FAST MULTIPOLE BOUNDARY ELEMENT SOLUTIONS WITH INEXACT KRYLOV ITERATIONS AND RELAXATION STRATEGIES

SIMON LAYTON

Dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

FAST MULTIPOLE BOUNDARY ELEMENT SOLUTIONS WITH INEXACT KRYLOV ITERATIONS AND RELAXATION STRATEGIES

by

SIMON LAYTON

B.Sc., University of Bristol, 2008
M.S., Boston University, 2011

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2013

Approved by

First Reader

Lorena Barba, PhD
Assistant Professor of Mechanical Engineering

Second Reader

Sheryl Grace, PhD
Associate Professor of Mechanical Engineering

Third Reader

Paul Barbone, PhD
Associate Professor of Mechanical Engineering

Fourth Reader

Emily Ryan, PhD
Assistant Professor of Mechanical Engineering

To my parents Dennis and Wendy, and my beloved fiancée Elizabeth

Acknowledgments

I would like to express wholehearted gratitude to my academic advisor over the last 6 years, Prof. Barba for her support and guidance throughout my studies in two universities, 3 degrees and 2 continents. In particular I would like to thank her for the belief she showed in me by bringing me from Bristol to Boston University in order to continue my studies. I deeply appreciate the freedom I have been given in my research, along with the invaluable advice and support that has been key to my successes.

I would also like to thank the members of my committee, Professors Paul Barbone, Sheryl Grace and Emily Ryan for their time and insights during these final months of my studies.

With the contributions, ideas and friendship of Christopher Cooper, Anush Krishnan, Brent Parham and Olivier Mesnard in Prof. Barba's research group, the time I have spent here has flown by, and they have always been willing to help in any way necessary, be it debugging code, reading papers or spending time relaxing away from the office. Special thanks must also go to Rio Yokota, who was always available for advice on coding, FMM or any other subject.

I also take this opportunity to thank those who have helped and funded me through my time at Boston University – Dan Kamalic for his patience in helping me with the use of his compute clusters. My continued funding has been greatly appreciated, and I would like to thank Boston University, the Massachusetts Green High-Performance Computing Center (MGHPCC), National Science Foundation (NSF) and the Office of Naval Research (ONR) for their contributions.

I would like to thank my fiancée, Elizabeth Matos for the love and support she has shown me, both pulling me through the last parts of my studies, and giving me an incredible future to look forward to. Finally, I wish to show my appreciation

for my parents, Dennis and Wendy Layton for their unending and unconditional encouragement and belief, for always being there for me, and for not begrudging me this time far away from them in Boston.

FAST MULTIPOLE BOUNDARY ELEMENT SOLUTIONS WITH INEXACT KRYLOV ITERATIONS AND RELAXATION STRATEGIES

SIMON LAYTON

Boston University, College of Engineering, 2013

Major Professor: Lorena A. Barba, PhD,
Assistant Professor of Mechanical Engineering

ABSTRACT

Boundary element methods (BEM) have been used for years to solve a multitude of engineering problems, ranging from Bioelectrostatics, to fluid flows over micro-electromechanical devices and deformations of cell membranes. Only requiring the discretization of a surface into panels rather than the entire domain, they effectively reduce the dimensionality of a problem by one.

This reduction in dimensionality nevertheless comes at a cost. BEM requires the solution of a large, dense linear system with each matrix element formed of an integral between two panels, often performed used an iterative solver based on Krylov subspace methods. This requires the repeated calculation of a matrix vector product that can be approximated using a hierarchical approximation known as the fast multipole method (FMM). While adding complexity, this reduces order of the time-to-solution from $\mathcal{O}(cN^2)$ to $\mathcal{O}(cN)$, where c is some function of the condition number of the dense matrix.

This thesis obtains algorithmic speedups for the solutions of FMM-BEM systems by applying the mathematical theory behind inexact matrix-vector products to our solver, implementing a relaxation scheme to control the error incurred by the FMM

in order to minimize the total time-to-solution. The theory is extensively verified for both Laplace equation and Stokes flow problems, with an investigation to determine how further problems may benefit from the addition of a relaxed solver. We also present experiments for the Stokes flow around both single and multiple red blood cells, an area of ongoing research, showing good speedups that would be applicable for any other code that chose to implement a similar relaxed solver. All of these results are obtained with an easy-to-use, extensible and open-source FMM-BEM code.

Contents

List of Figures

List of Abbreviations

BEM	.....	Boundary Element Method
FMM	.....	Fast Multipole Method
GMRES	.....	Generalized minimal residual
FGMRES	.....	Flexible generalized minimal residual
BLAS	.....	Basic Linear Algebra Subroutines
P2P	.....	Particle-to-Particle
P2M	.....	Particle-to-Multipole
M2M	.....	Multipole-to-Multipole
M2L	.....	Multipole-to-Local
L2L	.....	Local-to-Local
L2P	.....	Local-to-Particle

List of Algorithms

Chapter 1

Introduction

1.1 Introduction

As engineering problems and simulations become ever larger, and we continually look for new and better ways to simulate them, maximizing both the size of problem we can compute, and the speed at which we can obtain an answer. Problems become more complex as they move from 2 dimensional approximations to 3 dimensional simulations, leading to the natural question of how to both reduce the amount of work that has to be done, and how efficiently we can do it.

Boundary element methods (BEM) were introduced in the 1960s (standard introductory texts however, came later (Brebbia and Dominguez, 1992)) for a subset of problems, reducing the computational domain from the entirety of the needed physical space (volume / area) to the boundary of the region of interest (surface / curve), discretized into panels. The reduction in dimensionality changes a problem from 3 or 2 dimensions, to one dimension lower.

BEM's major downside is that it requires the solution of a dense system of linear equations, $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{R}^{N \times N}$ and N is the number of discretized panels. Due to the density of A, in the best case with an iterative solver, work and memory requirements for this solution scale as the size of the matrix, or in asymptotic notation, $\mathcal{O}(cN^2)$ time, where c is a function of the condition number of A, while a traditional dense solve, such as Gauss elimination would require $\mathcal{O}(N^3)$ time. This scaling is in comparison to (for instance) finite element, volume or difference codes, which still

require the solution of a linear system, but in their case A is sparse, allowing solutions to be found in $\mathcal{O}(cN)$ time, where N is now the number of degrees of freedom used, c is again a function of the condition number of A, and in general this N is significantly larger than the N for BEM problems. The $\mathcal{O}(cN^2)$ scaling causes BEM to become unusable for larger and more complex problems.

Initially introduced to quickly calculate potential fields for particle methods, Treecodes were developed in the late 1980s (Barnes and Hut, 1986). The computational domain is hierarchically split into near and far-field components, then far-field interactions are approximated with multipole expansions. These multipole expansions are faster than evaluating individual particles, but they are only convergent for distant targets. The near-field is still computed directly, preserving accuracy. Importantly, this splitting and approximation reduces the time taken to evaluate the interactions between N particles to $\mathcal{O}(N \log N)$. The fast multipole method (FMM) was introduced soon after as a further improvement, modifying the far-field evaluation by translating multipoles into local expansions (Greengard, 1987) that are accurate close to the expansion center and evaluating them instead of directly using the multipoles. This translation further reduced the complexity to the optimal $\mathcal{O}(N)$. The FMM was later named as one of the top 10 scientific algorithms of the 20th century (Dongarra and Sullivan, 2000).

Several years later, the potential of using fast methods to accelerate BEM was realized for the Laplace(Nabors et al., 1994) and Stokes (Gomez and Powert, 1997) equations. By using a fast method within an iterative solution of the BEM linear system, the scaling was reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(cN)$, making larger simulations feasible. While this modification adds complexity to the BEM, it allows users to perform much larger experiments, putting them on a more level playing field to more traditional computational methods, such as finite elements.

Parallel to the development of BEM and FMM, the linear algebra community was working on iterative solvers. The widely used GMRES algorithm (Saad and Schultz, 1986), published in 1986, iteratively solves $Ax = b$ for a general $A \in \mathbb{R}^{N \times N}$, by generating a Krylov subspace through repeated matrix-vector multiplication, and finding the vector x_k within this space that minimises $||Ax_k - b||$. The cost of this method scales as the cost of the product of computing the matrix-vector products and a function of the condition number of A, denoted as c, thus when used in a standard BEM the cost would be $\mathcal{O}(cN^2)$, while for the FMM-BEM, the cost is $\mathcal{O}(cN)$.

Where the iterative solution of $Ax = b$ dominates solution time, methods have been proposed to reduce both the number of iterations needed, and the cost of each of those iterations. In general the convergence rate of a solver depends on the condition number of A. The most commonly used way to reduce the condition number is to use a preconditioner. These can be as simple as using the inverse of the diagonal of A, or as complex as performing a complete additional linear solve.

The theory behind inexact matrix-vector products in the context of Krylov solvers (Simoncini and Szyld, 2003; Bouras and Frayssé, 2005) was initially developed to explain experimental results where error was introduced into the matvec within a solver, yet the correct results were obtained with no loss of convergence. While this concept is well developed within the linear algebra community, there is no work that we are aware of on applying the theory to FMM-BEM problems by *relaxing* the accuracy of the FMM-accelerated matvecs.

1.2 Objective

This thesis aims to apply current inexact Krylov and relaxation method theory to the field of FMM-BEM problems. To achieve this goal we first perform extensive verification tests on a comparatively simple test problem, before using this problem to

thoroughly examine the performance characteristics of utilizing a relaxation method. These experiments allow us to provide guidelines to help identify more potential problems for the use of a relaxed solver. Next, we verify the relaxed solver on a more complicated example, then use a problem within the engineering field to illustrate the practical performance gains of relaxation methods.

1.3 Present Contribution

This study presents the following work

1. An extendible, easy-to-use modern code for FMM, BEM and FMM-BEM problems, complete with both exact and inexact solvers, simple preconditioners, relaxation schemes and the ability to solve multiple kinds of problems including the Laplace and Stokes equations. This includes the first fully open-source FMM-BEM solver for the Stokes equation.

2. Use and verification of inexact Krylov solver theory with FMM-BEM problems for the first time. This study also provides guidelines for identifying problems that might benefit from relaxation schemes.

3. Extensive performance testing for both simple (Laplace) and more complex (Stokes) BEM problems, including the field of blood flow, showing speedups from the use of a relaxed solver.

1.4 Contents of the Thesis

This thesis is organized as follows. In chapter 2, some background is introduced for the component methods of this research work: BEM, FMM, Krylov iterative solvers and inexact matrix-vector products. Next, in chapter 3, we discuss the implemented

numerical methods that have been used in the current work, concentrating on numerical integrals along with a relaxation scheme and preconditioners. Chapter 4 discusses some implementation-specific details for the FMM used in this thesis. We then use the techniques from chapters 3 and 4 to problems involving the Laplace equation in chapter 5, presenting convergence tests, relaxation verification and performance results. Using a Laplace problem we experiment further with the parameter space of relaxation, obtaining some general rules by which one can determine whether a given problem will benefit from a relaxed solver. In chapter 6 these rules are applied to a more significant problem, Stokes flow. Experiments demonstrate both convergence of the method for this new equation and speedups with the relaxed solver. The final results-based section, chapter 7, explores the field of blood flow using the solver from chapter 6 to explore the performance of our relaxed solver for this more complex and relevant engineering problem. Finally in chapter 8, we discuss the expected impact of our completed work, along with the next steps that could be taken.

Chapter 2

Background

2.1 Boundary Element Methods (BEM)

The boundary element method (BEM) is based around a technique to solve linear partial differential equations by formulating them as boundary integrals. This formulation requires that only the surface of a body of interest (such as an object in a medium) be discretized, and the solution is obtained on that surface. As a post-processing step, this solution can then be computed both inside and outside the body. The lower discretization requirement is an advantage over traditional grid-based methods that require the entire domain to be meshed. Furthermore, regardless of infinite or bounded domains, the solution process and discretization is exactly the same.

For example, we show the boundary integral form of the Laplace equation, $\nabla^2\phi = 0$ — this will be derived much later in §5.1, but for now it is useful to see the form of the equation, where $G = 1/4\pi r$ is the free-space Green's function for the Laplace equation.

$$\frac{1}{2}\phi + \int_\Gamma \phi \frac{\partial G}{\partial \hat{n}} \, \mathrm{d}\Gamma = \int_\Gamma \frac{\partial \phi}{\partial \hat{n}} G \, \mathrm{d}\Gamma. \tag{2.1}$$

We can see that all terms involved are now only on the boundary, Γ, and when we eventually form a linear equation, the unknowns will be boundary values of ϕ and $\partial\phi/\partial\hat{n}$.

The exact manner in which we choose to solve equation (2.1) can now be discussed. First, Γ is dicretized into *panels*. Two pieces of nomenclature are now introduced – we integrate over a *source panel*, j with respect to a *target*, i, that can be a single point in space or a panel, depending on the formulation of BEM used. The 2 main families of methods for BEM, are *collocation* and *Galerkin* formulations. Collocation explicitly enforces equation (2.1) at a discrete set of points, $\mathbf{x}_i$, generally the centers of the target elements. By contrast, Galerkin formulations enforce the solution in a weighted average sense over an element, Γ_i. This is done by multiplying the function over the target panel with a test / shape basis function and performing an additional integral over the panel. Collocation BEM can be obtained from a Galerkin formulation by choosing a Dirac delta function as the test function.

Finally, BEM can use many different types of panel, each giving different convergence and error properties. We now describe two of the most commonly used panel types, *constant* and *linear*. (Brebbia and Dominguez, 1992)

1. *Constant* – Values (such as ϕ and $\partial\phi/\partial\hat{n}$) are considered to be constant across the entirety of the element. This is the most basic method, and allows the surface integrals encountered in BEM to be simplified greatly. For example, the integral $\int_\Gamma \phi \partial G/\partial\hat{n}\ \mathrm{d}\Gamma$ from equation (2.1) would be calculated as $\phi \int_\Gamma \partial G/\partial\hat{n}\ \mathrm{d}\Gamma$, as the value of ϕ is unchanging across the panel.

2. *Linear* – Values vary linearly across the panel. More complex than constant elements, values must be calculated at every vertex of every panel in the domain, resulting in a larger linear system. The trade-off however, is that fewer panels should be needed to obtain the same quality of results as constant elements.

There are many more types of panels that can be used for BEM, varying in both the geometric sense, being curved or flat, and in terms of the distribution of values over the panel, such as the constant and linear distributions described above.

In this work, we use a collocation method with constant panels for all experiments, in keeping with FMM-BEM studies such as (Liu, 2009) — the extension of FMM-BEM to linear elements is rare in the literature, and does not appear to be a fully-developed field.

We discretize equation (2.1) by exchanging integrals over Γ to sums over N discretized panels, Γ_j and take advantage of constant panels by moving the specified values, ϕ_j and $\partial\phi/\partial\hat{n}$ outside the surface integrals to get equation (2.2).

$$\frac{1}{2}\phi_i - \sum_{j=1}^{N} \phi_j \int_{\Gamma_j} \frac{\partial G_{ij}}{\partial \hat{n}} \, \mathrm{d}\Gamma' = \sum_{j=1}^{N} \frac{\partial \phi_j}{\partial \hat{n}_j} \int_{\Gamma_j} G_{ij} \, \mathrm{d}\Gamma' \tag{2.2}$$

From this form, solving for ϕ with a set $\partial\phi/\partial\hat{n}$ is known as a *second-kind* equation, and solving for $\partial\phi/\partial\hat{n}$ for set ϕ is a *first-kind* equation.

Given either ϕ or $\partial\phi/\partial\hat{n}$ for each panel in equation (2.2) we assemble the dense linear system

$$A\mathbf{x} = \mathbf{b}, \tag{2.3}$$

where for constant elements, A_{ij} is either $\int_{\Gamma_j} \partial G_{ij}/\partial\hat{n} \, \mathrm{d}\Gamma'$ or $\int_{\Gamma_j} G_{ij} \, \mathrm{d}\Gamma'$ depending on whether ϕ or $\partial\phi/\partial\hat{n}$ respectively is known on panel j.

The techniques used for solving this form of equation quickly will be described throughout the next chapters of this thesis, including the evaluations of the different integrals, in §3.1. However, it is useful to note at this point that for target points far away from a source panel we can approximate the surface integrals using Gauss quadrature, turning them into a simple repeated evaluation of G or $\partial G/\partial\hat{n}$. The influence of all panels at a chosen target can then be thought of as an N-body sum.

2.2 Krylov Subspace Methods

In many applications, including BEM, it is not feasible to solve the linear system $Ax = b$ in a direct fashion, due to either time or memory constraints. In these situations, iterative solvers are used, many of which (including conjugate gradient (CG) and generalized minimal residual (GMRES)) are based on Krylov subspaces. A Krylov subspace is a linear subspace which is spanned by the images of b under powers of A. An *order-r* subspace is comprised of the first r images, denoted by:

$$K_r(A, b) = \text{span}\{b, Ab, A^2b, ..., A^{r-1}b\}. \tag{2.4}$$

This family of methods derives from the Cayley-Hamilton theorem, which states that you can express the inverse of a matrix A, as a linear combination of its powers.

As the vectors in K_r can be close to linearly dependant, an orthogonalization scheme is always used, such as the Arnoldi method in GMRES(Saad and Schultz, 1986).

We choose to use GMRES as our Krylov solver instead of the other commonly used non-symmetric method, BICGSTAB for one simple reason — we expect the most expensive part of each iteration to be the matrix-vector product, and BICGSTAB requires two such products to GMRES' one. However, the same relaxation principles explored later should be applicable to any Krylov solver.

A pseudocode implementation of a canonical right-preconditioned GMRES (Saad and Schultz, 1986) is given in algorithm 1.

From this description of the GMRES algorithm, and applying it to the BEM, we can see that (as expected) the greatest cost per iteration is going to be the matrix vector product (matvec), $w \leftarrow A \cdot x$, taking $\mathcal{O}(N^2)$ time in a naive implementation. We now look to accelerate these matvecs using the fast multipole method, taking advantage of the BEM matrix's form, intending to reduce the cost of each matvec to $\mathcal{O}(N)$ time.

Algorithm 1 Right-Preconditioned GMRES (Saad and Schultz, 1986)

Require: Matrix A, initial guess x_0, right-hand side b, desired tolerance ϵ, order of the Krylov space k, preconditioner M.
Initialize $\bar{H}_m \in \mathbb{R}^{(k+1)\times k} = 0 \; \forall i, j$
$r_0 \leftarrow A \cdot x_0 - b$
$\beta \leftarrow ||r_0||_2, \;\; v_1 \leftarrow r_0/\beta$
for j=1,...,k **do**
  $z_j \leftarrow M^{-1} \cdot v_j$
  $w \leftarrow A \cdot z_j$
  for i=1,..,j **do**
    $h_{i,j} \leftarrow (w, v_i)$
    $w \leftarrow w - h_{i,j} \cdot v_i$
  $h_{j+1,j} \leftarrow ||w||_2$
  $v_{j+1} \leftarrow w/h_{j+1,j}$
$V_k \leftarrow [v_1, ..., v_k]$
$y_k = \arg\min_y ||\beta e_1 - \bar{H}_k y||_2$ and $e_1 = [1, 0, ..., 0]$
$x_k \leftarrow x_0 + M^{-1} V_k y_k$

2.3 Fast Multipole Methods (FMM)

The fast evaluation of large particle fields that decay in space has been a major computational problem in many disciplines. A non-definitive list would include Molecular dynamics (Plimpton, 1995), astrophysics simulations (Barnes and Hut, 1986) and vortex methods for computational fluids (Yokota and Barba, 2013). However, due to the $\mathcal{O}(N^2)$ computational complexity of directly performing this evaluation, equation (2.5), simulations beyond a certain size were impossible.

$$\phi_i = \sum_{j=0,\, j\neq i}^{N} q_i \cdot \mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) \;\; = \;\; \sum_{j=0}^{N} \mathbb{K}_{ij} q_j, \;\; \mathbb{K}_{i,j} = \mathbb{K}(\mathbf{x}_i, \mathbf{x}_j) \tag{2.5}$$

In the majority of cases, $\mathbb{K}$ represents a *kernel*, generally the Green's function of an equation, for instance Laplace ($\nabla^2 \mathbf{u} = 0$) or Helmholtz ($\nabla^2 \mathbf{u} + k^2 \mathbf{u} = 0$). Example kernels for Laplace, Yukawa and Helmholtz are shown in equations (2.6), (2.7) and (2.8).

$$\mathbb{K}_{\text{Laplace}} = \frac{1}{4\pi|\mathbf{x}_i - \mathbf{x}_j|} \tag{2.6}$$

$$\mathbb{K}_{\text{Yukawa}} = \frac{e^{-k|\mathbf{x}_i - \mathbf{x}_j|}}{4\pi|\mathbf{x}_i - \mathbf{x}_j|} \tag{2.7}$$

$$\mathbb{K}_{\text{Helmholtz}} = \frac{e^{-ik|\mathbf{x}_i - \mathbf{x}_j|}}{4\pi|\mathbf{x}_i - \mathbf{x}_j|} \tag{2.8}$$

This is a good place to note that the Green's functions from BEM formulations from §2.1, G and $\partial G/\partial \hat{n}$ can often be expressed in terms of a kernel (such as one of equations (2.6-2.8)), and so we can approximate the BEM matrix A and accelerate the repeated matrix-vector products in an iterative solve using an FMM.

The first fast method was introduced for the gravitational potential, equation (2.6), based on splitting decaying kernels into two components representing the near and far-field (Barnes and Hut, 1986):

$$\phi_i = \phi_{near} + \phi_{far}, \quad \phi_{near} \gg \phi_{far}. \tag{2.9}$$

This splitting allows us to approximate the smaller ϕ_{far} term by choosing a truncation parameter, p which dictates the accuracy of a series expansion representing sources in the far-field, and θ_{MAC}, a metric to determine whether a source belongs to the near or far-field. For accuracy, direct evaluation is carried out in the near-field. The approach relies on adaptively decomposing the space hierarchically into clusters (quadtree in 2D, octtree in 3D) as shown in 2D in figure 2·1, and approximating the particles contained within each cluster with a multipole expansion which only converges far from the cluster center. Next, these multipoles are shifted and accumulated up the tree from child to parent clusters, until every cluster holds an expansion representing all sources contained by itself and its descendants.

The tree structure is then traversed, summing influences of near clusters directly

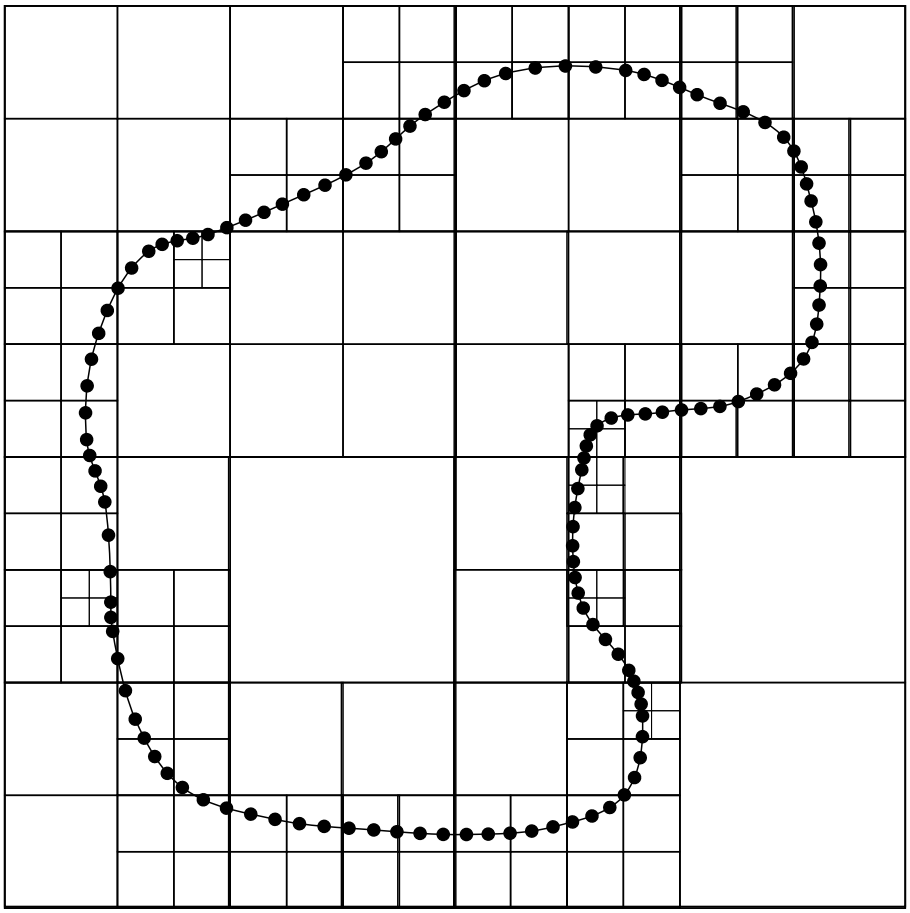


Figure 2·1: Adaptive quadtree in 2 dimensions

(particle-particle) and evaluating the multipole approximations of distant clusters (cluster-particle). This delineation is controlled by a *multipole acceptance criteria*, where a specified θ_{MAC} is compared to $\theta = b/d$, where d is the distance between clusters and b is a function of the size of the interacting clusters. If $\theta < \theta_{\text{MAC}}$ then the interaction is classed as short-range. This approach, represented by figure 2·2 gives a computational complexity of $\mathcal{O}(N \log N)$, enabling large calculations to be run in a reasonable timeframe.

This method was later extended into the fast multipole method (FMM) by Greengard (Greengard, 1987) to add local expansions which converge within a short distance of the expansion center. Instead of directly evaluating the multipole expansions of far

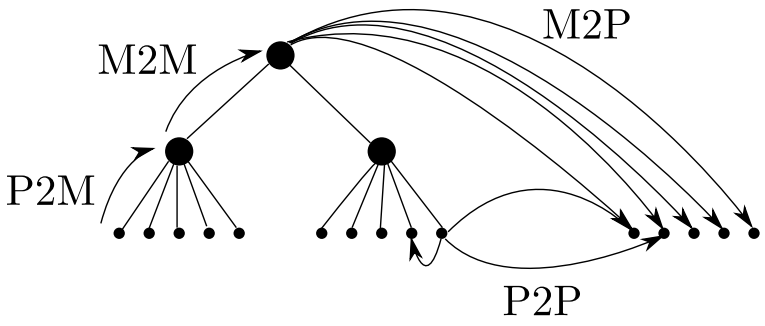


Figure 2·2: Treecode

clusters, the multipoles are translated and converted into local expansions (cluster-cluster), before being translated to the lowest nodes of the tree and evaluated against target points. Due to this modification, the overall complexity was reduced to the optimal $\mathcal{O}(N)$. A graphical representation of the algorithm is given in figure 2·3. Later, support for other kernels, including Yukawa (Boschitsch et al., 1999; Huang et al., 2009) and Helmholtz (Greengard et al., 1998) were added.

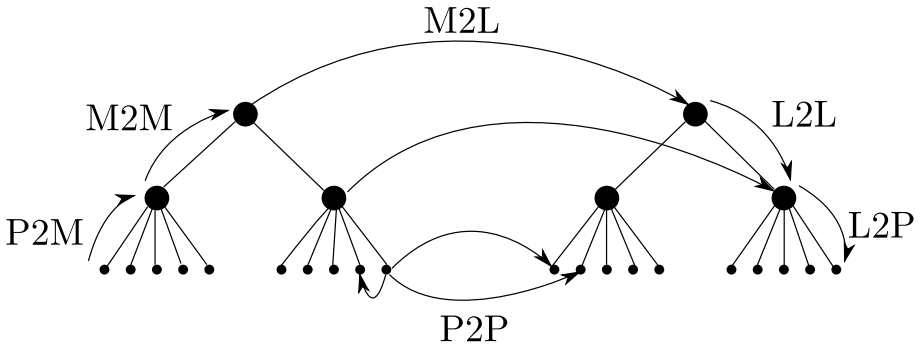


Figure 2·3: FMM

The FMM can be used for any problem that has an n-body sum of the form of equation (2.5) and a suitable Green's function. During each iterative solve step within the BEM we repeatedly compute sums of this kind due to evaluating integrals using Gauss quadrature — repeated Green's function evaluations. Therefore, the FMM can

be used to accelerate this process.

FMM and treecodes can be thought of as having 7 separate operators (or kernels) described in §2.3.1, and 3 distinct "phases", summarised below in §2.3.2. How these operators pertain to a tree structure is shown in figures 2·2 and 2·3 for treecodes and FMMs respectively.

2.3.1 Operators

There are 7 separate operators that constitute a standard FMM, each performing a distinct operation, from directly evaluating the influence from one particle on another, to creating a multipole expansion, to translating series expansions up, down or across the tree. These operators are summarised below.

- *Particle to Particle (P2P)* – Calculate directly (using an $\mathcal{O}(N^2)$ method) the interactions between source and target particles.
- *Particle to Multipole (P2M)* – Generate a multipole expansion about a cluster center of all source particles contained within that cluster.
- *Multipole to Multipole (M2M)* – Translate a multipole expansion by shifting it to a different center.
- *Multipole to Particle (M2P)* – Evaluate a multipole expansion at a *well-separated* target particle.
- *Multipole to Local (M2L)* – Translate a multipole expansion to a *well-separated* center point, and convert into a local expansion.
- *Local to Local (L2L)* – Translate a local expansion to a different center.
- *Local to Particle (L2P)* – Evaluate a local expansion at nearby particles.

2.3.2 Phases of the FMM

To perform a standard FMM, we require 4 distinct phases, performed in order. Each of these phases is shown below, along with pseudocode for each phase.

- *Tree Creation*, (algorithm 2) – The domain is partitioned hierarchically based on N_{CRIT}. Links between cells and their parents / children are created at this point for fast traversal later.

Algorithm 2 Tree Construction

```
Require: Maximum # of particles per cluster NCRIT, root cluster
  root = root_cluster
  for sources, j do
      ADD_TO_CLUSTER(j, root)

  procedure ADD_TO_CLUSTER(point, cluster)
      if NUM_POINTS(cluster) < NCRIT then
          cluster.APPEND(point)
      else
          if not cluster.split then
              SPLIT(cluster)                          ▷ Split into 8 child clusters
          oct = OCTANT(point, cluster)                ▷ Get correct child cluster
          ADD_TO_CLUSTER(point, cluster.child[oct])
```

- *Upward*, (algorithm 3) – Multipole expansions are created at all leaf (lowest level) clusters (P2M). Then, expansions are translated up the tree to parent clusters (M2M). At the end of this stage, all clusters contain a multipole representation of all particles approximated by themselves and their descendants.

- *Interaction*, (algorithm 4) – Both long and short range interactions are calculated, based on interacting source and target clusters against each other. If the ratio $\theta = b/d$, where d is the distance between 2 cluster centers, and b is a function of the cluster sizes is less than some specified θ_{MAC}, then direct particle-particle (P2P) evaluation is used. Otherwise, either cluster-particle (M2P) in

Algorithm 3 Upward sweep

```
Require: Cells C, kernel K.
  for i=1, ..., SIZE(C) do
    if IS_LEAF(C_i) then
      K.P2M(points(C_i))          ▷ Create multipole expansions at leaf cells
    else
      K.M2M(C_i, parent(C_i))     ▷ Translate multipole expansions up the tree
```

the case of a treecode, or cluster-cluster (M2L) operators in the case of an FMM are evaluated.

- *Downward*, (algorithm 5) – For a treecode, this phase is empty. For an FMM, local expansions are propagated downwards through the tree (L2L), passing from a parent cluster to its children. At the leaf nodes, these local expansions are evaluated at all target points within each relevant cluster (L2P).

2.3.3 Computational Complexity

While we've already stated that the FMM has $\mathcal{O}(N)$ scaling, we will now demonstrate it by breaking down the algorithm into distinct sections, giving the complexity of each section and thus the total complexity.

First of all, we define some useful values:

N	Number of particles
l	Number of levels in the Octree, $\approx \log_8 N$
p	Truncation point of expansions
$c_{\text{P2M}}(p)$	Cost of a creating multipole expansion (single source)
$c_{\text{M2M}}(p)$	Cost of a single multipole-multipole translation
$c_{\text{M2L}}(p)$	Cost of a single multipole-local translation
$c_{\text{L2L}}(p)$	Cost of a single local-local translation
$c_{\text{L2P}}(p)$	Cost of evaluating a local expansion (single target)
N_{M2L}	Number of M2L translations (= 189)

Using these, we can form an estimate for the total runtime of the FMM in terms of the cost of our expansions, which will then be specialized for a pair of different

Algorithm 4 Interaction Stage

```
pairQ = [ROOT(source), ROOT(target)]
while not EMPTY(pairQ) do
    pair = POP_FRONT(pairQ)
    b_1, b_2 = pair.first, pair.second
    if IS_LEAF(b_1) then
        if IS_LEAF(b_2) then                          ▷ Both leaves – direct evaluation
            P2P(b_1, b_2)
        else
            for bc in CHILDREN(b_2) do
                INTERACT(b_1, bc)
    else if IS_LEAF(b_2) then
        for bc in CHILDREN(b_1) do
            INTERACT(bc, b_2)
    else
        INTERACT(...)                          ▷ Iterate over smaller box's children

procedure INTERACT(Box_1, Box_2, pairQ)                          ▷ Interact two boxes
    if ACCEPT_MAC(Box_1, Box_2) then                             ▷ Long-range interactions
        if FMM then
            M2L(Box_1, Box_2)          ▷ Translate and convert multipole expansion
        if TREECODE then
            M2P(Box_1, Box_2)                    ▷ Evaluate multipole at far points
    else
        pairQ.PUSH_BACK(pair(Box_1, Box_2))      ▷ Not accepted, defer for splitting
```

Algorithm 5 Downward Sweep

```
for all clusters, c do
    if not IS_LEAF(c) then
        for ci in CHILDREN(c) do
            L2L(c, ci)                  ▷ Translate local expansions down the tree
    else
        L2P(c)                                ▷ Evaluate local expansions at leaf cells
```

expansion types.

$$\begin{aligned} C_{\text{FMM}} &= N \cdot c_{\text{P2M}}(p) \\ &+ N \cdot c_{\text{M2M}}(p) \\ &+ N \cdot N_{\text{M2L}} \cdot c_{\text{M2L}}(p) \\ &+ N \cdot c_{\text{L2L}}(p) \\ &+ N \cdot c_{\text{L2P}}(p) \\ &+ N \cdot 27 \cdot N_{\text{CRIT}}^2 \end{aligned} \tag{2.10}$$

Clearly this shows that the total complexity is $\mathcal{O}(N)$ as long as $p \neq p(N)$, something that holds for all applications discussed in this thesis, and $N_{\text{CRIT}} \ll N$, which holds for any reasonably created tree. (A notable application where $p = p(N)$ would be acoustics, using expansions for the Helmholtz Green's function). With a standard FMM formulation, $N_{\text{M2L}} = 189$. We can look at two examples, firstly Cartesian expansions, equation (2.11)

$$\phi(\mathbf{x}_i) = \sum_{||\mathbf{k}||=0}^{\mathbf{P}} \frac{1}{\mathbf{k}!} D_y^k(\mathbf{x_i}, \mathbf{y}) \psi(\mathbf{y}_j - \mathbf{y}_c)^{\mathbf{k}}, \tag{2.11}$$

where $\mathbf{k}$ is a multi-index variable, $\mathbf{k} = \{k_1, k_2, k_3\}$, $\mathbf{k}! = k_1!k_2!k_3!$, $\mathbf{y}^k = y_1^{k_1} y_2^{k_2} y_3^{k_3}$ and $D_y^k = D_{y_1}^{k_1} D_{y_2}^{k_2} D_{y_3}^{k_3}$ is the derivative operator. The other expansion we use as an example are spherical harmonic expansions, shown for Laplace's equation in equation (2.12).

$$\phi(\mathbf{x}_i) = \sum_{n=0}^{p} \sum_{m=-n}^{n} \frac{Y_n^m(\theta_i, \phi_i)}{r_i^{n+1}} \left\{ \sum_j^N q_j \rho_j^n Y_n^{-m}(\alpha_i, \beta_i) \right\} \tag{2.12}$$

Operation	Cost (Cartesian)	Cost (Spherical)
c_{P2M}	$\mathcal{O}(p^3)$	$\mathcal{O}(p^2)$
c_{M2M}	$\mathcal{O}(p^6)$	$\mathcal{O}(p^4)$
c_{M2L}	$\mathcal{O}(p^6)$	$\mathcal{O}(p^4)$
c_{L2L}	$\mathcal{O}(p^6)$	$\mathcal{O}(p^4)$
c_{L2P}	$\mathcal{O}(p^3)$	$\mathcal{O}(p^2)$

When we apply these costs to equation (2.10) we can immediately see that the most expensive expansion related cost will be the M2L translations due to the large constant multiplier, given by $N \cdot N_{\text{M2L}} \cdot \mathcal{O}(p^4)$ for spherical expansions and $N \cdot N_{\text{M2L}} \cdot \mathcal{O}(p^6)$ for cartesian. The other most expensive part of the evaluation will be the direct near-field computation, given by $N \cdot 27 \cdot N_{\text{CRIT}}^2$. The fastest FMM will usually involve trying to create a tree that keeps these two costs as equal as possible, known as *well-balanced*. This balance comes from trying to ensure that N_{CRIT} does not become so large that the algorithm starts to scale as $\mathcal{O}(N^2)$, while making sure that the tree does not have so many levels that the number of M2L translations becomes too large and dominates the run time. The high cost of M2L translations is going to be one of the main motivators for the relaxation schemes introduced later in §2.5.

2.4 Preconditioners

In all the experiments performed in this work, an important metric is the number of iterations taken by our linear solver to converge to a solution. Given the cost of each iteration, this number will also heavily influence the total time-to-solution. The number of iterations will be most directly influenced by the *condition number* of the matrix, $\kappa(A)$, given here for a normal matrix

$$\kappa(A) = \left| \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \right|, \tag{2.13}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of A respectively. As the ratio of these eigenvalues gets closer to 1, the matrix becomes more *well-conditioned*

and easier to solve (i.e. less solver iterations are needed).

In the context of Krylov solvers, one way to try and reduce the condition number is to apply a preconditioner, indicated in algorithm 1 by the step $Mz_j = v_j$ or

$$z_j \leftarrow M^{-1} \cdot v_j, \tag{2.14}$$

with M as some approximation to A. A simple example of M is simply the diagonal of A, that is,

$$M(A) = \text{diag}(A) \tag{2.15}$$

In this case, we can explicitly form M^{-1}, and applying the preconditioner is a simple element-wise multiplication between M^{-1} and v_j (`axpy` in standard dense linear algebra notation).

In general we may have to solve the $Mz_j = v_j$ system using a choice of linear solver (including another Krylov solver). The only additional overhead for this kind of so-call "flexible" preconditioner (Saad, 1993) from the viewpoint of the GMRES solver, is a slightly increased memory footprint. There are multiple ways of modifying GMRES to accommodate this type of preconditioning; The one used in this thesis is FGMRES, algorithm 6 (Saad, 1993), simply GMRES as described in algorithm 1, with two small changes: $V_k \leftarrow [v_1, ..., v_k]$ changes to $Z_k \leftarrow [z_1, ..., z_k]$ and we use Z_k instead of V_k when updating the approximation x_k. The FGMRES algorithm is shown in pseudocode as algorithm 6.

Notice that the main cost per iteration is still the matvec $w \leftarrow A \cdot z_j$, so the additional overhead of using the FGMRES variant is insignificant.

Algorithm 6 Right-Preconditioned FGMRES (Saad, 1993)

Require: Matrix A, initial guess x_0, right-hand side b, desired tolerance ϵ, order of the Krylov space k, preconditioner M.
Initialize $\bar{H}_m \in \mathbb{R}^{(k+1)\times k} = 0 \; \forall i, j$
$r_0 \leftarrow A \cdot x_0 - b$
$\beta \leftarrow ||r_0||_2, \;\; v_1 \leftarrow r_0/\beta$
for j=1,...,k **do**
  $z_j \leftarrow M^{-1} \cdot v_j$
  $w \leftarrow A \cdot z_j$
  for i=1,..,j **do**
    $h_{i,j} \leftarrow (w, v_i)$
    $w \leftarrow w - h_{i,j} \cdot v_i$
  $h_{j+1,j} \leftarrow ||w||_2$
  $v_{j+1} \leftarrow w/h_{j+1,j}$
$Z_k \leftarrow [z_1, ..., z_k]$
$y_k = \arg\min_y ||\beta e_1 - \bar{H}_k y||_2$ and $e_1 = [1, 0, ..., 0]$
$x_k \leftarrow x_0 + Z_k y_k$

2.5 Inexact Matrix-Vector Products

Consider the case where a matrix-vector product is not computed exactly, such as when using FMM within GMRES or FGMRES. In this situation, $y \leftarrow A \cdot x$ becomes $y \leftarrow (A + \epsilon_A) \cdot x$ with ϵ_A representing the errors induced by the FMM approximation. It can be shown (Simoncini and Szyld, 2003) that not only will we still converge to the correct solution of the linear system, but that $||\epsilon_A||$ can grow as the residual becomes closer to the final desired tolerance.

By introducing an error term on the i-th iteration to our matrix A, E_i, instead of the standard Arnoldi iteration used in GMRES, $AV_m = V_{m+1}H_m$, where H_m is the upper-Hessenberg matrix produced from the Arnoldi iterations and $V_m = [v_1, v_2, \cdots, v_m]$, we use the *inexact Arnoldi* iteration,

$$[(A + E_1)v_1, \cdots, (A + E_m)v_m] = [v_1, \cdots, v_m]H_m. \tag{2.16}$$

Assuming that the calculation of the initial residual, $r_0 = b - Ax_0$ is exact, and $\beta = ||r_0||$, we can write the m-th iteration of the GMRES procedure as $x_m = x_0 + \delta_m x$, with $\delta_m x = V_m y_m$ and y_m is the solution to $\min_y ||H_m y - \beta e_1||$. We diverge from the standard exact GMRES by introducing a perturbation matrix, $G_m = [E_1 v_1, \cdots, E_m v_m]$. From here, we can write the inexact Arnoldi iteration as an exact Arnoldi iteration with

$$(A + G_m V_m^T) V_m = V_{m+1} H_m \tag{2.17}$$

This shows that the quantities calculated in the first i steps of an inexact Arnoldi iteration are exactly the same as the first i steps of an exact iteration with $A = (A + G_m V M^T)$. From this, we can use the standard GMRES results to show that the residual r_m will decrease as i grows, establishing the convergence of our inexact GMRES.

The convergence of an inexact GMRES solver has also been shown experimentally for dense linear algebra (Van den Eshof and Sleijpen, 2004; Bouras and Frayssé, 2005; Simoncini and Szyld, 2003), and was implemented by explicitly adding a perturbation to A, such that $A' \leftarrow A + A_\epsilon$ and $||A_\epsilon||$ is controlled as part of the relaxation strategy. A different approach (Sidje and Winkles, 2011) has been used for sparse matrices, namely specifiying a "drop-tolerance", ϵ, where if $||A_{ij}|| \leq \epsilon$, then that term is not counted when performing the matrix-vector products within GMRES.

We take a different approach with the FMM, given that we can control error by using p and θ_{MAC} as described in §2.3. Given these values it is natural that we use them to adjust the accuracy of the FMM for each iteration, reducing the accuracy to perform the minimum amount of work necessary, speeding up the calculation. We expect this to give a speedup due to the scaling of the FMM — the two main contributions to the FMM's runtime are, from §2.3, the near-field calculation between

boxes, taking $N \cdot 27 \cdot N_{\text{CRIT}}^2$ time, and the M2L translation stage, which takes $N \cdot N_{\text{M2L}} \cdot c_{\text{M2L}}(p)$ time. Given that $c_{\text{M2L}}(p) = \mathcal{O}(p^4)$ for a standard spherical harmonics expansion and $\mathcal{O}(p^6)$ for Cartesian series, the reduction in time can be significant as p is reduced. For example, moving from $p = 10$ to $p = 1$ is a potential $10^4\times$ reduction in work for the spherical harmonics case. While this level of acceleration is unlikely to be obtained in practise it shows the potential of reducing p throughout the process of an iterative solve.

This method of controlling (and increasing) induced errors throughout the iterations of a linear solve is known as a *relaxation scheme*. In chapter 3 we will describe the exact nature of a relaxation scheme, and look at the remaining numerical methods that will be used in order to perform experiments on a variety of sample BEM problems to demonstrate the application of relaxation schemes for FMM-BEM. These experiments will constitute the first such work with relaxation schemes and FMM-BEM solvers.

Chapter 3

Numerical Methods

This section describes in detail the exact numerical methods used in this thesis, along with relevant formulae and implementation details. We start by discussing the different types of numerical integrals that will be used, including analytical and semi-analytical routines for singular integrals. Finally, we provide details about the relaxation schemes that will form the basis of our experimentation, along with pre-conditioners that will be used along with and in competition to relaxation schemes.

3.1 Numerical Integration

For all the BEM formulations described in this work, we must integrate some function, $f(\mathbf{x}, \mathbf{y})$ over a source panel, Γ, $\mathbf{y} \in \Gamma$ with respect to a target point, $\mathbf{x}$ in order to generate a matrix element. This section deals with how to perform these integrals with sources and targets in different parts of the domain.

$$\int_{\Gamma} f(\mathbf{x}, \mathbf{y}) \, \mathrm{d}\Gamma. \tag{3.1}$$

We can separate the types of integrals we need to perform in terms of the distance between the source panel and target point. We introduce the following terms to describe this separation:

- *Far* – Integrals where the source and target panels are far apart require much less accuracy than either of the 2 other categories. In order to accelerate the far-field

evaluation for an FMM, semi-analytical and analytical integrals are unsuitable, and in these cases, Gauss quadrature rules of low order ($\{1, 3, 4, 7\}$ quadrature points) are sufficient, fast, and suitable for acceleration using an FMM.

- *Near-singular* – This kind of integral occurs when the source panel and target point are close-by (defined somewhat arbitrarily), but not the same. The integral is non-singular, but still requires high accuracy. High-precision versions of standard quadrature rules are suitable in these cases, or the analytical / semi-analytical methods developed for singular integrals can be re-used.

- *Singular* – The integral of a singular function over a region that includes a singularity. In the case of BEM, this occurs when the target point is within the source panel. Due to the singularity, most standard approximation techniques cannot be used, and so special methods must be developed, such as analytical or semi-analytical integrals.

The domain for each of these methods is shown graphically in figure 3·1, and methods for each along with an implemented example are described in the following sections.

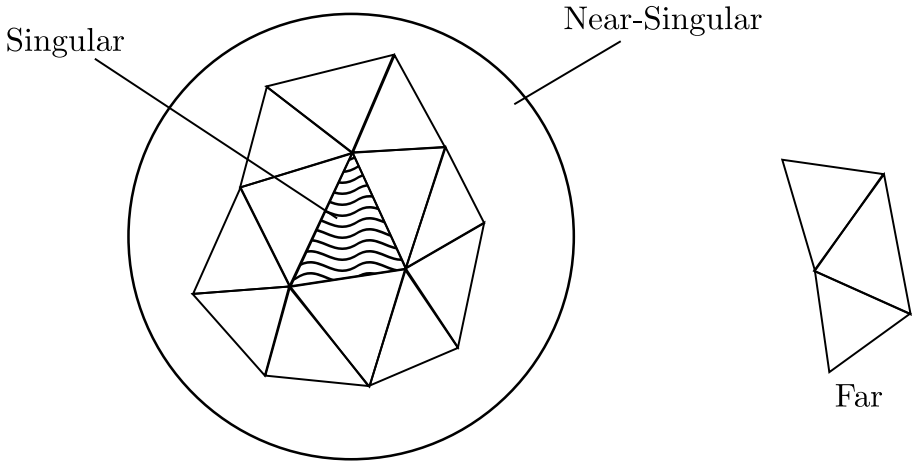


Figure 3·1: Integration domains

3.1.1 Gauss Quadrature

Gaussian quadrature is an approximation to a definite integral, formed by a weighted sum of K function evaluations at K *gauss points*, x_k and *gauss weights*, w_k within the integration region. As a 1D example, consider the integral of a single-valued function, $f(x)$ over a line segment, $[a, b]$.

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{k=1}^{K} w_k f(x_k), \quad w_k \in [0,1],\ x_k \in [a,b]. \tag{3.2}$$

This method is generalizable over n-dimensions, so for a multi-variate function $f(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^d$ defined in some region, $\mathcal{D}$, 3.2 changes very little, to

$$\int_{\mathcal{D}} f(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \sum_{k=1}^{K} w_k f(\mathbf{x}_k), \quad w_k \in [0,1],\ \mathbf{x}_k \in \mathcal{D}. \tag{3.3}$$

For an example of this, we can look at the integrals resulting from the BEM formulation for the Laplace equation, given later in equations 5.14 and 5.15. In this case, $\mathbf{x}_i$ is a target point, and j denotes a particular panel. Using a rule of the form from 3.3 these integrals turn into

$$\int_{\Gamma} G(\mathbf{x}_i, \mathbf{x}_j)\,\mathrm{d}\Gamma_j \approx \sum_{k=1}^{K} w_k \cdot A_j \cdot \frac{1}{|\mathbf{x}_i - \mathbf{x}_k|}, \quad \mathbf{x}_k \in \Gamma_j, \tag{3.4}$$

$$\int_{\Gamma} \frac{\partial G(\mathbf{x}_i, \mathbf{x}_j)}{\partial \hat{n}_j}\,\mathrm{d}\Gamma_j \approx \sum_{k=1}^{K} w_k \cdot A_j \cdot \frac{d\mathbf{x} \cdot \hat{n}_j}{|\mathbf{x}_i - \mathbf{x}_k|^3}, \quad \mathbf{x}_k \in \Gamma_j, \tag{3.5}$$

where w_k are area-normalized Gauss quadrature weights and A_j is the area of Γ_j. The accuracy of the integral can be controlled by changing the number of quadrature points used, K. Common values for far integrals include $K = 3, 4$, while near-singular, many more Gauss points, for instance $K = 19, 25$ may be necessary. It is worth noting

here that both integrals are now expressed in terms of repeated evaluations of $1/r$ and $\nabla(1/r) \cdot \hat{n}_j$, making them suitable for use with an FMM and enabling us to speedup the solution to the linear system.

Each quadrature point is expressed in terms of a set of normalized co-efficients and an area-normalized weight. Given a set of co-efficients $\xi_1,\ \xi_2,\ \xi_3$, we can express the associated quadrature point, $(x_p, y_p, z_p)^T$ on a triangle defined by 3 vertices, $(x_1, y_1, z_1),\ (x_2, y_2, z_2),\ (x_3, y_3, z_3)$ in the following way:

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{pmatrix} \times \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}. \tag{3.6}$$

This allows us to use the same sets of quadrature co-efficients for every integral without any changes. We use both standard low-order quadrature rules, with $k = 3, 4, 7$, but also have much higher-precision options available for near-singular integrals. Our default rule for near-singular integrals uses $k = 25$, while even higher precisions are available, for instance $k = 79$.

Example co-efficients for $K = 3$ for a triangular panel are given in table 3.1.

k	ξ_1^k	ξ_2^k	ξ_3^k	w_k
1	1/2	1/2	0	1/3
2	0	1/2	1/2	1/3
3	1/2	0	1/2	1/3

Table 3.1: Co-efficients and weights for Gauss integration over a triangular panel, $K = 3$

3.1.2 Semi-Analytical

The first approach for singular and near-singular integrals, will deal with the case that the function being integrated is (excluding constants) only a function of r, that is, $f = f(r)$, and $\int_a^b f(r)\ \mathrm{d}r$ can be computed analytically.

The technique we will describe relies on the fact that the integral over a polygon can be decomposed into a *signed summation* of the triangles resulting from the projection of the target point onto the polygon plane, denoted P, and two of the polygon vertices. In this way

$$\int_{\Gamma} f(r)\ \mathrm{d}r = \sum_{i=0}^{\#\text{vertices}} s_i \int_{PV_iV_{i+1}} f(r)\ \mathrm{d}r, \tag{3.7}$$

where s_i is the signing term, given by 1 if v_iv_{i+1} is clockwise from the target point, and -1 otherwise. This composition is shown in figure 3·2.

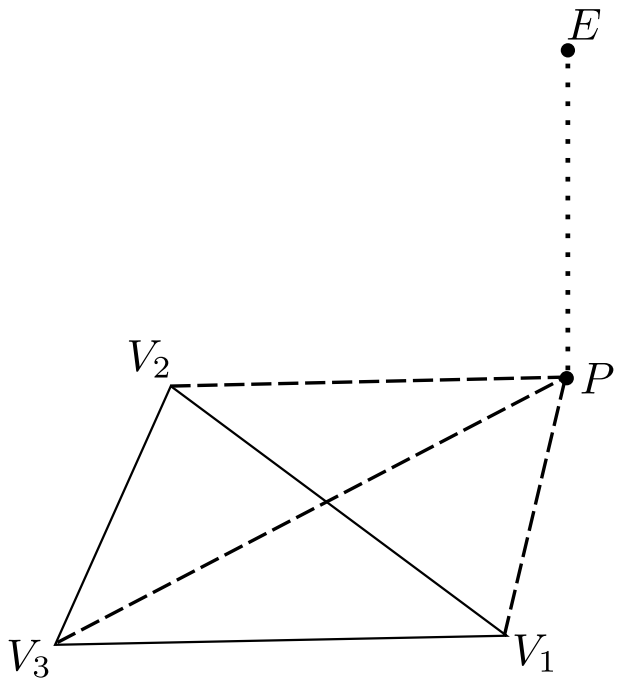


Figure 3·2: Integral decomposition into triangles

For each of the individual integrals, we set up the integration domain, figure 3·3,

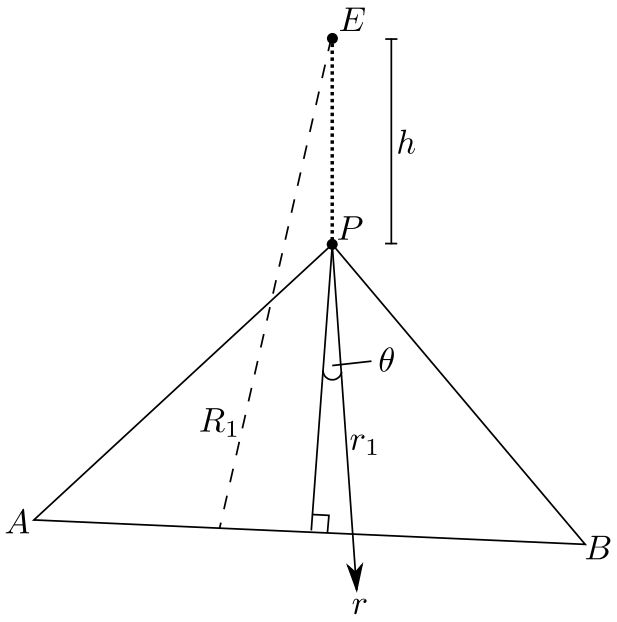


Figure 3·3: Individual triangular integral setup

noting that $R = \sqrt{r^2 + h^2}$ and then

$$\begin{aligned} \frac{\mathrm{d}R}{\mathrm{d}r} &= \frac{r}{\sqrt{r^2+h^2}} \\ \mathrm{d}R &= \frac{r}{R}\,\mathrm{d}r \\ R\,\mathrm{d}R &= r\,\mathrm{d}r \end{aligned}$$

If we denote the integral over this triangle by I, then we can now write

$$\begin{aligned} I &= \int_{\theta_1}^{\theta_2} \int_{0}^{r_1(\theta)} f(r) \cdot r\,\mathrm{d}r\,\mathrm{d}\theta \\ &= \int_{\theta_1}^{\theta_2} \int_{h}^{R_1(\theta)} f(R) \cdot R\,\mathrm{d}R\,\mathrm{d}\theta. \end{aligned} \tag{3.8}$$

Similarly, to find the integral of the derivative of $f(r)$ in a direction, $\partial I/\partial x_i$, we can perform

$$\frac{\partial I}{\partial x_i} = \frac{\partial}{\partial x_i} \int_{\theta_1}^{\theta_2} \int_{h}^{R_1(\theta)} f(R) \cdot R \, \mathrm{d}R \, \mathrm{d}\theta \tag{3.9}$$

The inner integrals in equations 3.8 and 3.9 over R will be performed analytically, then the outer integral over θ will be done using gaussian quadrature. As a first example, we look at the Green's function for Laplace's equation, $f(R) = 1/R$

$$\begin{aligned}
\int_{h}^{R_1(\theta)} f(R) \cdot R \, \mathrm{d}R &= \int_{h}^{R_1(\theta)} 1 \, \mathrm{d}R \\
&= R_1(\theta) - h. \\
\frac{\partial}{\partial \hat{n}} \int_{h}^{R_1(\theta)} f(R) \cdot R \, \mathrm{d}R &= \frac{\partial R_1(\theta)}{\partial \hat{n}} - \frac{\partial h}{\partial \hat{n}} \\
&= \frac{z}{R(\theta)} - \mathrm{sign}(z)
\end{aligned}$$

Next, we look at the Yukawa equation, $f(R) = \exp(-kR)/R$

$$\begin{aligned}
\int_{h}^{R_1(\theta)} f(R) \cdot R \, \mathrm{d}R &= \int_{h}^{R_1(\theta)} e^{-kR} \, \mathrm{d}R \\
&= -\frac{1}{k} \left[e^{-kR_1(\theta)} - e^{-kh} \right] \\
\frac{\partial}{\partial \hat{n}} \int_{h}^{R_1(\theta)} f(R) \cdot R \, \mathrm{d}R &= \frac{z \cdot e^{-kR(\theta)}}{R(\theta)} - e^{-kR(\theta)} \cdot \mathrm{sign}(z)
\end{aligned}$$

Now that we have the singular integrals available for Laplace and modified-Helmholtz, it is time to tackle the integrals that this semi-analytical method is unsuitable for – the Green's functions for the Stokes equation. This will be handled with an analytical expression.

3.1.3 Analytical

For functions that cannot be expressed purely in terms of $f(r)$, we need a different approach to the semi-analytical one described in 3.1.2. In these cases, we have chosen to use a purely analytical method, described by S. Fata for both Laplace (Fata, 2009) and linear elasticity (Fata, 2011), and trivially adaptable for the Stokes equations.

We begin with a triangle described by its 3 vertices, $\mathbf{y}_1, \mathbf{y}_2$ and $\mathbf{y}_3$, defined in a clockwise direction. In this way, the 3 line components of the triangle are described in terms of the vertices, with $L_1 = [\mathbf{y}_1, \mathbf{y}_2]$, $L_2 = [\mathbf{y}_2, \mathbf{y}_3]$, $L_3 = [\mathbf{y}_3, \mathbf{y}_1]$. The plane in which the triangle lies is denoted by E_q. Next, we compute an orthonormal companion reference $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, designed so that $\mathbf{e}_1$ is parallel to L_1, and $\mathbf{e}_3$ is perpendicular to the edges of the triangle and points in the direction of the outward normal. The plane spanned by $\mathbf{e}_1$ and $\mathbf{e}_2$ is referred to by P_{E_q}. Thus, we introduce a new co-ordinate system, $\mathbf{x}; \xi, \zeta, \eta$ that is associated with E_q, shown in figure 3·4.

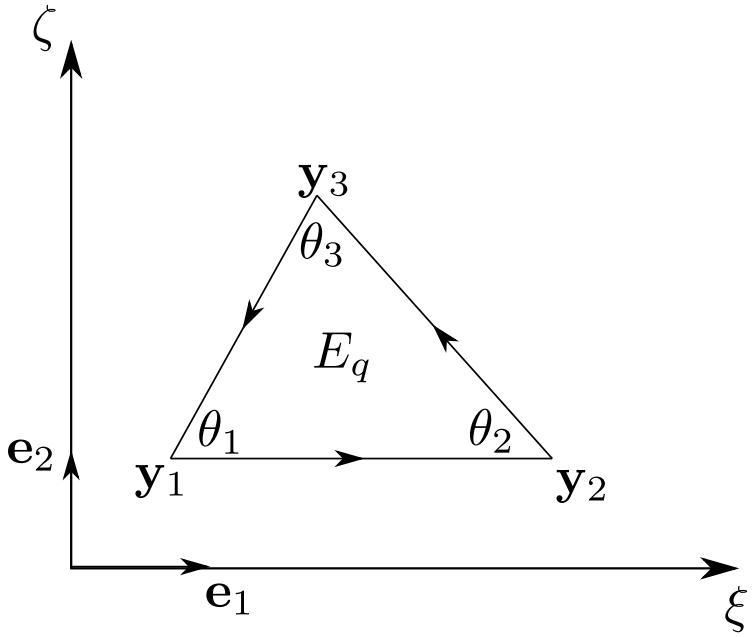


Figure 3·4: Local co-ordinate system for analytical integrals

Using this co-ordinate system, we can describe important quantities which are

used to describe our integrals in the forms shown in equations 3.10 and 3.11.

$$\begin{aligned} \mathbf{r} &= \mathbf{y}-\mathbf{x} = \xi\mathbf{e}_1 + \zeta\mathbf{e}_2 + \eta\mathbf{e}_3 \\ r &= ||\mathbf{y}-\mathbf{x}|| = \sqrt{\xi^2+\zeta^2+\eta^2}, \end{aligned}$$

$$\int_{\Gamma_j} G(\mathbf{x},\mathbf{y})\mathrm{d}\Gamma_j = \frac{1}{4\pi} I_1 \tag{3.10}$$

$$\int_{\Gamma_j} \frac{\partial G_j(\mathbf{x})}{\partial \hat{n}}\mathrm{d}\Gamma_j = -\frac{\eta}{4\pi} I_3, \tag{3.11}$$

where the generic integrals $I_1,\ I_3$ refer to:

$$I_1 = \int_{E_q} \frac{1}{r}\mathrm{d}s, \quad I_3 = \int_{E_q} \frac{1}{r^3}\mathrm{d}s \tag{3.12}$$

To express these generic integrals, we must define some more quantities over the panel. First, $\theta_i \in (0,\pi)$ is the inclusion angle at vertex i, such that $\theta_1+\theta_2+\theta_3=\pi$, next we define $\alpha_1 = 0,\ \alpha_2 = \pi-\theta_2,\ \alpha_3 = \pi+\theta_1$. Using these values of θ_i and α_i we can constuct a set of 3 local orthonormal bases centred at the origin, given by $\{\mathbf{e}_{p_i}, \mathbf{e}_{q_i}, \mathbf{e}_3\}$, such that $\mathbf{e}_{p_i}$ is parallel to the previously defined line L_i. This gives us local coordinated systems, denoted by $\{p_i, q_i, \eta\}$ that are associated with each edge of the triangle. In this form, we have

$$p_i = \xi\cos\alpha_i + \zeta\sin\alpha_i, \quad q_i = -\xi\sin\alpha_i + \zeta\cos\alpha_i. \tag{3.13}$$

More local coordinate systems can be formed from this construction, $\underset{\sim}{\mathbf{x}} = (p_i, q_i)$ defined for each of the sides of the triangle, allowing us to write a local coordinate (p_i^j, q_i^j, η) in terms of a basis $\{\mathbf{e}_{p_i}, \mathbf{e}_{q_i}, \mathbf{e}_3\}$. The distance ρ_j between a vertex, $\mathbf{y}_i$ and

a source point, $\mathbf{x}$ can now be written as

$$\rho_j = \sqrt{(p_i^j)^2 + (q_i^j)^2 + \eta^2}. \tag{3.14}$$

To make the following equations simpler, we define

$$\tilde{\rho}_1 = \rho_1 - \rho_2, \;\; \tilde{\rho}_2 = \rho_2 - \rho_3, \;\; \tilde{\rho_3} = \rho_3 - \rho_1, \;\; d_i = (q_i)^2 + \eta^2, \;\; i = 1, 2, 3. \tag{3.15}$$

More quantities are now defined over each side, L_i,

$$\gamma_i(\mathbf{x}) = \arctan\left(\frac{-2p_i^i q_i \eta \rho_i}{(q_i)^2(\rho_i)^2 - (p_i^i)^2\eta^2}\right) - \arctan\left(\frac{-2p_i^{i+1} q_i \eta \rho_{i+1}}{(q_i)^2(\rho_{i+1})^2 - (p_i^{i+1})^2\eta^2}\right) \tag{3.16}$$

$$\begin{aligned}
\chi_i(\mathbf{x}) &= \ln(p_i^i + \rho_i) - \ln(p_i^{i+1} + \rho_{i+1}), & (3.17)\\
\delta_i(\mathbf{x}) &= \frac{p_i^i}{\rho_i} - \frac{p_i^{i+1}}{\rho_{i+1}}, & (3.18)\\
\mathcal{L}_i(\mathbf{x}) &= \frac{1}{\rho_i} - \frac{1}{\rho_{i+1}}. & (3.19)
\end{aligned}$$

Discarding all explicit dependancies on $\mathbf{x}$ for brevity's sake, we define the last couple of quantities needed, $\theta_0(\mathbf{x})$, dependant on the location of the center of the source panel and $\theta(\mathbf{x})$.

$$\theta_0(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in P_{E_q} \setminus \bar{E}_q \\ \pi & \text{if } \mathbf{x} \in \mathcal{L}_q \setminus \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\} \\ 2\pi & \text{if } \mathbf{x} \in E_q \\ \theta_i & \text{if } \mathbf{x} = \mathbf{y}_i, \;\; i = 1, 2, 3 \end{cases} \tag{3.20}$$

$$\theta(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{3} \gamma_i(\mathbf{x}) + \text{sign}(\eta)\theta_0(\mathbf{x}) \tag{3.21}$$

This allows us to finally define I_1 and I_3 as:

$$I_1 = \sum_{i=1}^{3} q_i \chi_i(\mathbf{x}) - \eta\theta(\mathbf{x}) \tag{3.22}$$

$$I_3 = \frac{1}{\eta}\theta(\mathbf{x}), \tag{3.23}$$

allowing us to compute the desired integrals.

This work on analytical integrals for potential problems was later extended to linear elasticity integrals (Fata, 2011), which can then be devolved into Stokes kernels by setting the poisson ratio, $\nu = 1/2$. By using the same quantities over the triangle as from the potential integrals, we can express these Stokes integrals in a similar fashion

$$U(\mathbf{x},\mathbf{y}) = \frac{1}{8\pi\mu}\left(\frac{\mathbf{I}}{r} + \frac{(\mathbf{x}-\mathbf{y})\otimes(\mathbf{x}-\mathbf{y})}{r^3}\right) = \frac{1}{8\pi\mu} I_U \tag{3.24}$$

$$T(\mathbf{x},\mathbf{y}) = -\frac{3}{4\pi}\left(\frac{(\mathbf{x}-\mathbf{y})\otimes(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})\cdot\hat{n}}{r^5}\right) = -\frac{3}{4\pi} I_T \tag{3.25}$$

With I_U and I_T defined in terms of more generic integrals

$$\begin{aligned} I_U = {} & (I_1 + I_3^{\xi\xi})\mathbf{e}_1\otimes\mathbf{e}_1 + I_3^{\xi\zeta}\mathbf{e}_1\otimes\mathbf{e}_2 + \eta I_3^{\xi}\mathbf{e}_1\otimes\mathbf{e}_3 \\ & + I_3^{\xi\zeta}\mathbf{e}_1\otimes\mathbf{e}_2 + (I_1 + I_3^{\zeta\zeta})\mathbf{e}_2\otimes\mathbf{e}_2 + \eta I_3^{\zeta}\mathbf{e}_2\otimes\mathbf{e}_3 \\ & + \eta I_3^{\xi}\mathbf{e}_1\otimes\mathbf{e}_3 + \eta I_3^{\xi}\mathbf{e}_2\otimes\mathbf{e}_3 + (I_1 + \eta^2 I_3)\mathbf{e}_3\otimes\mathbf{e}_3 \end{aligned} \tag{3.26}$$

$$\begin{aligned} I_T = {} & -3\eta I_5^{\xi\xi}\mathbf{e}_1\otimes\mathbf{e}_1 - 3\eta I_5^{\xi\zeta}\mathbf{e}_1\otimes\mathbf{e}_2 - 3\eta^2 I_5^{\xi}\mathbf{e}_1\otimes\mathbf{e}_3 \\ & - 3\eta I_5^{\xi\zeta}\mathbf{e}_1\otimes\mathbf{e}_2 - 3\eta I_5^{\zeta\zeta}\mathbf{e}_2\otimes\mathbf{e}_2 - 3\eta^2 I_5^{\zeta}\mathbf{e}_2\otimes\mathbf{e}_3 \\ & - 3\eta^2 I_5^{\xi}\mathbf{e}_1\otimes\mathbf{e}_3 - 3\eta^2 I_5^{\zeta}\mathbf{e}_2\otimes\mathbf{e}_3 - 3\eta^2 I_5\mathbf{e}_3\otimes\mathbf{e}_3 \end{aligned} \tag{3.27}$$

where the new generic integrals are defined as:

$$
\begin{aligned}
I_5 &= \frac{1}{3\eta^2}\sum_{i=1}^{3}\frac{q_i}{d_i}\delta_i(\mathbf{x}) + \frac{1}{3\eta^3}\theta(\mathbf{x}) \\
I_3^{\xi} &= \sum_{i=1}^{3}\chi_i(\mathbf{x})\sin\alpha_i, \quad I_3^{\zeta} = -\sum_{i=1}^{3}\chi_i(\mathbf{x})\cos\alpha_i \\
I_5^{\xi} &= \frac{1}{3}\sum_{i=1}^{3}\frac{\delta_i(\mathbf{x})}{d_i}\sin\alpha_i, \quad I_5^{\zeta} = -\frac{1}{3}\sum_{i=1}^{3}\frac{\delta_i(\mathbf{x})}{d_i}\cos\alpha_i \\
I_5^{\xi\xi} &= -\frac{1}{3}\sum_{i=1}^{3}\left(\mathcal{L}_i(\mathbf{x})\cos\alpha_i + \frac{q_i}{d_i}\delta_i(\mathbf{x})\sin\alpha_i\right)\sin\alpha_i + \frac{1}{3\eta}\theta(\mathbf{x}) \\
I_5^{\zeta\zeta} &= \frac{1}{3}\sum_{i=1}^{3}\left(\mathcal{L}_i(\mathbf{x})\sin\alpha_i + \frac{q_i}{d_i}\delta_i(\mathbf{x})\cos\alpha_i\right)\cos\alpha_i + \frac{1}{3\eta}\theta(\mathbf{x}) \\
I_5^{\xi\zeta} &= \frac{1}{3}\sum_{i=1}^{3}\left(\mathcal{L}_i(\mathbf{x})\sin\alpha_i + \frac{q_i}{d_i}\delta_i(\mathbf{x})\cos\alpha_i\right)\sin\alpha_i
\end{aligned}
$$

Finally, we have all the integration tools in order to evaluate integral expressions over the entire domain. We can now evaluate $A\mathbf{x}$ in each GMRES iteration for all the kernels we will consider, and so we can concentrate on the linear solver itself.

3.2 Relaxation Strategies

All testing results presented in this thesis are obtained using relaxation strategies during our GMRES / FGMRES iterations, and are the first such work for FMM-BEM applications. For each iteration k of the solver, we define an accuracy, η_k that must be fulfilled by our FMM-accelerated matvec in order to continue converging. We have chosen to concentrate on the following definition for η_k, given by (Bouras and Frayssé, 2005).

$$\eta_k = \frac{\epsilon_{\text{global}}}{r_k}, \tag{3.28}$$

where ϵ_{global} is the desired tolerance of the solution, and r_k is the residual at iteration

k.

Given these accuracies, we need some way of imposing them on our fast method. Given that the error from said method, $\epsilon_{\text{fast}} = \epsilon_{\text{fast}}(\theta_{\text{MAC}}, p)$, we choose to modify only one parameter, holding θ_{MAC} constant, and only modifying p.

As an example, we choose spherical harmonics kernels for the Laplace Green's function. From (Greengard and Rokhlin, 1987) we can express a given term in the multipole expansion, M_n^m from source points $(\rho_i, \alpha_i, \beta_i)$ as

$$M_n^m = \sum_{i=1}^{N} q_i \cdot \rho_i^n \cdot Y_n^m(\alpha_i, \beta_i), \tag{3.29}$$

and the error incurred from this approximation, for series truncation $p > 1$ at a point, $P = P(r, \theta, \phi)$,

$$\left| \phi(P) - \sum_{n=0}^{p} \sum_{m=-n}^{n} \frac{M_n^m}{r^{n+1}} \cdot Y_n^m(\theta, \phi) \right| \leq \frac{\sum_{i=1}^{N} q_i}{r - a} \left(\frac{a}{r} \right)^{p+1}, \tag{3.30}$$

where a is the size of the cluster and r is the distance between the multipole center and the target particle. Similar bounds are derived (Greengard, 1987) for all of the translation and evaluation operators. Using the nature of the octree space decomposition, we can form the following expression for the number of terms needed for a given accuracy,

$$p \sim \lceil -\log_2(\epsilon) \rceil. \tag{3.31}$$

While in practice this presents a very loose error bound, often over-approximating the value of p needed for a desired accuracy, the fact that it requires no knowledge of the tree or interactions between boxes is advantageous, and allows us to choose a new p in $\mathcal{O}(1)$ time.

3.3 Preconditioners

In this work we will make use of a small number of fairly simple preconditioners, implemented for each of our test problems. In each case, for the right-preconditioned GMRES we are providing a solution to the system

$$Mz_j = v_j, \tag{3.32}$$

where M is some approximation of A, such that solving for z_j is cheaper than actually solving $Az_j = v_j$, or M^{-1} is easily computable.

The first, and most basic preconditioner that can be used is a diagonal preconditioner, where

$$M = \text{diag}(A). \tag{3.33}$$

With this form of M, M^{-1} becomes $M_{ii}^{-1} = (A_{ii})^{-1}$. For the Laplace equation, $(A_i i)^{-1}$ is simply $1/A_i i$, while for Stokes problems, where $A_{ii} \in \mathbb{R}^{3\times3}$, each element of M^{-1} is the 3×3 matrix inverse of A_{ii}. Note that this is a *constant* preconditioner with no inner solve needed, and thus can be used in both GMRES and FGMRES solvers.

Next, we look at 2 preconditioners that are made possible by the FMM's decomposition of space. First is a block-diagonal form, shown in (Liu, 2009) where M is a sparse matrix taking into account only particle-particle interactions within a leaf of the FMM's tree hierarchy. This takes into account only the closest interactions between panels. We denote this sparse matrix as A_{leaf}, and precompute at the start of all calculations. Each block of this matrix is independent, and so we could explicitly compute the inverse (using BLAS or similar). However, we do not do this currently, and instead solve $A_{\text{block}} z_j = v_j$ using an inner GMRES solver. Due to the inner solve, this preconditioner is considered to be *flexible*, and so FGMRES must be used as the outer solver. In the matrix form shown below, A_i denotes the block matrix formed

by interactions within leaf i of the tree.

$$A_{\text{leaf}} = \begin{pmatrix} A_1 & & & & \\ & A_2 & & & \\ & & A_3 & & \\ & & & \ddots & \\ & & & & A_N \end{pmatrix}, \quad A_{\text{leaf}}^{-1} = \begin{pmatrix} A_1^{-1} & & & & \\ & A_2^{-1} & & & \\ & & A_3^{-1} & & \\ & & & \ddots & \\ & & & & A_N^{-1} \end{pmatrix} \tag{3.34}$$

In a similar vein, the third preconditioner we experiment with takes into account all close-range interactions within the FMM (Chaillat et al., 2011). This preconditioner involves only P2P interactions, and no far-field approximations of any kind. By necessity, the interactions are precomputed and stored as the sparse-matrix A_{local}. We are unable to efficiently compute the inverse (which is not guaranteed to be sparse), so, as with the block-diagonal preconditioner described above, we use an additional GMRES solver to obtain z_j. Again, this is a *flexible* preconditioner an FGMRES must be used. For this case, in the notation used below, A_{ij} is the block matrix formed from interactions between leaves i and j.

$$A_{\text{local}} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & & \\ A_{21} & A_{22} & A_{23} & & \\ & A_{32} & A_{33} & A_{34} & \\ & & & \ddots & \\ & & & A_{N,N-1} & A_{NN} \end{pmatrix} \tag{3.35}$$

This ends up being another sparse approximation to the full linear system, albeit one more dense than the block-diagonal preconditioner. Intuitively, we could expect this preconditioner to work the best in terms of reducing iteration counts as it better represents A, however this will come at a cost – each inner GMRES will be more expensive due to the larger number of non-zeros.

Chapter 4

FMM-BEM Implementation Details

We have previously discussed the concepts of the FMM (§2.3) and BEM (§2.1), and some of the numerical techniques we will be using (§3.1). We now discuss implementation specifics for the FMM, and in particular, specifics for the FMM-BEM.

4.1 Data Types

The types of data we will deal with in the FMM-BEM, and how we handle them is very important. We will summarize the important types that we will deal with in both FMM and FMM-BEM, highlighting the differences.

- `point_type` – A d-dimensional point in space, $\mathbf{x} \in \mathbb{R}^d$.
- `source_type` – A source for the evaluation. For instance, a planetary body in astrophysics, a point vortex in vortex methods.
- `target_type` – A target for the evaluation – this can be the same as a source or simply a point in space, depending on the type of system being evaluated.
- `multipole_type` – Type of the multipole expansions, e.g. a vector of complex numbers (Spherical harmonics expansions) or real numbers (cartesian expansions)
- `local_type` – Analogous to `multipole_type` for the FMM's local expansions.

Before we explicitly set these types, we can gain insight into what they might look like by thinking about the nature of what we are solving in an FMM-BEM problem.

First, we are approximating a matrix from values on a discretized surface mesh, where each element contributes a term based on an integral over its surface with respect to a target surface. This implies that our `source_type` is not going to be a point in space, but rather some kind of panel. It will then be easiest to represent each target as a panel too, even though we will only care about the central point in a collocation BEM — this gives us the advantage of having sources and targets the same, simplifying the FMM evaluations. Next, each panel can contribute one of 2 types of value – fixed potential or charge in the case of Laplace, velocity or traction for Stokes and displacement or traction force for linear elasticity problems. Approximations to these different contributions must be kept separate, implying we should have a pair of expansions, one for each type of boundary condition.

The first BEM-specific change we make, is to consider a panel-type to serve as our source and target-types. Represented (at a minimum) as a set of vertices (v_1, v_2, v_3), the actual implementation of the panel is irrelevant so long as standard information can be obtained from it – quadrature points, area, normal etc. These panels will require implementation-specific P2P, P2M routines, and potentially M2P and L2P as well. An example of the code description for a constant triangle is shown below

```
// Constant triangular panel
struct ConstantTriangle
{
  // important values
  double area;
  Point3D normal;
  // pre-computed quadrature points for triangle
  vector<Point3D> quadrature_points;
  BoundaryConditionType BC;

  // create Panel from 3 vertices
  Panel(Point3D v1, Point3D v2, Point3D v3);
```

```
14
    // create Panel from existing panel
16  Panel(Panel other);
};
```

The change for multipole and local types is much each easier – if we consider (in C++notation) a single multipole to have type `multipole_type`, then the BEM specific type would be either `multipole_type[2]`, or `std::vector<multipole_type>`.

Table 4.1 summarizes the changes in types between the FMM for a particular equation, in this case Laplace, and the FMM-BEM.

Data type	FMM	FMM-BEM
point_type	`Point3D`	`Point3D`
source_type	`Point3D`	`Panel`
target_type	`Point3D`	`Panel`
multipole_type	`vector<complex>`	`vector<complex>[2]`
local_type	`vector<complex>`	`vector<complex>[2]`

Table 4.1: Data types for Laplace FMM and FMM-BEM

4.2 FMM Operators

We have already discussed how the data types change from FMM to FMM-BEM, now we look at how the FMM operators must change to accommodate these typing changes. The first operator we will look at is P2P, the interaction between a single source and single target. This is generating the element (i, j) if we were to form a full-rank matrix. For the BEM, this is calculated with

$$\mathbb{K}(i,j) = \int_{\Gamma_j} f(\mathbf{x}_i, \mathbf{x}_j) \mathrm{d}\Gamma, \tag{4.1}$$

with $\mathbf{x}_i$ within the target panel, and $\mathbf{x}_j$ within the source. For the singular case, $i = j$, we use specific routines (see §3.1) and Gauss quadrature for all other cases. Thus a panel need simply provide some way of accessing it's relevant quadrature points, whether explicitly storing them, or calculating on-the-fly.

Next, we look at P2M - this calculates the multipole approximation from a single panel at some arbitrary point. For panels, this is as easy as iterating over the component quadrature points of each panel, treating each of them as a single source would be in a standard FMM.

Translation operators, M2M, M2L and L2L area identical to the FMM counterparts, with the exception that every translation is done twice — once for each type of boundary condition.

Finally, the evaluation operators, M2P and L2P will change based on the exact implementation of the panel. For constant panels, they will be identical to the FMM case, with a single point being evaluating at the center of the target panel. For more complex formulations, they may require exposing more target points, rendering the situation analogous to that for P2M – the panel exposes multiple points, each of which are evaluated individually as a single point.

4.3 Near-Field Sparse-Matrix

One of the major advantages of using the FMM to form an approximation of the full-rank BEM matrix, is to reduce the overall memory consumption by never explicitly storing the matrix. However, we can trade-off some of these memory savings in order to make our matrix-vector products faster. We do this by explicitly constructing the matrix of only the near-field interactions. This allows us to only perform the near-field calculation (all of the integrals) once, as each further calculation becomes a simple matrix-vector product. This is the same as calculating A_{local} from §3.3, but instead

of using the matrix as a preconditioner, we store it and use it instead of re-evaluating the near-field every solver iteration.

To save memory, we use the CSR sparse-matrix format, which consists of 3 arrays — offsets corresponding to the beginning and end of rows, which reference to segments of the other 2 arrays, containing the column index and value of every non-zero value. An example matrix and corresponding CSR representation is shown below.

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix} \tag{4.2}$$

offsets	0	2	3	5	
indices	0	1	1	0	2
values	1	2	1	3	1

Table 4.2: CSR representation of A

In the case of the near-field, every row corresponds to a source panel, and every column entry signifies a direct (P2P) interaction between the source and that target. The total storage for this matrix form in double precision is: $(\#\text{rows} \times 4) + (\text{nnz} \times 12)$ bytes . While this form of near-field interaction can be used for a standard FMM, it can cause some notable problems:

- If sources move, the sparse-matrix has to be re-created, including the non-zero (sparsity) pattern – there are no corresponding savings for multiple iterations.
- In uniform distributions of sources, each individual source will on average interact with $27 \times N_{\text{CRIT}}$ other particles, resulting in very memory-intensive matrices. For instance, $N = 10^4,\ N_{\text{CRIT}} = 400$ would result in a matrix taking 1236MB. Increasing N to 10^5 would give a matrix taking 12.36GB. We can use this technique more readily for BEM applications, as the nature of the surface mesh

will result in less than 27 near-field cluster-cluster interactions, reducing that number to around 9, with an equivalent reduction in memory consumption.

4.4 Lazy Evaluators

In the FMM-BEM, we have the advantage that once we create the spatial decomposition of the domain, it doesn't change throughout the iterations of the linear solve. This means we can traverse the tree a single time, storing the resulting interactions in lists, and simply run through those lists once for every iteration.For example, the upward sweep shown in algorithm 3 turns into algorithm 7 when performed in a lazy fashion.

Analogous variations on algorithms 4 and 5 can also be created.

Algorithm 7 Lazy Upward sweep

```
Require: Cells C, kernel K.
  P2M_list = []
  M2M_list = []
  // Once during construction
  for i=1, ..., SIZE(C) do
      if IS_LEAF(C_i) then
          P2M_LIST.INSERT(i)                    ▷ Queue P2M operation for leaf
      else
          p = PAIR(i, parent(C_i).index)        ▷ Queue M2M operation up tree
          M2M_LIST.INSERT(p)
  // Each Iteration
  for Entries i in P2M_list do
      K.P2M(C_i)                                ▷ Evaluate queued P2M
  for Entries i in M2M_list do
      K.M2M(C_i.first, C_i.second)              ▷ Evaluate queued M2M
```

A further advantage to this style of evaluation beyond saving the cost of repeated tree traversals, is that we can process lists of queued operations in parallel very easily. The most obvious candidates for this treatment are lists for P2M and L2P, where every operator call only involves a single cell, and every cell is completely independent. All other lists can still be performed in parallel, albeit with more care taken to ensure no

race conditions (multiple parallel processes attempting to change the same value(s) simultaneously) occur.

4.5 Performance

While we have asserted in §2.3 that the FMM scales as $\mathcal{O}(N)$, we should confirm that our implementation scales correctly. All kernels should scale equally, so we choose to use the Laplace kernel, utilizing spherical harmonic expansions, with $p = 5$. We choose to keep $N_{\text{CRIT}} = 126$ for all cases, and values of N are chosen such that all data points are equispaced on a log-log plot, shown below in figure 4·1.

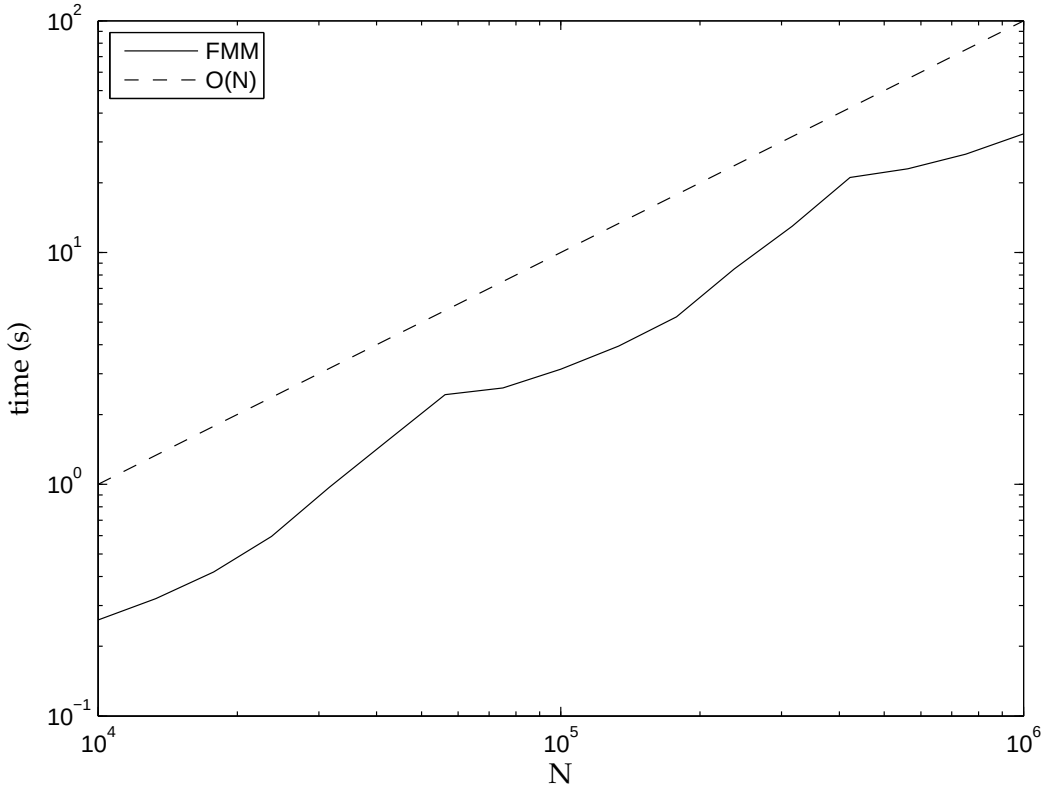


Figure 4·1: Scaling of the FMM with respect to problem size, N

This plot shows the overall scaling of the FMM as $\mathcal{O}(N)$. The minor peaks are caused by the nature of the hierarchical decomposition – when a new level is added

it changes the balance of near and far-field computation, which is vital to the overall $\mathcal{O}(N)$ behavior. We could eliminate these peaks by tweaking N_{CRIT} for every test value of N, but the general trend of timing has already been shown.

Chapter 5

BEM with inexact GMRES for the Laplace equation

To properly evaluate the utility of our relaxed FMM-BEM method, we must apply it to a variety of examples, both to test for correctness (obtaining the correct answer and maintaining all convergence properties) and to evaluate the "real-world" potential for speedup.

The first such example we will look at is the solution to Laplace's equation, consisting of obtaining $\phi(\mathbf{x})$ such that:

$$\nabla^2\phi(\mathbf{x}) = 0 \tag{5.1}$$

This equation can be used to model several applications, including electrostatics (Yokota et al., 2010), potential flow (Klaseboer et al., 2011) and heat transfer (Panti, 2008; Majchrzak and Freus, 2003), and using FMM-BEM in particular (Nabors et al., 1994).

5.1 BEM

To obtain a BEM formulation for Laplace, we start with the weak formulation of 5.1, with some weighting function, W,

$$\int_{\Omega} \left[\nabla^2\phi\, W\right]\, \mathrm{d}\Omega = 0. \tag{5.2}$$

We can write this in a different form,

$$\int_{\Omega} \left[\nabla(\nabla\phi\, W) - \nabla\phi\nabla W\right]\, \mathrm{d}\Omega = 0. \tag{5.3}$$

Applying Gauss' divergence theorem to the first term, we convert it into a surface integral, while simultaneous re-writing the second term, $\nabla\phi\nabla W = \nabla(\phi\nabla W) - \phi\nabla^2 W$ gives us

$$\int_{\Gamma} \left[\nabla\phi\, W\cdot\hat{n}\right]\, \mathrm{d}\Gamma - \int_{\Omega} \left[\nabla(\phi\nabla W) - \phi\nabla^2 W\right]\, \mathrm{d}\Omega = 0. \tag{5.4}$$

Utilizing Gauss' divergence theorem once again, and choosing $W = G = 1/4\pi r$, the Green's function for the Laplace equation (while noting that $\nabla^2 G = -\delta$), we obtain

$$\int_{\Gamma} \frac{\partial\phi}{\partial\hat{n}} G\, \mathrm{d}\Gamma - \int_{\Gamma} \phi\frac{\partial G}{\partial\hat{n}}\, \mathrm{d}\Gamma - \int_{\Omega} \phi\nabla^2 G\, \mathrm{d}\Omega = 0. \tag{5.5}$$

Taking advantage of $\nabla^2 G = -\delta$, we obtain the near-final form

$$\int_{\Gamma} \frac{\partial\phi}{\partial\hat{n}} G\, \mathrm{d}\Gamma - \int_{\Gamma} \phi\frac{\partial G}{\partial\hat{n}}\, \mathrm{d}\Gamma - \phi = 0. \tag{5.6}$$

.

The final step we must perform is to take our target point, i, to the boundary, where we want to form a linear system. At i we augment the domain by a small hemisphere around the point with radius ϵ. We can then consider i as $\epsilon \to 0$. The first integral to be treated is over G as it presents a lower order singularity than the integral over $\partial G/\partial\hat{n}$.

$$\lim_{\epsilon\to 0}\int_{\Gamma_\epsilon} G\frac{\partial\phi}{\partial\hat{n}}\,\mathrm{d}\Gamma = \lim_{\epsilon\to 0}\int_{\Gamma_\epsilon}\frac{1}{4\pi\epsilon}\frac{\partial\phi}{\partial\hat{n}}\,\mathrm{d}\Gamma \tag{5.7}$$

$$= \lim_{\epsilon\to 0}\frac{2\epsilon^2}{4\pi\epsilon}\frac{\partial\phi}{\partial\hat{n}} = 0 \tag{5.8}$$

In a similar fashion, we take the integral over $\partial G/\partial\hat{n}$ to the surface,

$$\lim_{\epsilon\to 0}\int_{\Gamma_\epsilon}\phi\frac{\partial G}{\partial\hat{n}}\,\mathrm{d}\Gamma = -\lim_{\epsilon\to 0}\int_{\Gamma_\epsilon}\phi\frac{1}{4\pi\epsilon^2} \tag{5.9}$$

$$= -\lim_{\epsilon\to 0}\phi\frac{2\pi\epsilon^2}{4\pi\epsilon^2} = -\frac{1}{2}\phi. \tag{5.10}$$

Utilizing these two results and applying them to equation 5.6 with some rearrangement, we obtain the final form for the Laplace equation:

$$\frac{1}{2}\phi + \int_\Gamma \phi\frac{\partial G}{\partial\hat{n}}\,\mathrm{d}\Gamma = \int_\Gamma \frac{\partial\phi}{\partial\hat{n}}G\,\mathrm{d}\Gamma. \tag{5.11}$$

The constant of $1/2$ holds as long as target points are located on a smooth surface, that is to say, not on the edge or corner of a panel. As noted in §2.1, we use constant, flat panels with collocation, resulting in targets in the center of panels, allowing us to use this formulation with no changes.

Finally, as we have specified constant elements, we can bring the ϕ_j and $\partial\phi_j/\partial\hat{n}$ terms outside their relevant integrals, and form the surface integrals as sums over discretized panels to give the final expression we will solve:

$$\frac{1}{2}\phi_i = \sum_j^N \frac{\partial\phi_j}{\partial\hat{n}_j}\int_\Gamma G_{ij}\mathrm{d}\Gamma_j - \sum_j^N \phi_j\int_\Gamma \frac{\partial G_{ij}}{\partial\hat{n}_j}\,\mathrm{d}\Gamma_j. \tag{5.12}$$

To find every ϕ_i and $\partial\phi_i/\partial\hat{n}_i$, we create a system of linear equations $A\mathbf{x} = \mathbf{b}$

where the elements A_{ij} are given by:

$$A_{ij} = \begin{cases} \int_\Gamma G_{ij} \, \mathrm{d}\Gamma_j, & \phi \text{ specified on panel } j \\ \int_\Gamma \frac{\partial G_{ij}}{\partial \hat{n}_j} \, \mathrm{d}\Gamma_j, & \frac{\partial \phi}{\partial \hat{n}} \text{ specified on panel } j \end{cases} \tag{5.13}$$

and $\mathbf{b}$ is formed from the known terms on the boundary – for instance, if ϕ is specified on a panel j, then $\phi_j \int_{\Gamma_j} \partial G_{ij}/\partial \hat{n}_j \, \mathrm{d}\Gamma_j$ will be added to b_i.

Expanding out these operators into the actual forms of G_{ij} and $\partial G_i j/\partial \hat{n}_j$ we obtain expressions in terms of $1/r$ and $\hat{n}_j \cdot \nabla(1/r)$.

$$\int_\Gamma G_{ij} \, \mathrm{d}\Gamma_j = \int_\Gamma \frac{1}{|\mathbf{x}_i - \mathbf{x}_j|} \, \mathrm{d}\Gamma_j \tag{5.14}$$

$$\int_\Gamma \frac{\partial G_{ij}}{\partial \hat{n}_j} \, \mathrm{d}\Gamma_j = \int_\Gamma \frac{d\mathbf{x} \cdot \hat{n}_j}{|\mathbf{x}_i - \mathbf{x}_j|^3} \, \mathrm{d}\Gamma_j \tag{5.15}$$

Exactly how to evaluate these integrals numerically has been discussed in detail in §3.1, so we just need to deal with the final details of our scheme – the far-field approximations used for the FMM.

5.2 Expansions

While there are many different ways to approximate the Laplace Green's function, we choose to use spherical harmonics due to their superior scaling at high values of p (translations such as M2L scale as $\mathcal{O}(p^4)$ instead of $\mathcal{O}(p^6)$ for cartesian expansions). This choice of expansion will give us distinct multipole (singular) and local (regular) representations, convergent in different parts of the domain.

Remember that the potential at a point $\mathbf{x}_i$ from N sources, $\mathbf{x}_j$, denoted here (to avoid clashing with the spherical co-ordinate ϕ) by $\Phi(\mathbf{x}_i)$ is given by

$$\Phi(\mathbf{x}_i) = \sum_{j=0}^{N} \frac{q_j}{|\mathbf{x}_i - \mathbf{x}_j|} \tag{5.16}$$

We begin by taking two points, $\mathbf{x}_i,\ \mathbf{x}_j \in \mathbb{R}^3$, and an intermediate point between them, $\mathbf{x}_*$. Next, we look at the pair of vectors, $\mathbf{x}_i - \mathbf{x}_* = \mathbf{x}_{i*}$ and $\mathbf{x}_j - \mathbf{x}_* = \mathbf{x}_{j*}$ in spherical coordinates.

$$\begin{aligned} \mathbf{x}_{i*} &= \mathbf{x}_{i*}(r, \theta, \phi) \\ \mathbf{x}_{j*} &= \mathbf{x}_{j*}(\rho, \alpha, \beta) \end{aligned}$$

If we let γ be the angle between $\mathbf{x}_{i*}$ and $\mathbf{x}_{j*}$, we can write the distance between these two points, r' as

$$r'^2 = r^2 + \rho^2 - 2r\rho\cos\gamma, \tag{5.17}$$

where $\cos\gamma = \cos\theta\cos\alpha + sin\theta\sin\alpha\cos(\phi - \beta)$. By setting $\mu = \rho/r$ and $u = \cos\gamma$ we directly write

$$\frac{1}{r'} = \frac{1}{r\sqrt{1 - 2u\mu + \mu^2}}. \tag{5.18}$$

For $\mu > 1$ we can expand $1/r'$ in terms of μ^n, resulting in a Legendre polynomial of order n, given by

$$\frac{1}{\sqrt{1 - 2u\mu + \mu^2}} = \sum_{n=0}^{\infty} P_n(u)\mu^n, \tag{5.19}$$

and so we can write our expression for $1/r'$ as

$$\frac{1}{r'} = \sum_{n=0}^{\infty} \frac{\rho^n}{r^{n+1}} P_n(u). \tag{5.20}$$

It is worth noting that our expansion is still coupled in terms of $\mathbf{x}_i$ and $\mathbf{x}_j$, so we use

the "well-known" result (Abramowitz and Stegun, 1964) that Legendre polynomials can be expressed in terms of spherical harmonics

$$P_n(u) = \frac{4\pi}{2n+1} \sum_{m=-n}^{n} Y_n^{-m}(\alpha,\beta) Y_n^m(\theta,\phi), \tag{5.21}$$

with

$$Y_n^m(\theta,\phi) = \sqrt{\frac{2n+1}{4\pi}} \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} P_n^{|m|}(\cos\theta) e^{im\phi}. \tag{5.22}$$

In this form for Y_n^m, the associated Legendre polynomials, P_n^m must be calculated using Rodrigues' formula (Rodrigues, 1815)

$$P_n(x) = \frac{1}{2^n n!} \frac{\mathrm{d}}{\mathrm{d}x^n} (x^2-1)^n \tag{5.23}$$

By combining equations 5.16, 5.20 and 5.21, we get two very similar forms

$$\Phi(\mathbf{x}_i) = \sum_{n=0}^{p} \sum_{m=-n}^{n} \frac{Y_n^m(\theta_i,\phi_i)}{r_i^{n+1}} \underbrace{\left\{ \sum_{j}^{N} q_j \rho_j^n Y_n^{-m}(\alpha_i,\beta_i) \right\}}_{M_n^m} \tag{5.24}$$

$$\Phi(\mathbf{x}_i) = \sum_{n=0}^{p} \sum_{m=-n}^{n} r_i^n Y_n^m(\theta_i,\phi_i) \underbrace{\left\{ \sum_{j}^{N} q_j \frac{Y_n^{-m}(\alpha_i,\beta_i)}{\rho_j^{n+1}} \right\}}_{L_n^m}, \tag{5.25}$$

with M_n^m and L_n^m as the multipole (singular) and local (regular) coefficients respectively.

While the approximation of $G = 1/4\pi r$ is easy with the FMM, forming multipole expansions for $\partial G/\partial \hat{n}$ requires a little more work. Working from $\partial G/\partial \hat{n} = \nabla G \cdot \hat{n}$, to form the desired multipole expansion requires the computation of $\partial M_n^m(r,\theta,\phi)/\partial \hat{n}$. Taking each derivative in turn, and initially working in spherical coordinates,

$$\begin{aligned} \mathrm{d}r &= \frac{n}{r} M_n^m \\ \mathrm{d}\theta &= r^n \frac{\partial Y_n^m(\theta,\phi)}{\partial \theta} \\ \mathrm{d}\varphi &= -imM_n^m. \end{aligned}$$

We can now use a simple conversion into cartesian coordinates in order to perform the dot product with $\hat{n}$:

$$\begin{pmatrix} \mathrm{d}x \\ \mathrm{d}y \\ \mathrm{d}z \end{pmatrix} = \begin{pmatrix} \sin\theta\sin\phi & (\cos\theta\cos\phi)/r & -(\sin\theta\sin\phi)/r \\ \sin\theta\sin\phi & (\cos\theta\sin\phi)/r & (\sin\theta\cos\phi)/r \\ \cos\theta & -(\sin\theta)/r & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathrm{d}r \\ \mathrm{d}\theta \\ \mathrm{d}\phi \end{pmatrix}$$

To translate and convert these expansions, we reproduce without proof the formulae from (Greengard, 1987) for multipole-multipole (5.26), multipole-local (5.27) and local-local translations (5.28), where $A_n^m = (-1)^n/(n-m)!(n+m)!$.

$$M_j^k = \sum_{n=0}^{j} \sum_{m=-n}^{n} \frac{\hat{M}_{j-n}^{k-m} i^{|k|-|m|-|k-m|} A_n^m A_{j-n}^{k-m} \rho^n Y_n^{-m}(\alpha,\beta)}{(-1)^n A_j^k} \quad \text{(M2M)} \quad (5.26)$$

$$L_j^k = \sum_{n=0}^{j} \sum_{m=-n}^{n} \frac{M_n^m i^{|k-m|-|k|-|m|} A_n^m A^k j Y_{j+n}^{m-k}(\alpha,\beta)}{(-1)^{j+k} A_{j+n}^{m-k} \rho^{j+n+1}} \quad \text{(M2L)} \quad (5.27)$$

$$L_j^k = \sum_{n=0}^{j} \sum_{m=-n}^{n} \frac{\hat{L}_n^m i^{|m|-|k|-|m-k|} A_{n-j}^{m-k} A_j^k \rho^{n-j} Y_{n-j}^{m-k}(\alpha,\beta)}{A_n^m} \quad \text{(L2L).} \quad (5.28)$$

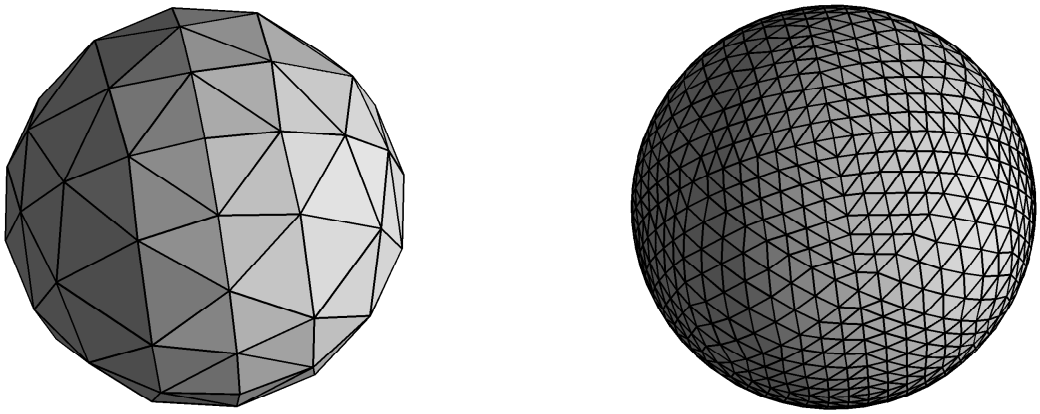
We note that there are more efficient versions of these translations, for instance, factorized translation operators that rely on the ability to rotate spherical harmonics to a chosen orientation – this would allow all $Y_n^m(\alpha,\beta)$ terms in equations 5.26-5.28 to be replaced with $Y_n^m(0,0)$, which can be precomputed. At the cost of extra memory

for precomputed values, this reduces the cost from $\mathcal{O}(p^4)$ to $\mathcal{O}(p^3)$ (Greengard and Rokhlin, 1997). Further algorithmic gains can be made using a plane wave expansion formulation for the M2L operator, where we convert multipoles into plane waves ($\mathcal{O}(p^3)$), translate them ($\mathcal{O}(p^2)$) and finally convert them back into local expansions ($\mathcal{O}(p^3)$). This method keeps the overall complexity of the M2L at $\mathcal{O}(p^3)$, but with lower constant terms (Greengard and Rokhlin, 1997).

5.3 Convergence

As an initial test of our FMM-BEM implementation, we wish to verify its convergence to a known solution based on the spatial resolution of our mesh. To do this, we use simple tests of constant potential and charge on the surface of a sphere. We use the analytical solution of $\phi = \partial\phi/\partial\hat{n} = 1$. The presence of an analytical solution makes this problem perfect for convergence testing for both 1st-kind (solving for ϕ with $\partial\phi/\partial\hat{n}$ known) and 2nd-kind (solving for $\partial\phi/\partial\hat{n}$ with ϕ known). The geometry is produced by forming an initial approximation for the sphere using 8 triangles, then recursively splitting each triangle into 4 smaller panels. In this way we can control the spatial discretization of the geometry and use it to test real-world convergence of our BEM for both first and second-kind equations. Two examples of the spherical domain are shown in figures 5·1a and 5·1b.

All tests were performed using a canonical right-preconditioned GMRES (algorithm 1) implementation, using our `FMM_Plan` framework (see appendix A) for the matrix-vector products. We used spherical harmonic expansions for the far-field, and the semi-analytical integral described in §3.1.2 for singular integrals. High precision Gauss quadrature was used for near-singular integrals. In all cases, $\theta_{\text{MAC}} = 0.5$, $p = 10$ and a solver tolerance of 10^{-6} was used in order to minimize all but discretization errors.



(a) 128 panels (b) 2048 panels

Figure 5·1: Triangular discretizations of a sphere

Figures 5·2 and 5·3 are encouraging, as they display the correct orders of convergence that we expect compared to comparable codes, namely $\mathcal{O}(1/N)$ for the 2nd-kind solve, while achieving slightly better than $\mathcal{O}(1/\sqrt{N})$ in the 1st-kind solve. This implies that our BEM formulation is correct, the singular / near-singular integrals are accurate, and the far-field approximation using the FMM is also giving the expected answer. This provides the basis we will use to continue experimenting with our code, and ensures that when we use relaxed solvers, we can test their convergence, and be sure that we are still getting the correct answer.

5.4 Relaxation

We are trying to minimize the time taken to solve BEM problems while not sacrificing accuracy, so it is vital to see how the modifications affect our code's behavior. First, we look at an example problem and see how the residual changes with GMRES iterations, and the p required to continue convergence.

Figure 5·4 shows how both $||r_k||$ and p change over the course of a solve. In this problem, 32768 panels were used to solve the first-kind equation on a sphere

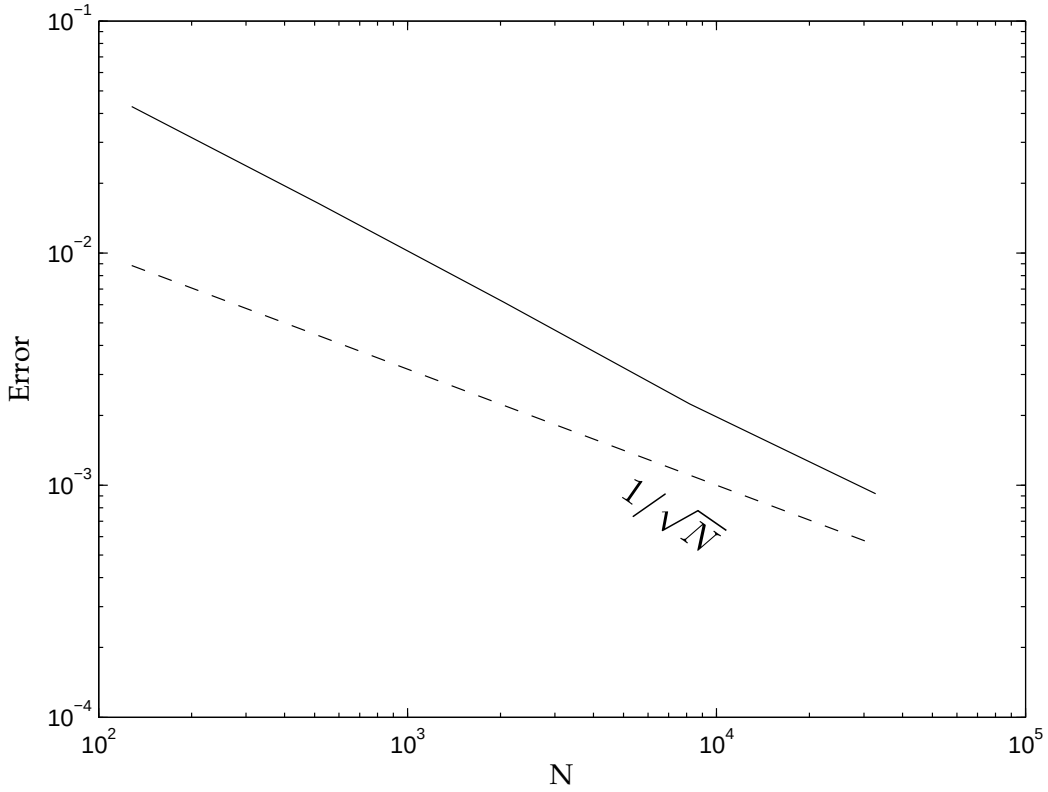


Figure 5·2: Convergence of 1st-kind solve for Laplace equation on a sphere, solving with $p = 10$, solver tolerance of 10^{-6}.

to a tolerance of 10^{-5} with an initial p set to 8. Clearly, as the residual drops, the p required to maintain convergence of the solver drops significantly, down from $p = 8$ at the first iteration to $p = 3$ at the 7th. Kernel translation operators in this implementation scale from $\mathcal{O}(p^4)$ for spherical harmonics to $\mathcal{O}(p^6)$ for Cartesian kernels, thus this drop in p can result in large savings in terms of the work being performed.

Next, we compare problems with and without a relaxation strategy. Figure 5·5 shows the results of tests from 8192 to 131072 panels with a solver tolerance of 10^{-5} using a multi-threaded evaluator on 4 cores. For all experiments in this section N_{CRIT} was chosen for each test to minimize the solution time. Only the time spent solving

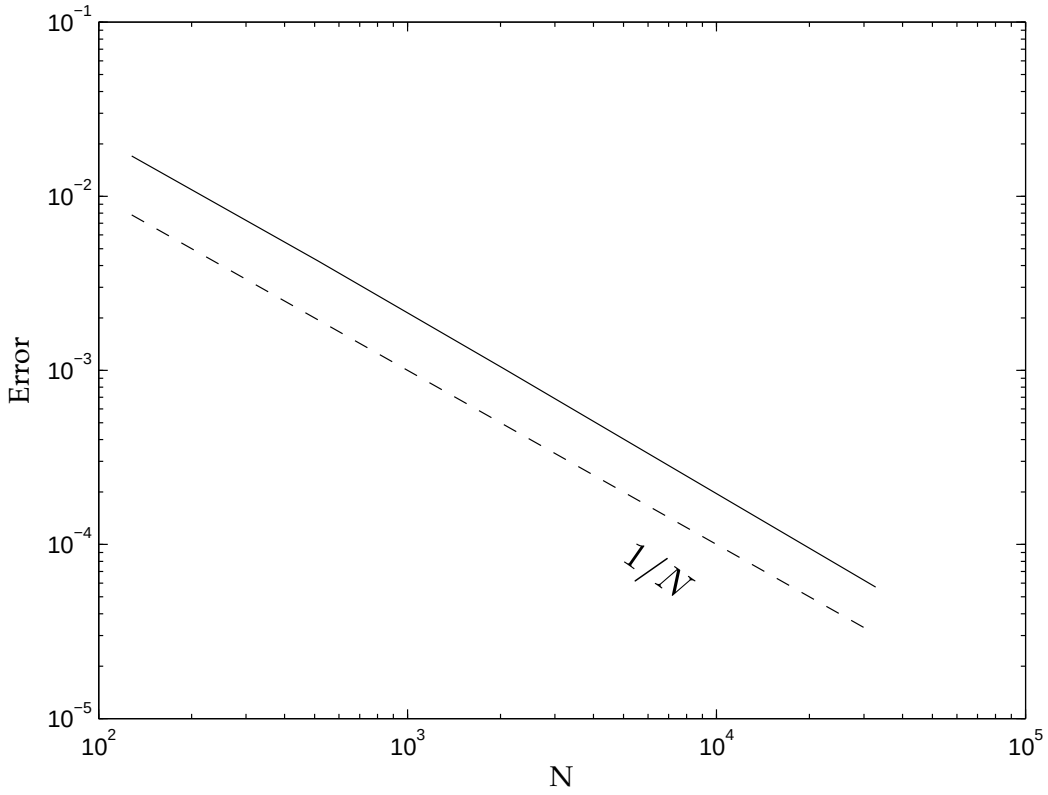


Figure 5·3: Converge of 2nd-kind solve for Laplace equation on a sphere, solving with $p = 10$, solver tolerance of 10^{-6}.

$Ax = b$, t_{solve} is plotted. While there are many metrics that could be used to establish performance, we choose to use time-to-solution in all cases for the following reasons:

1. Reporting times normalized on the number of iterations will both show the same speedups (due to the identical number of iterations required for relaxed and fixed-p solvers), and show exactly the same speedups as the total time-to-solution.

2. Times, rather than operation counts (whether total or per-iteration) are used to abstract away as much as possible the implementation details — we could potentially use a "reference operations" count, the operations required to perform a direct ($\mathcal{O}(N^2)$ solve), normalized by either total time (operations per second)

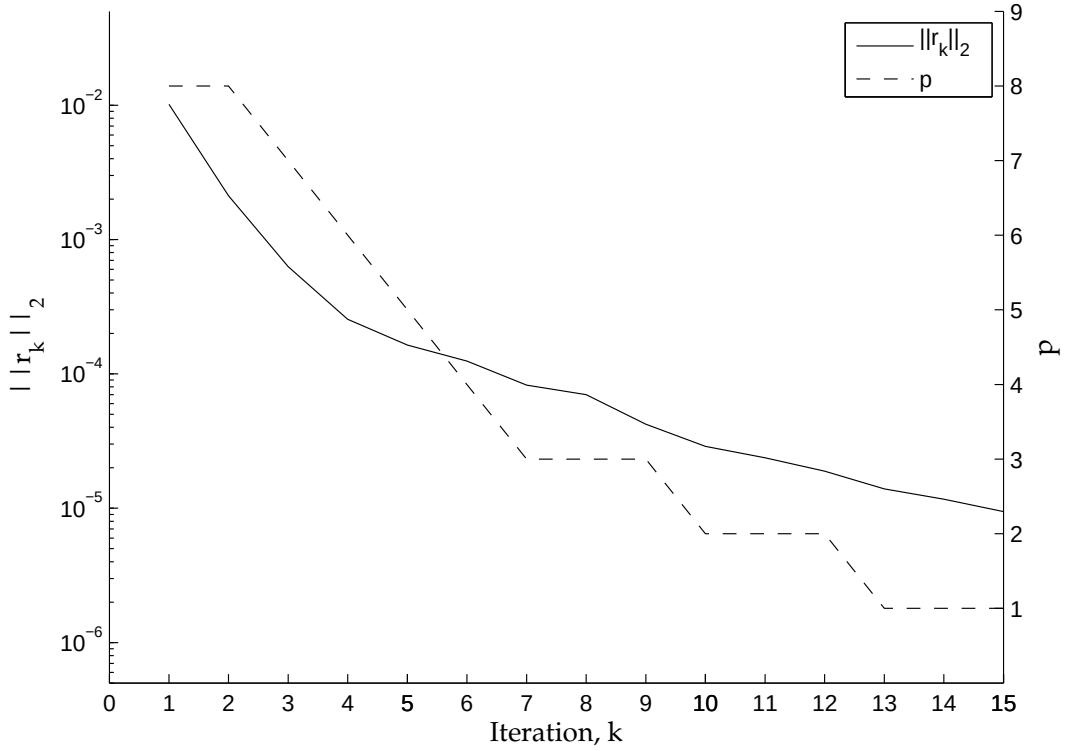


Figure 5·4: Behaviour of the residual $||r_k||$ and necessary p against GMRES iteration number

or by iterations (average operations per second per iteration), but neither of these options provide any real advantage to simply reporting times.

3. Showing times / operation counts for individual iterations is too unwieldy — instead of presenting a single value of speedup for each experiment, either multiple values (up to 70+ iterations in some of the later experiments in this thesis) or a plot would need to be presented.

From tables 5.1 and 5.2, we can see that there is a benefit from using a relaxation scheme for these problems. For problem sizes of 8192 panels and above, we see an average speedup of around 1.4× for 1st-kind solves and 1.2× for 2nd-kind problems. While these are not huge speedups, they show the first successful application of a relaxation scheme to FMM-BEM, as well as increasing speedups as problem sizes become

N	Non-Relaxed N_{CRIT}	Non-Relaxed t_{solve}	Relaxed N_{CRIT}	Relaxed t_{solve}	Speedup
2048	400	0.27	200	0.40	0.675
8192	400	2.58	200	1.7	1.52
32768	400	7.1	200	5.07	1.40
131072	400	30.1	200	20.72	1.45

Table 5.1: Speedups for Laplace 1st-kind relaxation, $p = 8$, solver tolerance of 10^{-5}

N	Non-Relaxed N_{CRIT}	Non-Relaxed t_{solve}	Relaxed N_{CRIT}	Relaxed t_{solve}	Speedup
2048	400	0.13	200	0.64	0.20
8192	400	1.49	200	1.19	1.25
32768	400	6.77	200	5.92	1.14
131072	400	30.01	200	24.2	1.24

Table 5.2: Speedups for Laplace 2nd-kind relaxation, $p = 8$, solver tolerance of 10^{-5}

larger.

Most interestingly, these results illustrate the differing parameters needed for optimal run times. The best choice of N_{CRIT} changes from 400 in the non-relaxed case, to 200 when a relaxed solver is used. This is logical, as for non-relaxed solves the best runtimes will be obtained by balancing the near and far-field evaluations (P2P and M2L). However, when we relax the system, the time taken for the far-field will decrease as a consequence of reducing p, while the amount of time for the P2P will remain constant (as an aside, this could also be relaxed, but would necessitate creating a new tree every iteration). This means that to minimize time-to-solution for relaxed cases, we want to reduce this constant P2P cost by adjusting N_{CRIT}.

Now that we have established that relaxation is successful in the sense that it a) converges to the correct answer, and b) provides a speedup in solve times over using

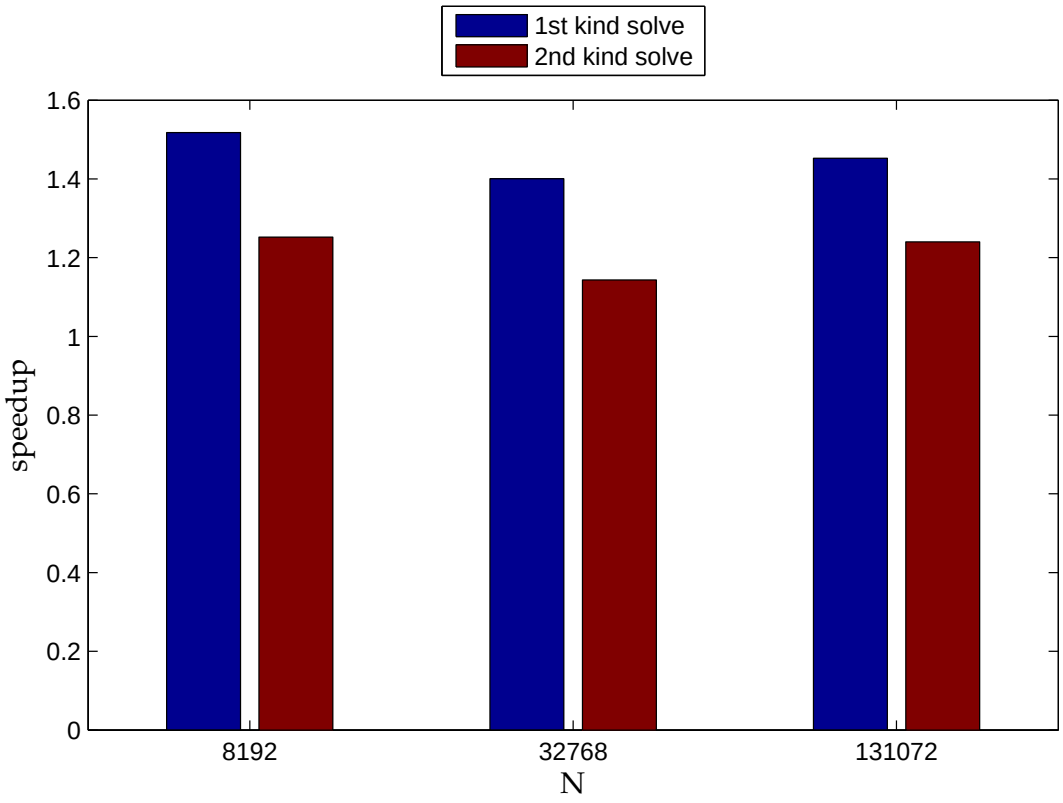


Figure 5·5: Speedup using relaxation strategy

a fixed p, we can investigate the kinds of situations that will provide the greatest benefits. To do this, we propose 2 hypotheses based on our initial results:

1. If a higher accuracy (and necessarily higher p) is needed, the benefits of relaxation will be greater, due to the ratio of work between the starting and minimum p becoming greater. This hypothesis will be tested by solving a 1st-kind equation for 32768 panels, with varying values of starting p. To eliminate potential discrepancies in iteration count between values of p, and to recognize that not all problems are as simple to solve as the sphere, we enforce 10 GMRES iterations, resulting in a solve to 5.5×10^{-5} tolerance.

2. As more iterations are required, relaxed solves will spend more time at low values of p, and thus the speedups will be greater. We will test this hypothesis

by artificially increasing the number of iterations in an analogous way to how we fixed the iteration count at 10 earlier. For this particular problem, increasing the number of iterations is equivalent to desiring a lower exit tolerance for the solver.

We now test both hypotheses, starting with the first; the relation between initial p and overall speedup, using the sphere test case and forcing the solver to perform 10 iterations.

p	Relaxed		Non-Relaxed		Speedup
	N_{CRIT}	t_{solve}	N_{CRIT}	t_{solve}	
5	100	4.21	100	6.34	1.51
8	100	6.24	400	12.4	1.99
10	150	8.76	400	18.5	2.11
12	150	13.2	600	25.3	1.92
15	150	19.3	600	38.3	1.98

Table 5.3: Laplace 1st-kind relaxation speedup with respect to p for sphere with 32768 panels.

The results in table 5.3 and summarized in figure 5·6 show a general trend – at the lowest values of p speedup is smaller, but it increases and levels out at approximately $2\times$ for larger tests. Looking at the times taken for individual iterations, this behaviour seems consistent. For the fastest total time, we desire an unbalanced tree on the first iteration (that is, where the time taken for near and far-fields is not equal), with the amount of P2P kept artificially low.

While this means that later, low-p iterations are very quick, it generally results in the first, high-p, iterations being slower than the equivalent fixed-p case. Thus, the speedups obtained are purely from the low cost of the last iterations – with higher p, the speedup in these iterations is greater, resulting in lower overall times to solution.

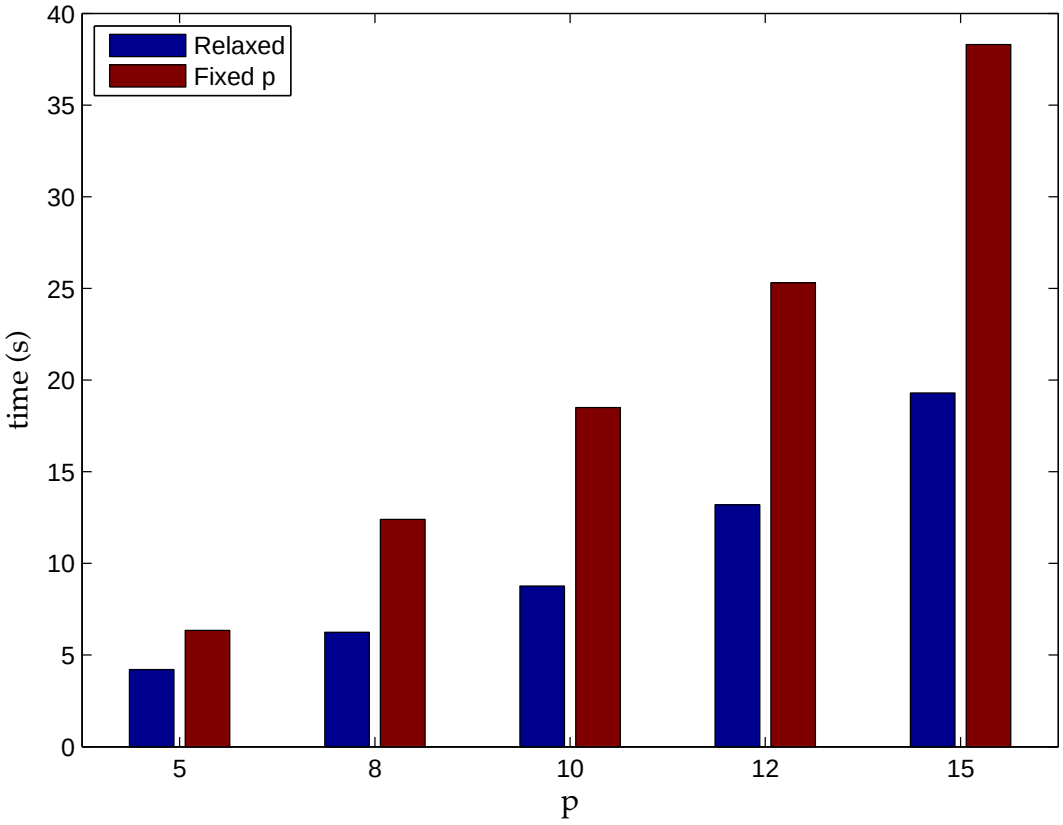


Figure 5·6: Time for 1st-kind Laplace equation solve on a sphere for 32768 panels with varying initial p for relaxed and fixed-p solvers. Iteration count capped at 10 for all cases.

The corollary of this, is that for lower initial p the speedups from later iterations is smaller, thus the overall speedup is smaller.

To test the performance of relaxation with respect to higher iteration counts, we use the sphere problem again, artificially setting the desired number of GMRES iterations. This has the effect of progressively solving to lower tolerances, while eliminating potential discrepancies in iteration counts between the relaxed and fixed p experiments. Table 5.3 shows these results. For all tests, the solver tolerance was disabled and initial $p = 10$. In all cases, a multi-threaded evaluator using 4 cores was used.

Iterations	Relaxed N_{CRIT}	Relaxed t_{solve}	Non-Relaxed N_{CRIT}	Non-Relaxed t_{solve}	Speedup
5	100	8.57	400	10.0	1.17
10	100	9.81	400	18.4	1.88
15	100	11.1	400	26.9	2.42
20	100	12.4	400	35.3	2.85
25	100	13.7	400	43.8	3.20
50	100	20.6	400	86.2	4.18

Table 5.4: Laplace 1st-kind relaxation speedup with respect to iteration count for sphere with 32768 panels. $p = 10$

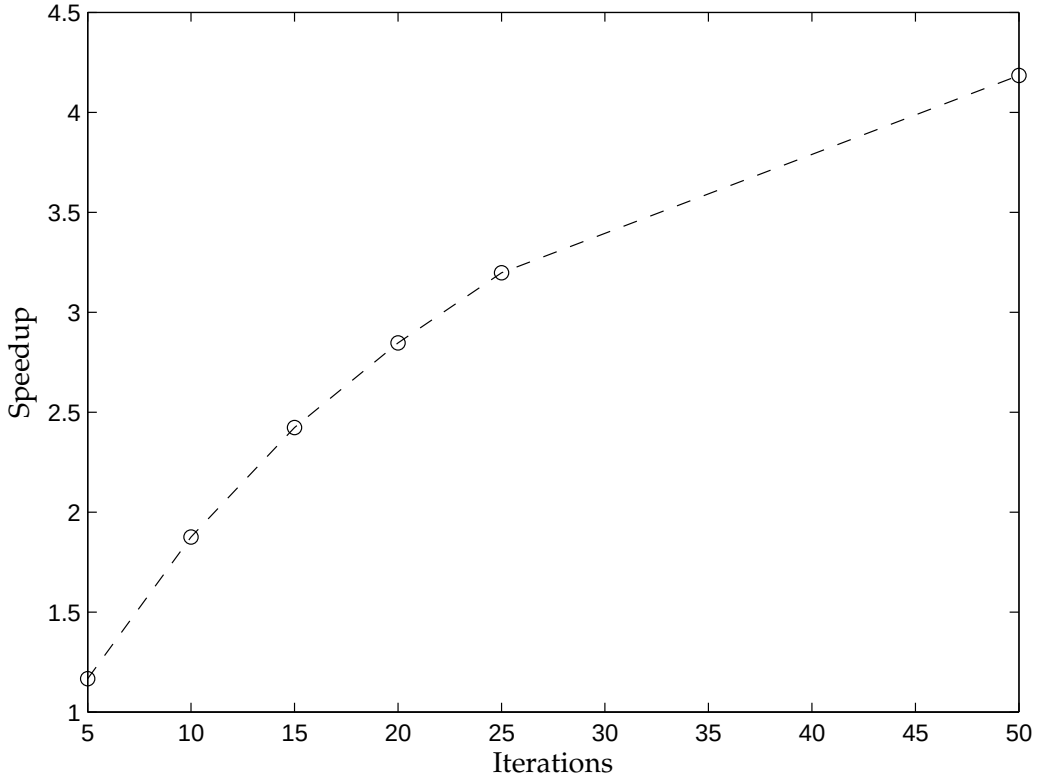


Figure 5·7: Speedup of 1st-kind Laplace solve on a sphere for 32768 panels with varying initial iteration count. $p = 10$ for all cases.

From table 5.4 and figure 5·7, the upward trend of speedup with respect to itera-

tion count is very clear. From modest initial speedups, each increase in iterations gives a corresponding increase in acceleration. Further, we can see that each non-relaxed iteration adds approximately $1.68s$ to t_{solve}, while each relaxed iteration adds a mere $0.276s$. Thus we can predict speedup as the iteration count continues to increase, with a $4.73\times$ acceleration predicted at 75 iterations, and $4.94\times$ at 100 iterations.

Combining our 2 earlier hypotheses and supporting results for both, we can now state that the optimal conditions for speedup from using a relaxation scheme are both a high iteration count, and high desired precision. Any problems exhibiting both these quantities should show significant speedups over standard solvers using a fixed p.

5.5 Conclusions

In this section, we have laid the groundwork for applying relaxation schemes to FMM-BEM problems. We have shown that our implementation is accurate, and that it converges as expected spatially. Beyond that, we have demonstrated the application of a relaxation scheme to FMM-BEM, the first work of its kind. The parameter space for determining the expected speedup from relaxation has been explored, giving us confidence that any problems that require high accuracy and present tricky linear systems, resulting in many GMRES iterations, will give speedups in the order of $2-4\times$, a significant result.

Chapter 6

BEM with inexact GMRES for the Stokes equation

Now that we have demonstrated the ability of relaxed GMRES to correctly solve FMM-BEM problems based on Laplace's equation and shown the speedups that this approach can give, we can continue to more substantial problems. Results from §5.4 suggest that as p increases the potential speedup from relaxation also increases. This implies that the benefits from relaxed solvers will increase with a corresponding need for accuracy. The next conclusion we can draw from the Laplace equation tests, is that we can correlate speedups with the number of iterations needed to solve – in GMRES solvers, we spend a majority of iterations with residual values close to the desired final tolerance. As the linear system increases in difficulty to solve, we have more low-precision iterations and it is these iterations that will give us the greatest speedup.

The Stokes equation is a vector equation, and significantly more involved than Laplace, involving $62K$ operations for each K-th order integration between two panels, compared to $8K$ for Laplace. It is used in applications such as biomedicine (Rahimian et al., 2010) and MEMS (Microelectromechanical systems)(Fachinotti et al., 2007). The use of FMM-BEM for this equation is also well established (Liu, 2009; Gomez and Powert, 1997). The difficulty of the resulting problems comes from both the added complications from the equation itself, requiring solving for 3 components (of velocity or surface traction) at every target panel, along with the more difficult linear systems that arise. Additional to these sources of complications, high

precision is required in order to maintain accuracy (Liu, 2009). This combination of tough linear system, high accuracy and added difficulties within the integrals that must be computed, means we anticipate significant speedups from relaxing the solve, at least on par with those experienced in our Laplace experiments.

6.1 Equation

We start from the unsteady Navier-Stokes equations for fluid motion in terms of the velocity, $\mathbf{u}$, pressure, p, viscocity μ and fluid density ρ,

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u}, \tag{6.1}$$

and concentrate on the case where the Reynolds number, $Re_L = \rho UL/\mu$ is small, or in other words, diffusion effects $\nabla^2\mathbf{u}$ are much larger than convective effects, $\mathbf{u}\cdot\nabla\mathbf{u}$.

$$\nabla^2\mathbf{u} \gg \mathbf{u}\cdot\nabla\mathbf{u} \tag{6.2}$$

In this case, we obtain 2 equations, the Stokes equation (6.3) and the linearized Navier-Stokes equation 6.4 for the steady and unsteady cases respectively.

$$\mu\nabla^2\mathbf{u} = \nabla p \tag{6.3}$$

$$\rho\frac{\partial \mathbf{u}}{\partial t} = -\nabla p + \mu\nabla^2\mathbf{u}. \tag{6.4}$$

Similar to the Laplace equation, we want the fundamental solution(s) for equation 6.3, and these are the Stokeslet 6.5 and the stresslet 6.7, shown in both matrix and tensor notation forms.

$$S_{ij}(\mathbf{x},\mathbf{y}) = \frac{\delta_{ij}}{r} + \frac{(x_i - y_i)(x_j - y_j)}{r^3} \tag{6.5}$$

$$= \frac{1}{r^3}\begin{pmatrix} r^2 + dx^2 & dx \cdot dy & dx \cdot dz \\ dx \cdot dy & r^2 + dy^2 & dy \cdot dz \\ dx \cdot dz & dy \cdot dz & r^2 + dz^2 \end{pmatrix} \tag{6.6}$$

$$T_{ijk}(\mathbf{x},\mathbf{y},\hat{n}) = 6\frac{(x_i - y_i)(x_j - y_j)(x_k - y_k)n_k}{r^5} \tag{6.7}$$

$$= 6\frac{(\mathrm{d}\mathbf{x} \cdot \hat{\mathbf{n}})}{r^5}\begin{pmatrix} dx^2 & dx \cdot dy & dx \cdot dz \\ dx \cdot dy & dy^2 & dy \cdot dz \\ dx \cdot dz & dy \cdot dz & dz^2 \end{pmatrix} \tag{6.8}$$

These fundamental solutions will be used in the next section when we formulate a BEM form of the Stokes equation, suitable for solving with the framework used throughout this thesis.

6.2 BEM

Similar to how we began from Green's second identity while deriving the boundary integral form of the Laplace equation, we start here from Lorentz's reciprocal relation, similar to standard derivations (Pozrikidis, 1992; Liu, 2009).

$$\nabla \cdot (\mathbf{u}' \cdot \sigma - \mathbf{u} \cdot \sigma') = 0 \tag{6.9}$$

$$\frac{\partial}{\partial x_k}(u_i'\sigma_{ik} - u_i\sigma_{ik}') = 0, \tag{6.10}$$

where $\mathbf{u}$ and $\mathbf{u}'$ are solutions to Stokes equations, and σ and σ' are the associated stress tensors. Looking at the domain in figure 6·1, if we identify $\mathbf{u}'$ as the velocity

induced from a point source with strength $\mathbf{g}$, we can obtain

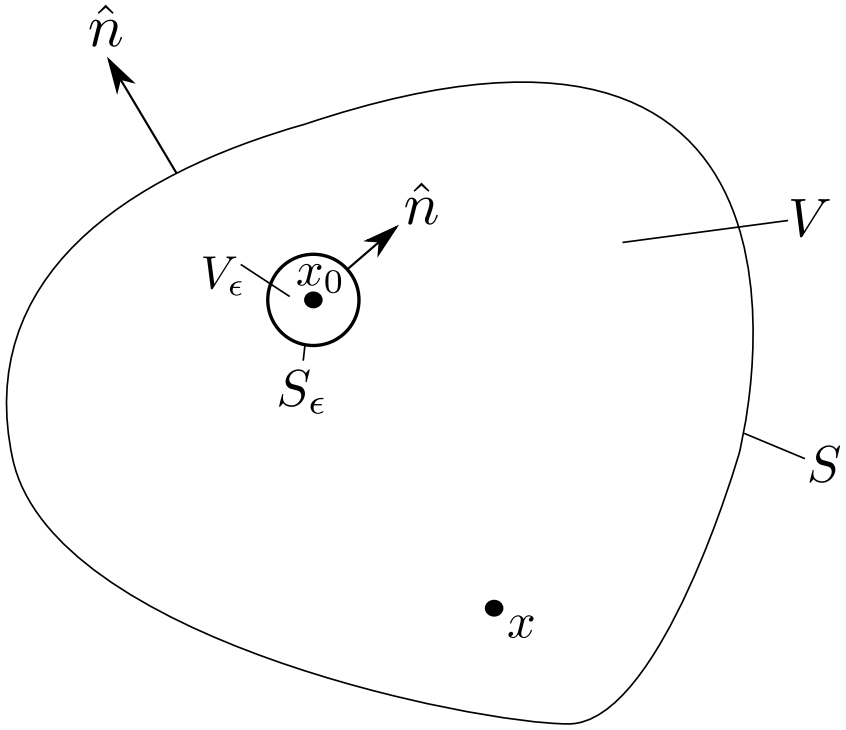


Figure 6·1: Domain used for derivation of Stokes boundary integral equation

$$u_i(\mathbf{x}) = \frac{1}{8\pi\mu} G_{ij}(\mathbf{x}, \mathbf{x}_0) g_j, \quad \sigma'_{ijk} = \frac{1}{8\pi} T_{ij}(\mathbf{x}, \mathbf{x}_0, \hat{n}) g_j. \tag{6.11}$$

Discarding $\mathbf{g}$ as a constant, and substituting 6.11 into 6.10, we get

$$\frac{\partial}{\partial x_k} \left(G_{ij}(\mathbf{x}, \mathbf{x}_0)\sigma_{ik}(\mathbf{x}) - \mu u_i(\mathbf{x}) T_{ijk}(\mathbf{x}, \mathbf{x}_0) \right) = 0. \tag{6.12}$$

Next, we integrate 6.12 over a volume, V, set $\mathbf{x}_0$ outside of V and apply the divergence theorem

$$\int_\Gamma \left[G_{ij}(\mathbf{x}, \mathbf{x}_0)\sigma_{ik}(\mathbf{x}) - \mu u_i(\mathbf{x}) T_{ijk}(\mathbf{x}, \mathbf{x}_0) \right] \hat{n}_k(\mathbf{x}) \mathrm{d}\mathbf{S}(\mathbf{x}) = 0. \tag{6.13}$$

Next, moving $\mathbf{x}_0$ towards the surface, we select V_ϵ as a small spherical volume around $\mathbf{x}_0$ with radius ϵ. Integrating over $V - V_\epsilon$ and using the divergence theorem once more

$$\int_{\Gamma, S_\epsilon} \left[G_i j(\mathbf{x}, \mathbf{x}_0) \sigma_{ik}(\mathbf{x}) - \mu u_i(\mathbf{x}) T_{ijk}(\mathbf{x}, \mathbf{x}_0) \right] \hat{n}_k(\mathbf{x}) \mathrm{d}\mathbf{S}(\mathbf{x}) = 0. \tag{6.14}$$

Letting $\epsilon \to 0$, $\mathbf{G}, \mathbf{T}$ reduce to the Stokeslet and stresslet,

$$G_{ij} = S_{ij} = \frac{\delta_{ij}}{\epsilon} + \frac{\hat{x}_i \hat{x}_j}{\epsilon^3}, \quad T_{ijk} = -6 \frac{\hat{x}_i \hat{x}_j \hat{x}_k}{\epsilon^5}, \tag{6.15}$$

with $\hat{x}_i = x_i - x_{0i}$. Over the volume S_ϵ, $\hat{n} = \hat{x}/\epsilon$ and $\mathrm{d}\mathbf{S} = \epsilon^2 \mathrm{d}\Omega$. Substituting 6.15 into 6.14 we now obtain

$$\begin{aligned} \int_{\Gamma, S_\epsilon} \left[G_i j(\mathbf{x}, \mathbf{x}_0) \sigma_{ik}(\mathbf{x}) - \mu u_i(\mathbf{x}) T_{ijk}(\mathbf{x}, \mathbf{x}_0) \right] \hat{n}_k(\mathbf{x}) \mathrm{d}\mathbf{S}(\mathbf{x}) = \\ - \int_{S_\epsilon} \left[\left(\delta_{ij} + \frac{\hat{x}_i \hat{x}_j}{\epsilon^2} \right) \sigma_{ik}(\mathbf{x}) + 6 \mu u_i(\mathbf{x}) \frac{\hat{x}_i \hat{x}_j \hat{x}_k}{\epsilon^4} \right] \hat{x}_k \, \mathrm{d}\Gamma. \end{aligned} \tag{6.16}$$

When we take the limit $\epsilon \to 0$, $\mathbf{u}$ and σ over the volume S_ϵ tend to $\mathbf{u}(\mathbf{x}_0)$ and $\sigma(\mathbf{x}_0)$. We also note that as $\epsilon \to 0$ the stress term on the right-hand-side of 6.16 will decrease linearly, while the velocity term will tend to a constant,

$$-6 \mu u_i(\mathbf{x}_0) \frac{1}{\epsilon^4} \int_{S_\epsilon} \hat{x}_i \hat{x}_j \, \mathrm{d}S(\mathbf{x}). \tag{6.17}$$

Applying the divergence theorem to the surface integral in 6.17, we get

$$\int_{S_\epsilon} \hat{x}_i \hat{x}_j \, \mathrm{d}S(\mathbf{x}) = \epsilon \int_{S_\epsilon} \hat{x}_i \hat{n}_j \, \mathrm{d}S(\mathbf{x}) = \epsilon \int_{V_\epsilon} \frac{\partial \hat{x}_i}{\partial \hat{x}_j} \, \mathrm{d}V(\mathbf{x}) = \delta_{ij} \frac{4}{3} \pi \epsilon^4. \tag{6.18}$$

Finally plugging 6.18 into 6.16 (with the right hand side collapsed to 6.17), we get the expected boundary integral form:

$$u_j(\mathbf{x_0}) = -\frac{1}{8\pi\mu}\int_\Gamma \sigma_{ik}(\mathbf{x})n_k(\mathbf{x})G_{ij}(\mathbf{x},\mathbf{x}_0)\ \mathrm{d}S(\mathbf{x}) + \frac{1}{8\pi}\int_\Gamma u_i(\mathbf{x})T_{ijk}(\mathbf{x},\mathbf{x}_0)n_k(\mathbf{x})\ \mathrm{d}S(\mathbf{x}). \tag{6.19}$$

Setting $t = \sigma \cdot \hat{n}$ and taking $\mathbf{x}_0$ to a smooth part of the boundary, $\int_\Gamma u_i(\mathbf{x})T_{ijk}(\mathbf{x},\mathbf{x}_0)n_k(\mathbf{x})\ \mathrm{d}\Gamma(\mathbf{x}) \to 0$, and $\int_\Gamma \sigma_{ik}(\mathbf{x})n_k(\mathbf{x})G_{ij}(\mathbf{x},\mathbf{x}_0)\ \mathrm{d}\Gamma(\mathbf{x}) \to 1/2$. This gives the final form of the boundary integral equation before discretization

$$\frac{1}{2}u(\mathbf{x_0}) = -\frac{1}{8\pi\mu}\int_\Gamma t_i(\mathbf{x})G_{ij}(\mathbf{x},\mathbf{x}_0)\ \mathrm{d}\Gamma + \frac{1}{8\pi}\int_\Gamma u_i(\mathbf{x})T_{ijk}(\mathbf{x},\mathbf{x}_0)n_k(\mathbf{x})\ \mathrm{d}\Gamma. \tag{6.20}$$

Finally discretizing with constant panels we obtain

$$\frac{1}{2}u_i(\mathbf{x_0}) = -\frac{1}{8\pi\mu}\sum_{j=1}^{N} t_j \int_\Gamma G_{ij}\ \mathrm{d}\Gamma_j + \frac{1}{8\pi}\sum_{j=0}^{N} u_j \int_\Gamma T_{ijk}\cdot n_k\ \mathrm{d}\Gamma_j. \tag{6.21}$$

6.3 Expansions

To approximate the Stokeslet and stresslet we use a method based on decomposing 6.5 and 6.7 into sets of harmonic equations, each suitable to be approximated using the Laplace FMM from §5.2 and recombined to give the full answer (Tornberg and Greengard, 2008).

First, we can write the Stokeslet S_{ij} as

$$S_{ij} = \left(\delta_{ij} - (x_j - y_j)\frac{\partial}{\partial x_i}\right)\frac{1}{|\mathbf{x}_i - \mathbf{x}_j|}, \tag{6.22}$$

and $F_i^m = F_i(\mathbf{x}_m)$ as

$$F_i^m = \sum_{n=0}^{N} S_{ij}(\mathbf{x}_m, \mathbf{x}_n)\cdot \mathbf{f}_n \tag{6.23}$$

Then, we can perform more manipulations to get another alternate form,

$$F_i^m = \sum_{j=1}^{3} \left[\left(\delta_{ij} - x_j^m \frac{\partial}{\partial x_i} \right) \sum_{n=1}^{N} \frac{f_j^n}{r_{nm}} \right] + \frac{\partial}{\partial x_i} \sum_{n=1}^{N} \frac{\mathbf{x}^n \cdot \mathbf{f}^n}{r_{nm}}. \tag{6.24}$$

This decomposition lets us approximate the Green's function operator with 4 Laplace FMMs for $1/r$, with source strengths $f_1^n,\ f_2^n,\ f_3^n,\ (\mathbf{x}^n \cdot \mathbf{f}^n)$. Similarly, we can rewrite the stresslet, D_{ij} in a similar form:

$$\begin{aligned} D_{ij}(\mathbf{x}, \mathbf{y}, \hat{n}) = \frac{1}{6} \sum_{k=1}^{3} \Bigg[& \left(\delta_{ij} - (x_j - y_j) \frac{\partial}{\partial x_i} \right) \frac{(x_k - y_k)\hat{n}_k}{|\mathbf{x} - \mathbf{y}|^3} \\ & + \left(\delta_{ik} - (x_k - y_k) \frac{\partial}{\partial x_i} \right) \frac{(x_j - y_j)\hat{n}_k}{|\mathbf{x} - \mathbf{y}|^3} \Bigg] \end{aligned} \tag{6.25}$$

and $G_i^m = G_i(\mathbf{x}_m)$ as

$$G_i^m = \sum_{n=0}^{N} D(\mathbf{x}_m, \mathbf{x}_n, \hat{n}_n) \cdot \mathbf{g}_n \tag{6.26}$$

$$\begin{aligned} G_i^m = \frac{1}{6} \sum_{j=1}^{3} \Bigg[& \left(\delta_{ij} - x_j^m \frac{\partial}{\partial x_i} \right) \sum_{n=1}^{N} \left\{ \frac{(\mathbf{r}_{nm} \cdot \hat{\mathbf{n}}^n) g_j^n}{r_{nm}^3} + \frac{(\mathbf{r}_{nm} \cdot \mathbf{g}^n)\hat{n}_j^n}{r_{nm}^3} \right\} \Bigg] \\ & + \frac{1}{6} \frac{\partial}{\partial x_i} \sum_{n=1}^{N} \left\{ \frac{(\mathbf{r}_{nm} \cdot \hat{\mathbf{n}}^n)(\mathbf{x}^n \cdot \mathbf{g}^n)}{r_{nm}^3} + \frac{(\mathbf{r}_{nm} \cdot \mathbf{g}^n)(\hat{\mathbf{n}}^n \cdot \mathbf{x}^n)}{r_{nm}^3} \right\} \end{aligned} \tag{6.27}$$

Similar to the Stokeslet case, we are able to approximate the Stresslet using 4 harmonic approximations, this time for $1/r^3$, where every particle contributes 2 dipoles to each approximation.

$$(\mathbf{r}_{nm} \cdot \hat{\mathbf{n}}^n) g_1^n + (\mathbf{r}_{nm} \cdot \mathbf{g}^n)\hat{n}_1^n \tag{6.28}$$

$$(\mathbf{r}_{nm} \cdot \hat{\mathbf{n}}^n) g_2^n + (\mathbf{r}_{nm} \cdot \mathbf{g}^n)\hat{n}_2^n \tag{6.29}$$

$$(\mathbf{r}_{nm} \cdot \hat{\mathbf{n}}^n) g_3^n + (\mathbf{r}_{nm} \cdot \mathbf{g}^n)\hat{n}_3^n \tag{6.30}$$

$$(\mathbf{r}_{nm} \cdot \hat{\mathbf{n}}^n)(\mathbf{x}^n \cdot \mathbf{g}^n) + (\mathbf{r}_{nm} \cdot \mathbf{g}^n)(\hat{\mathbf{n}}^n \cdot \mathbf{x}^n) \tag{6.31}$$

When we come to evaluate all of these expansions, we note that both the Stokeslet and stresslet are of the same form, just with different source strengths for each expansions. Assuming our 4 expansions $M_i,\ i = 1..4$ are ordered in the same way we defined the influences of particles earlier,

$$\phi_i^m = \sum_{j=1}^{3} \left(\delta_{ij} - x_j^m \frac{\partial}{\partial x_i} \right) M_j + \frac{\partial}{\partial x_i} M_4. \tag{6.32}$$

6.4 Convergence

Just as for the Laplace equation, we wish to show our method is working as intended, and thus converges to the correct accuracy as a function of the spatial discretization. For the Stokes equations, we choose to simulate the low-Reynolds number flow over a sphere. A classical problem in Stokes flow, when placed in a uniform flow of speed u_x, the drag over the sphere can be found analytically to be $F_d = 6\pi\mu R u_x$.

Imposing $\mathbf{u} = (1, 0, 0)^T$ at the center of every panel, we solve a first-kind equation for the traction force, $\mathbf{t}$, from which we can then get the total drag force from the integral

$$F_d = \int_\Gamma t_x \, \mathrm{d}\Gamma' = \sum_{j=1}^{N} t_{x_j} \cdot A_j \tag{6.33}$$

In these tests, we set $R = 1$, $u_x = 1$ and $\mu = 10^{-3}$, giving a drag force of $F_d = 0.01885$. For each case, p was increased until the solution stopped improving. Figure 6·2 shows the t_x, the traction force exerted in the x-direction on the sphere

It is worth pointing out that while we do have some oscillations in the traction force, their magnitude is small ($\mathcal{O}(10^{-4})$), and they do not affect the derived quantity (in this case, total drag force) that we are interested in.

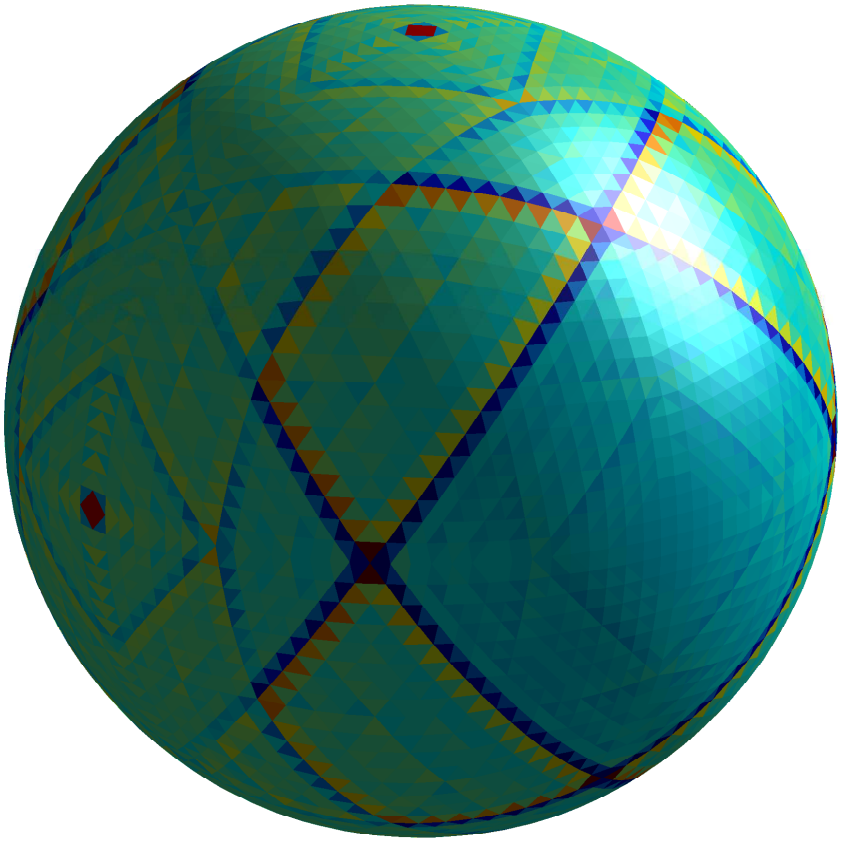


Figure 6·2: Traction force t_x exerted in the x-direction on a sphere, 32768 panels, $p = 16$, $p_{\min} = 5$ solved to 10^{-5} tolerance.

As shown in figure 6·3 we see that for 1st-kind equations (likely to be the most common for Stokes problems – the velocity is prescribed and we wish to find the surface traction), we converge as $\mathcal{O}(1/\sqrt{N})$, in-line with the results from §5.3 for the Laplace equation.

6.5 Relaxation

Similar to the Laplace equation, we are interested in showing both that using a relaxation scheme converges to the same solution (and thus error) as a non-relaxed

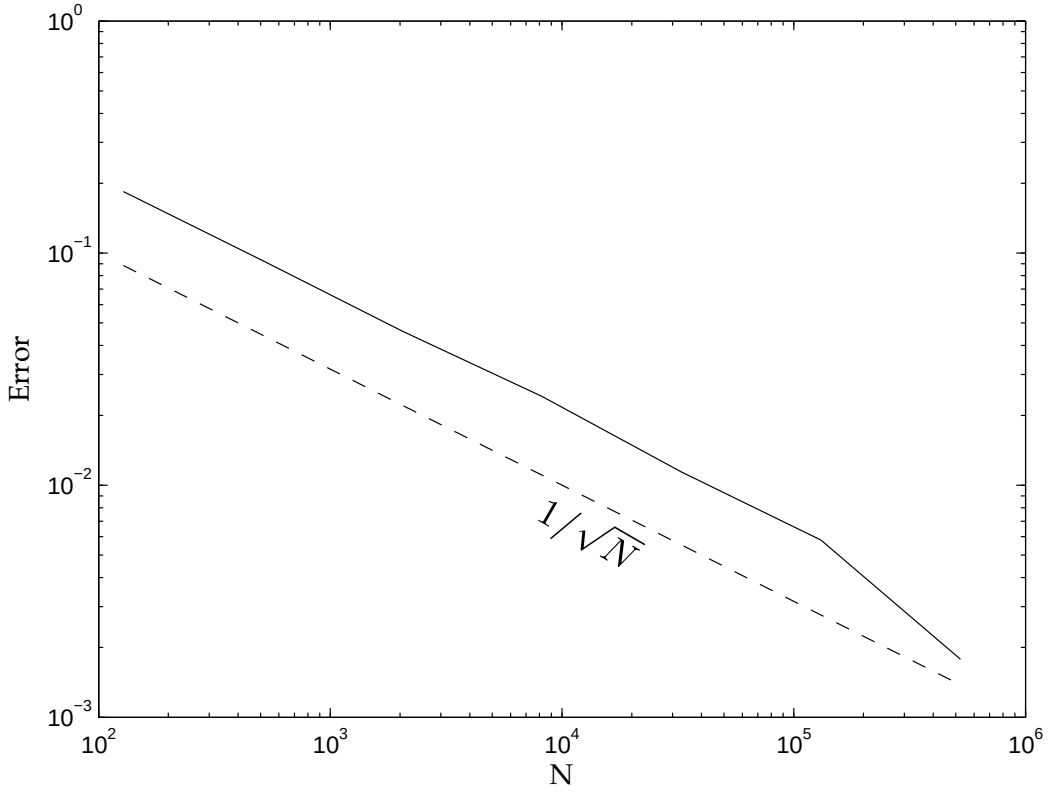


Figure 6·3: Convergence of first-kind equation for Stokes flow around a sphere. Initial $p = 16$, linear system solved to 10^{-5} tolerance.

run, and that we get a benefit in terms of speed.

First, to estimate the benefits we might obtain, we look at the residual history of a solve for the traction force on a sphere, the same test used in §6.4. Figure 6·4 shows that beyond an initial fast reduction in residual in the first 3-4 iterations, before convergence slows until we reach the final residual after 27 iterations. This is encouraging for the use of relaxation, as we quickly reach a residual that will allow us to dramatically reduce the accuracy needed for the matvec.

It is this reduction in accuracy that we anticipate producing large time savings for the Stokes equations. As the FMM expansions for Stokes consist of 4 Laplace expansions, and each expansion translation scales as $\mathcal{O}(p^4)$, the potential savings are

large. For instance, starting a problem with $p = 16$ as above, and finishing at $p = 5$, the far-field evaluation for each low-accuracy iteration will be approximately $100\times$ faster than the high-precision iteration.

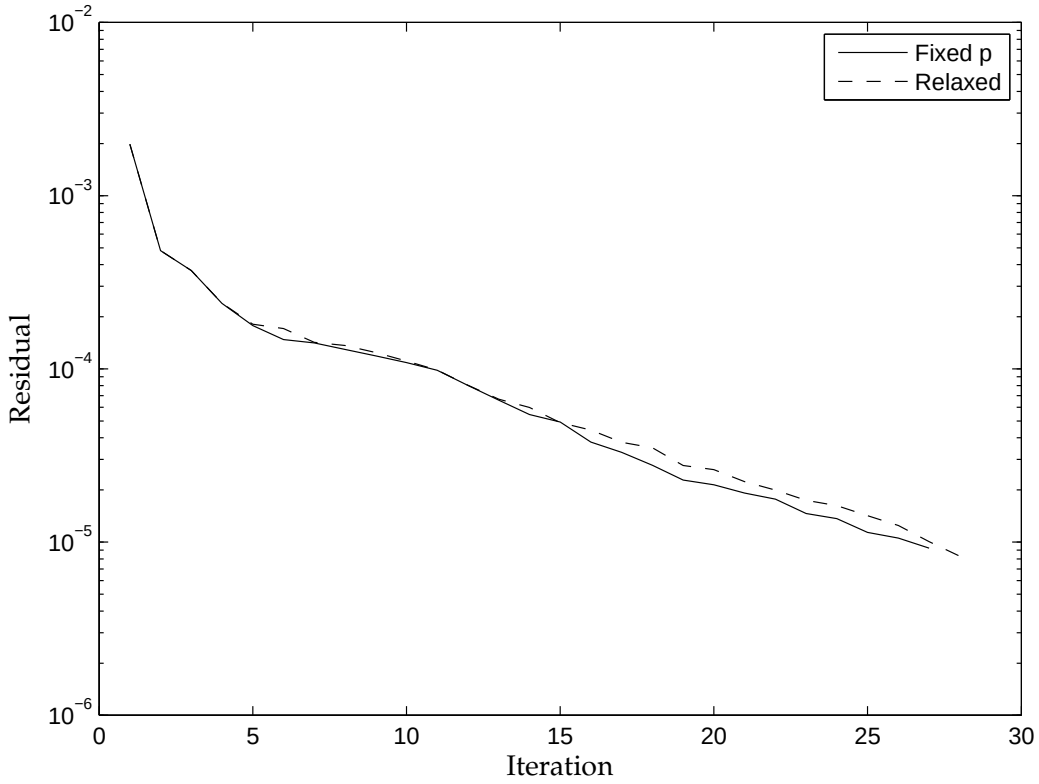


Figure 6·4: Residual history solving for surface traction on the surface of a sphere, 10^{-5} solver tolerance, 8192 panels, $p = 16$.

During this period of initial testing, we confirmed the departure from standard FMM practice as regards to choosing N_{CRIT} and p that we first encountered in §5.4 for the Laplace equation. Traditionally, we would try to balance the amount of time spent on the near-field (P2P) and far-field (dominated by M2L). While this works well for the case where we keep p constant, it requires an adjustment for the relaxed case. In order for the optimal time to solution, we actually want to balance near and far-fields for the value of p we spend most time on — in the majority of cases, this is *the lowest instance of p*. Figure 6·5 shows the residual history of a solver over a

sphere, along with the breakdown of P2P and M2L for each iteration.

During the verification of our relaxed solver, we encountered a new issue unseen for the Laplace equation in chapter 5, the need to enforce a *minimum* p in order to maintain accuracy. We ran extensive tests on the smallest value of p that could be allowed within the relaxed solver, whilst still maintaining both accuracy and convergence properties. Some of these results are summarized in table 6.1.

	2048 panels		8192 panels		32768 panels	
$p_{\min}$	Error	it	Error	it	Error	it
5	4.70×10^{-2}	22	2.44×10^{-2}	28	1.34×10^{-2}	28
4	4.49×10^{-2}	23	2.51×10^{-2}	29	1.25×10^{-2}	29
3	4.64×10^{-2}	22	2.78×10^{-2}	29	1.39×10^{-2}	29
2	4.95×10^{-2}	25	2.62×10^{-2}	33	3.18×10^{-2}	39
1	4.96×10^{-2}	25	2.99×10^{-2}	51	4.77×10^{-2}	53

Table 6.1: Effect of $p_{\min}$ on accuracy and convergence for Stokes flow around a sphere for differing values of N. Error is on the total drag force in the x-direction, F_x.

These results show that as we discretize more finely, the value of $p_{\min}$ becomes ever more important. While for 2048 panels, the effect in terms of both accuracy and iteration count is small, it becomes more noticeable at 8192 panels – with $p_{\min} = 1$ we require almost twice as many iterations to converge to our tolerance with respect to $p_{\min} = 5$. With 32768 panels, the problems become even more noticeable, with accuracy severely affected at $p_{\min} = 2$ and below. This loss of accuracy is so large that for $p_{\min} = 1$ the error is almost $4\times$ worse than for $p_{\min} = 5$.

Due to this behaviour, we choose to prioritize accuracy over some speed gains and use $p_{\min} = 5$ for all further Stokes flow cases seen in this thesis. We will lose some potential speedups from iterations at even lower values of p, but for all cases tried in the course of the experimentation for this thesis, $p_{\min} = 5$ has produced accurate results with good convergence properties.

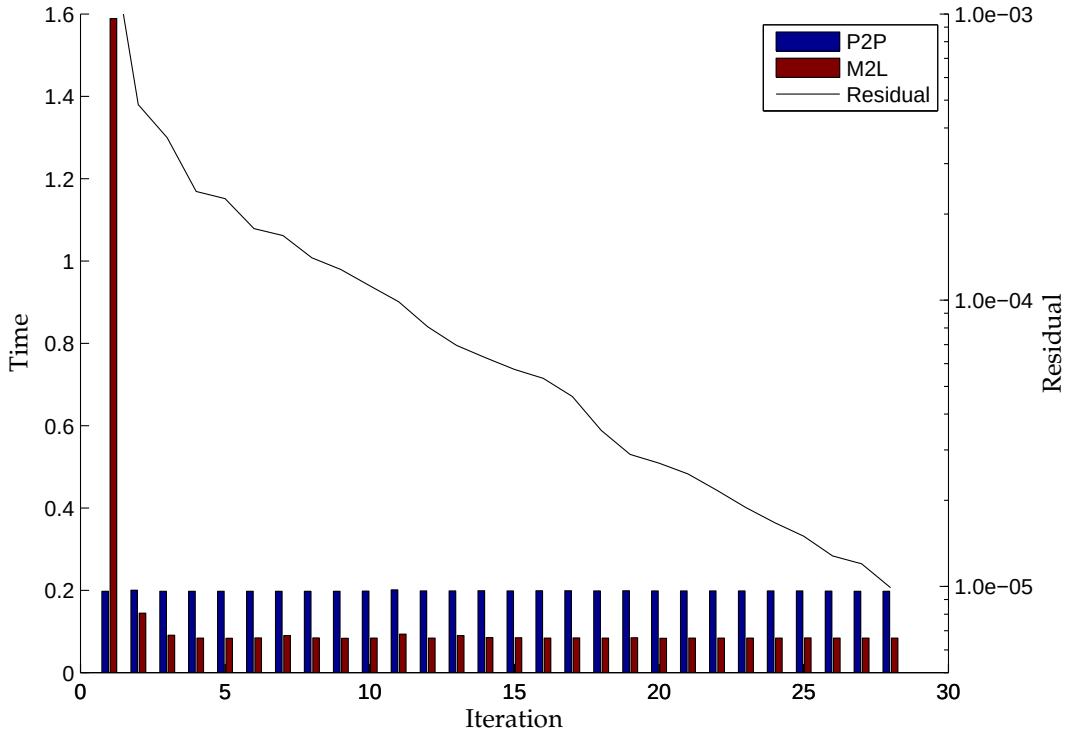


Figure 6·5: Residual history solving for surface traction on the surface of a sphere, with time breakdown between P2P and M2L. 10^{-5} solver tolerance, 8192 panels, $p = 16$.

The results shown in figure 6·5 indicate that we spend the vast majority of iterations at the minimum p, so, confirming our observation from chapter 5 and the Laplace equation, it is in our best interests to balance the computation for low p case. In practise, this means setting N_{CRIT} to the lowest possible in order to maintain the accuracy of the FMM, as the M2L component becomes very fast at the lowest values of p.

After exploring the behavior of the relaxed solver, we can finally present speedup results. Figure 6·6 shows the speedup of the relaxed solver over keeping p fixed for $N = 2048$ to 131072 (6144 to 393216 unknowns).

These results demonstrate that our optimism about the potential of relaxed solvers for Stokse problems was clearly warranted. Speedups for problems of 8192 panels and

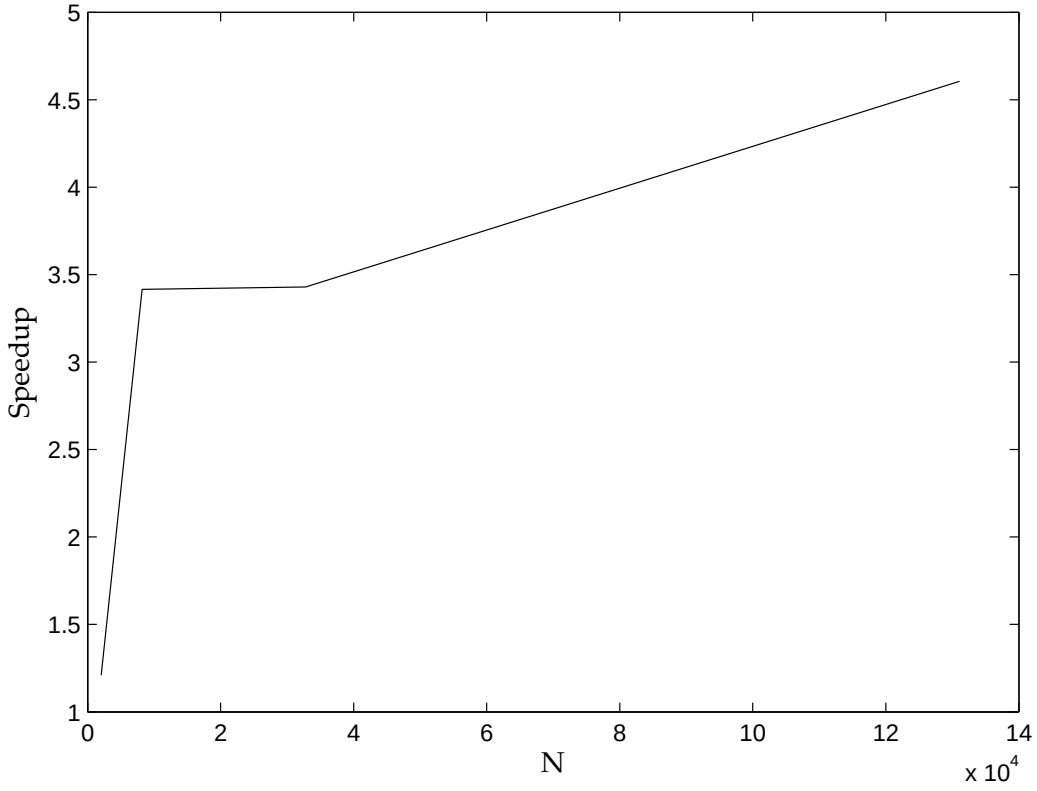


Figure 6·6: Speedup for solving first-kind Stokes equation on the surface of a sphere, varying N. 10^{-5} solver tolerance, $p = 16$.

more are 3.5× or greater, and as problem sizes grow, the speedup increased to greater than 4×. This additional speedup for large problems is due to the disparity in N_{CRIT} values needed for relaxed and fixed p cases, and the use of the sparse matrix near-field form from §4.3 – the higher N_{CRIT} for fixed p problems prohibits the use of the significantly faster sparse matrix approach due to memory consumption.

6.6 Conclusions

In this section of the thesis, we have extended the work from chapter 5 to the Stokes equations, a significantly more difficult problem. Our implementation has been shown to converge to the correct solution for a standard test case, and we have demonstrated

that as predicted, the combination of high-accuracy and large iteration counts for this type of problem has resulted in good speedups from a relaxed solver, reducing the time-to-solution by a factor of $3-4\times$. Further to these performance results, we have also shown the importance of setting a minimum value of p, $p_{\min}$, that we cannot relax below, in order to maintain accuracy. For all tests performed, $p_{\min}=5$ was good for minimizing iteration counts while maintaining accuracy. For general problems, the choice of $p_{\min}$ may be more difficult, requiring either a conservative choice to ensure accuracy, or multiple experiments with varying values of $p_{\min}$ so that any changes in the solution can be observed.

Chapter 7

BEM with inexact GMRES for Red Blood Cell problems

To show "real-world" performance, we choose an application closer to contemporary Stokes BEM research than flow past a sphere. The behavior of both flow past red blood cells (RBCs), in our case ethrocytes, and the effect of this flow on the cells themselves has prompted research based on a variety of methods, from traditional fluid solvers to boundary integral formulations like the one presented in this thesis. These studies are of particular use to the medical field — by better understanding the microcirculation of blood, topics such as anti-coagulation therapy and stroke research would greatly benefit (Rahimian et al., 2010). To summarize state-of-the-art in the boundary integral representation of RBCs, we briefly discuss 3 papers.

The first was an early attempt at a large-scale simulation (Zinchenko and Davis, 2003), using $\mathcal{O}(1000)$ panels per elliptical cell (Vessicle) and utilizing a periodic domain of $\mathcal{O}(100s)$ of cells in order to approximate blood cells in plasma using an emulsion model. Extending this idea, the winner of the prestigious Gordon Bell prize in 2011 (Rahimian et al., 2010) used large numbers of low-definition elliptical vessicles ($\mathcal{O}(100)$ panels per cell) with a coupled Stokes and elasticity formulation to simulate blood drops – the largest simulation was performed with 200 million cells, and 90 billion unknowns (velocity and forces). This corresponds to roughly 40 drops of blood. Finally, the simulation closest in capability to the work in this thesis involves 40 ethrocyte cells comprising of 7500 panels each (Liu, 2009). This experiment

was non time-dependant, and did not involve any cell deformation, making it directly comparable to our work.

These experiments are characterized by their size — the computational capacity required, the number of cells used and the amount of time taken for simulations. The application of a relaxation scheme would give the option of either reducing both the computational and time requirements, or performing experiments with more cells or the same amount, but better discretized cells. Both options would provide a significant gain in capability for any suitable code.

While research often includes deformation of the cells by using a linear elasticity BEM and unsteady flow to show the time-dependant evolution of cells in blood flow, we choose to demonstrate only the Stokes-flow part. The equations for linear elasticity can be treated in the same way as those for the Stokes equation — we integrate over tensors, the result is a vector quantity, and we can use an FMM to improve the scaling of N-body products by a decomposition into multiple harmonic FMMs. Thus, speedups from relaxation for the purely Stokes problem can be viewed as analogous to speedups that would be required for both linear elasticity and the combined problem. Unsteady flow would involve repeated BEM solves of comparable or equal difficulty, so we can view any speedup for a single solve (single time-step) as applicable for every solve at every time-step.

7.1 Geometry and Single Cell

While there are many ways of obtaining a surface mesh for the ethrocyte, we choose a parameterization method (Krüger, 2012) in order to re-use our existing geometry creation routines for a sphere. The method is based on applying a simple transform to every vertex, such that a vertex, $v = v(x, y, z),\ x, y, z \in [-1, 1]$ is transformed into $v' = v'(x', y', z(\rho'))$, where $x' = x \cdot r,\ y' = y \cdot r,\ \rho = \sqrt{x'^2 + y'^2}$ and

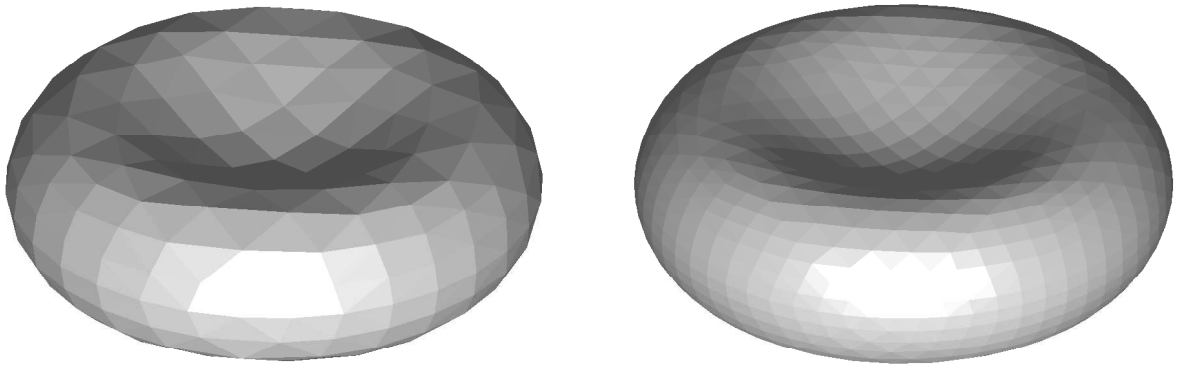
$$z(\rho) = \pm\frac{1}{2}\sqrt{1-\left(\frac{\rho}{r}\right)^2}\left(C_0 + C_2\left(\frac{\rho}{r}\right)^2 + C_4\left(\frac{\rho}{r}\right)^4\right). \tag{7.1}$$

In this form, the rotation axis of symmetry is along the z-axis, we choose the $\pm$ based on whether the original vertex has $\pm z$, and the constants are shown in table 7.1:

Constant	Value
r	$3.91\mu m$
C_0	$0.81\mu m$
C_2	$7.83\mu m$
C_4	$-4.39\mu m$

Table 7.1: Constants for equation (7.1)

This formulation provides a smooth, well-resolved surface mesh, with triangles of relatively uniform size and shape, shown in figures 7·1a, 7·1b.



(a) 512 panels (b) 2048 panels

Figure 7·1: Triangular discretizations of an ethrocyte, parameterized from a sphere.

When placed in a flow with $U_x = 1,\ \mu = 10^{-3}$, we obtain the traction force in the x-direction, shown in figure 7·2.

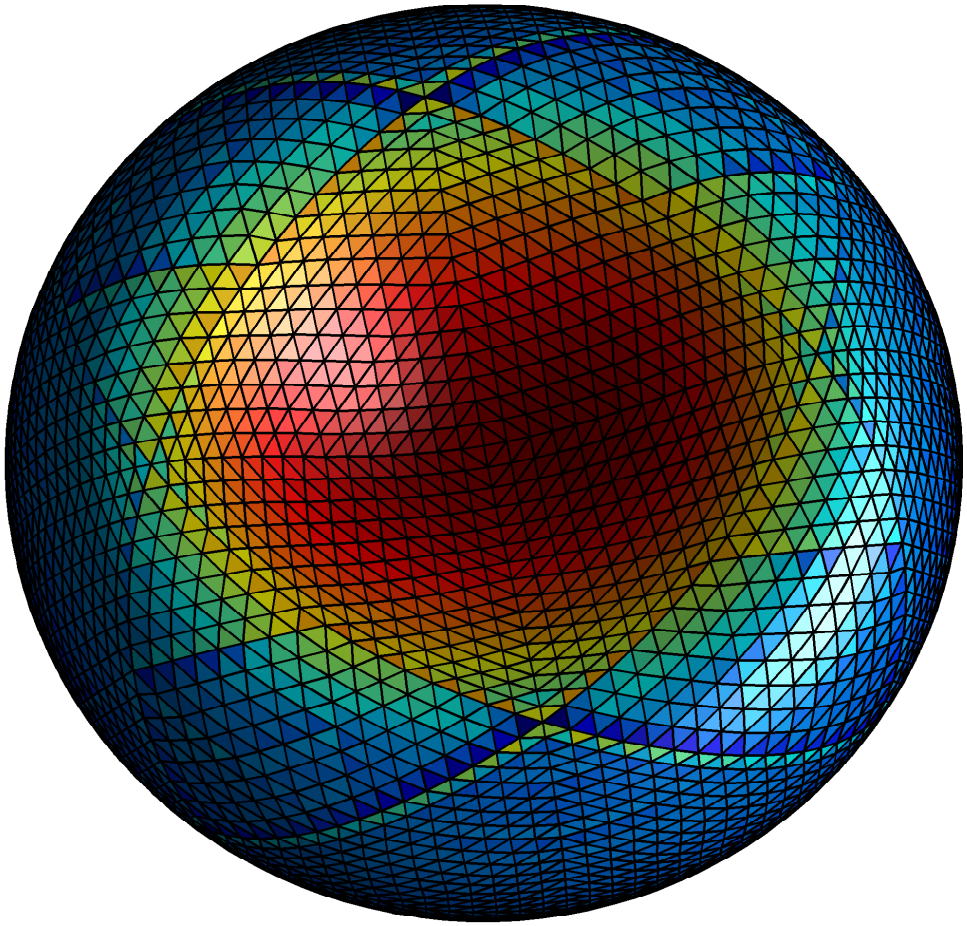


Figure 7·2: Traction force t_x exerted in the x-direction on an Ethrocyte with 8192 panels, $p = 16$, $p_{\min} = 5$ solved to 10^{-5} tolerance

While we do not have any kind of analytic expression for any of the flow quantities, we can still test the correctness of our method by checking if our calculated results converge as expected to some extrapolated value. As we tested the convergence of the drag force for convergence purposes with the sphere, it is a natural choice to use it for the ethrocyte as well. To obtain an extrapolated value for the drag, we use Richardson extrapolation (Roache, 1998), choosing the calculated drag force from 3 different meshes, with a constant refinement factor, C (in this case 4). Then, to

calculate the extrapolated force value, $\bar{f}$, we use the formula below, with f_1 being the coarsest mesh, and f_3 the finest.

$$\bar{f} = \frac{f_1 f_3 - f_2^2}{f - 1 - 2f_2 + f_3} \tag{7.2}$$

Another useful quantity we obtain from this procedure, is the *observed order of convergence*, a measure of the true convergence obtained with our method. Denoted by p, this is given by

$$p = \frac{\ln\left(\frac{f_2 - f_1}{f_3 - f_2}\right)}{\ln C}. \tag{7.3}$$

Using the mesh, f_x value pairs given in table 7.2, we calculate the converged drag force to be $f_x = -0.0834$, and the observed order of convergence to be 0.481. This shows both that our solution is converging spatially to a value, and that the rate of convergence is $\mathcal{O}(\sqrt{N})$, the same as observed for the sphere test case.

N	f_x
512	-0.059
2048	-0.071
8192	-0.077

Table 7.2: Values of mesh size and calculated drag force for convergence study

To further illustrate this important result, we can use the extrapolated f_x as an "exact value" in order to calculate errors at each of our mesh points. This error is plotted in figure 7·3, where the first 3 points were used to calculate the extrapolated value, and the final, largest point was used as a test.

This result shows that we are observing the same order or convergence for the ethrocyte case as for the simpler sphere — this gives us more confidence that our FMM-BEM solver is behaving as expected.

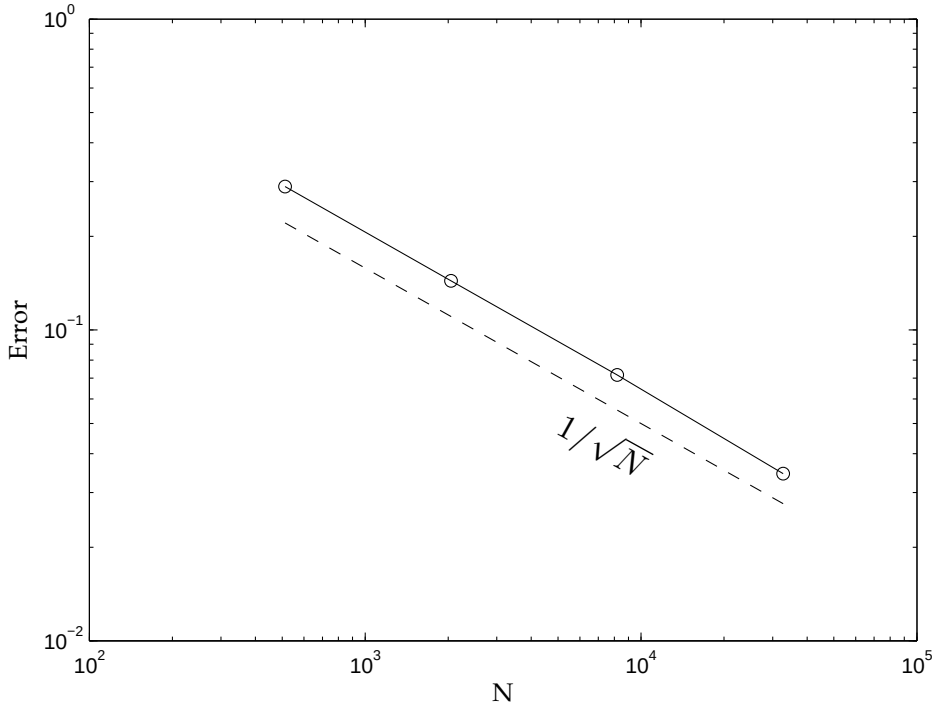


Figure 7·3: Observed convergence for Ethrocyte with respect to extrapolated value

7.2 Multiple Cells

While the flow around a single ethrocyte is more interesting than flow over a sphere, it is still not of much interest. For a more substantial problem, we look at multiple ethrocytes together, representing a single instant of blood flow in an artery / vein.

Due to the nature of the BEM formulation, the only difference between a single ethrocyte and multiple ones, is that the total number of panels is going to increase, and if we were to be using a dense matrix instead of the FMM, it would increase in size from $(3N) \times (3N)$ to $(3N_cN) \times (3N_cN)$, where N_c is the number of cells we choose to use. In the FMM case, we simply have more source and target panels, and due to the $\mathcal{O}(N)$ scaling, the time taken for each matrix-vector product should increase linearly.

In order to generate an interesting combination of cells, we repeatedly take a single ethrocyte, then apply a random rotation and shift in space, ensuring that cells do not

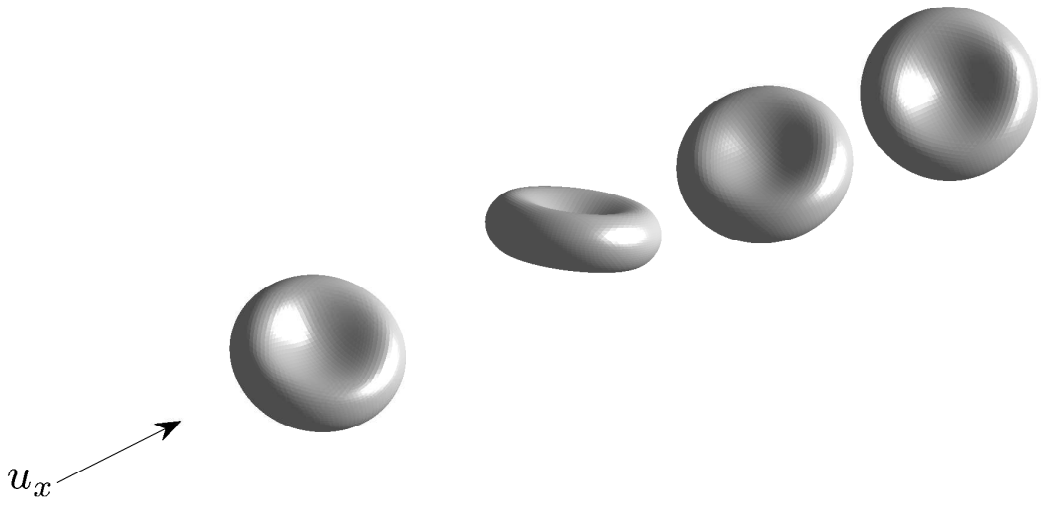


Figure 7·4: Example surfaces for 4 Ethrocytes in fluid

overlap. An example of this is shown in figure 7·4.

Given our ability to generate geometries with arbitrary numbers of cells, we can perform a number of interesting experiments. First, we study the properties of the resulting linear system (conditioning of the system), especially how the number of iterations changes as we increase N_c. Next, we examine the benefits of both preconditioners and relaxation (and combinations of both) on a non-trivial problem.

7.2.1 Convergence Properties

As we move from a single ethrocyte to many, we might suspect that the resulting linear system would become increasingly harder to solve. This can result from the fact that the system has increased in size by a factor of the number of cells, or from the natural assumption that as we add cells that are disconnected from each other, it becomes tougher to solve the system.

We would prefer to investigate this topic by calculating the condition number of actual problems, but for problem sizes that would be interesting (8192 panels / cell), the amount of memory required to calculate the condition would be very large.

For instance, the aforementioned 8192 panel case gives a dense matrix with 6×10^8 entries, taking 4.6GB in double precision. Raising the number of panels to 32768, we have 9.7×10^9 entries in our matrix, taking 73.7GB in double precision, a prohibitive amount.

While there has been some theoretical work on the condition number of matrices arising from 2D BEM problems, (Dijkstra and Mattheij, 2006; Dijkstra and Mattheij, 2007), and some numerical notes involving condition numbers for 3D MEMS problems (Stokes) (Frangi and di. Gioia, 2005), there are no analytical formulations to approximate the condition number for 3D Stokes, and the problems we are interested in are too large to produce a dense matrix representation and directly obtain the condition number. Due to these restrictions, we choose to approach this problem in terms of problem size and the number of iterations required to obtain a solution with a desired residual, using the iteration count as a proxy to the condition number.

First, as a baseline, we will look at the iterations required to solve the system for a single ethrocyte, whilst refining the surface mesh.

In figure 7·5 we can see that as we increase from 128 panels to 8192, the number of iterations needed increases from 19 to 37, while as we further refine the mesh up to $131,072$ panels we see the number of iterations flatten out. This implies that beyond a certain point, the condition number of our matrix stops increasing, although without being able to test a larger system, we cannot confirm this.

Now that we know how the convergence behaviour changes with respect to the number of panels per ethrocyte, we can look at how it changes with respect to multiple cells. We take advantage of the nature of our geometry creation (that is, the refinement factor is 4) to both test how the convergence changes as we increase the number of cells, as well as the effect of the number of panels per cell. In all cases $p = 16,\ p_{\min} = 5$ and the solver tolerance is set to 10^{-5}.

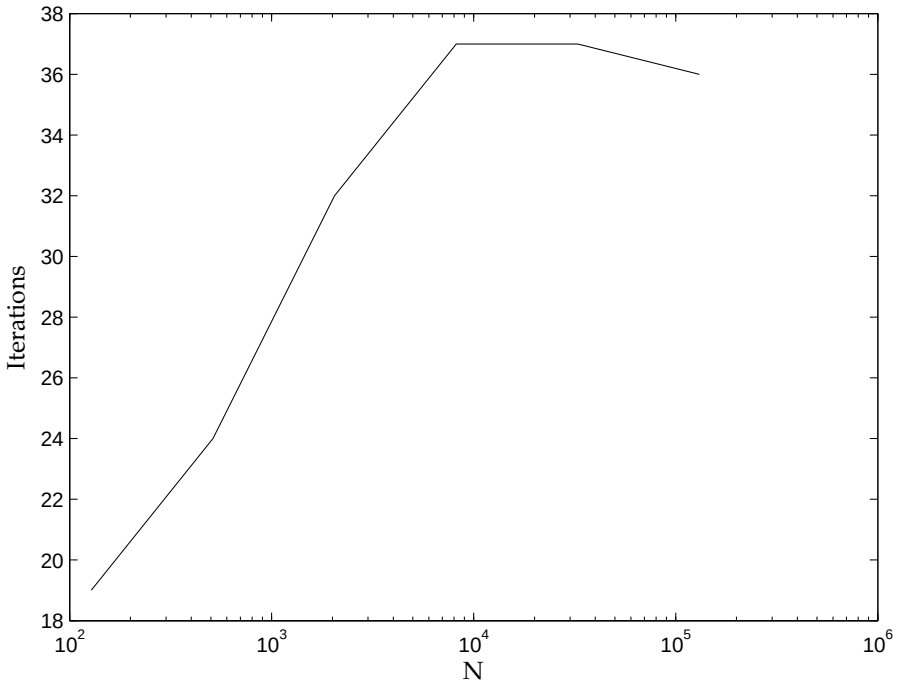


Figure 7·5: Iterations for Ethrocyte system to converge. $p = 16$, desired residual 10^{-5}

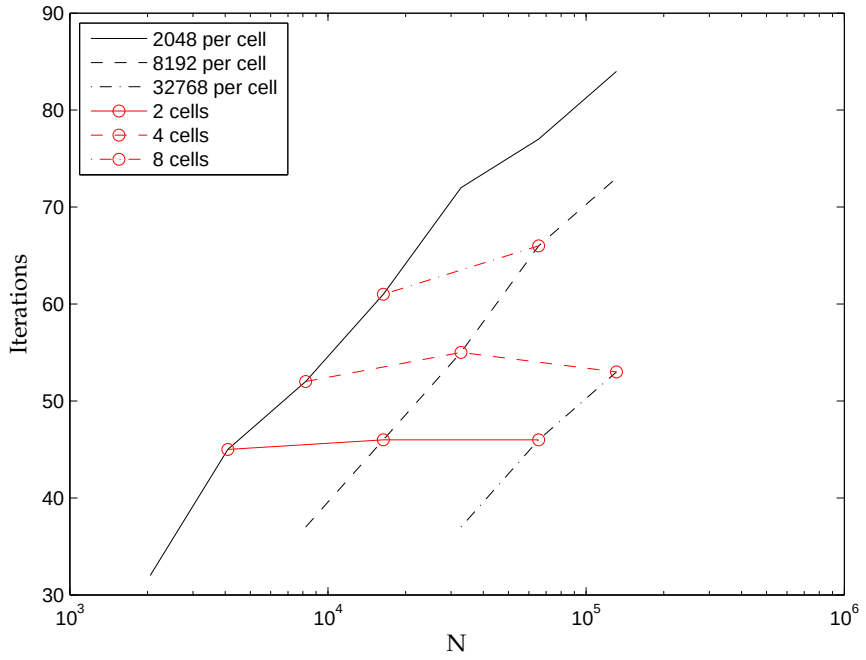


Figure 7·6: Iterations for system from multiple Ethrocyte to converge. $p = 16$, desired residual 10^{-5}

Figure 7·6 shows how the number of iterations needed to converge changes with respect to the number of cells used and with respect to the number of panels per cell. In all 3 cases, 2048, 8192 or 32768 panels per cell, we see a sharp increase in the number of iterations as the number of cells increases. Interestingly, if we look at the number of iterations required for a set number of cells, they are very similar for each case – it makes little difference how many panels comprise each cell, the convergence properties are dominated by the number of cells. For instance, the system for 2 cells takes 45, 46 and 46 iterations for 2048, 8192 and 32768 panels per cell. Similarly, for 4 cells, 52, 55 and 53 iterations were needed for the analogous cases.

This is encouraging for the use of relaxation, as state-of-the-art simulations often involve 40 or more ethrocytes (Liu, 2009; Rahimian et al., 2010), implying that we are likely to require a large number of iterations, potentially $\mathcal{O}(100)$, giving large potential gains in speed.

7.2.2 Relaxation

Now that we have established the convergence properties of systems involving ethrocytes, we start to look at the advantages of using a relaxation scheme. With the results shown in figures 7·5 and 7·6, we expect a large number of iterations to be needed in all cases. In addition to this, from figure 6·5 in our experiments on a spherical object, we expect to spend large numbers of iterations at a smaller value of p. Finally, figure 6·5, shows the decrease in time for the far-field calculation when we reach our minimum p.

Combining these 3 observations, we can expect savings from applying a relaxation scheme for a single ethrocyte, with the gains increasing as a function of the number of cells. In all comparisons, the best parameters will be used – for non-relaxed runs the near and far-field evaluations will be as balanced as possible, while for relaxed tests parameters will be chosen to give the shortest runtime. In every case we only

look at the time taken in the solution phase (within the GMRES solver) as the setup (including generating the right-hand-side of the linear system) is 1) amortized over the total run, and 2) always performed at high-precision, so the behavior is identical for relaxed and fixed-p cases. Small problems will not be tested, as the fastest solution will also be using a direct method, which inherently cannot use relaxation.

All experiments were performed with the settings described in table 7.3.

Variable	Setting
p_{initial}	16
p_{min}	5
solver tolerance	10^{-5}
Near-field	Sparse matrix
Threads	4
Solver	GMRES
Preconditioner	None

Table 7.3: Ethrocyte relaxation test settings

Looking at the results in 7.4, we see that we obtain a consistent $3-4\times$ speedup over non-relaxed results. The outlying result for 131072 panels is due to being unable to use the significantly faster sparse-matrix representation of the near field. The variation in the other speedup numbers is a natural result of the behaviour of the FMM when we choose different values of N_{CRIT} (and generate different octrees).

These results backup our original assumption that due to the combination of high numbers of iterations largely spent at low values of p, and the large savings from using that low p, we should see large time savings. Combining this observation with the known significant increase in iterations for the multiple ethrocyte case (7·6), we expect to see time savings of a similar, if not greater degree.

[1]Due to memory restrictions, the sparse-matrix representation of the near-field could not be used, resulting in a much slower P2P evaluation

N	# unknowns	Non-relaxed		Relaxed		Speedup
		N_{CRIT}	t_{solve}	N_{CRIT}	t_{solve}	
2048	6144	200	44.5	100	11.0	4.05
8192	24576	400	177	150	52.4	3.37
32768	98304	400	848	150	223	3.80
131072	393216	600	6386[1]	150	874	7.31[1]

Table 7.4: Single Ethrocyte relaxation test results, performed with initial $p = 16$, $p_{\min} = 5$, solved to 10^{-5} tolerance.

2048 panels / cell

N	# unknowns	N_c	Non-relaxed		Relaxed		Speedup
			N_{CRIT}	t_{solve}	N_{CRIT}	t_{solve}	
2048	6144	1	200	44.5	100	11.0	4.05
8192	24576	4	200	236	150	59.8	3.95
32768	98304	16	400	1261	150	331	3.81
131072	393216	64	300	9982[1]	100	1606	6.22[1]

Table 7.5: Multiple Ethrocyte relaxation test results 2048 panels / cell

8192 panels / cell

N	# unknowns	N_c	Non-relaxed		Relaxed		speedup
			N_{CRIT}	t_{solve}	N_{CRIT}	t_{solve}	
8192	24576	1	400	177	150	52.4	3.37
32768	98304	4	400	1375	150	315	4.37
131072	393216	16	300	15980[1]	100	1692	9.44[1]

Table 7.6: Multiple Ethrocyte relaxation test results for 8192 panels / cell

32768 panels / cell

N	# unknowns	N_c	Non-relaxed		Relaxed		speedup
			N_{CRIT}	t_{solve}	N_{CRIT}	t_{solve}	
32768	98304	1	400	848	150	223	3.80
131072	393216	4	300	9629[1]	100	1247	7.72[1]

Table 7.7: Multiple Ethrocyte relaxation test results for 32768 panels / cell

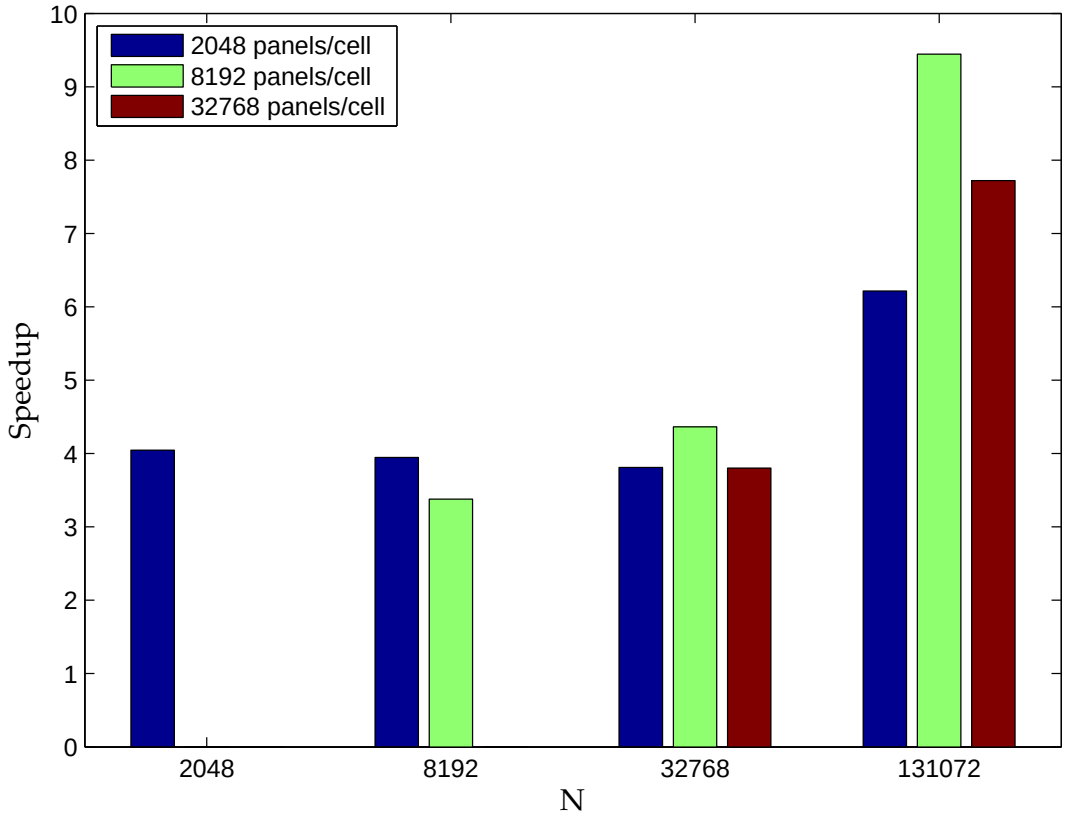


Figure 7·7: Speedups from multiple ethrocyte tests with varying panels / cell and number of cells. $p = 16$, desired residual 10^{-5}

Tables 7.5-7.7 present timing results for both relaxed and non-relaxed solvers for systems involving multiple ethrocytes. The results are also summarized in figure 7·7. The first result to note is that the smallest speedup obtained was 3.37×, for a single cell of 8192 panels. This shows that even for a relatively small test problem, the speedup due to relaxation is large. Second, for tests where both solvers could utilize the sparse near-field representation, the average speedup is 3.89×, a major gain. Finally, we see again that for large problems the memory efficiency of the lower N_{CRIT} for relaxed solves is vital, as for these cases the speedup jumps to between $6 - 9\times$.

All of these multiple ethrocyte results confirm our expectations from the single

cell tests, that we can obtain significant and worthwhile speedups simply from the use of a relaxation scheme and modifying N_{CRIT} to maintain a reasonable balance between P2P and M2L over the course of an iterative solve.

7.2.3 Preconditioners

Even with the results demonstrated in §7.2.2, the traditional manner of reducing solution time is to use a preconditioner to lower the iteration count, so we must consider the following questions:

1. Is our relaxation scheme faster than a traditional preconditioned problem solved with fixed p?

2. Does our relaxation scheme benefit additionally when used in a preconditioned system?

To find answers to these 2 queries, we perform some experiments on multiple ethrocytes, using the block-diagonal and local preconditioners described in §3.3. As an initial test, we perform a small calculation, with 4 cells, each of 2048 panels. This results in a total of 8192 panels, and a linear system with 24576 unknowns. This is sizable enough to draw some initial conclusions, yet small enough that it can be performed in $1-2$ minutes in the relaxed case (and < 5 minutes with the non-relaxed solver).

Table 7.8 presents iteration counts and total solution times for both relaxed and non-relaxed solvers for preconditioned and non-preconditioned solvers. In all experiments $N_{\text{CRIT}} = 150$ for relaxation, and $N_{\text{CRIT}} = 400$ for non-relaxed cases.

From these results, it is immediately obvious that while the preconditioners increase the rate of convergence, by up to 50% in the case of the local preconditioner, the time taken for the solve does not necessarily reflect this decrease in iterations. The block-diagonal preconditioner is the most successful of the preconditioned cases,

Preconditioner	Relaxed			Fixed p		
	iterations	t_{solve}	speedup	iterations	t_{solve}	speedup
Unpreconditioned	52	59.9	-/-	52	238.0	-/-
Block-Diagonal	45	60.8	1.18/0.98	45	212.7	1.18/1.13
Local	37	115.8	1.54/0.52	37	241.6	1.54/0.99

Table 7.8: Preconditioning tests for relaxed and non-relaxed solvers – speedup is in terms of iterations (first value) and t_{solve}

with a 1.13× speedup in the non-relaxed case (compared to a 1.18× reduction in iterations), however when using relaxation it was slower than not using a preconditioner, albeit to an insignificant degree.

These initial results suggest that while the local preconditioner is effective at increasing the rate of convergence, at this point it cannot be used, as all potential savings from the reduced number of iterations are cancelled out by the cost of performing the inner GMRES solve. However, the block-diagonal preconditioner shows promise, working well for the fixed p case, while not providing any benefit in total solve time for the relaxed example.

It must be mentioned here that these preconditioners are working in the *algorithmic* sense, the *implementation* is highly important, and could radically change the current results. Taking the block-diagonal case as an example, we can immediately see the following potential speed-ups:

1. Explicitly create the inverse of A_{block} as described in §3.3, changing the application of the preconditioner from a GMRES solve (involving 2 matrix-vector products, and multiple other operations) to a single matrix-vector product. Given the nature of A_{block}, A_{block}^{-1} would require exactly the same amount of storage, while giving a clear speed advantage, at the expense of setup time, due

to the cost of inverting many dense matrices of size $N_{\text{CRIT}} \times N_{\text{CRIT}}$.

2. Faster sparse matrix operations, notable the matrix-vector product. This could be done in any number of ways, notable examples would be using vectorization (such as SSE instructions), or an additional accelerator, for instance a GPU. These options would reduce the cost of applying the preconditioner, while maintaining the reduced iteration count, giving a larger overall speedup.

The implementation of one or both of these suggestions might provide a more useful scheme, but at this point, the fastest time-to solution is for an un-preconditioned solve using a relaxation scheme.

7.3 Conclusions

We have shown compelling results for relaxation schemes in the solution of systems involving Stokes flow around one or more Ethrocytes. These speedups are consistently in the $3-4\times$ range seen in chapter 6, and are consistent between both the number of ethrocytes used and the level of discretization used per ethrocyte. Further, we have demonstrated that our relaxed solver outperforms a traditionally preconditioned linear system.

We can also compare this work to the most similar example in literature (Liu, 2009). As previously mentioned, 40 non-deformable cells of 7500 panels each were used, giving 300k total panels and 900k degrees of freedom, roughly $2\times$ the size of problem we can currently perform. Most simulation variables were given, with the notable exception of N_{CRIT}. The exact formulation of Stokes BEM used is not given. This system was solved in 730 minutes, taking 49 iterations at $p = 15$, solving to a tolerance of 10^{-4}. We can replicate the majority of these options, although for 40 cells we must use 2048 panels per cell, giving 81920 total panels. This experiment took 33 iterations to converge and 1793s for the fixed-p case, and 639s for the relaxed

solver, giving a speedup of 2.80×. While this speedup is lower than those reported for our experiments due to the change in solver tolerance and thus fewer iterations, it would still give an approximated run time of 260 minutes.

The reduction in solution time from an algorithmic change presents an increase in capability for FMM-BEM methods that should be applicable to a majority of existing codes. The ability to both perform experiments in 1/3 to 1/4 the time allows more test cases to be run, or, potentially, a higher resolution case can be run in the same timeframe. This presents advantages for all use cases – if time constrained, experiments can be performed much quicker, while if accuracy constrained, higher precision experiments can be performed without the previous cost in computational time.

Chapter 8

Conclusions

8.1 Achievements and Findings

The theory of inexact-Krylov iterations has been applied for the first time for FMM-BEM problems and verified against standard GMRES solvers. It has been tested for both Laplace and Stokes problems and the convergence of the associated linear solves, along with the resulting accuracy of the result has been found to be comparable in all cases. Further, this new method has been applied to the field of the flow of red blood cells (ethrocytes), and has demonstrated significant *algorithmic* performance gains in this engineering example. More detailed conclusions can be summarized as follows.

For the simplest test problem explored, experiments using the Laplace equation showed the validity of using a relaxed GMRES solver for FMM-BEM. In all cases, the relaxed solver provided reduced time-to-solution, while maintaining comparable iteration counts, convergence rates and final residuals. In the course of a parameter study involving these equations, we determined the following general rules for the use of a relaxed method:

- Higher required precisions will result in more significant reductions to time-to-solution, due to the strong scaling with respect to series truncation value, p in the FMM.
- Problems that require large numbers of iterations in order to reach a satisfactory solution tolerance will benefit more from relaxed schemes – the nature of GMRES

results in a large number of iterations requiring comparatively low accuracy from the matrix-vector product; iterations that benefit greatly from relaxation's reduced accuracy requirement.

- The traditional thinking within the FMM community that the computation time for near and far-fields should be balanced, requires modifications for use within a relaxed solver. As the desired tolerance, and thus required p decreases, the cost of the far-field reduces significantly, while the near-field cost stays constant. Thus, for minimized time-to-solution, the best configuration is to shrink the near-field to the smallest possible whilst retaining accuracy. The higher cost of initial iterations compared to balanced evaluations is made up for by the larger number of low-cost iterations during the remainder of the solve.

We next used a more difficult problem, creeping or Stokes flow, to test our conclusions from the Laplace experiments, and to show the applicability of the inexact GMRES to a tougher, more relevant problem. In all cases, once again, the relaxed solver gave comparable iteration counts and final residuals, while providing significant speedups, in the order of $3 - 4\times$, for time-to-solution. These experiments also contributed the idea that a *minimum* value of p may need to be set for tough problems, in order to maintain overall accuracy.

Finally, the insights provided from test problems were applied to a more significant, real-world application, the flow of red blood cells. These experiments demonstrated that for a relevant simulation, the relaxed GMRES solver can provide significant reductions in solution time, often by a factor of $4\times$, resulting in huge time savings. As a corollary to the reduced time, the relaxed solver also provides a more memory-efficient method, due to the smaller near-field, and thus reduced memory footprint of the near-field sparse matrix.

In summary, the following discoveries were made in this work:

1. The theory behind inexact Krylov solvers is fully applicable to FMM-BEM applications, and can provide significant benefits in terms of time-to-solution, while requiring little effort to add to an existing FMM-BEM code.

2. Based on the accuracy required, and number of iterations required to solve the linear system, we can predict whether a problem will benefit or not from relaxed solvers.

3. Relaxation can be applied successfully to more difficult systems, such as those from the Stokes equations. After these experiments we can say with some confidence that relaxation is likely to work with any equation, as long as accuracy and iteration count requirements are met.

4. There are significant *algorithmic* performance benefits from using a relaxed solver. Speedups obtained from the current implementation for Stokes problems were consistently in the $3-4\times$ range, and due to the algorithmic nature of the improvement, should be comparable for other implementations.

8.2 Further Work

8.2.1 Distributed-memory Parallelization

We already have a basic shared-memory parallel evaluator for the FMM-BEM, however for large problems we run in problems with memory consumption, especially when trying to use the sparse-matrix form of the near-field shown in §4.3. Moving to a full parallel code would allow us to compute much larger problems due to both the addition of extra computing power, and access to more memory.

8.2.2 Hypersingular / Dual Formulation for Stokes

While we have used a standard, or "conventional" formulation for our Stokes BEM, it is also possible to reformulate the problem in terms of a Hypersingular BEM, so-named for the existence of a new hypersingular operator that appears. This formulation can have problems with the uniqueness of solutions, so a combination of both conventional and hypersingular formulations was proposed (Liu, 2009) in order to maintain the better linear system characteristics of the hypersingular form, while retaining a unique solution. This combined approach would change the convergence properties of the BEM by reducing the number of iterations, while increasing the amount of work per iteration, and thus potential savings from relaxation, due to the additional operators required.

8.2.3 Higher-Order Elements

While this thesis concentrates on BEM with flat, constant-value triangles, there exist other, higher-order elements. Common examples of these would be flat panels with either linear or quadratic distributions of value across them, or even curved panels. These additions would provide better accuracy with fewer panels for some classes of problems, at the cost of more computational work for the P2P and P2M operators, changing the balance of near and far-field computation for relaxed problems.

8.2.4 Linear-Elasticity Equations

A natural extension of the work on the Stokes equations would be extending to the linear elasticity equations. There already exist formulations of the FMM based around decomposition into harmonic FMMs (Fu et al., 1998), similar to those for Stokes. Further, the analytical integration routines used in §3.1.3 were originally for linear elasticity, and were modified for our use for Stokes problems. Not only would this be a relatively simple addition to the work, but it would open up whole new application

areas, both stand-alone and in conjunction with the red-blood cell work in chapter 7. In particular, the application of ethrocytes moving and deforming within blood vessels would be enabled with this set of additional FMM kernels.

8.2.5 FMM Algorithmic improvements

As stated in §5.2, there exist several algorithmic improvements for the Laplace FMM used for all problems in this thesis. These improvements lower the complexity of translation operators from $\mathcal{O}(p^4)$ to $\mathcal{O}(p^3)$, making the FMM more efficient, especially for high accuracy (high p) cases. While we would still expect relaxation schemes to provide a significant benefit to solution times, the balance of near and far-field computations would be likely to change.

Appendix A

FMM_plan

A.1 Description

`FMM_plan` is an open-source software library for fast multipole methods, implemented in C++. It provides a simple user interface and modular design for easy development.

A.2 Software Components

`FMM_plan` is comprised of 5 major components:

1. `FMM_plan` – User interface for the library, hides all internal details from users.
2. `Executor` – Dictates runtime behaviour of the FMM: contains source and target trees, evaluators and storage for bodies and series expansions. All access to underlying data handled through here (also known as a `Context` within evaluators).
3. `Tree` – Hierarchical decomposition of the space. Provides access to boxes and interactions between them (parents, children).
4. `Evaluator` – Traverses the tree and calls kernel operators
5. `Kernel` – Contains all operators (P2P, M2L etc.) – Implements specific equations, such as Laplace, Stokes, Yukawa etc.

We will now go into more detail about each of these components, and useful terms.

A.2.1 Important terms

This section will briefly introduce important terms that will be used throughout the following descriptions.

- *Point* (`point_type`) – A simple point in space. Used, for instance, within the `tree`, where the full definition of the body is unnecessary. A simple example of this would be: `typedef Vec<3,double> point_type`.

- *Body* (`source_type, target_type`) – An individual element that will be evaluated. Examples include points in 3D space (for astrophysics), point vortices and panels (for BEM calculations). Examples include:

  `typedef TriangularPanel source_type //BEM` and

  `typedef point_type source_type //FMM`

- *Box* (`box_type`) – A physical subdivision of space, containing some number of child boxes or points.

- *Multipole Expansion* (`multipole_type`) – Singular expansion used in both treecode and FMM to approximate the influence of far-away boxes. For instance, for spherical harmonic expansions:

  `typedef std::vector<std::complex<double>> multipole_type`

- *Local Expansion* (`multipole_type`) – Regular expansion used in FMM to approximate the influence of far-away boxes. Translated and converted from a multipole expansion. For spherical harmonic expansions:

  `typedef std::vector<std::complex<double>> local_type`.

A.2.2 Plan

As the main forward-facing part of the library, the plan is designed to be very simple with all details abstracted away (and chosen through an options object). A single line

is required to create the entire FMM framework, and another line to get the result, as shown in A·1.

```
// define a kernel_type
typedef SphericalLaplace kernel_type;
// construct a kernel, in this case with p=5
kernel_type K(5);
// construct a plan with sources == targets
FMM_plan<kernel_type> plan(K,sources,options);
// construct a plan with sources != targets
FMM_plan<kernel_type> plan(K,sources,targets,options);
// execute a plan with given charges
auto results = plan.execute(charges);
```

Figure A·1: *FMM_plan* interface

A.2.3 Executor / Context

This object contains the abstraction of all details within the FMM. It holds the tree, all evaluators, and provides the interface between the evaluators and all underlying data (sorted sources, charges and targets, multipole and local expansions etc.). It does this by providing the public interface shown in A·2.

A.2.4 Tree

Defines the concept of a `Box`, and decomposes the space into these boxes based on a maximum number of bodies per box, N_{CRIT}. Provides an interface to those boxes, A·3. Default is an Octree ($3D$), but anything that provides the correct interface could be used.

A.2.5 Evaluator

Controls traversal of the tree, and calls kernel operators. Can be constructed with arbitrary state, and has a single called method, A·4. Obtains data for operator calls through the Context's public interface. The evaluator can be customized to provide

```
// MAC satisfied between two boxes?
bool accept_multipole(const box_type& source, const box_type& target);
// get multipole / local expansions for a given box
multipole_type& multipole_expansion(const box_type& box);
local_type& local_expansion(const box_type& box);
// box center
point_type center(const box_type& box);
// iterators to sources
body_source_iterator source_begin(const box_type& box);
body_source_iterator source_end(const box_type& box);
// iterators to charges
body_charge_iterator charge_begin(const box_type& box);
body_charge_iterator charge_end(const box_type& box);
// iterators to targets
body_target_iterator target_begin(const box_type& box);
body_target_iterator target_end(const box_type& box);
// iterators to results
body_result_iterator result_begin(const box_type& box);
body_result_iterator result_end(const box_type& box);
```

Figure A·2: Methods exposed by executor / context

any kind of evaluation, including treecode / FMM, only considering the near-field (for preconditioners, kernels with negligible far-fields), or modifying the manner in which domains are evaluated – the sparse-matrix form of the near-field from §4.3 is implemented using a customized evaluator.

A.2.6 Kernel

This contains all the operator methods required for a treecode FMM: P2P, M2L etc. Can keep arbitrary state (for instance precomputed translation matrices), and can offer different granularities of computation for different architectures. The following types must be defined:

- `point_type` – A point in physical space (i.e. (x, y, z)).
- `kernel_value_type` – result of $\mathbb{K}(x_i, x_j)$.
- `source_type` – Source body, i.e. point for FMM, panel for BEM. Must be convertible to `point_type` by: `static_cast<point_type>(source_type)`

- `charge_type` – Charge associated with each source
- `target_type` – Analogous to `source_type` with the same casting restriction.
- `result_type` – Product of `charge_type * kernel_value_type`.
- `multipole_type` – Multipole expansion type
- `local_type` – Local expansion type

To perform a calculation, the following operators must be defined:

- *Treecode* – P2P, P2M, M2M, M2P
- *FMM* – P2P, P2M, M2M, M2L, L2L, L2P

and must have signatures defined in A·5

The kernels currently available are:

Equation	$\mathbb{K}$	Expansion	Treecode/FMM	Param.	Class
Laplace	$\frac{1}{r}$	Spherical Cartesian	Both	p	`LaplaceSpherical` `LaplaceCartesian`
Yukawa	$\frac{e^{-kr}}{r}$	Spherical Cartesian	Treecode Both	$p,\ k$	`YukawaSpherical` `YukawaCartesian`
Helmholtz	$\frac{e^{-ikr}}{r}$	Spherical	FMM	$p,\ k,\ \epsilon$	`HelmholtzSpherical`
Stokes	(6.5)	Spherical	Both	p	`StokesSpherical`

Table A.1: Available kernels

A.3 Example

Now all of the software components have been introduced, it is worthwhile to demonstrate how a traditional treecode / FMM maps to our library. The main components are:

1. Construct the tree
2. Create multipole expansions at leaf boxes
3. Translate multipole expansions up the tree
4. Long-range interactions (M2P for treecode, M2L for FMM)
5. Translate local expansions down the tree and evaluate them (FMM only)
6. Perform local evaluations (P2P)

At a high level, we can think of this as two main "phases":

1. Create a tree,
2. Traverse tree, calling operators.

In our library, first a `FMM_Plan` object is created – this constructs the `Tree` internally, as well as setting up the `Executor/Context` and the desired combination of `Evaluator`s. When the plan's `execute(charges)` method is called, the `Evaluator` traverses the `Tree` and calls the appropriate combination of `Kernel`s using appropriate data requested from the `Context`. In it worth noting that as long as each component presents the appropriate interface, the underlying implementation is unimportant.

```
// Box and Box methods
//
// get box data
unsigned Box::index() const;
unsigned Box::level() const;
point_type Box::extents() const;
unsigned Box::num_children() const;
bool Box::is_leaf() const;
point_type Box::center() const;
Box Box::parent() const;
body_iterator Box::body_begin() const;
body_iterator Box::body_end() const;
box_iterator Box::child_begin() const;
box_iterator Box::child_end() const;
// Equality operators (based on box index)
bool Box::operator==(const Box& b) const;
bool Box::operator<(const Box& b) const;

// Construct a tree
template <typename SourceIterator>
void construct_tree(SourceIterator s_begin, SourceIterator s_end,
                    unsigned NCRIT);
// bounding box this tree encompasses
bounding_box<point_type> bounding_box() const;
// number of bodies, boxes, levels contained in the tree
unsigned size() const;
unsigned boxes() const;
unsigned levels() const;
// check if a box is contained within this tree
bool contains(const box_type& box) const;
// root box
box_type& root() const;
// iterators to values stored in the tree (entire tree)
body_iterator body_begin() const;
body_iterator body_end() const;
box_iterator box_begin() const;
box_iterator box_end() const;
// iterators to boxes on a given level
box_iterator box_begin(unsigned level) const;
box_iterator box_end(unsigned level) const;
// get a box from its index
box_type& box(int idx) const;
```

Figure A·3: Tree methods

```
// execute the evaluator with a given context
void execute(context_type& context) const;
```

Figure A·4: Evaluator execution

```
// P2P - One of (non-exhaustive):
kernel_value_type operator()(const target_type& t,
                             const source_type& s) const;
template <typename SourceIter, typename ChargeIter,
          typename TargetIter, typename ResultIter>
void P2P(SourceIter s_first, SourceIter s_last, ChargeIter c_first,
         TargetIter t_first, TargetIter t_last,
         ResultIter r_first) const;

// P2M - One of:
void P2M(const source_type& source, const charge_type& charge,
         const point_type& center, multipole_type& M) const;
template <typename SourceIter, typename ChargeIter>
void P2M(SourceIter s_first, SourceIter s_last, ChargeIter c_first,
         const point_type& center, multipole_type& M) const;

// M2M
void M2M(const multipole_type& source, multipole_type& target,
         const point_type& translation) const;

// M2P - one of:
void M2P(const multipole_type& M, const point_type& center,
         const target_type& target, result_type& result) const;
template <typename TargetIter, typename ResultIter>
void M2P(const multipole_type& M, const point_type& center,
         TargetIter t_first, TargetIter t_last,
         ResultIter r_first) const;

// M2L
void M2L(const multipole_type& source, local_type& target,
         const point_type& translation) const;

// L2L
void L2L(const local_type& source, local_type& target,
         const point_type& translation) const;

// L2P - one of:
void L2P(const local_type& source, const point_type& center,
         const target_type& target, result_type& result) const;
template <typename TargetIter, typename ResultIter>
void L2P(const local_type& source, const point_type& center,
         TargetIter t_first, TargetIter t_last,
         ResultIter r_first) const;
```

Figure A·5: Kernel operators

References

Abramowitz, M. and Stegun, I., editors (1964). *Handbook of Mathematical Functions*. National Bureau of Standards, second edition.

Barnes, J. and Hut, P. (1986). A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449.

Boschitsch, A. H., Fenley, M. O., and Olson, W. K. (1999). A fast adaptive multipole algorithm for calculating screened Coulomb (Yukawa) interactions. *Journal of Computational Physics*, 151:212–241.

Bouras, A. and Frayssé, V. (2005). Inexact matrix-vector products in krylov methods for solving linear systems: A relaxation strategy. *SIAM Journal on Matrix Analysis and Applications*, 26(3):660–678.

Brebbia, C. and Dominguez, I. (1992). *Boundary Elements An Introductory Course*. WIT Press, 2nd edition.

Chaillat, S., Semblat, J., and Bonnet, M. (2011). A preconditioned 3-d multi-region fast multipole solver for seismic wave propagation in complex geometries. *Communications in Computational Physics*, 11(2):594–609.

Dijkstra, W. and Mattheij, R. M. M. (2006). The condition number of the BEM-matrix arising from Laplace's equation. Technical report, OAI Repository of the Technische Universiteit Eindhoven (TU/e)

Dijkstra, W. and Mattheij, R. M. M. (2007). Condition number of the BEM matrix arising from the stokes equations in 2D. Technical report, Eindhoven University of Technology.

Dongarra, J. and Sullivan, F. (2000). The top 10 algorithms of the twentieth century. *Computing in Science and Engineering*, 2(1):22–23.

Fachinotti, V. D., Cardona, A., D'Elia, J., and Paquay, S. (2007). BEM for the analysis of fluid flow around MEMS. *Mecánica Computacional*, XXIV: 1104–19

Fata, S. N. (2009). Explicit expressions for 3D boundary integrals in potential theory. *International Journal for Numerical Methods in Engineering*, 78:32–47.

Fata, S. N. (2011). Explicit expressions for three-dimensional boundary integrals in linear elasticity. *Journal of Computational and Applied Mathematics*, 235(15):4480–4495

Frangi, A. and di. Gioia, A. (2005). Multipole BEM for the evaluation of damping forces on MEMS. *Computational Mechanics*, 37(1): 24–31

Fu, Y., Klimkowski, K. J., Rodin, G. J., Berger, E., Browne, J. C., Singer, J. K., Van De Geijn, R. A., and Vemangati, K. S. (1998). A fast solution method for three-dimensional many-particle problems of linear elasticity. *International Journal for Numerical Methods in Engineering*, 42(7): 1215–1229.

Gomez, J. and Powert, H. (1997). A multipole direct and indirect bem for 2d cavity flow at low reynolds number. *Engineering Analysis with Boundary Elements*, 19(1):17–31.

Greengard, L. (1987). *The rapid evaluation of potential fields in particle systems.* The MIT Press.

Greengard, L., Huang, J., Rokhlin, V., and Wandzura, S. (1998). Accelerating fast multipole methods for the Helmholtz equation at low frequencies. *IEEE Computational Science and Engineering*, 5(3):32–38.

Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348.

Greengard, L. and Rokhlin, V. (1997). A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6.1 (1997): 229–269.

Huang, J., Jia, J., and Zhang, B. (2009). FMM-Yukawa: An adaptive fast multipole method for screened Coulomb interactions. *Computational Physics Communications*, 180(11):2331–2338.

Klaseboer, E., Manica, R., Chan, D. Y. C., and Khoo, B. C. (2011). BEM simulations of potential flow with viscous effects as applied to a rising bubble. *Engineering Analysis with Boundary Elements*, 35:480–494.

Krüger, T. (2012). *Computer Simulation Study of Collective Phenomena in Dense Suspensions of Red Blood Cells under Shear.* Springer.

Liu, Y. (2009). *Fast multipole boundary element method: Theory and applications in engineering.* Cambridge University Press.

Majchrzak, E. and Freus, K. (2003). Boundary element method in the inverse problems of steady heat transfer. Technical report, Scientific Reseach of the Institue of Mathematics and Computer Science.

Nabors, K., Korsmeyer, F. T., Leighton, F. T., and White, J. (1994). 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.

Panti, E. (2008). Boundary element method for heat transfer in a buried pipe. Master's thesis, University of Reading.

Plimpton, S. (1995). Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19.

Pozrikidis, C. (1992). *Boundary integral and singularity methods for linearized viscous flow*. Cambridge University Press.

Rahimian, A., Lashuk, I., Veerapaneni, S., Chandramowlishwaran, A., Malhotra, D., Moon, L., Sampath, R., Shringarpure, A., Vetter, J., Vuduc, R., Zorin, D., and Biros, G. (2010). Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11. IEEE Computer Society.

Roache, P. J. (1998). *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers.

Rodrigues, O. (1815). *De l'attraction des sphéroïdes*. PhD thesis.

Saad, Y. (1993). A flexible inner-outer preconditioned gmres algorithm. *SIAM Journal of Scientific Computing*, 14(2):461–469.

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

Sidje, R. and Winkles, N. (2011). Evaluation of the performance of inexact GMRES. *Journal of Computational and Applied Mathematics*, 235(8): 1956–1975

Simoncini, V. and Szyld, D. (2003). Theory of inexact krylov subspace methods and applications to scientific computing. *SIAM Journal on Scientific Computing*, 25(2):454–477.

Tornberg, A.-K. and Greengard, L. (2008). A fast multipole method for the three-dimensional stokes equations. *Journal of Computational Physics*, 227:1613–1619.

Van den Eshof, J. and Sleijpen, G. (2004). Inexact krylov subspace methods for linear systems. *SIAM Journal on Matrix Analysis Applications*, 26(1):125–153.

Yokota, R., Bardhan, J. P., Knepley, M. G., Barba, L. A., and Hamada, T. (2010). Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. *Compututational Physics Communications*, 182.6 (2011): 1272–1283

Yokota, R. and Barba, L. A. (2013). FMM-based vortex method for simulation of isotropic turbulence on GPUs, compared with a spectral method. *Computers & Fluids*, 80:17–27.

Zinchenko, A. and Davis, R. (2003). Large-scale simulations of concentrated emulsion flows. *Philosophical Transactions, Seria A, Mathematical, Physical and Engineering Sciences*, 361(1806): 813–845

CURRICULUM VITAE

Simon Layton

Research Interests

Fast multipole methods (FMM) and fast multipole accelerated Boundary element methods. High- performance computing using heterogenous architectures especially NVIDIAs CUDA environment. Classical algebraic multigrid for engineering applications, including computational fluid dynamics, most notably immersed boundary techniques. Also interested in sparse linear algebra, notably sparse matrix-matrix products and iterative linear solvers. Further experience in fast multipole and boundary element methods.

Internships

May-August 2012 – DevTech Compute intern, NVIDIA Worked on integrating the internal NVAMG library into a large open source scientific code. Work included implementing new features to the library using CUDA and optimizing these codes. Also worked on the interception and analysis of BLAS calls in a major structural analysis code, including work on moving suitable calls to cuBLAS and analyzing and profiling full simulation runs on sample industrial problems.

June-September 2011 – Emerging Applications intern, NVIDIA Worked with Jonathan Cohen on finite-volume methods (1 mo.) and algebraic multigrid (AMG) in CUDA (remaining time) Work on AMG included a complete port of the previous CPU only code from within the group to CUDA, testing and verification against the Hypre open-source package.

Education

PhD, Mechanical Engineering, Boston University, Expected September 2013 Thesis: *Fast multipole boundary element methods with inexact Krylov iterations and relaxation strategies* – Use of fast multipole methods as an inexact matrix-vector product for boundary element methods when solved using Krylov iterative methods. The theory behind inexact matrix-vector products is applied to adaptively control the error produced by the fast multipole method in order to minimize the time to solution for boundary element formulations of example engineering problems involving the Laplace and Stokes equations.

MS, Mechanical Engineering, Boston University, Awarded January 2011
Concentrations: Computational Fluid Dynamics, Fast Evaluation Algorithms, Heterogenous Computing

BSc, Mathematics and Computer Science, University of Bristol, 2008
Concentrations: Applied Mathematics, Algorithms, Numerical Methods

Publications

- "How to obtain efficient GPU kernels: an illustration using FMM & FGT algorithms", F. A. Cruz, S. K. Layton, L. A. Barba, *Comput. Phys. Commun.*, Volume 182, Issue 10, p.2084-2098 (2011)

Conference Contributions and Talks

- "Classical algebraic multigrid for CFD with CUDA", GTC 2012, San Jose, CA
- "Classical algebraic multigrid for engineering applications", CUDA Fast Forward, NVIDIA Booth, SC 11, Seattle, WA, 14th Nov. 2011
- "Classical algebraic multigrid using CUDA", GPU@BU Workshop, Nov. 8th 2011, Boston University
- "cuIBM - a GPU-accelerated immersed boundary method", S. K. Layton, A. Krishnan, L. A. Barba, International Conference in Parallel CFD, Barcelona, Spain, 16-20 May 2011.
- "The parallel Fast Gauss Transform in an heterogenous computing environment", S. K. Layton, F. A. Cruz, L. A. Barba, US National Congress in Computational Mechanics, USNCCM'09; Columbus, OH, July 2009.

Workshops

Pan-American Advanced Studies Institute (PASI) - Universidad Santa Maria, Valparaiso, Chile (January 2011) - Fully funded by NSF

Fellowships

- Deans Fellow, Boston University College of Engineering, 2008-2009
- Graduate Teaching Fellow, Boston University College of Engineering, Fall 2009

Research

2009-Present – Research Assistant at Boston University for Prof. L. Barba, working on heterogenous computing using CUDA with varied applications, including fast multipole accelerated boundary element methods, relaxed Krylov solvers, algebraic multigrid, fast evaluation of sums of Gaussians, Computational Fluid dynamics and high order non-oscillatory finite difference schemes for systems of conservation equations.

2008-09 – Research Assistant at Boston University for Prof. L. Barba, working on fast evaluation of Gaussians and heterogenous computing using CUDA.

2007-08 – 9 month final mathematics research project with Prof. L. Barba: "Implementation and numerical experimentation of fast algorithms for particle methods with Gaussians"

Teaching

2009 Teaching assistant for Fluid Mechanic course, ME303, Boston University

Skills and Qualifications

- Use and administration of Linux, Mac OS X, Microsoft Windows systems
- Programming in C, C++, CUDA, Python, bash, Java, Matlab, Mathematica
- Use of Boost, OpenMP, MPI, Cusp, Thrust and varied other software libraries
- Familiar with CFD packages such as OpenFOAM and Gerris