as opposed to $|\eta| \leq 1$, the relationship between the associated Legendre polynomials of the first kind and the hypergeometric function is slightly different. In particular, there is no factor of $(-1)^m$:

$$P_n^m(x) = \frac{(n+m)!}{2^m m!(n-m)!}\left(x^2-1\right)^{m/2} F\left(m-n, n+m+1; m+1; \frac{1-x}{2}\right). \tag{7.29}$$

Following a procedure similar to the one followed for the angle functions of the first kind, we have

$$R_{mn}^{(1)}(c,\xi) = k_{mn}^{(1)}(c)^{-1}\left(\xi^2-1\right)^{m/2}\sum_{k=0}^{\infty}(-1)^k c_{2k}^{mn}(c)\left(\xi^2-1\right)^k, \quad n-m = \text{even}, \tag{7.30}$$

$$R_{mn}^{(1)}(c,\xi) = k_{mn}^{(1)}(c)^{-1}\xi\left(\xi^2-1\right)^{m/2}\sum_{k=0}^{\infty}(-1)^k c_{2k}^{mn}(c)\left(\xi^2-1\right)^k, \quad n-m = \text{odd}, \tag{7.31}$$

where $c_{2k}^{mn}(c)$ is the same as before. For the radial functions of the second kind,

$$R_{mn}^{(2)}(c,\xi) = k_{mn}^{(2)}(c)^{-1}S_{mn}^{(2)}(c,\xi) = k_{mn}^{(2)}(c)^{-1}\sum_{r=-\infty}^{\infty}{}' d_r^{mn}(c)\,Q_{m+r}^{m}(\xi). \tag{7.32}$$

7.2.3 Calculating the Characteristic Value and Expansion Coefficients

All of the expressions introduced in the previous sections require, either directly or indirectly, the characteristic value, $\lambda_{mn}(c)$, and expansion coefficients, $d_r^{mn}(c)$. Below, we derive expressions for computing them.

There are several methods for computing the characteristic value, but here, we use a combination of two: method (1) involves solving for the eigenvalues of a tridiagonal matrix [102]; and method (2) involves solving for the roots of a transcendental equation [77]. To begin, method (1) is used to compute an approximate value for the characteristic value. Method (2) is then used to compute a more accurate value for the characteristic value using the approximate value computed by method (1) as a starting point. This procedure is

similar to the one used in [98]. In our software, we use double precision for method (1) and arbitrary precision for method (2).

Both methods rely on the following recurrence relation, which can be obtained by plugging Eq. (7.1) into Eq. (6.5):

$$\alpha_r d_{r+2}^{mn}(c) + (\beta_r - \lambda_{mn}(c)) d_r^{mn}(c) + \gamma_r d_{r-2}^{mn}(c) = 0, \tag{7.33}$$

where

$$\alpha_r = \frac{(2m+r+2)(2m+r+1)}{(2m+2r+5)(2m+2r+3)} c^2, \tag{7.34}$$

$$\beta_r = (m+r)(m+r+1) + \frac{2(m+r)(m+r+1) - 2m^2 - 1}{(2m+2r-1)(2m+2r+3)} c^2, \tag{7.35}$$

$$\gamma_r = \frac{r(r-1)}{(2m+2r-3)(2m+2r-1)} c^2. \tag{7.36}$$

For method (1), this recurrence relation is rearranged slightly:

$$\alpha_r d_{r+2}^{mn}(c) + \beta_r d_r^{mn}(c) + \gamma_r d_{r-2}^{mn}(c) = \lambda_{mn}(c) d_r^{mn}(c). \tag{7.37}$$

In matrix form,

$$\begin{bmatrix} \beta_0 & \alpha_0 & & \\ \gamma_2 & \beta_2 & \alpha_2 & \\ & \gamma_4 & \beta_4 & \alpha_4 \\ & & & \ddots \end{bmatrix} \begin{bmatrix} d_0^{mn}(c) \\ d_2^{mn}(c) \\ d_4^{mn}(c) \\ \vdots \end{bmatrix} = \lambda_{mn}(c) \begin{bmatrix} d_0^{mn}(c) \\ d_2^{mn}(c) \\ d_4^{mn}(c) \\ \vdots \end{bmatrix}, \quad n - m = \text{even}, \tag{7.38}$$

$$\begin{bmatrix} \beta_1 & \alpha_1 & & \\ \gamma_3 & \beta_3 & \alpha_3 & \\ & \gamma_5 & \beta_5 & \alpha_5 \\ & & & \ddots \end{bmatrix} \begin{bmatrix} d_1^{mn}(c) \\ d_3^{mn}(c) \\ d_5^{mn}(c) \\ \vdots \end{bmatrix} = \lambda_{mn}(c) \begin{bmatrix} d_1^{mn}(c) \\ d_3^{mn}(c) \\ d_5^{mn}(c) \\ \vdots \end{bmatrix}, \quad n-m = \text{odd}. \tag{7.39}$$

When $n-m =$ even, the eigenvalues are $\lambda_{mn}(c)$ for $n = m, m+2, m+4, \ldots$, and when $n-m =$ odd, the eigenvalues are $\lambda_{mn}(c)$ for $n = m+1, m+3, m+5, \ldots$. Thus, we can compute $\lambda_{mn}(c)$ by plugging these tridiagonal matrices into an eigenvalue solver. In our software, we use the `eig` function in MATLAB.

In method (2), the recurrence relation in Eq. (7.33) is divided through by $d_r^{mn}(c)$, which yields

$$\alpha_r \frac{d_{r+2}^{mn}(c)}{d_r^{mn}(c)} + \beta_r - \lambda_{mn}(c) + \gamma_r \frac{d_{r-2}^{mn}(c)}{d_r^{mn}(c)} = 0. \tag{7.40}$$

Setting

$$N_r^m = -\alpha_{r-2} \frac{d_r^{mn}(c)}{d_{r-2}^{mn}(c)}, \quad \gamma_r^m = \beta_r, \quad \beta_r^m = \gamma_r \alpha_{r-2} \tag{7.41}$$

allows us to write Eq. (7.40) as

$$-N_{r+2}^m + \gamma_r^m - \lambda_{mn}(c) - \frac{\beta_r^m}{N_r^m} = 0. \tag{7.42}$$

Rearranging one way leads to a continued fraction in decreasing r:

$$N_r^m = \gamma_{r-2}^m - \lambda_{mn}(c) - \frac{\beta_{r-2}^m}{\gamma_{r-4}^m - \lambda_{mn}(c) - \dfrac{\beta_{r-4}^m}{\gamma_{r-6}^m - \lambda_{mn}(c) - \cdots}}. \tag{7.43}$$

Rearranging the other way leads to a continued fraction in increasing r:

$$N_r^m = \frac{\beta_r^m}{\gamma_r^m - \lambda_{mn}(c) - \dfrac{\beta_{r+2}^m}{\gamma_{r+2}^m - \lambda_{mn}(c) - \cdots}}. \tag{7.44}$$

The two expression for N_r^m should be equal to each other. Setting $r = n - m + 2$ in Eqs. (7.43) and (7.44),

$$U_1(\lambda_{mn}(c)) = \gamma_{n-m}^m - \lambda_{mn}(c) - \frac{\beta_{n-m}^m}{\gamma_{n-m-2}^m - \lambda_{mn}(c) - \dfrac{\beta_{n-m-2}^m}{\gamma_{n-m-4}^m - \lambda_{mn}(c) - \cdots}}, \tag{7.45}$$

$$U_2(\lambda_{mn}(c)) = -\frac{\beta_{n-m+2}^m}{\gamma_{n-m+2}^m - \lambda_{mn}(c) - \dfrac{\beta_{n-m+4}^m}{\gamma_{n-m+4}^m - \lambda_{mn}(c) - \cdots}}. \tag{7.46}$$

Adding these together yields a transcendental equation in $\lambda_{mn}(c)$:

$$U(\lambda_{mn}(c)) = U_1(\lambda_{mn}(c)) + U_2(\lambda_{mn}(c)) = 0. \tag{7.47}$$

We can compute $\lambda_{mn}(c)$ by solving this transcendental equation. In our software, we use the secant method.

Once $\lambda_{mn}(c)$ is known, Eqs. (7.41) and (7.44) can be used to compute $d_r^{mn}(c)$. The expansion coefficients are unique up to a constant factor, though, so the following

normalization scheme is used. When $n-m=$ even,

$$S_{mn}^{(1)}(c,0)=P_n^m(0), \tag{7.48}$$

$$\sum_{r=0}^{\infty}{}' d_r^{mn}(c)\frac{(-1)^{r/2}(2m+r)!}{2^r\left(\frac{2m+r}{2}\right)!\left(\frac{r}{2}\right)!}=\frac{(-1)^{(n-m)/2}(n+m)!}{2^{n-m}\left(\frac{n+m}{2}\right)!\left(\frac{n-m}{2}\right)!}. \tag{7.49}$$

When $n-m=$ odd,

$$S_{mn}^{(1)\prime}(c,0)=P_n^{m\prime}(0), \tag{7.50}$$

$$\sum_{r=1}^{\infty}{}' d_r^{mn}(c)\frac{(-1)^{(r-1)/2}(2m+r+1)!}{2^r\left(\frac{2m+r+1}{2}\right)!\left(\frac{r-1}{2}\right)!}=\frac{(-1)^{(n-m-1)/2}(n+m+1)!}{2^{n-m}\left(\frac{n+m+1}{2}\right)!\left(\frac{n-m-1}{2}\right)!}. \tag{7.51}$$

To use Eq. (7.32), $d_r^{mn}(c)$ must be computed for negative r as well. To begin, Eq. (7.33) is rewritten as

$$A_{r+2}^m d_{r+2}^{mn}(c)+B_r^m d_r^{mn}(c)+C_{r-2}^m d_{r-2}^{mn}(c)=0, \tag{7.52}$$

where

$$A_r^m=\alpha_{r-2},\quad B_r^m=\beta_r-\lambda_{mn}(c),\quad C_r^m=\gamma_{r+2}. \tag{7.53}$$

Rearranging,

$$\frac{d_r^{mn}(c)}{d_{r+2}^{mn}(c)} = -\frac{A_{r+2}^m}{B_r^m + C_{r-2}^m \dfrac{d_{r-2}^{mn}(c)}{d_r^{mn}(c)}}, \tag{7.54}$$

which can be expanded as a continued fraction in decreasing r:

$$\frac{d_r^{mn}(c)}{d_{r+2}^{mn}(c)} = -\frac{A_{r+2}^m}{B_r^m -} \frac{C_{r-2}^m A_r^m}{B_{r-2}^m -} \frac{C_{r-4}^m A_{r-2}^m}{B_{r-4}^m -} \cdots . \tag{7.55}$$

Because $A_r^m = 0$ when $r = -2m$ or $r = -2m+1$, this continued fraction ends:

$$\frac{d_r^{mn}(c)}{d_{r+2}^{mn}(c)} = -\frac{A_{r+2}^m}{B_r^m -} \frac{C_{r-2}^m A_r^m}{B_{r-2}^m -} \frac{C_{r-4}^m A_{r-2}^m}{B_{r-4}^m -} \cdots \frac{A_{-2m+2}^m}{B_{-2m}^m + C_{-2m-2}^m} \tag{7.56}$$

when $n - m =$ even, and

$$\frac{d_r^{mn}(c)}{d_{r+2}^{mn}(c)} = -\frac{A_{r+2}^m}{B_r^m -} \frac{C_{r-2}^m A_r^m}{B_{r-2}^m -} \frac{C_{r-4}^m A_{r-2}^m}{B_{r-4}^m -} \cdots \frac{A_{-2m+3}^m}{B_{-2m+1}^m + C_{-2m-1}^m} \tag{7.57}$$

when $n - m =$ odd. This also means that $d_r^{mn}(c) \to 0$ when $r \leq -2m-2$ for $n - m =$ even and $r \leq -2m-1$ for $n - m =$ odd. However, $Q_{m+r}^m(\xi) \to \infty$ in these cases, and $d_r^{mn}(c)\, Q_{m+r}^m(\xi) < \infty$:

$$d_r^{mn}(c)\, Q_{m+r}^m(\xi) = d_{r|\epsilon}^{mn}(c)\, P_{-r-m-1}^m(\xi). \tag{7.58}$$

The sum in Eq. (7.32) is, therefore, separated into two pieces:

$$R_{mn}^{(2)}\left(c,\xi\right)={k_{mn}^{(2)}}^{-1}\left(\sum_{r=-\infty}^{-2m-2,-2m-1}{}^{\prime}\;d_{r|\epsilon}^{mn}\left(c\right)P_{-r-m-1}^{m}\left(\xi\right)\;+\;\sum_{r=-2m,-2m+1}^{\infty}{}^{\prime}\;d_{r}^{mn}\left(c\right)Q_{m+r}^{m}\left(\xi\right)\right), \tag{7.59}$$

where $d_{-2m-2|\epsilon}^{mn}\left(c\right)$ and $d_{-2m-1|\epsilon}^{mn}\left(c\right)$ are computed using

$$\frac{d_{-2m-2|\epsilon}^{mn}\left(c\right)}{d_{-2m}^{mn}\left(c\right)}=\frac{c^{2}}{\left(2m-1\right)\left(2m+1\right)}\frac{1}{B_{-2m-2}^{m}-}\frac{C_{-2m-4}^{m}A_{-2m-2}^{m}}{B_{-2m-4}^{m}-}\frac{C_{-2m-6}^{m}A_{-2m-4}^{m}}{B_{-2m-6}^{m}-}\cdots, \tag{7.60}$$

$$\frac{d_{-2m-1|\epsilon}^{mn}\left(c\right)}{d_{-2m+1}^{mn}\left(c\right)}=-\frac{c^{2}}{\left(2m-1\right)\left(2m-3\right)}\frac{1}{B_{-2m-1}^{m}-}\frac{C_{-2m-3}^{m}A_{-2m-1}^{m}}{B_{-2m-3}^{m}-}\frac{C_{-2m-5}^{m}A_{-2m-3}^{m}}{B_{-2m-5}^{m}-}\cdots. \tag{7.61}$$

For $r<-2m-2$ when $n-m=$ even and $r<-2m-1$ when $n-m=$ odd, the remaining expansion coefficients can be computed using

$$\frac{d_{r|\epsilon}^{mn}\left(c\right)}{d_{r+2|\epsilon}^{mn}\left(c\right)}=-\frac{A_{r+2}^{m}}{B_{r}^{m}-}\frac{C_{r-2}^{m}A_{r}^{m}}{B_{r-2}^{m}-}\frac{C_{r-4}^{m}A_{r-2}^{m}}{B_{r-4}^{m}-}\cdots. \tag{7.62}$$

7.3 Oblate Spheroidal Wave Functions

7.3.1 Angle Functions

The expressions for the prolate spheroidal angle functions, including those written in terms of the associated Legendre polynomials as well as those expanded as power series,

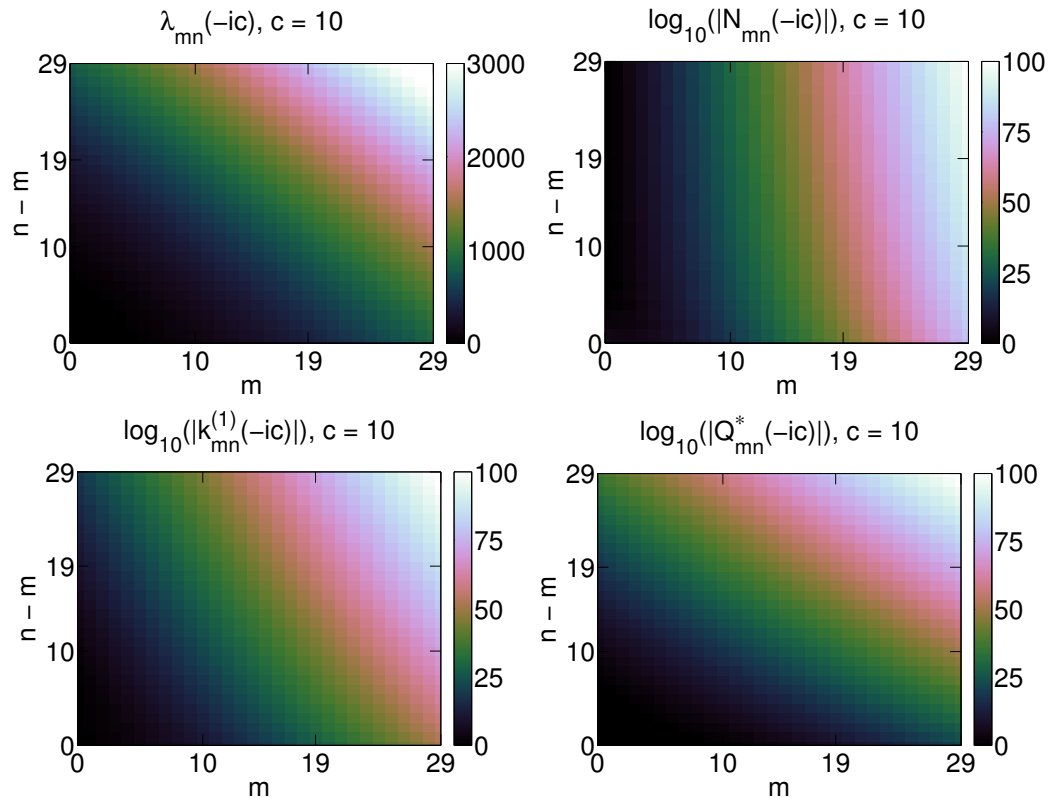

Figure 7.3: Characteristic and other special values for the oblate spheroidal wave functions for $c = 10$, $m = 0, 1, \ldots, 29$, and $n = m, m+1, \ldots, m+29$.

can be transformed into those for the oblate spheroidal angle functions by letting $c \to -ic$.

7.3.2 Radial Functions

The oblate spheroidal radial functions of the first and second kinds can be written in terms of the spherical Bessel and Neumann functions, respectively:

$$R_{mn}^{(1)}\left(-ic, i\xi\right) = \frac{1}{F_{mn}\left(-ic\right)}\left(1+\frac{1}{\xi^2}\right)^{m/2} \times \sum_{r=0,1}^{\infty}{}' \left(-1\right)^{(r-(n-m))/2} d_r^{mn}\left(-ic\right) \frac{(2m+r)!}{r!} j_{m+r}\left(c\xi\right), \tag{7.63}$$

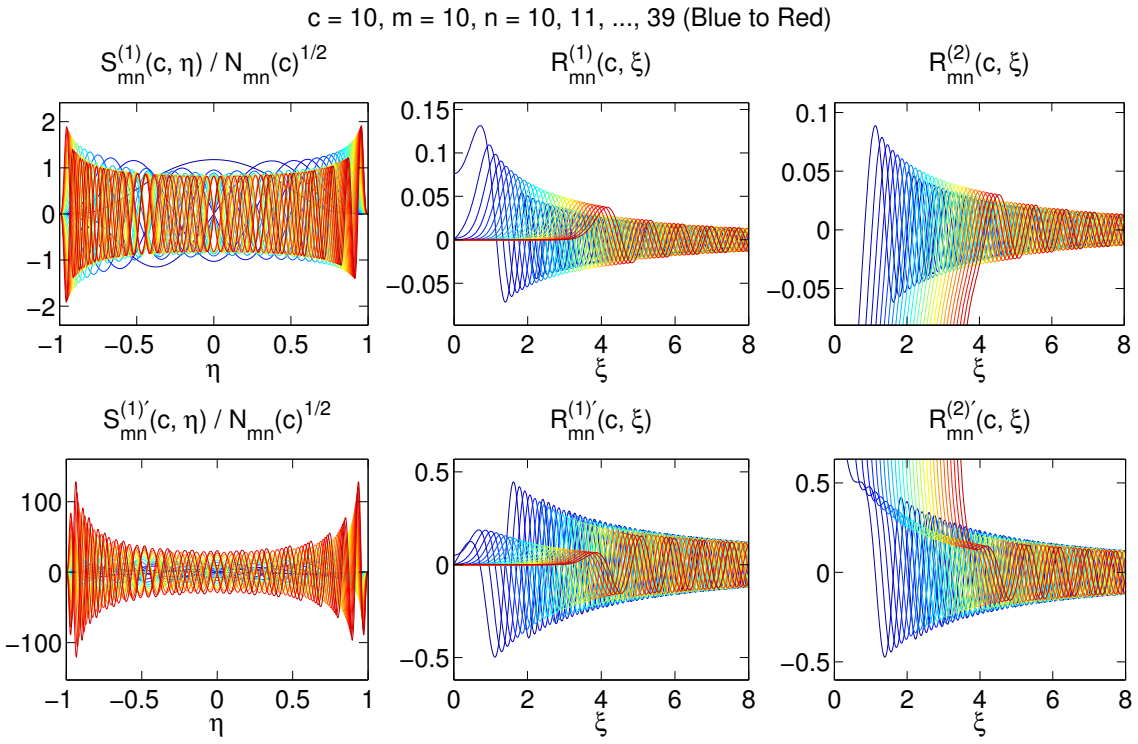

Figure 7.4: The oblate spheroidal wave functions and their derivatives for $c = 10$, $m = 10$, and $n = 10, 11, \ldots, 39$.

$$R_{mn}^{(2)}\left(-ic, i\xi\right) = \frac{1}{F_{mn}\left(-ic\right)}\left(1+\frac{1}{\xi^2}\right)^{m/2} \times \sum_{r=0,1}^{\infty\prime} (-1)^{(r-(n-m))/2}\, d_r^{mn}\left(-ic\right) \frac{(2m+r)!}{r!} y_{m+r}\left(c\xi\right), \tag{7.64}$$

where

$$F_{mn}\left(-ic\right) = \sum_{r=0,1}^{\infty\prime} d_r^{mn}\left(-ic\right) \frac{(2m+r)!}{r!}. \tag{7.65}$$

The radial functions of the third and fourth kinds are linear combinations of those of the first and second kinds:

$$R_{mn}^{(3)}\left(-ic, i\xi\right) = R_{mn}^{(1)}\left(-ic, i\xi\right) + iR_{mn}^{(2)}\left(-ic, i\xi\right), \tag{7.66}$$

$$R_{mn}^{(4)}\left(-ic, i\xi\right) = R_{mn}^{(1)}\left(-ic, i\xi\right) - iR_{mn}^{(2)}\left(-ic, i\xi\right). \tag{7.67}$$

The Wronskian of the radial functions of the first and second kinds is given by

$$\begin{aligned} W_{mn}\left(-ic, i\xi\right) &= R_{mn}^{(1)}\left(-ic, i\xi\right)\frac{\partial}{\partial\xi}R_{mn}^{(2)}\left(-ic, i\xi\right) - \frac{\partial}{\partial\xi}R_{mn}^{(1)}\left(-ic, i\xi\right)R_{mn}^{(2)}\left(-ic, i\xi\right) \\ &= \frac{1}{c\left(\xi^2+1\right)} \end{aligned} \tag{7.68}$$

and is useful for validating computed values of these functions.

The radial functions are related to the angle functions by

$$S_{mn}^{(1)}\left(-ic, iz\right) = k_{mn}^{(1)}\left(-ic\right)R_{mn}^{(1)}\left(-ic, iz\right), \quad S_{mn}^{(2)}\left(-ic, iz\right) = k_{mn}^{(1)}\left(-ic\right)R_{mn}^{(2)}\left(-ic, iz\right), \tag{7.69}$$

where $k_{mn}^{(1)}\left(-ic\right)$ and $k_{mn}^{(2)}\left(-ic\right)$ are given by the same expressions as in the prolate case, provided that $c \to -ic$. The expression for the radial functions of the first kind using the spherical Bessel functions converges and is accurate for all values of ξ, except for $\xi = 0$, where the expression is undefined due to a divide by zero. The expression for the radial functions of the second kind using the spherical Neumann functions converges very slowly for values of ξ near 0 and is, therefore, inaccurate in these cases. Thus, these relationships can be used to construct secondary methods for computing these functions.

The same procedures used in the prolate case for doing so can also be used here, provided that $c, \xi \to -ic, i\xi$ where necessary.

A third method for computing the radial functions of the second kind can also be constructed using a power series:

$$R_{mn}^{(2)}\left(-ic, i\xi\right) = Q_{mn}^{*}\left(-ic\right) R_{mn}^{(1)}\left(-ic, i\xi\right)\left(\arctan\left(\xi\right) - \frac{\pi}{2}\right) + g_{mn}\left(-ic, i\xi\right), \quad (7.70)$$

where

$$Q_{mn}^{*}\left(-ic\right) = \frac{\left(i^{-m} k_{mn}^{(1)}\left(-ic\right)\right)^{2}}{c} \sum_{r=0}^{m} \alpha_{r}^{mn}\left(-ic\right) \frac{(2m-2r)!}{r!\left(2^{m-r}\left(m-r\right)!\right)^{2}}, \quad n-m = \text{even}, \quad (7.71)$$

$$Q_{mn}^{*}\left(-ic\right) = -\frac{\left(i^{-(m+1)} k_{mn}^{(1)}\left(-ic\right)\right)^{2}}{c} \sum_{r=0}^{m} \alpha_{r}^{mn}\left(-ic\right) \frac{(2m-2r+1)!}{r!\left(2^{m-r}\left(m-r\right)!\right)^{2}}, \quad n-m = \text{odd}, \quad (7.72)$$

$$\alpha_{r}^{mn}\left(-ic\right) = \left[\frac{d^{r}}{dx^{r}} \frac{1}{\left(\sum_{k=0}^{\infty} c_{2k}^{mn}\left(-ic\right) x^{k}\right)^{2}}\right]_{x=0}, \quad (7.73)$$

$$g_{mn}\left(-ic, i\xi\right) = \xi\left(\xi^{2}+1\right)^{-m/2} \sum_{r=0}^{\infty} B_{2r}^{mn}\left(-ic\right) \xi^{2r}, \quad n-m = \text{even}, \quad (7.74)$$

$$g_{mn}\left(-ic, i\xi\right) = \left(\xi^{2}+1\right)^{-m/2} \sum_{r=0}^{\infty} B_{2r}^{mn}\left(-ic\right) \xi^{2r}, \quad n-m = \text{odd}. \quad (7.75)$$

To compute $\alpha_{r}^{mn}\left(-ic\right)$, we use the following procedure, which was described in [92] and uses some properties of Cauchy products. Let $C_{k} = c_{2k}^{mn}\left(-ic\right)$, and expand the

denominator in Eq. (7.73) as

$$\left(\sum_{k=0}^{\infty} c_{2k}^{mn}\left(-ic\right)x^{k}\right)^{2} = \sum_{n=0}^{\infty}\sum_{k=0}^{n} C_k x^k C_{n-k} x^{n-k} = \sum_{n=0}^{\infty} B_n x^n, \tag{7.76}$$

$$B_n = \sum_{k=0}^{n} C_k C_{n-k}, \tag{7.77}$$

$$\frac{1}{\left(\sum_{k=0}^{\infty} c_{2k}^{mn}\left(-ic\right)x^{k}\right)^{2}} = \frac{1}{\sum_{n=0}^{\infty} B_n x^n} = \sum_{n=0}^{\infty} A_n x^n, \tag{7.78}$$

$$\sum_{n=0}^{\infty} A_n x^n \sum_{n=0}^{\infty} B_n x^n = 1, \tag{7.79}$$

$$\sum_{n=0}^{\infty}\left(\sum_{k=0}^{n} A_k B_{n-k}\right)x^n = 1. \tag{7.80}$$

In order for this equality to hold,

$$A_0 B_0 = 1, \quad \sum_{k=0}^{n} A_k B_{n-k} = 0, \quad n > 0. \tag{7.81}$$

Rearranging,

$$A_0 = \frac{1}{B_0}, \quad A_n = -\frac{1}{B_0}\sum_{k=0}^{n-1} A_k B_{n-k}, \quad n > 0. \tag{7.82}$$

We can now compute $\alpha_r^{mn}\left(-ic\right)$:

$$\alpha_r^{mn}\left(-ic\right) = \left[\frac{d^r}{dx^r}\sum_{n=0}^{\infty} A_n x^n\right]_{x=0} = A_r r!. \tag{7.83}$$

Plugging Eq. (7.70) into the oblate version of Eq. (6.6) yields the following recurrence

relation in $B_{2r}^{mn}(-ic)$:

$$\alpha_{2r}B_{2r+2}^{mn}(-ic)+\beta_{2r}B_{2r}^{mn}(-ic)+\gamma_{2r}B_{2r-2}^{mn}(-ic)=h_{2r}, \tag{7.84}$$

where

$$\alpha_{2r}=(2r+2)(2r+3), \tag{7.85}$$

$$\beta_{2r}=(2r+1)(2r-2m+2)+m(m-1)-\lambda_{mn}(-ic), \tag{7.86}$$

$$\gamma_{2r}=c^2, \tag{7.87}$$

$$\begin{gathered}h_{2r}=-2Q_{mn}^{*}(-ic)\left(i^{-m}k_{mn}^{(1)}(-ic)\right)^{-1}\times\\ \sum_{k=r-m+1}^{\infty}c_{2k}^{mn}(-ic)(m+2k)\frac{(m+k-1)!}{(m+k-1-r)!r!}\end{gathered} \tag{7.88}$$

when $n-m=$ even, and

$$\alpha_{2r}=(2r+1)(2r+2), \tag{7.89}$$

$$\beta_{2r}=2r(2r-2m+1)+m(m-1)-\lambda_{mn}(-ic), \tag{7.90}$$

$$\gamma_{2r}=c^2, \tag{7.91}$$

$$\begin{gathered}h_{2r}=-2Q_{mn}^{*}(-ic)\left(i^{-(m+1)}k_{mn}^{(1)}(-ic)\right)^{-1}\times\\ \left(\sum_{k=r-m}^{\infty}c_{2k}^{mn}(-ic)(m+2k+1)\frac{(m+k)!}{(m+k-r)!r!}-\right.\\ \left.\sum_{k=r-m+1}^{\infty}c_{2k}^{mn}(-ic)(m+2k)\frac{(m+k-1)!}{(m+k-1-r)!r!}\right)\end{gathered} \tag{7.92}$$

when $n-m=$ odd. Given a starting value, $B_0^{mn}(-ic)$, Eq. (7.84) can be used to compute $B_{2r}^{mn}(-ic)$. The starting values are

$$B_0^{mn}(-ic) = \left(cR_{mn}^{(1)}(-ic,i0)\right)^{-1} - Q_{mn}^{*}(-ic)\,R_{mn}^{(1)}(-ic,i0)\,, \quad n-m=\text{even}, \tag{7.93}$$

$$B_0^{mn}(-ic) = -\left(cR_{mn}^{(1)\prime}(-ic,i0)\right)^{-1}, \quad n-m=\text{odd}. \tag{7.94}$$

7.3.3 Calculating the Characteristic Value and Expansion Coefficients

The procedures for computing the characteristic value and expansion coefficients for the oblate case are exactly the same as those for the prolate case, provided that $c \to -ic$.

7.4 Implementation Details

7.4.1 Using the MPFR Library

Our code uses the GNU MPFR library, which provides interfaces and routines for performing arbitrary precision arithmetic [89]. We created a C++ class, `real`, which encapsulates many of the features provided by GNU MPFR. These features include: basic arithmetic by overloading the +, −, ⋆, and / operators; comparisons by overloading the >, >=, <, <=, ==, and != operators; and some elementary functions, including `abs`, `atan`, `cos`, `log`, `pow`, and `sin`. GNU MPFR allows the programmer to specify the precision to use for these operations by setting the number of bits of precision. As a reference, single and double precision arithmetic found in most programming languages have 24 and 53 bits

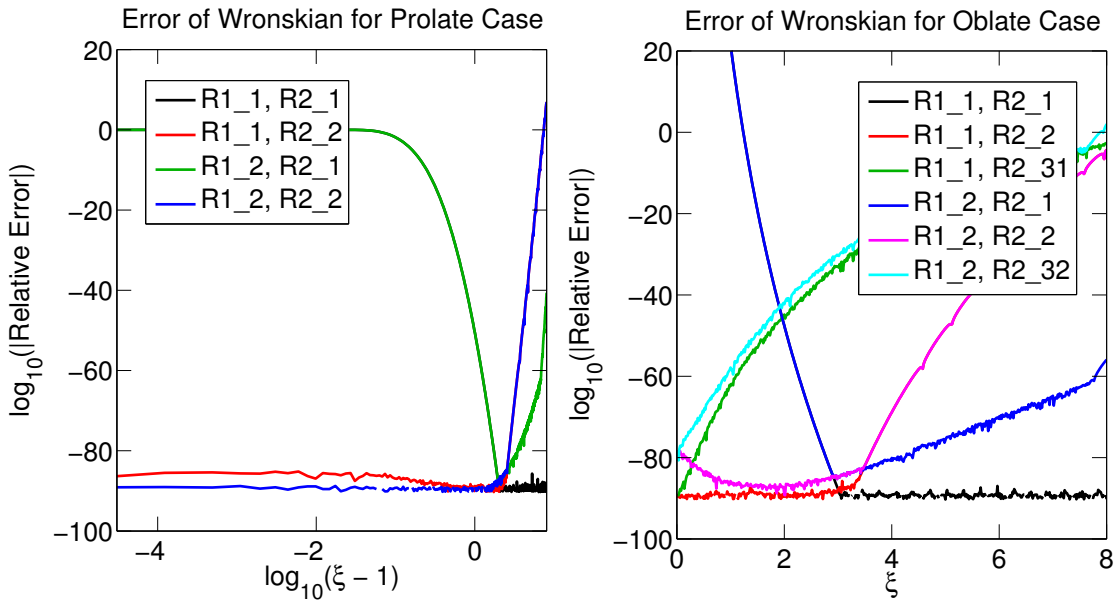

Figure 7.5: The relative error of the computed Wronskian when using different combinations of the methods for computing the prolate (left) and oblate (right) spheroidal radial functions for $c = 10$, $m = 10$, and $n = 39$.

of precision, respectively. We experimented with several different levels of precision (as low as 24 and as high as 5000 bits of precision). In general, as we increased the precision, the accuracy of the computations increased. For lower c, m, and n, single and double precision were good enough. However, for higher values, using such low precision yielded very large errors, and only by increasing the precision were these errors reduced.

7.4.2 Using the Wronskian

Two methods were given for computing the prolate spheroidal radial functions of the first kind. One method uses the spherical Bessel functions, and the other method uses a power series. Call these methods `R1_1` and `R1_2`, respectively. Likewise, two methods were given for computing the prolate spheroidal radial functions of the second kind. One method uses the spherical Neumann functions, and the other method uses the

associated Legendre polynomials of the second kind. Call these methods `R2_1` and `R2_2`, respectively. In each of these pairs of methods, one is better for certain values of ξ than the other, and vice versa. In particular, `R1_1` is better for larger ξ and `R1_2` is better for smaller ξ. Similarly, `R2_1` is better for larger ξ and `R2_2` is better for smaller ξ. However, when to use which method is not always clear: the exact value of ξ below which one is better and above which the other is better is different for different values of c, m, and n. We solved this dilemma in the following manner. For a given ξ, all four methods are used to compute their respective functions, and the combination that yields the smallest error in the computed Wronskian is used. An example of this can be seen in Figure 7.5. The combination, `R1_2, R2_2`, was superior for smaller ξ and the combination, `R1_1, R2_1`, was superior for larger ξ.

This procedure is also used for the oblate spheroidal radial functions. There are two methods for computing the oblate spheroidal radial functions of the first kind, one that uses the spherical Bessel functions and one that uses a power series. Call these methods `R1_1` and `R1_2`, respectively. There are three methods for computing the oblate spheroidal radial functions of the second kind, one that uses the spherical Neumann functions, one that uses the associated Legendre polynomials of the second kind, and one that uses a power series. Call these methods `R2_1`, `R2_2`, and `R2_3`, respectively. Internally, `R2_3` uses the oblate spheroidal radial functions of the first kind, so when `R1_1` is used, call this method `R2_31`, and when `R1_2` is used, call this method `R2_32`. Thus, there are eight possible combinations, but only six are considered: `R1_1` and `R2_1`; `R1_1` and `R2_2`; `R1_1` and `R2_31`; `R1_2` and `R2_1`; `R1_2` and `R2_2`; and `R1_2` and `R2_32`. For a given ξ, of these six, the one that yields the smallest error in the computed Wronskian is

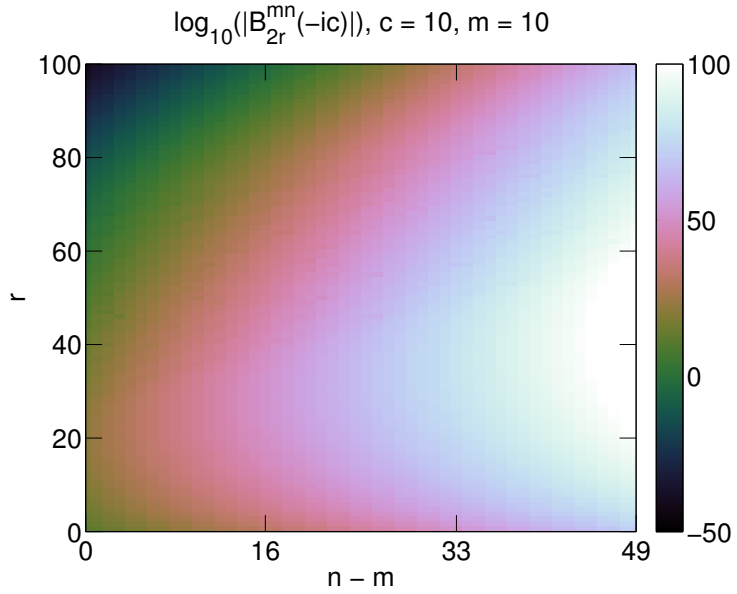

Figure 7.6: The order of magnitude of the expansion coefficients, $B_{2r}^{mn}\left(-ic\right)$, for $c = 25$, $m = 49$, and $n = 49, 50, \ldots, 98$.

used. An example of this can be seen in Figure 7.5. The combination, `R1_1, R2_2`, was superior for smaller ξ and the combination, `R1_1, R2_1`, was superior for larger ξ.

7.4.3 Solving Forward and Backward Recurrences

Several recurrence relations need to be solved in order to compute the characteristic and other special values, expansion coefficients, and special functions required by many of the expressions for the spheroidal wave functions. These recurrence relations can either have a starting value (e.g., for $B_{2r}^{mn}\left(-ic\right)$) or some kind of normalization scheme (e.g., for $d_r^{mn}\left(ic\right)$). Except for one case, all of the recurrence relations encountered in the previous sections were homogeneous. Depending on whether the solution to a particular recurrence relation grows or decays, the method of solution is different. For solutions that grow (e.g., the spherical Neumann functions), the forward recurrence approach is used (i.e., the recurrence relation is used directly to compute succeeding values). For solutions that decay

(e.g., $d_r^{mn}(ic)$), the forward recurrence approach is numerically unstable. The continued fraction approach, which is described in [77], is used instead.

Nonhomogeneous recurrence relations are more complicated. For solutions that grow, the forward recurrence approach can still be used. However, for solutions that decay, the continued fraction approach no longer works. Instead, we use the tridiagonal matrix method described in [103, 104]: the recurrence relation is written for each index, these are combined into a tridiagonal system of equations, and this system is inverted. The case of computing the expansion coefficients, $B_{2r}^{mn}(-ic)$, is further complicated by the fact that, for many values of c, m, and n, $B_{2r}^{mn}(-ic)$ grows for lower r, but decays for higher r. See, for example, Figure 7.6. Thus, the forward recurrence approach is used when $B_{2r}^{mn}(-ic)$ is growing, and the tridiagonal matrix method is used when $B_{2r}^{mn}(-ic)$ is decaying.

7.5 Conclusion

The spheroidal wave functions are among the most complicated special functions around. However, because the solutions to so many interesting problems require them, software for computing them accurately is needed. We have developed computational software for doing so using C++, MATLAB, and GNU MPFR, a library for performing arbitrary precision arithmetic. This chapter described the prolate and oblate spheroidal coordinate systems and wave functions, methods for deriving analytical expressions for computing them, and our software that implements these expressions. Our software includes many novel features. Some of these features include using arbitrary precision arithmetic and using the Wronskian to choose from several different methods for computing

the spheroidal radial functions to improve their accuracy. We have made our software freely available from our webpage.

Chapter 8: Conclusion

The Laplace and Helmholtz equations in three dimensions are two of the most important, perhaps the most important, partial differential equations (PDEs) in science, and govern problems in a large number of disciplines, including electromagnetism, acoustics, astrophysics, molecular dynamics, and aerodynamics, among others. The boundary element method (BEM) is a powerful method for solving these PDEs. The BEM reduces the dimensionality of the problem by one, allows for the treatment of complex boundary shapes, and treats thin surfaces and multi-domain problems well. Unfortunately, the BEM also suffers from a few problems. The entries in the system matrices require the computation of certain boundary integrals, which can be difficult to do accurately, especially in the Galerkin formulation. These matrices are also dense, and when solved conventionally via direct matrix decompositions, require O (N^2) storage and run in O (N^3), where N is the number of discretization unknowns. This can effectively restrict the size of a problem. This dissertation addressed these issues by making three contributions.

First, we presented novel methods inspired by the fast multipole method (FMM) for computing all the boundary integrals that arise in the Galerkin formulation to any accuracy. Integrals involving completely geometrically separated triangles are non-singular, and are computed using a technique based on spherical harmonics and multipole expansions

and translations, which require the integration of polynomial functions over the triangles. Integrals involving cases where the triangles have common vertices, edges, or are coincident are treated via scaling and symmetry arguments, combined with automatic recursive geometric decomposition of the integrals. We have parallelized our code using C++ and OpenMP, and also ported the C++ code to run on the GPU using CUDA. The GPU version of the code, which ran on a NVIDIA Tesla K20c, ran $6\times$ faster than the CPU version of the code, which ran on a Quad Intel Xeon Dual E5-2690 (32 cores).

Second, we used the FMM to accelerate the BEM. The FMM is usually designed around monopole and dipole sources, not the integral expressions in the BEM. To apply the FMM to these expressions, the internal logic of the FMM must be changed, but this can be difficult to do. To overcome this difficulty, we presented the correction factor matrix method, which works by approximating the integrals using a quadrature. The quadrature points are treated as monopole and dipole sources, which are plugged directly into current FMM codes. Any inaccuracies from the quadrature are corrected during a correction factor step. We carefully explored all of the sources of error in the method, identifying the range of values that all of the available parameters in the method should have to ensure the best accuracy. In the example problem for the Laplace equation, the errors were reduced to well below 0.01%, and in the example problem for the Helmholtz equation, the errors were reduced to around 1%. The correction factor matrix method and the use of the FMM reduced the quadratic and cubic scalings of the unaccelerated BEM to linear, which allowed for very large problems. In fact, in the example problem for the Laplace equation, a mesh with over one million boundary elements took only minutes to solve on a single workstation.

Third, we developed computational software for calculating the solutions to acoustic scattering problems involving spheroids and disks. We used this software to verify the accuracy of the BEM for the Helmholtz equation. The software uses spheroidal wave functions to analytically build the solutions to these problems. However, the spheroidal wave functions are notoriously difficult to compute. Because of this, practically no programming language comes equipped with the means to compute them. This makes problems that require their use hard to tackle. For this reason, we had to also develop computational software for computing the spheroidal wave functions. This software has many features, including: using arbitrary precision arithmetic; adaptively choosing the number of expansion coefficients to compute and use; and using the Wronskian to choose from several different methods for computing the spheroidal radial functions to improve their accuracy.

The product of these three contributions was a fast and accurate BEM solver for the Laplace and Helmholtz equations. Where possible, the companion code for this dissertation has been released online and is available for download.

There are a number of interesting problems that remain open and could possibly be the topic of future work. In the example problems in Chap. 5, the boundaries were simple shapes. In the case of the Laplace equation, the boundary was a cube, and in the case of the Helmholtz equation, the boundary was a disk. Moreover, the discretizations of these boundaries were "good", meaning the triangles in the mesh were all acute or right-angled, and they were all roughly the same size. A real-world problem with a more complicated geometry may not allow for such a "good" mesh. For example, a problem may be composed of more complicated shapes that have sharp corners, holes, or many

different levels of scale, where some features are very big and some features are very small, perhaps in close proximity to each other. These kinds of geometries would be more difficult to discretize, requiring triangles of many different sizes, some of which could be very obtuse. The effect of these "bad" meshes on the different methods presented in this dissertation remains an open question, and would require additional work to answer.

The use of these "bad" meshes usually results in linear systems that are poorly conditioned. This primarily affects the performance of the iterative solver. Since the convergence rates for iterative solvers largely depend on the condition number of the linear system, this can greatly affect how quickly a solution is reached. In particular, higher condition numbers require more iterations, and lower condition numbers require fewer. Since more iterations take longer, keeping the iteration counts a low as possible is important. The number of iterations can be reduced by using a preconditioner. A preconditioner works by reducing the condition number of the linear system. Instead of solving $\mathbf{Ax} = \mathbf{b}$, we solve $\mathrm{P}^{-1}\mathbf{Ax} = \mathrm{P}^{-1}\mathbf{b}$. By carefully choosing P, the new system matrix, $\mathrm{P}^{-1}\mathrm{A}$, has a lower condition number, thus reducing the number of iterations. Exploring different preconditioners could be an interesting topic of future work. Even for "good" meshes, this could be helpful. For the example problems in Chap. 5, the iteration counts were between 50 and 100 for the largest mesh sizes. Reducing these numbers would provide a significant speedup.

To get an idea of how more complicated geometries might affect the methods presented in this dissertation, we modeled the electric-field cage described in [2]. The electric-field cage, shown in Fig. 8.1 on the left, is a large parallel plate capacitor with "guard tubes" spaced evenly between the plates. This generates a uniform electric field

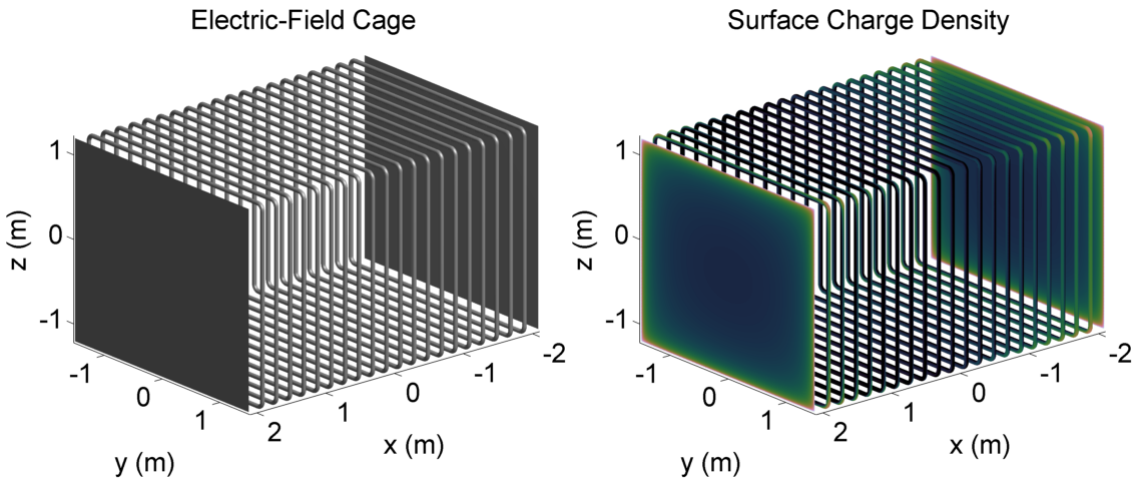

Figure 8.1: On the left: a rendering of the electric-field cage. On the right: the charge density over the surface of the electric-field cage.

between the plates, which is used for characterizing and calibrating electric-field sensors. We modeled the cage in a bipolar configuration, where the $+x$ plate is energized to 2.1 V and the $-x$ plate is energized to -2.1 V. The "guard tubes" are energized to voltages that enforce a linear gradient field. The resulting surface charge density can be seen in Fig. 8.1 on the right. A close-up of the mesh and this surface charge density at the corner of the cage can be seen in Fig. 8.2. While the geometry is much more complicated in this example, including multiple disconnected boundaries, some of which have holes, the mesh is actually very good. The triangles are all acute or right-angled, and they are all roughly the same size. The mesh we used had 1,109,680 triangles. The constant collocation, constant Galerkin, and linear Galerkin took 14.9, 18.8 and 33.7 minutes to solve, respectively, and required 69, 82, and 154 iterations to converge, respectively. In this example, the performance of the iterative solver comparable to the simpler examples in Chap. 5.

There are a number of interesting topics of future work related to the BEM for

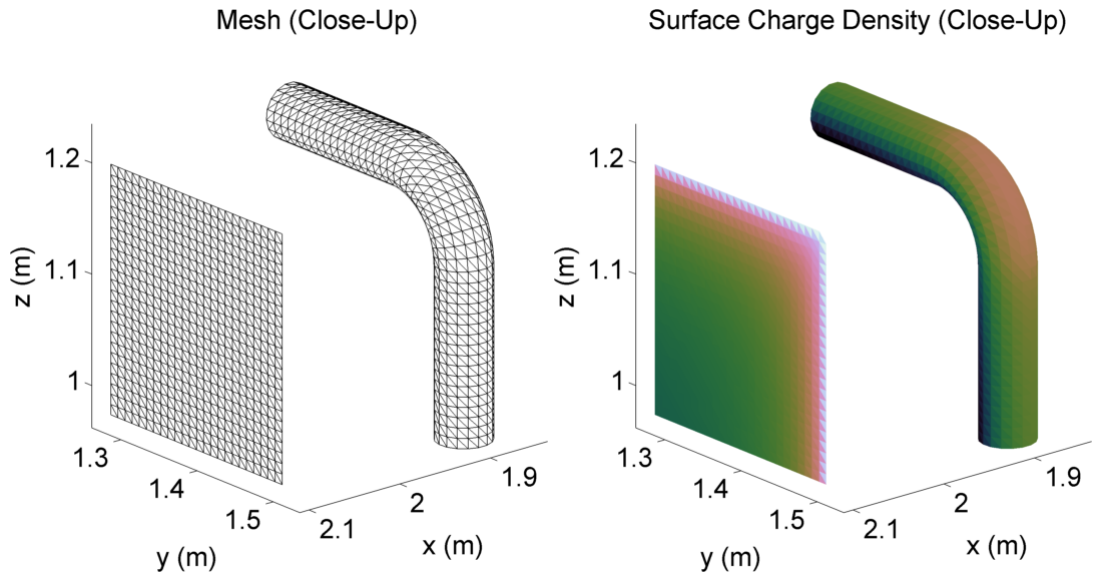

Figure 8.2: On the left: a close-up of the mesh of the electric-field cage. On the right: a close-up of the charge density over the surface of the electric-field cage.

the Helmholtz equation. Currently, the BEM for the Helmholtz equation only supports constant collocation. While the accuracy was decent for this method, extending the BEM to support constant and linear Galerkin would likely reduce them to better levels. This would require computing double surface integrals for the Green's function for the Helmholtz equation. The singularity subtraction method used for the single surface integrals in the collocation method could be used here again. The subdivision and scaling method from Chap. 3 would be used for the singular part, and Gaussian quadrature would be used for the regular part. Another interesting topic is solving multi-domain problems for the Helmholtz equation. The indirect BEM allows for some multi-domain problems, but not all. In particular, the indirect BEM is unable to account for changes in the wavenumber across boundaries because the Green's function is different on one side on the boundary than the other. This kind of problem appears when solving a transmission problem in acoustics or electromagnetics, where a wave strikes a surface, and the problem is to compute how much

of that wave passes through the surface and into the material. One way to work around this problem is to solve two problems separately, one for one side of the boundary and another for the other, and then couple the problems together. The BEM can be used to solve both of these problems, or another method, including volumetric methods, could be used to solve one of the problems. Implementing these multi-domain methods would be an interesting line of future work as well.

Appendix A: Contour Integration

A.1 Constant and Linear Planar Polygonal Elements

Implementing the BEM for the Laplace equation requires a method for computing the single- and double-layer potentials and gradients due to a linear source density distribution over a triangular element:

$$L(\mathbf{y}) = \int_{\mathbf{x}\in\mathrm{T}} (\sigma_0 + \mathbf{p}\cdot(\mathbf{x}-\mathbf{x}_1))\, G(\mathbf{x},\mathbf{y})\, dS(\mathbf{x}), \tag{A.1}$$

$$M(\mathbf{y}) = \int_{\mathbf{x}\in\mathrm{T}} (\sigma_0 + \mathbf{p}\cdot(\mathbf{x}-\mathbf{x}_1))\, (\mathbf{n}\cdot\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y}))\, dS(\mathbf{x}), \tag{A.2}$$

$$\nabla_{\mathbf{y}} L(\mathbf{y}) = \nabla_{\mathbf{y}} \int_{\mathbf{x}\in\mathrm{T}} (\sigma_0 + \mathbf{p}\cdot(\mathbf{x}-\mathbf{x}_1))\, G(\mathbf{x},\mathbf{y})\, dS(\mathbf{x}), \tag{A.3}$$

$$\nabla_{\mathbf{y}} M(\mathbf{y}) = \nabla_{\mathbf{y}} \int_{\mathbf{x}\in\mathrm{T}} (\sigma_0 + \mathbf{p}\cdot(\mathbf{x}-\mathbf{x}_1))\, (\mathbf{n}\cdot\nabla_{\mathbf{x}} G(\mathbf{x},\mathbf{y}))\, dS(\mathbf{x}), \tag{A.4}$$

where the three vertices of the triangle, T, are $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$. Note that, in this appendix, the meaning of σ_0 has changed very slightly compared to the other chapters. Here, σ_0 is the source density at $\mathbf{x}_1$, not the origin. In this appendix, a method is derived that transforms the surface integrals over the triangle into line integrals around the contour of the triangle. As a result, this method is called the contour integration method. The method is capable of

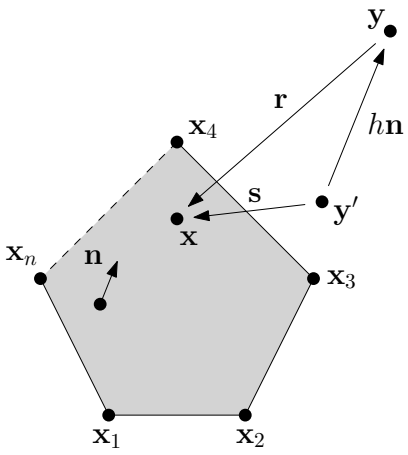

Figure A.1: The planar polygonal element.

treating not only triangular elements, but planar polygonal elements as well.

The planar polygonal element, shown in Fig. A.1, has n vertices: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. The evaluation point, $\mathbf{y}$, is projected onto the plane that contains this element:

$$\mathbf{y} = \mathbf{y}' + h\mathbf{n}. \tag{A.5}$$

Assume $h \geq 0$, meaning the evaluation point is on the positive side of the element. When this is not the case, the vertices of the element can be reversed. The displacement vector is given by

$$\mathbf{r} = \mathbf{x} - \mathbf{y}, \tag{A.6}$$

$$r = |r| = \left(s^2 + h^2\right)^{1/2}, \tag{A.7}$$

where

$$s = |\mathbf{s}| = |\mathbf{x} - \mathbf{y}'|. \tag{A.8}$$

The source density distribution over the polygonal element is

$$\sigma(\mathbf{x}) = \sigma_0 + \mathbf{p} \cdot (\mathbf{x} - \mathbf{x}_1) . \tag{A.9}$$

In particular, the source density at the projected evaluation point, $\mathbf{y}'$, is

$$\sigma_1 = \sigma_0 + \mathbf{p} \cdot (\mathbf{y}' - \mathbf{x}_1) . \tag{A.10}$$

A.2 Single-Layer Potential

A.2.1 Line Integrals

The single-layer potential is given by

$$L(\mathbf{y}) = \int_{\mathrm{S}} (\sigma_1 + \mathbf{p} \cdot \mathbf{s}) \, G(\mathbf{x}, \mathbf{y}) \, dS(\mathbf{x}) , \tag{A.11}$$

where

$$G(\mathbf{x}, \mathbf{y}) = G(\mathbf{x} - \mathbf{y}) = G(\mathbf{r}) = G(|\mathbf{r}|) = G(r) \tag{A.12}$$

is the Green's function. Using the divergence theorem,

$$L(\mathbf{y}) = \int_{\mathrm{S}} \nabla_{\mathbf{x}} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}) \, dS(\mathbf{x}) = \int_{\mathrm{C}} \mathbf{n}' \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}) \, dl(\mathbf{x}) . \tag{A.13}$$

We need to pick an $\mathbf{F}$ such that

$$\nabla_{\mathbf{x}} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}) = (\sigma_1 + \mathbf{p} \cdot \mathbf{s}) G(\mathbf{x}, \mathbf{y}). \tag{A.14}$$

Before doing that, though, let's replace the arguments of G and $\mathbf{F}$:

$$\nabla_{\mathbf{x}} \cdot \mathbf{F}(\mathbf{s}; h) = (\sigma_1 + \mathbf{p} \cdot \mathbf{s}) G\left(\left(s^2 + h^2\right)^{1/2}\right). \tag{A.15}$$

Try

$$\mathbf{F} = \sigma_1 f \mathbf{s} + g\mathbf{p}, \quad f = f(s; h), \quad g = g(s; h). \tag{A.16}$$

Plugging these into Eq. (A.14), we have

$$\nabla_{\mathbf{x}} \cdot (\sigma_1 f \mathbf{s} + g\mathbf{p}) = (\sigma_1 + \mathbf{p} \cdot \mathbf{s}) G\left(\left(s^2 + h^2\right)^{1/2}\right), \tag{A.17}$$

$$\nabla_{\mathbf{x}} \cdot (\sigma_1 f \mathbf{s}) + \nabla_{\mathbf{x}} \cdot (g\mathbf{p}) = \sigma_1 G\left(\left(s^2 + h^2\right)^{1/2}\right) + (\mathbf{p} \cdot \mathbf{s}) G\left(\left(s^2 + h^2\right)^{1/2}\right). \tag{A.18}$$

There's a lot of freedom here in choosing f and g. In fact, we can divide this equation into two, one for f and one for g:

$$\nabla_{\mathbf{x}} \cdot (\sigma_1 f \mathbf{s}) = \sigma_1 G\left(\left(s^2 + h^2\right)^{1/2}\right), \tag{A.19}$$

$$\nabla_{\mathbf{x}} \cdot (g\mathbf{p}) = (\mathbf{p} \cdot \mathbf{s}) G\left(\left(s^2 + h^2\right)^{1/2}\right). \tag{A.20}$$

The solutions to these will solve the original. First, let's solve for f:

$$\nabla_{\mathbf{x}} \cdot (\sigma_1 f \mathbf{s}) = \sigma_1 G\left(\left(s^2+h^2\right)^{1/2}\right), \tag{A.21}$$

$$\nabla_{\mathbf{x}} f \cdot \mathbf{s} + f \nabla_{\mathbf{x}} \cdot \mathbf{s} = G\left(\left(s^2+h^2\right)^{1/2}\right). \tag{A.22}$$

Since f is a function of only s and $\nabla_{\mathbf{x}} \cdot \mathbf{s} = 2$,

$$s\frac{df}{ds} + 2f = G\left(\left(s^2+h^2\right)^{1/2}\right). \tag{A.23}$$

Suppose f is of the form,

$$f = \frac{\hat{f}}{s^2}, \tag{A.24}$$

$$\frac{df}{ds} = \frac{d\hat{f}}{ds}s^{-2} - 2\hat{f}s^{-3}, \tag{A.25}$$

$$s\left(\frac{d\hat{f}}{ds}s^{-2} - 2\hat{f}s^{-3}\right) + 2\frac{\hat{f}}{s^2} = G\left(\left(s^2+h^2\right)^{1/2}\right), \tag{A.26}$$

$$\frac{1}{s}\frac{d\hat{f}}{ds} = G\left(\left(s^2+h^2\right)^{1/2}\right), \tag{A.27}$$

$$\hat{f} = \int sG\left(\left(s^2+h^2\right)^{1/2}\right) ds, \tag{A.28}$$

$$f = \frac{1}{s^2}\int sG\left(\left(s^2+h^2\right)^{1/2}\right) ds. \tag{A.29}$$

Make a change of variables:

$$t = \left(s^2 + h^2\right)^{1/2}, \tag{A.30}$$

$$f = \frac{1}{s^2} \int tG\left(t\right) dt. \tag{A.31}$$

We are free to choose the limits of integration. In order to eliminate a possible singularity when $s = 0$, but $h \neq 0$,

$$f = \frac{1}{s^2} \int_h^r tG\left(t\right) dt. \tag{A.32}$$

Second, let's solve for g:

$$\nabla_{\mathbf{x}} \cdot \left(g\mathbf{p}\right) = \left(\mathbf{p} \cdot \mathbf{s}\right) G\left(\left(s^2 + h^2\right)^{1/2}\right), \tag{A.33}$$

$$\nabla_{\mathbf{x}} g \cdot \mathbf{p} = \left(\mathbf{p} \cdot \mathbf{s}\right) G\left(\left(s^2 + h^2\right)^{1/2}\right). \tag{A.34}$$

Since g is a function of only s,

$$\left(\hat{\mathbf{s}} \frac{d}{ds}\right) g \cdot \mathbf{p} = \left(\mathbf{p} \cdot \mathbf{s}\right) G\left(\left(s^2 + h^2\right)^{1/2}\right), \tag{A.35}$$

$$\left(\mathbf{p} \cdot \mathbf{s}\right) \frac{1}{s} \frac{dg}{ds} = \left(\mathbf{p} \cdot \mathbf{s}\right) G\left(\left(s^2 + h^2\right)^{1/2}\right), \tag{A.36}$$

$$\frac{1}{s} \frac{dg}{ds} = G\left(\left(s^2 + h^2\right)^{1/2}\right), \tag{A.37}$$

$$g = \int sG\left(\left(s^2 + h^2\right)^{1/2}\right) ds. \tag{A.38}$$

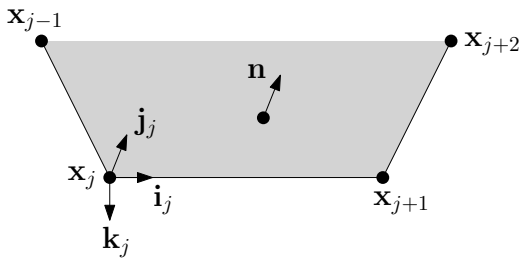

Figure A.2: The local, right-handed coordinate system for the jth edge.

Making the same change of variables as before,

$$g = \int tG\left(t\right) dt. \tag{A.39}$$

Unlike before, this expression, no matter the limits of integration, is nonsingular everywhere. So,

$$g = \int_0^r tG\left(t\right) dt. \tag{A.40}$$

In summary,

$$\mathbf{F} = \sigma_1 f \mathbf{s} + g\mathbf{p}, \quad f = \frac{1}{s^2}\int_h^r tG\left(t\right) dt, \quad g = \int_0^r tG\left(t\right) dt. \tag{A.41}$$

The single-layer potential is now given by

$$L\left(\mathbf{y}\right) = \int_{\mathrm{C}} \mathbf{n}' \cdot \mathbf{F}\left(\mathbf{x}, \mathbf{y}\right) dl\left(\mathbf{x}\right). \tag{A.42}$$

Divide the integral into n pieces, one for each edge:

$$L\left(\mathbf{y}\right) = \sum_{j=1}^{n} \int_{\mathrm{C}_j} \mathbf{n}_j \cdot \mathbf{F}\left(\mathbf{x}, \mathbf{y}\right) dl\left(\mathbf{x}\right) = \sum_{j=1}^{n} I_j\left(\mathbf{y}\right), \tag{A.43}$$

where C_j starts at $\mathbf{x}_j$ and ends at $\mathbf{x}_{j+1}$ ($\mathbf{x}_{n+1} = \mathbf{x}_1$). Consider the jth integral:

$$I_j(\mathbf{y}) = \int_{\mathrm{C}_j} \mathbf{n}_j \cdot \mathbf{F}(\mathbf{x}, \mathbf{y})\, dl(\mathbf{x}). \tag{A.44}$$

Create a local, right-handed coordinate system, which is centered at $\mathbf{x}_j$ and has orthogonal axes given by (see Fig. A.2)

$$\mathbf{i}_j = \frac{\mathbf{x}_{j+1} - \mathbf{x}_j}{l_j}, \quad l_j = |\mathbf{x}_{j+1} - \mathbf{x}_j|, \quad \mathbf{j}_j = \mathbf{n}, \quad \mathbf{k}_j = \mathbf{i}_j \times \mathbf{j}_j = \mathbf{n}_j. \tag{A.45}$$

Write the evaluation point in this local coordinate system:

$$\mathbf{y} = \mathbf{x}_j + x'\mathbf{i}_j + y'\mathbf{j}_j + z'\mathbf{k}_j, \tag{A.46}$$

$$x' = \mathbf{i}_j \cdot (\mathbf{y} - \mathbf{x}_j), \quad y' = \mathbf{j}_j \cdot (\mathbf{y} - \mathbf{x}_j), \quad z' = \mathbf{k}_j \cdot (\mathbf{y} - \mathbf{x}_j), \tag{A.47}$$

Recall that

$$\mathbf{y} = \mathbf{y}' + h\mathbf{n}. \tag{A.48}$$

Note that $h\mathbf{n} = y'\mathbf{j}_j$ (which means $y' = h \geq 0$), so

$$\mathbf{x}_j + x'\mathbf{i}_j + y'\mathbf{j}_j + z'\mathbf{k}_j = \mathbf{y}' + y'\mathbf{j}_j, \tag{A.49}$$

$$\mathbf{y}' = \mathbf{x}_j + x'\mathbf{i}_j + z'\mathbf{k}_j. \tag{A.50}$$

The integration point is

$$\mathbf{x} = \mathbf{x}_j + x\mathbf{i}_j, \tag{A.51}$$

where x goes from 0 to l_j, and

$$\mathbf{s} = \mathbf{x} - \mathbf{y}' = (\mathbf{x}_j + x\mathbf{i}_j) - (\mathbf{x}_j + x'\mathbf{i}_j + z'\mathbf{k}_j) = (x - x')\,\mathbf{i}_j - z'\mathbf{k}_j. \tag{A.52}$$

Now, let's return to the integral:

$$I_j(\mathbf{y}) = \int_{\mathrm{C}_j} \mathbf{n}_j \cdot \mathbf{F}(\mathbf{x}, \mathbf{y})\, dl(\mathbf{x}), \tag{A.53}$$

$$I_j(\mathbf{y}) = \int_{x=0}^{l_j} \mathbf{k}_j \cdot (\sigma_1 f \mathbf{s} + g\mathbf{p})\, dx, \tag{A.54}$$

$$I_j(\mathbf{y}) = \int_{x=0}^{l_j} (\mathbf{k}_j \cdot (\sigma_1 f \mathbf{s}) + \mathbf{k}_j \cdot (g\mathbf{p}))\, dx, \tag{A.55}$$

$$I_j(\mathbf{y}) = \int_{x=0}^{l_j} (\sigma_1 (\mathbf{k}_j \cdot \mathbf{s})\, f + (\mathbf{k}_j \cdot \mathbf{p})\, g)\, dx, \tag{A.56}$$

$$I_j(\mathbf{y}) = \int_{x=0}^{l_j} (-\sigma_1 z' f + (\mathbf{k}_j \cdot \mathbf{p})\, g)\, dx. \tag{A.57}$$

Explicity name all of the arguments:

$$I_j(\mathbf{y}) = \int_{x=0}^{l_j} (-\sigma_1 z' f(s;h) + (\mathbf{k}_j \cdot \mathbf{p})\, g(s;h))\, dx, \tag{A.58}$$

$$\begin{aligned} I_j(\mathbf{y}) = \int_{x=0}^{l_j} \Bigg(& -\sigma_1 z' f\left(\left((x-x')^2 + z'^2\right)^{1/2}; y'\right) + \\ & (\mathbf{k}_j \cdot \mathbf{p})\, g\left(\left((x-x')^2 + z'^2\right)^{1/2}; y'\right)\Bigg)\, dx. \end{aligned} \tag{A.59}$$

Make a change of variables:

$$x - x' \to x, \tag{A.60}$$

$$I_j(\mathbf{y}) = \int_{x=-x'}^{l_j - x'} \left(-\sigma_1 z' f\left(\left(x^2 + z'^2 \right)^{1/2}; y' \right) + (\mathbf{k}_j \cdot \mathbf{p})\, g\left(\left(x^2 + z'^2 \right)^{1/2}; y' \right) \right) dx, \tag{A.61}$$

$$I_j(\mathbf{y}) = H_j(l_j - x'; y', z') - H_j(-x'; y', z'), \tag{A.62}$$

where $H_j = H_j(x; y', z')$ is the following primitive:

$$H_j = \int \left(-\sigma_1 z' f\left(\left(x^2 + z'^2 \right)^{1/2}; y' \right) + (\mathbf{k}_j \cdot \mathbf{p})\, g\left(\left(x^2 + z'^2 \right)^{1/2}; y' \right) \right) dx. \tag{A.63}$$

In summary,

$$I_j(\mathbf{y}) = H_j(l_j - x'; y', z') - H_j(-x'; y', z'), \tag{A.64}$$

where

$$H_j = H_j^{\sigma_1} + H_j^{\mathbf{p}}, \tag{A.65}$$

and

$$H_j^{\sigma_1} = -\sigma_1 z' \int f\left(\left(x^2 + z'^2 \right)^{1/2}; y' \right) dx, \tag{A.66}$$

$$H_j^{\mathbf{p}} = (\mathbf{k}_j \cdot \mathbf{p}) \int g\left(\left(x^2 + z'^2 \right)^{1/2}; y' \right) dx. \tag{A.67}$$

A.2.2 Primitives

Let's compute the primitives, $H_j^{\sigma_1}$ and $H_j^{\mathbf{P}}$, for the Green's function for the Laplace equation. We have

$$G(r) = \frac{1}{4\pi r}, \tag{A.68}$$

$$f = \frac{1}{s^2}\int_h^r tG(t)\,dt = \frac{1}{4\pi s^2}\int_h^r dt = \frac{\left(s^2+h^2\right)^{1/2} - h}{4\pi s^2}, \tag{A.69}$$

$$g = \int_0^r tG(t)\,dt = \frac{1}{4\pi}\int_0^r dt = \frac{\left(s^2+h^2\right)^{1/2}}{4\pi}. \tag{A.70}$$

By choosing the limits as we did, as long as $h \neq 0$, the expression in Eq. (A.69) will not "blow up" as $s \to 0$. First, let's do Eq. (A.66):

$$H_j^{\sigma_1} = -\sigma_1 z' \int f\left(\left(x^2+z'^2\right)^{1/2}; y'\right)dx, \tag{A.71}$$

$$H_j^{\sigma_1} = -\frac{\sigma_1 z'}{4\pi}\int \frac{\left(x^2+y'^2+z'^2\right)^{1/2} - y'}{x^2+z'^2}dx, \tag{A.72}$$

$$H_j^{\sigma_1} = -\frac{\sigma_1}{4\pi}\left(z'\ln(x+r) + y'\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right)\right), \tag{A.73}$$

where

$$r = \left(x^2+y'^2+z'^2\right)^{1/2}. \tag{A.74}$$

Second, let's do Eq. (A.67):

$$H_j^{\mathbf{p}} = (\mathbf{k}_j \cdot \mathbf{p}) \int g\left(\left(x^2 + z'^2\right)^{1/2}; y'\right) dx, \tag{A.75}$$

$$H_j^{\mathbf{p}} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} \int \left(x^2 + y'^2 + z'^2\right)^{1/2} dx, \tag{A.76}$$

$$H_j^{\mathbf{p}} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{8\pi} \left(xr + \left(y'^2 + z'^2\right) \ln\left(x + r\right)\right). \tag{A.77}$$

A.2.3 Special Cases

The primitive, $H_j^{\sigma_1}$, has a singularity at $x = 0$ when $z' = 0$. Note, however, that there is a factor of z' in front of the original integral. Thus, when $z' = 0$, $H_j^{\sigma_1}$ is not used, and instead, the integral is computed directly as zero. In practice, when computing the primitives using double precision, for very small, but nonzero, z', the given expressions can still "blow up" or return NaNs. Thus, the integral is computed directly as zero when $|z'|$ is less than some small value, e.g., 10^{-5}.

For $H_j^{\mathbf{p}}$, there are no singularities, but the given expressions can "blow up" or return NaNs when $y', z' = 0$. In this case,

$$H_j^{\mathbf{p}} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} \int \left(x^2\right)^{1/2} dx, \tag{A.78}$$

$$H_j^{\mathbf{p}} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} \int |x|\, dx, \tag{A.79}$$

$$H_j^{\mathbf{p}} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{8\pi} \operatorname{sgn}(x)\, x^2. \tag{A.80}$$

Again, in practice, this expression is used when y' (remember, $y' \geq 0$) and $|z'|$ are both

less than some small value, e.g., 10^{-5}.

A.3 Gradient of Single-Layer Potential

A.3.1 Line Integrals

The gradient of the single-layer potential is given by

$$\nabla_{\mathbf{y}} L(\mathbf{y}) = \nabla_{\mathbf{y}} \sum_{j=1}^{n} I_j(\mathbf{y}) = \sum_{j=1}^{n} \nabla_{\mathbf{y}} I_j(\mathbf{y}). \tag{A.81}$$

Remember that

$$I_j(\mathbf{y}) = H_j(l_j - x'; y', z') - H_j(-x'; y', z'), \tag{A.82}$$

where

$$x' = \mathbf{i}_j \cdot (\mathbf{y} - \mathbf{x}_j), \quad y' = \mathbf{j}_j \cdot (\mathbf{y} - \mathbf{x}_j), \quad z' = \mathbf{k}_j \cdot (\mathbf{y} - \mathbf{x}_j). \tag{A.83}$$

Taking the gradient, we have

$$\nabla_{\mathbf{y}} I_j(\mathbf{y}) = \nabla_{\mathbf{y}} \left(H_j(l_j - x'; y', z') - H_j(-x'; y', z') \right). \tag{A.84}$$

We are taking the derivative with respect to the primed coordinates, so we need to apply the chain rule in the case of x'. The components of $\nabla_{\mathbf{y}} I_j$ are given by

$$\left[\nabla_{\mathbf{y}} I_j\left(\mathbf{y}\right)\right]_{x'} = -\left(\frac{dH_j}{dx}\left(l_j - x'; y', z'\right) - \frac{dH_j}{dx}\left(-x'; y', z'\right)\right), \tag{A.85}$$

$$\left[\nabla_{\mathbf{y}} I_j\left(\mathbf{y}\right)\right]_{y'} = \frac{dH_j}{dy'}\left(l_j - x'; y', z'\right) - \frac{dH_j}{dy'}\left(-x'; y', z'\right), \tag{A.86}$$

$$\left[\nabla_{\mathbf{y}} I_j\left(\mathbf{y}\right)\right]_{z'} = \frac{dH_j}{dz'}\left(l_j - x'; y', z'\right) - \frac{dH_j}{dz'}\left(-x'; y', z'\right). \tag{A.87}$$

A.3.2 Primitives

To compute $\nabla_{\mathbf{y}} L$, we need to compute the derivatives of the following primitives with respect to their arguments:

$$H_j = H_j\left(x; y', z'\right) = H_j^{\sigma_1} + H_j^{\mathbf{P}} = H_j^{\sigma_1}\left(x; y', z'\right) + H_j^{\mathbf{P}}\left(x; y', z'\right). \tag{A.88}$$

Let's take the derivative of $H_j^{\sigma_1}$ with respect to x:

$$\frac{dH_j^{\sigma_1}}{dx} = \frac{d}{dx}\left(-\frac{\sigma_1 z'}{4\pi}\int \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2} dx\right), \tag{A.89}$$

$$\begin{aligned}\frac{dH_j^{\sigma_1}}{dx} = &-\frac{d\sigma_1}{dx}\frac{z'}{4\pi}\int \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2} dx - \\ &\frac{\sigma_1 z'}{4\pi}\frac{d}{dx}\int \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2} dx.\end{aligned} \tag{A.90}$$

We know that

$$\sigma_1 = \sigma_0 + \mathbf{p} \cdot (\mathbf{y}' - \mathbf{x}_1) = \sigma_0 + \mathbf{p} \cdot ((\mathbf{x}_j + x'\mathbf{i}_j + z'\mathbf{k}_j) - \mathbf{x}_1), \tag{A.91}$$

$$\frac{d\sigma_1}{dx} = (\mathbf{i}_j \cdot \mathbf{p}) \frac{dx'}{dx} = -(\mathbf{i}_j \cdot \mathbf{p}), \quad \frac{d\sigma_1}{dz'} = \mathbf{k}_j \cdot \mathbf{p}. \tag{A.92}$$

So,

$$\begin{aligned} \frac{dH_j^{\sigma_1}}{dx} = \frac{(\mathbf{i}_j \cdot \mathbf{p})\, z'}{4\pi} \int \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2} dx - \\ \frac{\sigma_1 z'}{4\pi} \frac{d}{dx} \int \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2} dx. \end{aligned} \tag{A.93}$$

The integral on the left is the same as before, and the derivative on the right cancels the integral, leaving

$$\begin{aligned} \frac{dH_j^{\sigma_1}}{dx} = \frac{\mathbf{i}_j \cdot \mathbf{p}}{4\pi} \left(z' \ln(x + r) + y' \left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right) \right) \right) - \\ \frac{\sigma_1 z'}{4\pi} \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2}. \end{aligned} \tag{A.94}$$

Let's take the derivative of $H_j^{\sigma_1}$ with respect to y':

$$\frac{dH_j^{\sigma_1}}{dy'} = \frac{d}{dy'} \left(-\frac{\sigma_1 z'}{4\pi} \int \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2} dx \right). \tag{A.95}$$

Since σ_1 does not depend on y',

$$\frac{dH_j^{\sigma_1}}{dy'} = -\frac{\sigma_1 z'}{4\pi} \int \frac{d}{dy'} \frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2} dx, \tag{A.96}$$

$$\frac{dH_j^{\sigma_1}}{dy'} = -\frac{\sigma_1}{4\pi} \left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right) \right). \tag{A.97}$$

Let's take the derivative of $H_j^{\sigma_1}$ with respect to z':

$$\frac{dH_j^{\sigma_1}}{dz'} = \frac{d}{dz'}\left(-\frac{\sigma_1 z'}{4\pi}\int \frac{\left(x^2+y'^2+z'^2\right)^{1/2}-y'}{x^2+z'^2}dx\right) \tag{A.98}$$

$$\begin{aligned}\frac{dH_j^{\sigma_1}}{dz'} = -\frac{d\sigma_1}{dz'}\frac{z'}{4\pi}\int \frac{\left(x^2+y'^2+z'^2\right)^{1/2}-y'}{x^2+z'^2}dx - \\ \frac{\sigma_1}{4\pi}\int \frac{d}{dz'}\left(z'\frac{\left(x^2+y'^2+z'^2\right)^{1/2}-y'}{x^2+z'^2}\right)dx.\end{aligned} \tag{A.99}$$

The integral on the left is the same as before, and after solving the integral on the right, we have

$$\begin{aligned}\frac{dH_j^{\sigma_1}}{dz'} = -\frac{\mathbf{k}_j\cdot\mathbf{p}}{4\pi}\left(z'\ln\left(x+r\right)+y'\left(\arctan\left(\frac{y'x}{z'r}\right)-\arctan\left(\frac{x}{z'}\right)\right)\right) - \\ \frac{\sigma_1}{4\pi}\left(\ln\left(x+r\right)-\frac{x\left(r-y'\right)}{x^2+z'^2}\right).\end{aligned} \tag{A.100}$$

Let's take the derivative of H_j^{p} with respect to x:

$$\frac{dH_j^{\mathrm{p}}}{dx} = \frac{d}{dx}\left(\frac{\mathbf{k}_j\cdot\mathbf{p}}{4\pi}\int\left(x^2+y'^2+z'^2\right)^{1/2}dx\right), \tag{A.101}$$

$$\frac{dH_j^{\mathrm{p}}}{dx} = \frac{\mathbf{k}_j\cdot\mathbf{p}}{4\pi}\frac{d}{dx}\int\left(x^2+y'^2+z'^2\right)^{1/2}dx. \tag{A.102}$$

The derivative cancels the integral, leaving

$$\frac{dH_j^{\mathrm{p}}}{dx} = \frac{\mathbf{k}_j\cdot\mathbf{p}}{4\pi}\left(x^2+y'^2+z'^2\right)^{1/2}. \tag{A.103}$$

Let's take the derivative of $H_j^{\mathbf{p}}$ with respect to y':

$$\frac{dH_j^{\mathbf{p}}}{dy'} = \frac{d}{dy'}\left(\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\int \left(x^2 + y'^2 + z'^2\right)^{1/2} dx\right), \tag{A.104}$$

$$\frac{dH_j^{\mathbf{p}}}{dy'} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\int \frac{d}{dy'}\left(x^2 + y'^2 + z'^2\right)^{1/2} dx, \tag{A.105}$$

$$\frac{dH_j^{\mathbf{p}}}{dy'} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\int \frac{y'}{\left(x^2 + y'^2 + z'^2\right)^{1/2}} dx, \tag{A.106}$$

$$\frac{dH_j^{\mathbf{p}}}{dy'} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} y' \ln\left(x + r\right). \tag{A.107}$$

Let's take the derivative of $H_j^{\mathbf{p}}$ with respect to z':

$$\frac{dH_j^{\mathbf{p}}}{dz'} = \frac{d}{dz'}\left(\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\int \left(x^2 + y'^2 + z'^2\right)^{1/2} dx\right), \tag{A.108}$$

$$\frac{dH_j^{\mathbf{p}}}{dz'} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} z' \ln\left(x + r\right). \tag{A.109}$$

A.3.3 Special Cases

There are a number of special cases when computing the derivatives of the primitives when one or more arguments are zero. They are caused by singularities in some of the primitives that cause them to "blow up" or return NaNs. Many of these singularities are "fake" and can be corrected by careful analysis. Some, however, are real and, thus, uncorrectable. In these cases, they are caused by the governing physics. Let's go one by one.

First, let's look at the derivative of $H_j^{\sigma_1}$ with respect to x:

$$\frac{dH_j^{\sigma_1}}{dx} = \frac{\mathbf{i}_j \cdot \mathbf{p}}{4\pi}\left(z' \ln\left(x + r\right) + y'\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right)\right) - \frac{\sigma_1 z'}{4\pi}\frac{\left(x^2 + y'^2 + z'^2\right)^{1/2} - y'}{x^2 + z'^2}. \tag{A.110}$$

This has a singularity at $x = 0$ when $z' = 0$. However, like in the case of $H_j^{\sigma_1}$, there is a factor of z' in front of the original integral. Thus, when $z' = 0$, the integral is computed directly as zero.

Second, let's look at the derivative of $H_j^{\sigma_1}$ with respect to y':

$$\frac{dH_j^{\sigma_1}}{dy'} = -\frac{\sigma_1}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right). \tag{A.111}$$

This has a singularity at $x = 0$ when $z' = 0$. However, unlike for the derivative with respect to x, the derivative with respect to y' caused an extra factor of z' to pop out from inside of the original integral, canceling the one already there. This is not a problem, though, because we can use the arctangent subtraction formula to do the following:

$$\frac{dH_j^{\sigma_1}}{dy'} = -\frac{\sigma_1}{4\pi}\arctan\left(\left(\frac{y'x}{z'r} - \frac{x}{z'}\right)\left(1 + \frac{y'x}{z'r}\frac{x}{z'}\right)^{-1}\right), \tag{A.112}$$

$$\frac{dH_j^{\sigma_1}}{dy'} = -\frac{\sigma_1}{4\pi}\arctan\left(\frac{y'xz' - xz'r}{z'^2 r + y'x^2}\right). \tag{A.113}$$

Letting $z' \to 0$, the expression inside the arctangent goes to zero, leaving

$$\frac{dH_j^{\sigma_1}}{dy'} = 0. \tag{A.114}$$

Third, let's look at the derivative of $H_j^{\sigma_1}$ with respect to z':

$$\frac{dH_j^{\sigma_1}}{dz'} = -\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\left(z' \ln\left(x+r\right) + y'\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right)\right) - \frac{\sigma_1}{4\pi}\left(\ln\left(x+r\right) - \frac{x\left(r-y'\right)}{x^2+z'^2}\right). \tag{A.115}$$

This has a singularity at $x = 0$ when $z' = 0$. The left term in this expression is just like before. There's a factor of z' in front of the original integral, so the entire integral is zero when $z' = 0$. The right term is more complicated. The derivative with respect to z' eliminated that factor, so the entire integral doesn't go to zero automatically. Thus, the $\ln\left(x+r\right)$ does, indeed, cause the expression to "blow up". We can solve this problem by noting that, in order to solve the original integral, we must evaluate the primitive at $x = x_1 = -x'$ and $x = x_2 = l_j - x'$, and then subtract the former from the latter. Let's look at what happens when $y' = 0$:

$$\left(\ln\left(x_2+r_2\right) - \frac{x_2\left(r_2-y'\right)}{x_2^2+z'^2}\right) - \left(\ln\left(x_1+r_1\right) - \frac{x_1\left(r_1-y'\right)}{x_1^2+z'^2}\right), \tag{A.116}$$

$$\ln\left(\frac{x_2+r_2}{x_1+r_1}\right) - \left(\frac{x_2\left(r_2-y'\right)}{x_2^2+z'^2} - \frac{x_1\left(r_1-y'\right)}{x_1^2+z'^2}\right), \tag{A.117}$$

$$\ln\left(\frac{x_2+\left(x_2^2+y'^2+z'^2\right)^{1/2}}{x_1+\left(x_1^2+y'^2+z'^2\right)^{1/2}}\right) - \left(\frac{x_2\left(\left(x_2^2+y'^2+z'^2\right)^{1/2}-y'\right)}{x_2^2+z'^2} - \frac{x_1\left(\left(x_1^2+y'^2+z'^2\right)^{1/2}-y'\right)}{x_1^2+z'^2}\right), \tag{A.118}$$

$$\ln\left(\frac{x_2+\left(x_2^2+z'^2\right)^{1/2}}{x_1+\left(x_1^2+z'^2\right)^{1/2}}\right)-\left(\frac{x_2\left(x_2^2+z'^2\right)^{1/2}}{x_2^2+z'^2}-\frac{x_1\left(x_1^2+z'^2\right)^{1/2}}{x_1^2+z'^2}\right), \tag{A.119}$$

$$\ln\left(\frac{x_2+|x_2|\left(1+\dfrac{z'^2}{x_2^2}\right)^{1/2}}{x_1+|x_1|\left(1+\dfrac{z'^2}{x_1^2}\right)^{1/2}}\right)-\left(\frac{x_2}{\left(x_2^2+z'^2\right)^{1/2}}-\frac{x_1}{\left(x_1^2+z'^2\right)^{1/2}}\right). \tag{A.120}$$

For small z', using a binomial approximation,

$$\ln\left(\frac{x_2+|x_2|\left(1+\dfrac{1}{2}\dfrac{z'^2}{x_2^2}\right)}{x_1+|x_1|\left(1+\dfrac{1}{2}\dfrac{z'^2}{x_1^2}\right)}\right)-\left(\frac{x_2}{\left(x_2^2+z'^2\right)^{1/2}}-\frac{x_1}{\left(x_1^2+z'^2\right)^{1/2}}\right). \tag{A.121}$$

When $x_1, x_2 < 0$, $x_1 + |x_1| = 0$ and $x_2 + |x_2| = 0$, so

$$\ln\left(\frac{|x_2|\dfrac{1}{2}\dfrac{z'^2}{x_2^2}}{|x_1|\dfrac{1}{2}\dfrac{z'^2}{x_1^2}}\right)-\left(\frac{x_2}{\left(x_2^2+z'^2\right)^{1/2}}-\frac{x_1}{\left(x_1^2+z'^2\right)^{1/2}}\right). \tag{A.122}$$

Rearranging,

$$\ln\left(\frac{1/|x_2|}{1/|x_1|}\right)-\left(\frac{x_2}{\left(x_2^2+z'^2\right)^{1/2}}-\frac{x_1}{\left(x_1^2+z'^2\right)^{1/2}}\right). \tag{A.123}$$

Letting $z' \to 0$,

$$\ln\left(\frac{1/|x_2|}{1/|x_1|}\right) - \left(\frac{x_2}{|x_2|} - \frac{x_1}{|x_1|}\right). \tag{A.124}$$

Since x_1 and x_2 are both negative and, thus, have the same sign, we have

$$\ln\left(\frac{1/|x_2|}{1/|x_1|}\right). \tag{A.125}$$

Note how, when one of the xs is negative and the other is positive, we can't do what we just did in the previous three or four equations, so the expression does, indeed, "blow up". This indicates that, when evaluating at a point along an edge, $\nabla_{\mathbf{y}} L$ is singular along the direction perpendicular to that edge. Now, let's look at what happens when $y' \neq 0$.

$$\ln\left(x + r\right) - \frac{x\left(r - y'\right)}{x^2 + z'^2}. \tag{A.126}$$

When $z' = 0$,

$$\ln\left(x + \left(x^2 + y'^2\right)^{1/2}\right) - \frac{\left(x^2 + y'^2\right)^{1/2} - y'}{x}. \tag{A.127}$$

When computed directly, this expression will "blow up" when $x = 0$. However, using a binomial approximation,

$$\ln\left(x + \left(x^2 + y'^2\right)^{1/2}\right) - \frac{|y'|\left(1 + \frac{1}{2}\frac{x^2}{y'^2}\right) - y'}{x}. \tag{A.128}$$

Since $y' > 0$,

$$\ln\left(x + \left(x^2 + y'^2\right)^{1/2}\right) - \frac{x}{2y'}. \tag{A.129}$$

When $x = 0$,

$$\ln\left(y'\right). \tag{A.130}$$

Fourth, let's look at the derivative of $H_j^{\mathbf{p}}$ with respect to y':

$$\frac{dH_j^{\mathbf{p}}}{dy'} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} y' \ln\left(x + r\right). \tag{A.131}$$

This is completely regular as long as y' or z' are nonzero. However, when $y', z' = 0$, this expression can "blow up" or return NaNs. To solve this, set $y' = 0$ and $z' \neq 0$. The expression inside the logarithm is

$$x + \left(x^2 + z'^2\right)^{1/2} > 0. \tag{A.132}$$

Thus, the logarithm does not "blow up", and the factor of y' forces the entire expression to zero. Once this is done, we can let $z' \to 0$ as well, leaving us with zero for the entire expression when $y', z' = 0$.

Fifth, let's look at the derivative of $H_j^{\mathbf{p}}$ with respect to z':

$$\frac{dH_j^{\mathbf{p}}}{dz'} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} z' \ln\left(x + r\right). \tag{A.133}$$

This is the same as for the derivative of $H_j^{\mathbf{p}}$ with respect to y', except there's a factor of z' instead of y'. Again, this expression can "blow up" or return NaNs when $y', z' = 0$. We

can solve this like before. First, set $z' = 0$, but $y' \neq 0$, and then let $y' \to 0$. The expression will then be equal to zero.

A.4 Double-Layer Potential

A.4.1 Line Integrals

The double-layer potential is given by

$$M(\mathbf{y}) = \int_S (\sigma_1 + \mathbf{p} \cdot \mathbf{s}) (\mathbf{n} \cdot \nabla_{\mathbf{x}} G(\mathbf{x}, \mathbf{y})) \, dS(\mathbf{x}) . \tag{A.134}$$

Since $\nabla_{\mathbf{x}} G = -\nabla_{\mathbf{y}} G$,

$$M(\mathbf{y}) = \int_S (\sigma_1 + \mathbf{p} \cdot \mathbf{s}) (-\mathbf{n} \cdot \nabla_{\mathbf{y}} G(\mathbf{x}, \mathbf{y})) \, dS(\mathbf{x}) . \tag{A.135}$$

Rearranging and using what we derived in the previous sections, we have

$$M(\mathbf{y}) = -\mathbf{n} \cdot \nabla_{\mathbf{y}} \int_S (\sigma_1 + \mathbf{p} \cdot \mathbf{s}) G(\mathbf{x}, \mathbf{y}) \, dS(\mathbf{x}) , \tag{A.136}$$

$$M(\mathbf{y}) = -\mathbf{n} \cdot \nabla_{\mathbf{y}} L(\mathbf{y}) = -\mathbf{n} \cdot \nabla_{\mathbf{y}} \sum_{j=1}^{n} I_j(\mathbf{y}) = \sum_{j=1}^{n} (-\mathbf{n} \cdot \nabla_{\mathbf{y}} I_j(\mathbf{y})) , \tag{A.137}$$

$$M(\mathbf{y}) = \sum_{j=1}^{n} J_j(\mathbf{y}) , \tag{A.138}$$

where

$$J_j(\mathbf{y}) = -\mathbf{n} \cdot \nabla_{\mathbf{y}} I_j(\mathbf{y}) . \tag{A.139}$$

Remember that $\mathbf{n} = \mathbf{j}_j$ and

$$\nabla_{\mathbf{y}} = \frac{d}{dx'}\mathbf{i}_j + \frac{d}{dy'}\mathbf{j}_j + \frac{d}{dz'}\mathbf{k}_j, \tag{A.140}$$

so we have

$$J_j(\mathbf{y}) = -\frac{dI_j}{dy'}(\mathbf{y}). \tag{A.141}$$

A.4.2 Primitives

We know that

$$I_j(\mathbf{y}) = H_j(l_j - x'; y', z') - H_j(-x'; y', z'), \tag{A.142}$$

$$\frac{dI_j}{dy'}(\mathbf{y}) = \frac{dH_j}{dy'}(l_j - x'; y', z') - \frac{dH_j}{dy'}(-x'; y', z'). \tag{A.143}$$

Plugging in Eq. (A.141),

$$J_j(\mathbf{y}) = -\left(\frac{dH_j}{dy'}(l_j - x'; y', z') - \frac{dH_j}{dy'}(-x'; y', z')\right), \tag{A.144}$$

$$J_j(\mathbf{y}) = K_j(l_j - x'; y', z') - K(-x'; y', z'), \tag{A.145}$$

where

$$K_j = -\frac{dH_j}{dy'}. \tag{A.146}$$

Like in the case of the single-layer potential, we separate K_j into two pieces, one corresponding to σ_1 and one corresponding to $\mathbf{p}$:

$$K_j = K_j\left(x; y', z'\right) = K_j^{\sigma_1} + K_j^{\mathbf{p}} = K_j^{\sigma_1}\left(x; y', z'\right) + K_j^{\mathbf{p}}\left(x; y', z'\right), \tag{A.147}$$

where

$$K_j^{\sigma_1} = -\frac{dH_j^{\sigma_1}}{dy'} = \frac{\sigma_1}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right), \tag{A.148}$$

$$K_j^{\mathbf{p}} = -\frac{dH_j^{\mathbf{p}}}{dy'} = -\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} y' \ln\left(x + r\right). \tag{A.149}$$

A.4.3 Special Cases

Since $K_j^{\sigma_1}$ and $K_j^{\mathbf{p}}$ are the negatives of $dH_j^{\sigma_1}/dy'$ and $dH_j^{\mathbf{p}}/dy'$, respectively, the special cases for these new primitives are the same as those for the old ones.

A.5 Gradient of Double-Layer Potential

A.5.1 Line Integrals

The gradient of the double-layer potential is given by

$$\nabla M\left(\mathbf{y}\right) = \nabla_{\mathbf{y}} \sum_{j=1}^{n} J_j\left(\mathbf{y}\right) = \sum_{j=1}^{n} \nabla_{\mathbf{y}} J_j\left(\mathbf{y}\right). \tag{A.150}$$

Remember that

$$J_j\left(\mathbf{y}\right) = K_j\left(l_j - x'; y', z'\right) - K_j\left(-x'; y', z'\right), \tag{A.151}$$

where

$$x' = \mathbf{i}_j \cdot (\mathbf{y} - \mathbf{x}_j), \quad y' = \mathbf{j}_j \cdot (\mathbf{y} - \mathbf{x}_j), \quad z' = \mathbf{k}_j \cdot (\mathbf{y} - \mathbf{x}_j). \tag{A.152}$$

Taking the gradient, we have

$$\nabla_{\mathbf{y}} J_j(\mathbf{y}) = \nabla_{\mathbf{y}} \left(K_j(l_j - x'; y', z') - K_j(-x'; y', z') \right). \tag{A.153}$$

We are taking the gradient with respect to the primed coordinates, so we need to apply the chain rule in the case of x'. The components of $\nabla_{\mathbf{y}} J_j(\mathbf{y})$ are given by

$$\left[\nabla_{\mathbf{y}} J_j(\mathbf{y})\right]_{x'} = -\left(\frac{dK_j}{dx}(l_j - x'; y', z') - \frac{dK_j}{dx}(-x'; y', z') \right), \tag{A.154}$$

$$\left[\nabla_{\mathbf{y}} J_j(\mathbf{y})\right]_{y'} = \frac{dK_j}{dy'}(l_j - x'; y', z') - \frac{dK_j}{dy'}(-x'; y', z'), \tag{A.155}$$

$$\left[\nabla_{\mathbf{y}} J_j(\mathbf{y})\right]_{z'} = \frac{dK_j}{dz'}(l_j - x'; y', z') - \frac{dK_j}{dz'}(-x'; y', z'). \tag{A.156}$$

A.5.2 Primitives

To compute ∇M, we need to compute the derivatives of the following primitives with respect to their arguments:

$$K_j = K_j(x; y', z') = K_j^{\sigma_1} + K_j^{\mathbf{P}} = K_j^{\sigma_1}(x; y', z') + K_j^{\mathbf{P}}(x; y', z'). \tag{A.157}$$

Let's take the derivative of $K_j^{\sigma_1}$ with respect to x:

$$\frac{dK_j^{\sigma_1}}{dx} = \frac{d}{dx}\left(\frac{\sigma_1}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right)\right), \tag{A.158}$$

$$\begin{aligned}\frac{dK_j^{\sigma_1}}{dx} = \frac{d\sigma_1}{dx}\frac{1}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right) + \\ \frac{\sigma_1}{4\pi}\frac{d}{dx}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right).\end{aligned} \tag{A.159}$$

Since $d\sigma_1/dx = -(\mathbf{i}_j \cdot \mathbf{p})$,

$$\frac{dK_j^{\sigma_1}}{dx} = -\frac{\mathbf{i}_j \cdot \mathbf{p}}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right) + \frac{\sigma_1}{4\pi}\left(\frac{z'}{x^2+z'^2}\right)\left(\frac{y'-r}{r}\right). \tag{A.160}$$

Let's take the derivative of $K_j^{\sigma_1}$ with respect to y':

$$\frac{dK_j^{\sigma_1}}{dy'} = \frac{d}{dy'}\left(\frac{\sigma_1}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right)\right). \tag{A.161}$$

Since σ_1 does not depend on y',

$$\frac{dK_j^{\sigma_1}}{dy'} = \frac{\sigma_1}{4\pi}\frac{d}{dy'}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right), \tag{A.162}$$

$$\frac{dK_j^{\sigma_1}}{dy'} = \frac{\sigma_1}{4\pi}\frac{z'x}{\left(y'^2+z'^2\right)r}. \tag{A.163}$$

Let's take the derivative of $K_j^{\sigma_1}$ with respect to z':

$$\frac{dK_j^{\sigma_1}}{dz'} = \frac{d}{dz'}\left(\frac{\sigma_1}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right)\right), \tag{A.164}$$

$$\begin{aligned}\frac{dK_j^{\sigma_1}}{dz'} = \frac{d\sigma_1}{dz'}\frac{1}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right) + \\ \frac{\sigma_1}{4\pi}\frac{d}{dz'}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right).\end{aligned} \tag{A.165}$$

Since $d\sigma_1/dz' = \mathbf{k}_j \cdot \mathbf{p}$,

$$\begin{aligned}\frac{dK_j^{\sigma_1}}{dz'} = \frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\left(\arctan\left(\frac{y'x}{z'r}\right) - \arctan\left(\frac{x}{z'}\right)\right) + \\ \frac{\sigma_1}{4\pi}\left(-\frac{y'x\left(r^2 + z'^2\right)}{\left(x^2 + z'^2\right)\left(y'^2 + z'^2\right)r} + \frac{x}{x^2 + z'^2}\right).\end{aligned} \tag{A.166}$$

Let's take the derivative of $K_j^{\mathbf{p}}$ with respect to x:

$$\frac{dK_j^{\mathbf{p}}}{dx} = \frac{d}{dx}\left(-\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}y' \ln\left(x + r\right)\right), \tag{A.167}$$

$$\frac{dK_j^{\mathbf{p}}}{dx} = -\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\frac{y'}{r}. \tag{A.168}$$

Let's take the derivative of $K_j^{\mathbf{p}}$ with respect to y':

$$\frac{dK_j^{\mathbf{p}}}{dy'} = \frac{d}{dy'}\left(-\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}y' \ln\left(x + r\right)\right), \tag{A.169}$$

$$\frac{dK_j^{\mathbf{p}}}{dy'} = -\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi}\left(\ln\left(x + r\right) + \frac{y'^2}{r\left(x + r\right)}\right). \tag{A.170}$$

Let's take the derivative of $K_j^{\mathbf{p}}$ with respect to z':

$$\frac{dK_j^{\mathbf{p}}}{dz'} = \frac{d}{dz'}\left(-\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} y' \ln\left(x + r\right)\right), \tag{A.171}$$

$$\frac{dK_j^{\mathbf{p}}}{dz'} = -\frac{\mathbf{k}_j \cdot \mathbf{p}}{4\pi} \frac{y'z'}{r\left(x + r\right)}. \tag{A.172}$$

A.5.3 Special Cases

The special cases for the primitives for the gradient of the double-layer potential follow the same pattern as those for the gradient of the single-layer potential.

Appendix B: Spherical Basis Functions for the Laplace Equation

B.1 Spherical Coordinate System

The spherical coordinate system, shown in Figure B.1, is related to the Cartesian coordinate system by

$$x = r\sin(\theta)\cos(\phi), \quad y = r\sin(\theta)\sin(\phi), \quad z = r\cos(\theta). \tag{B.1}$$

B.2 Local and Multipole Expansions

The Green's function for the Laplace equation can be expanded as

$$G(\mathbf{x}-\mathbf{x}') = \frac{1}{4\pi|\mathbf{x}-\mathbf{x}'|} = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{1}{2n+1}\frac{r_<^n}{r_>^{n+1}}Y_n^{m*}(\theta',\phi')\,Y_n^m(\theta,\phi), \tag{B.2}$$

where (r',θ',ϕ') and (r,θ,ϕ) are the spherical coordinates of $\mathbf{x}'$ and $\mathbf{x}$, respectively, $r_< = \min(r',r)$, and $r_> = \max(r',r)$ (see Fig. B.1). The spherical harmonics are given by

$$Y_n^m(\theta,\phi) = (-1)^m\left(\frac{2n+1}{4\pi}\frac{(n-|m|)!}{(n+|m|)!}\right)^{1/2}P_n^{|m|}(\cos(\theta))\exp(im\phi), \tag{B.3}$$

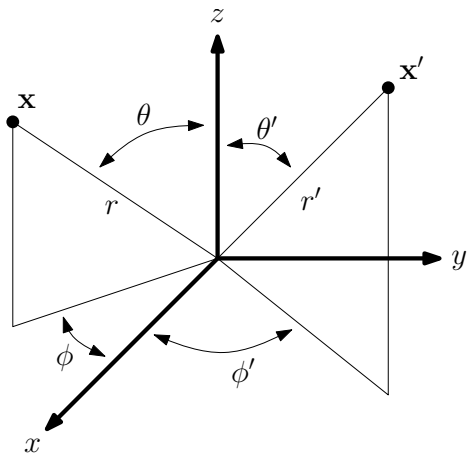

Figure B.1: The spherical coordinate system.

where P_n^m are the associated Legendre polynomials [67, 105]. Fig. B.2 shows a graphical representation of the spherical harmonics.

This expression can be used to build multipole and local expansions. For example, suppose we want to compute the potential at $\mathbf{x}$ due to a point source at $\mathbf{x}'$. When $r > r'$, we can build a multipole expansion:

$$\frac{1}{4\pi \left|\mathbf{x}-\mathbf{x}'\right|} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} R_n^{m*}\left(r', \theta', \phi'\right) S_n^m\left(r, \theta, \phi\right). \tag{B.4}$$

Here,

$$R_n^m\left(\mathbf{x}\right) = R_n^m\left(r, \theta, \phi\right) = \left(\frac{1}{2n+1}\right)^{1/2} r^n Y_n^m\left(\theta, \phi\right), \tag{B.5}$$

$$S_n^m\left(\mathbf{x}\right) = S_n^m\left(r, \theta, \phi\right) = \left(\frac{1}{2n+1}\right)^{1/2} \frac{1}{r^{n+1}} Y_n^m\left(\theta, \phi\right) \tag{B.6}$$

are the local and multipole expansion basis functions, respectively. Instead of centering the expansion around the origin, we can center the expansion around $\mathbf{x}^*$. When $|\mathbf{x}-\mathbf{x}^*| >$

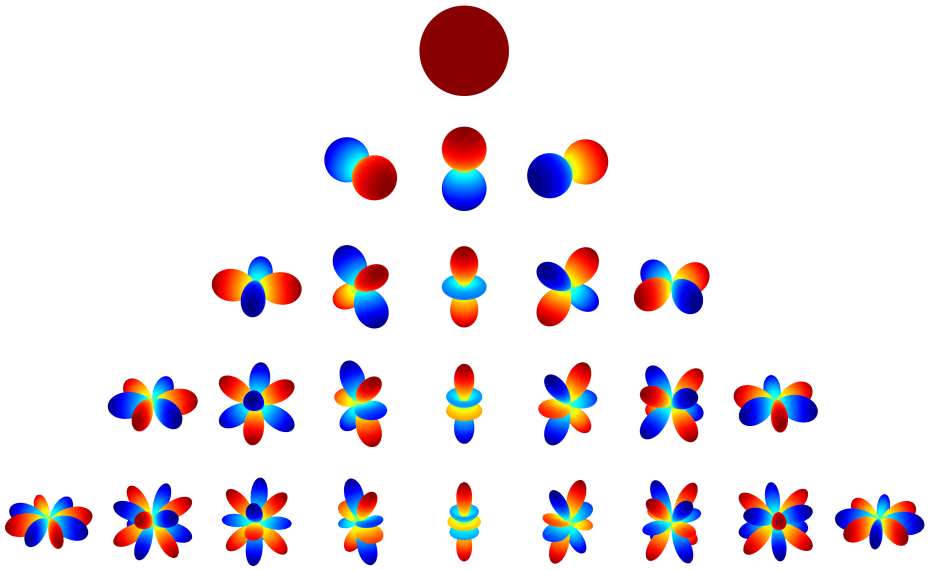

Figure B.2: A graphical representation of the spherical harmonics for $n = 0, 1, 2, 3, 4$ (top to bottom) and $m = -n, -n+1, \ldots, n$ (left to right).

$|\mathbf{x}' - \mathbf{x}^*|$,

$$\frac{1}{4\pi |\mathbf{x} - \mathbf{x}'|} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} R_n^{m*} (\mathbf{x}' - \mathbf{x}^*) S_n^m (\mathbf{x} - \mathbf{x}^*) . \tag{B.7}$$

Likewise, when $|\mathbf{x} - \mathbf{x}^*| < |\mathbf{x}' - \mathbf{x}^*|$, we can build a local expansion:

$$\frac{1}{4\pi |\mathbf{x} - \mathbf{x}'|} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} S_n^{m*} (\mathbf{x}' - \mathbf{x}^*) R_n^m (\mathbf{x} - \mathbf{x}^*) . \tag{B.8}$$

These expressions can be used to build multipole and local expansions for arbitrary source distributions. For example, let us build a multipole expansion for the source distribution, $\rho(\mathbf{x}')$, contained entirely inside an imaginary sphere of radius, r^*, centered around $\mathbf{x}^*$. For a point, $\mathbf{x}$, outside the sphere, the potential due to this source distribution is given by

$$\Phi(\mathbf{x}) = \int_{|\mathbf{x}' - \mathbf{x}^*| < r^*} \frac{\rho(\mathbf{x}')}{4\pi |\mathbf{x} - \mathbf{x}'|} dV(\mathbf{x}') . \tag{B.9}$$

Substituting Eq. (B.7) and rearranging,

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m S_n^m (\mathbf{x} - \mathbf{x}^*), \tag{B.10}$$

where

$$a_n^m = \int_{|\mathbf{x}'-\mathbf{x}^*|<r^*} \rho(\mathbf{x}') R_n^{m*} (\mathbf{x}' - \mathbf{x}^*) dV(\mathbf{x}'). \tag{B.11}$$

We can build a local expansion using the same procedure. Consider a different source distribution contained entirely outside the imaginary sphere. For a point, $\mathbf{x}$, inside the sphere, the potential due to this source distribution is given by

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m R_n^m (\mathbf{x} - \mathbf{x}^*), \tag{B.12}$$

where

$$a_n^m = \int_{|\mathbf{x}'-\mathbf{x}^*|>r^*} \rho(\mathbf{x}') S_n^{m*} (\mathbf{x}' - \mathbf{x}^*) dV(\mathbf{x}'). \tag{B.13}$$

B.3 Translations

Multipole and local expansions can be translated to other multipole and local expansions using the following three identities [67]:

$$S_n^m(\mathbf{r}+\mathbf{t}) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (S|S)_{nn'}^{mm'}(\mathbf{t})\, S_{n'}^{m'}(\mathbf{r}), \tag{B.14}$$

$$S_n^m(\mathbf{r}+\mathbf{t}) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (S|R)_{nn'}^{mm'}(\mathbf{t})\, R_{n'}^{m'}(\mathbf{r}), \tag{B.15}$$

$$R_n^m(\mathbf{r}+\mathbf{t}) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (R|R)_{nn'}^{mm'}(\mathbf{t})\, R_{n'}^{m'}(\mathbf{r}). \tag{B.16}$$

We want to use these three identities to translate an expansion centered around $\mathbf{x}^*$ to another expansion centered around $\mathbf{y}^*$. Setting $\mathbf{r} = \mathbf{x} - \mathbf{y}^*$ and $\mathbf{t} = \mathbf{y}^* - \mathbf{x}^*$, we have

$$S_n^m(\mathbf{x}-\mathbf{x}^*) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (S|S)_{nn'}^{mm'}(\mathbf{y}^*-\mathbf{x}^*)\, S_{n'}^{m'}(\mathbf{x}-\mathbf{y}^*), \tag{B.17}$$

$$S_n^m(\mathbf{x}-\mathbf{x}^*) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (S|R)_{nn'}^{mm'}(\mathbf{y}^*-\mathbf{x}^*)\, R_{n'}^{m'}(\mathbf{x}-\mathbf{y}^*), \tag{B.18}$$

$$R_n^m(\mathbf{x}-\mathbf{x}^*) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (R|R)_{nn'}^{mm'}(\mathbf{y}^*-\mathbf{x}^*)\, R_{n'}^{m'}(\mathbf{x}-\mathbf{y}^*). \tag{B.19}$$

Consider the following multipole expansion:

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m S_n^m(\mathbf{x}-\mathbf{x}^*). \tag{B.20}$$

Let's translate this multipole expansion, which is centered around $\mathbf{x}^*$, to another multipole

expansion centered around $\mathbf{y}^*$:

$$\Phi(\mathbf{x}) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m \left(\sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (S|S)_{nn'}^{mm'} (\mathbf{y}^* - \mathbf{x}^*) \right) S_{n'}^{m'} (\mathbf{x} - \mathbf{y}^*). \tag{B.21}$$

Rearranging,

$$\Phi(\mathbf{x}) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} b_{n'}^{m'} S_{n'}^{m'} (\mathbf{x} - \mathbf{y}^*), \tag{B.22}$$

where

$$b_{n'}^{m'} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (S|S)_{nn'}^{mm'} (\mathbf{y}^* - \mathbf{x}^*) a_n^m. \tag{B.23}$$

The mapping from one set of coefficients to another is linear and, thus, can be represented by a single matrix operation:

$$\begin{bmatrix} b_0^0 \\ b_1^{-1} \\ b_1^0 \\ b_1^1 \\ \vdots \end{bmatrix} = \begin{bmatrix} (S|S)_{0,0}^{0,0} & (S|S)_{1,0}^{-1,0} & (S|S)_{1,0}^{0,0} & (S|S)_{1,0}^{1,0} & \cdots \\ (S|S)_{0,1}^{0,-1} & (S|S)_{1,1}^{-1,-1} & (S|S)_{1,1}^{0,-1} & (S|S)_{1,1}^{1,-1} & \cdots \\ (S|S)_{0,1}^{0,0} & (S|S)_{1,1}^{-1,0} & (S|S)_{1,1}^{0,0} & (S|S)_{1,1}^{1,0} & \cdots \\ (S|S)_{0,1}^{0,1} & (S|S)_{1,1}^{-1,1} & (S|S)_{1,1}^{0,1} & (S|S)_{1,1}^{1,1} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} a_0^0 \\ a_1^{-1} \\ a_1^0 \\ a_1^1 \\ \vdots \end{bmatrix}. \tag{B.24}$$

The $(\mathbf{y}^* - \mathbf{x}^*)$s have been dropped to save space. More compactly,

$$\mathbf{b} = (S|S)(\mathbf{y}^* - \mathbf{x}^*)\mathbf{a}, \tag{B.25}$$

where $(S|S)(\mathbf{y}^* - \mathbf{x}^*)$ is the multipole-to-multipole translation matrix.

Consider the same multipole expansion from before:

$$\Phi\left(\mathbf{x}\right) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m S_n^m \left(\mathbf{x} - \mathbf{x}^*\right). \tag{B.26}$$

Let's translate this multipole expansion, which is centered around $\mathbf{x}^*$, to a local expansion centered around $\mathbf{y}^*$. We can do this using the same procedure as before, except the multipole-to-multipole translation matrix is replaced by a multipole-to-local translation matrix:

$$\mathbf{b} = \left(S|R\right)\left(\mathbf{y}^* - \mathbf{x}^*\right)\mathbf{a}, \tag{B.27}$$

where

$$\left(S|R\right)\left(\mathbf{y}^* - \mathbf{x}^*\right) = \begin{bmatrix} \left(S|R\right)_{0,0}^{0,0} & \left(S|R\right)_{1,0}^{-1,0} & \left(S|R\right)_{1,0}^{0,0} & \left(S|R\right)_{1,0}^{1,0} & \dots \\ \left(S|R\right)_{0,1}^{0,-1} & \left(S|R\right)_{1,1}^{-1,-1} & \left(S|R\right)_{1,1}^{0,-1} & \left(S|R\right)_{1,1}^{1,-1} & \dots \\ \left(S|R\right)_{0,1}^{0,0} & \left(S|R\right)_{1,1}^{-1,0} & \left(S|R\right)_{1,1}^{0,0} & \left(S|R\right)_{1,1}^{1,0} & \dots \\ \left(S|R\right)_{0,1}^{0,1} & \left(S|R\right)_{1,1}^{-1,1} & \left(S|R\right)_{1,1}^{0,1} & \left(S|R\right)_{1,1}^{1,1} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{B.28}$$

Consider the following local expansion:

$$\Phi\left(\mathbf{x}\right) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m R_n^m \left(\mathbf{x} - \mathbf{x}^*\right). \tag{B.29}$$

Let's translate this local expansion, which is centered around $\mathbf{x}^*$, to another local expansion centered around $\mathbf{y}^*$. We can do this using the same procedure as before, except a local-to-

local translation matrix is used:

$$\mathbf{b} = (R|R)\,(\mathbf{y}^* - \mathbf{x}^*)\,\mathbf{a}, \tag{B.30}$$

where

$$(R|R)\,(\mathbf{y}^* - \mathbf{x}^*) = \begin{bmatrix} (R|R)_{0,0}^{0,0} & (R|R)_{1,0}^{-1,0} & (R|R)_{1,0}^{0,0} & (R|R)_{1,0}^{1,0} & \dots \\ (R|R)_{0,1}^{0,-1} & (R|R)_{1,1}^{-1,-1} & (R|R)_{1,1}^{0,-1} & (R|R)_{1,1}^{1,-1} & \dots \\ (R|R)_{0,1}^{0,0} & (R|R)_{1,1}^{-1,0} & (R|R)_{1,1}^{0,0} & (R|R)_{1,1}^{1,0} & \dots \\ (R|R)_{0,1}^{0,1} & (R|R)_{1,1}^{-1,1} & (R|R)_{1,1}^{0,1} & (R|R)_{1,1}^{1,1} & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{B.31}$$

B.4 Translation Matrices

We need to compute the entries in the translation matrices, $(S|S)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*)$, $(S|R)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*)$, and $(R|R)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*)$. We know that

$$R_n^m\,(\mathbf{r}) = \alpha_n^m r^n Y_n^m\,(\theta, \phi)\,, \quad S_n^m\,(\mathbf{r}) = \beta_n^m \frac{1}{r^{n+1}} Y_n^m\,(\theta, \phi)\,. \tag{B.32}$$

The coefficients, α_n^m and β_n^m, were chosen earlier on as

$$\alpha_n^m = \left(\frac{1}{2n+1}\right)^{1/2}, \quad \beta_n^m = \left(\frac{1}{2n+1}\right)^{1/2}. \tag{B.33}$$

Some other coefficients we need are [67]

$$\alpha_{(1)n}^{m} = (-1)^{n} i^{-|m|} \left(\frac{4\pi}{(2n+1)\,(n-|m|)!\,(n+|m|)!} \right)^{1/2}, \tag{B.34}$$

$$\beta_{(1)n}^{m} = i^{|m|} \left(\frac{4\pi\,(n-|m|)!\,(n+|m|)!}{2n+1} \right)^{1/2}. \tag{B.35}$$

For $\alpha_n^m = \alpha_{(1)n}^m$ and $\beta_n^m = \beta_{(1)n}^m$,

$$(S|S)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*) = R_{n'-n}^{m-m'}\,(\mathbf{y}^* - \mathbf{x}^*)\,, \tag{B.36}$$

$$(S|R)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*) = S_{n+n'}^{m-m'}\,(\mathbf{y}^* - \mathbf{x}^*)\,, \tag{B.37}$$

$$(R|R)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*) = R_{n-n'}^{m-m'}\,(\mathbf{y}^* - \mathbf{x}^*)\,. \tag{B.38}$$

For arbitrary α_n^m and β_n^m,

$$(S|S)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*) = \frac{\beta_{(1)n'}^{m'}}{\beta_{n'}^{m'}} \frac{\alpha_{(1)(n'-n)}^{m-m'}}{\alpha_{n'-n}^{m-m'}} \frac{\beta_n^m}{\beta_{(1)n}^m} R_{n'-n}^{m-m'}\,(\mathbf{y}^* - \mathbf{x}^*)\,, \tag{B.39}$$

$$(S|R)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*) = \frac{\alpha_{(1)n'}^{m'}}{\alpha_{n'}^{m'}} \frac{\beta_{(1)(n+n')}^{m-m'}}{\beta_{n+n'}^{m-m'}} \frac{\beta_n^m}{\beta_{(1)n}^m} S_{n+n'}^{m-m'}\,(\mathbf{y}^* - \mathbf{x}^*)\,, \tag{B.40}$$

$$(R|R)_{nn'}^{mm'}\,(\mathbf{y}^* - \mathbf{x}^*) = \frac{\alpha_{(1)n'}^{m'}}{\alpha_{n'}^{m'}} \frac{\alpha_{(1)(n-n')}^{m-m'}}{\alpha_{n-n'}^{m-m'}} \frac{\alpha_n^m}{\alpha_{(1)n}^m} R_{n-n'}^{m-m'}\,(\mathbf{y}^* - \mathbf{x}^*)\,. \tag{B.41}$$

For $\alpha_n^m = 1$ and $\beta_n^m = 1$,

$$(S|S)_{nn'}^{mm'}\left(\mathbf{y}^* - \mathbf{x}^*\right) = \frac{\beta_{(1)n'}^{m'}\alpha_{(1)(n'-n)}^{m-m'}}{\beta_{(1)n}^{m}} R_{n'-n}^{m-m'}\left(\mathbf{y}^* - \mathbf{x}^*\right), \tag{B.42}$$

$$(S|R)_{nn'}^{mm'}\left(\mathbf{y}^* - \mathbf{x}^*\right) = \frac{\alpha_{(1)n'}^{m'}\beta_{(1)(n+n')}^{m-m'}}{\beta_{(1)n}^{m}} S_{n+n'}^{m-m'}\left(\mathbf{y}^* - \mathbf{x}^*\right), \tag{B.43}$$

$$(R|R)_{nn'}^{mm'}\left(\mathbf{y}^* - \mathbf{x}^*\right) = \frac{\alpha_{(1)n'}^{m'}\alpha_{(1)(n-n')}^{m-m'}}{\alpha_{(1)n}^{m}} R_{n-n'}^{m-m'}\left(\mathbf{y}^* - \mathbf{x}^*\right). \tag{B.44}$$

B.5 Faster Translation Schemes

When the multipole and local expansions are truncated at $n = p - 1$, there are p^2 coefficients in the expansion. Performing a translation, therefore, requires $\mathrm{O}\left(p^4\right)$ using the matrix method in the previous two sections. For even moderately large p, this can be computationally prohibitive. As a result, many translation schemes have been developed that achieve considerably better computational complexity. In particular, the point and shoot method runs in $\mathrm{O}\left(p^3\right)$. A good overview of such methods can be found in [67].

Bibliography

[1] T. F. Eibert and V. Hansen. On the calculation of potential integrals for linear source distributions on triangular domains. *IEEE Transactions on Antennas and Propagation*, 43(12):1499 – 1502, December 1995.

[2] D. M. Hull, S. J. Vinci, and Yongming Zhang. ARL electric-field cage modeling, design and calibration. *IEEE Conference on Electromagnetic Field Computation*, 2006.

[3] Ross Adelman and David Hull. US Army Research Laboratory power-line UAV modeling and simulation (ARL-PLUMS) software tool. Technical Report ARL-TR-7456, Army Research Laboratory, September 2015.

[4] Zhenyu Zhang, Isaak D. Mayergoyz, Nail A. Gumerov, and Ramani Duraiswami. Numerical analysis of plasmon resonances in nanoparticles based on fast multipole method. *IEEE Transactions on Magnetics*, 43(4):1465 – 1468, April 2007.

[5] Richard A. Kerr. A quickie birth for Jupiters and Saturns. *Science*, 298(5599):1698 – 1699, November 2002.

[6] Nail A. Gumerov and Ramani Duraiswami. Efficient FMM accelerated vortex methods in three dimensions via the Lamb-Helmholtz decomposition. *Journal of Computational Physics*, 240:310 – 328, May 2013.

[7] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Transactions on Graphics*, 26, 2007.

[8] Beate Muth, Gnther Of, Peter Eberhard, and Olaf Steinbach. Collision detection for complicated polyhedra using the fast multipole method or ray crossing. *Archive of Applied Mechanics*, 77(7):503 – 521, July 2007.

[9] Eugen Skudrzyk. *The Foundations of Acoustics*. Springer-Verlag, New York, 1971.

[10] John David Jackson. *Classical Electrodynamics*. John Wiley & Sons, Inc., 1999.

[11] B. P. Abbott et al. Observation of gravitational waves from a binary black hole merger. *Physical Review Letters*, 116, 2016.

[12] Roger F. Harrington. *Field Computation by Moment Methods*. The Macmillan Company, New York, 1968.

[13] C. Pozrikids. *A Practical Guide to Boundary Element Methods with the Software Library BEMLIB*. Chapman & Hall/CRC, New York, 2002.

[14] Walton C. Gibson. *The Method of Moments in Electromagnetics*. Chapman & Hall/CRC, New York, 2008.

[15] A. J. Burton and G. F. Miller. The application of integral equation methods to the numerical solution of some exterior boundary-value problems. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 323(1553):201 – 210, 1971.

[16] Ramesh Natarajan. An iterative scheme for dense, complex-symmetric, linear systems in acoustics boundary-element computations. *SIAM J. Sci. Comput.*, 19(5):1450 – 1470, September 1998.

[17] O. O. Ademoyero, M. C. Bartholomew-Biggs, and A. J. Davies. Computational linear algebra issues in the Galerkin boundary element method. *Computers & Mathematics with Applications*, 42(10 - 11):1267 – 1283, November/December 2001.

[18] F. Vipiana, D. R. Wilton, and W. A. Johnson. Advanced numerical schemes for the accurate evaluation of 4-D reaction integrals in the method of moments. *IEEE Transactions on Antennas and Propagation*, 61(11):5559 – 5566, August 2013.

[19] M. M. Botha. A family of augmented Duffy transformations for near-singularity cancellation quadrature. *IEEE Transactions on Antennas and Propagation*, 61(6):3123 – 3134, March 2013.

[20] S. Lopez-Pena, A. G. Polimeridis, and J. R. Mosig. On the analytic-numeric treatment of weakly singular integrals on arbitrary polygonal domains. *Progress In Electromagnetics Research*, 117:339 – 355, 2011.

[21] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325 – 348, December 1987.

[22] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*, 155(2):468 – 498, November 1999.

[23] Nail A. Gumerov and Ramani Duraiswami. A broadband fast multipole accelerated boundary element method for the 3D Helmholtz equation. *J. Acoust. Soc. Am.*, 125(1):191 – 205, January 2009.

[24] G. Of, O. Steinbach, and P. Urthaler. Fast evaluation of volume potentials in boundary element methods. *SIAM J. Sci. Comput.*, 32(2):585 – 602, 2010.

[25] Anh Duc Pham, Saida Mouhoubi, Marc Bonnet, and Cyrille Chazallon. Fast multipole method applied to symmetric Galerkin boundary element method for 3D elasticity and fracture problems. *Engineering Analysis with Boundary Elements*, 36(12):1838 – 1847, December 2012.

[26] B. Michiels, J. Fostier, I. Bogaert, and D. De Zutter. Full-wave simulations of electromagnetic scattering problems with billions of unknowns. *IEEE Transactions on Antennas and Propagation*, 63(2):796 – 799, February 2015.

[27] http://www.cims.nyu.edu/cmcl/software.html.

[28] http://www.fastmultipole.org.

[29] P. A. Martin. Acoustic scattering and radiation problems, and the null-field method. *Wave Motion*, 4(4):391 – 408, October 1982.

[30] S. M. Rao, A. W. Glisson, D. Wilton, and B. Vidula. A simple numerical solution procedure for statics problems involving arbitrary-shaped surfaces. *IEEE Transactions on Antennas and Propagation*, 27(5):604 – 608, September 1979.

[31] S. M. Rao, D. Wilton, and A. W. Glisson. Electromagnetic scattering by surfaces of arbitrary shape. *IEEE Transactions on Antennas and Propagation*, 30(3):409 – 418, May 1982.

[32] D. R. Wilton, S. M. Rao, A. W. Glisson, D. H. Schaubert, O. Al-Bundak, and C. M. Butler. Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains. *IEEE Transactions on Antennas and Propagation*, 32(3):276 – 281, March 1984.

[33] K. Davey and S. Hinduja. Analytical integration of linear three-dimensional triangular elements in BEM. *Applied Mathematical Modelling*, 13(8):450 – 461, August 1989.

[34] R. D. Graglia. On the numerical integration of the linear shape functions times the 3-D Green's function or its gradient on a plane triangle. *IEEE Transactions on Antennas and Propagation*, 41(10):1448 – 1455, October 1993.

[35] Joseph Katz and Allen Plotkin. *Low-Speed Aerodynamics*. Cambridge University Press, New York, NY, 2001.

[36] Li Li and T. F. Eibert. Radial-angular singularity cancellation transformations derived by variable separation. *IEEE Transactions on Antennas and Propagation*, 64(1):189 – 200, January 2016.

[37] M. M. Botha. Numerical integration scheme for the near-singular green function gradient on general triangles. *IEEE Transactions on Antennas and Propagation*, 63(10):4435 – 4445, October 2015.

[38] Diane P. O'Leary. *Scientific Computing with Case Studies*. Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[39] John Gilbert and Donald Kershaw. *Large-Scale Matrix Problems and the Numerical Solution of Partial Differential Equations*. Clarendon Press, Oxford, 1994.

[40] Kenk A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, 2003.

[41] David S. Watkins. *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods*. Society for Industrial and Applied Mathematics, Philadelphia, 2007.

[42] Youcef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3), 1986.

[43] Ross Adelman, Nail A. Gumerov, and Ramani Duraiswami. Computation of Galerkin double surface integrals in the 3-D boundary element method. *IEEE Transactions on Antennas and Propagation*, 64(6):2389 – 2400, June 2016.

[44] A. Aimi and M. Diligenti. Hypersingular kernel integration in 3D Galerkin boundary element method. *Journal of Computational and Applied Mathematics*, 138(1):51 – 72, January 2002.

[45] Zhongde Wang, J. Volakis, K. Saitou, and K. Kurabayashi. Comparison of semi-analytical formulations and Gaussian-quadrature rules for quasi-static double-surface potential integrals. *IEEE Antennas and Propagation Magazine*, 45(6):96 – 102, December 2003.

[46] P. Arcioni, M. Bressan, and L. Perregrini. On the evaluation of the double surface integrals arising in the application of the boundary integral method to 3-D problems. *IEEE Transactions on Microwave Theory and Techniques*, 45(3):436 – 439, May 1997.

[47] L. J. Gray and T. Kaplan. 3D Galerkin integration without Stokes' theorem. *Engineering Analysis with Boundary Elements*, 25(4 - 5):289 – 295, April 2001.

[48] Alberto Salvadori. Analytical integrations of hypersingular kernel in 3D BEM problems. *Computer Methods in Applied Mechanics and Engineering*, 190(31):3957 – 3975, April 2001.

[49] S. Jarvenpaa, M. Taskinen, and P. Yla-Oijala. Singularity subtraction technique for high-order polynomial vector basis functions on planar triangles. *IEEE Transactions on Antennas and Propagation*, 54(1):42 – 49, January 2006.

[50] M. A. Khayat, D. R. Wilton, and P. W. Fink. An improved transformation and optimized sampling scheme for the numerical evaluation of singular and near-singular potentials. *IEEE Antennas and Wireless Propagation Letters*, 7:377 – 380, 2008.

[51] S. Nintcheu Fata and L. J. Gray. Semi-analytic integration of hypersingular Galerkin BIEs for three-dimensional potential problems. *Journal of Computational and Applied Mathematics*, 231(2):561 – 576, September 2009.

[52] F. Vipiana and D. R. Wilton. Numerical evaluation via singularity cancellation schemes of near-singular integrals involving the gradient of Helmholtz-type potentials. *IEEE Transactions on Antennas and Propagation*, 61(3):1255 – 1265, November 2012.

[53] Athanasios G. Polimeridis and Juan R. Mosig. Evaluation of weakly singular integrals via generalized Cartesian product rules based on the double exponential formula. *IEEE Transactions on Antennas and Propagation*, 58(6):1980 – 1988, March 2010.

[54] S. E. Mousavi and N. Sukumar. Generalized Duffy transformation for integrating vertex singularities. *Computational Mechanics*, 45(2 - 3):127 – 140, 2010.

[55] S. Nintcheu Fata. Semi-analytic treatment of nearly-singular Galerkin surface integrals. *Applied Numerical Mathematics*, 60(10):974 – 993, October 2010.

[56] A. G. Polimeridis, J. M. Tamayo, J. M. Rius, and J. R. Mosig. Fast and accurate computation of hypersingular integrals in Galerkin surface integral equation formulations via the direct evaluation method. *IEEE Transactions on Antennas and Propagation*, 59(6):2329 – 2340, April 2011.

[57] E. Lezar and D. B. Davidson. GPU-accelerated method of moments by example: monostatic scattering. *IEEE Antennas and Propagation Magazine*, 52(6):120 – 135, December 2010.

[58] Andrzej Kuzelewski and Eugeniusz Zieniuk. GPU-based acceleration of computations in elasticity problems solving by parametric integral equations system. *Advances in Engineering Software*, 79:27 – 35, January 2015.

[59] Toru Takahashi and Tsuyoshi Hamada. GPU-accelerated boundary element method for Helmholtz' equation in three dimensions. *International Journal for Numerical Methods in Engineering*, 80(10):1295 – 1321, December 2009.

[60] Christopher C. Chabalko and Balakumar Balachandran. Implementation and benchmarking of two-dimensional vortex interactions on a graphics processing unit. *Journal of Aerospace Information Systems*, 11(6):372 – 385, 2014.

[61] G. Of, O. Steinbach, and W. L. Wendland. Applications of a fast multipole Galerkin in boundary element method in linear elastostatics. *Computing and Visualization in Science*, 8(3 - 4):201 – 209, December 2005.

[62] J. N. Newman. Distributions of sources and normal dipoles over a quadrilateral panel. *Journal of Engineering Mathematics*, 20(2):113 – 126, 1986.

[63] K. Nabors, F. T. Korsmeyer, F. T. Leighton, and J. White. Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory. *SIAM Journal on Scientific Computing*, 15(3):713 – 735, 1993.

[64] John P. Barrett, Joseph A. Formaggio, and Thomas J. Corona. The spherical multipole expansion of a triangle. 2014. preprint, http://arxiv.org/abs/1403.5362.

[65] Nail A. Gumerov and Ramani Duraiswami. A method to compute periodic sums. *Journal of Computational Physics*, 272:307 – 326, September 2014.

[66] Shaozhong Deng. Quadrature Formulas in Two Dimensions, Spring 2010. University of North Carolina-Charlotte Math 5172 Lecture.

[67] N. A. Gumerov and R. Duraiswami. Comparison of the efficiency of translation operators used in the fast multipole method for the 3D Laplace equation. Technical Report CS TR 4701/UMIACS TR-2005-09, University of Maryland, College Park, April 2005.

[68] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 227(18):8290 – 8313, September 2008.

[69] Ross Adelman, Nail A. Gumerov, and Ramani Duraiswami. FMM/GPU-accelerated boundary element method for the 3-D Laplace equation. to be published.

[70] Qi Hu, Nail A. Gumerov, and Ramani Duraiswami. Scalable fast multipole methods on distributed heterogeneous architectures. *In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, 2011.

[71] Rio Yokota, L.A. Barba, Tetsu Narumi, and Kenji Yasuoka. Petascale turbulence simulation using a highly parallel fast multipole method on GPUs. *Computer Physics Communications*, 184(3):445 – 455, 2013.

[72] Nail A. Gumerov, Adam E. O'Donovan, Ramani Duraiswami, and Dmitry N. Zotkin. Computation of the head-related transfer function via the fast multipole accelerated boundary element method and its spherical harmonic representation. *J. Acoust. Soc. Am.*, 127:370 – 386, January 2010.

[73] Nail A. Gumerov, Ross Adelman, and Ramani Duraiswami. Fast multipole accelerated indirect boundary elements for the Helmholtz equation. *Proc. Mtgs. Acoust.*, 19, June 2013.

[74] Ross Adelman, Nail A. Gumerov, and Ramani Duraiswami. Semi-analytical computation of acoustic scattering by spheroids and disks. *J. Acoust. Soc. Am.*, 136:EL405 – EL410, December 2014.

[75] Lord Rayleigh. On the passage of waves through aperatures in plane screens, and allied problems. *Philos. Mag.*, 43(5), 1897.

[76] C. J. Bouwkamp. On the freely vibrating circular disk and the diffraction by circular disks and apertures. *Physica*, 16(1), 1950.

[77] C. Flammer. *Spheroidal wave functions*. Dover Publications, Mineola, 2005.

[78] R. D. Spence. The diffraction of sound by circular disks and aperatures. *J. Acoust. Soc. Am.*, 20(4), 1948.

[79] A. Storruste and H. Wergeland. On two complementary diffraction problems. *Phys. Rev.*, 73, 1948.

[80] A. Leitner. Diffraction of sound by a circular disk. *J. Acoust. Soc. Am.*, 21(4), 1949.

[81] G. Chertock. Sound radiation from prolate spheroids. *J. Acoust. Soc. Am.*, 33(7), 1961.

[82] T. B. A. Senior. The scattering from acoustically hard and soft prolate spheroids for axial incidence. *Can. J. Phys.*, 41, 1966.

[83] J. J. Bowman, T. B. A. Senior, and P. L. E. Uslenghi. *Electromagnetic and acoustic scattering by simple shapes*. Hemisphere Publishing Corporation, New York, 1987.

[84] J. P. Barton, N. L. Wolff, H. Zhang, and C. Tarawneh. Near-field calculations for a rigid spheroid with an arbitrary incident acoustic field. *J. Acoust. Soc. Am.*, 113(3), 2003.

[85] J. A. Roumeliotis, A. D. Kotsis, and G. Kolezas. Acoustic scattering by an impenetrable spheroid. *Acoustical Physics*, 53(4), 2007.

[86] http://www.github.com/radelman/scattering.

[87] Ross Adelman, Nail A. Gumerov, and Ramani Duraiswami. Software for computing the spheroidal wave functions using arbitrary precision arithmetic. 2014. preprint, http://arxiv.org/abs/1408.0074.

[88] D. Slepian. Some comments on Fourier analysis, uncertainty and modeling. *SIAM Review*, 25(3), July 1983.

[89] L. Fousse, G. Hanrot, V. Lefevre, P. Pelissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, 33(2), June 2007.

[90] T. Do-Nhat and R. H. MacPhie. Accurate values of prolate spheroidal radial functions of the second kind. *Can. J. Phys.*, 75, 1997.

[91] Hans Volkmer. Chapter 30: Spheroidal wave functions. *NIST Digital Library of Mathematical Functions*, 2016.

[92] S. Zhang and J. Jin. *Computation of special functions*. Wiley-Interscience, 1996.

[93] W. J. Thompson. *Atlas for computing mathematical functions*. John Wiley and Sons, 1997.

[94] W. J. Thompson. Spheroidal wave functions. *Computing in Science and Engineering*, 1(3), 1999.

[95] A. L. Van Buren and J. E. Boisvert. Accurate calculation of prolate spheroidal radial functions of the first kind and their first derivatives. *Quart. Appl. Math.*, 60, 2002.

[96] A. L. Van Buren and J. E. Boisvert. Improved calculation of prolate spheroidal radial functions of the second kind and their first derivatives. *Quart. Appl. Math.*, 62, 2004.

[97] L. Li, M. Leong, T. Yeo, P. Kooi, and K. Tan. Computations of spheroidal harmonics with complex arguments: a review with an algorithm. *Phys. Rev.*, 58, 1998.

[98] P. E. Falloon, P. C. Abbott, and J. B. Wang. Theory and computation of the spheroidal wave functions. *J. Physics A*, 36, 2003.

[99] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 2002.

[100] D. X. Ogburn, C. L. Waters, M. D. Sciffer, J. A. Hogan, and P. C. Abbott. A finite difference construction of the spheroidal wave functions. *Computer Physics Communications*, 185, 2014.

[101] Zhu Huang, Jianping Xiao, and John P. Boyd. Adaptive radial basis function and hermite function pseudospectral methods for computing eigenvalues of the prolate spheroidal wave equation for very large bandwidth parameter. *Journal of Computational Physics*, 281:269 – 284, January 2015.

[102] D. B. Hodge. Eigenvalues and eigenfunctions of the spheroidal wave equation. *J. Math. Phys.*, 11, 1970.

[103] F. W. J. Olver. Numerical solution of second-order linear difference equations. *Journal of Research of the National Bureau of Standards*, 71B, 1967.

[104] F. W. J. Olver and D. J. Sookne. Note on backward recurrence algorithms. *Mathematics of Computation*, 26(120), October 1972.

[105] Milton Abramowitz and Irene Ann Stegun. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables*. United States Department of Commerce/National Bureau of Standards, 1974.