UNIVERSITY OF CALIFORNIA
Santa Barbara

Efficiently removing stiffness in the Immersed Boundary Method

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Mathematics

by

Jordan E. Fisher

Committee in Charge:

Professor Hector D. Ceniceros, Chair

Professor Paul J. Atzberger

Professor Gustavo Ponce

September 2011

The Dissertation of
Jordan E. Fisher is approved:

Professor Paul J. Atzberger

Professor Gustavo Ponce

Professor Hector D. Ceniceros, Committee Chairperson

September 2011

Efficiently removing stiffness in the Immersed Boundary Method

by

Jordan E. Fisher

To Medina, without whom I would have

been kicked out of school.

Acknowledgements

I would like to thank my advisor, Hector, for introducing me to the Immersed Boundary Method and for his support of my research. I would also like to thank our collaborator on these works, Alex Roma, for his insightful discussion and for being a generous host during my stays at Universidade de São Paulo. Thanks as well to the members of my dissertation committee, Gustavo Ponce and Paul Atzberger, for their support during my stay at UC Santa Barbara.

Thanks to Fuzzy Rogers and Paul Weakliem for helping with my computational needs, and to David Valdman for the insight that the amazing algorithm I 'discovered' was actually just an implementation of the Singular Value Decomposition.

Finally, thanks to Peter Kuchment, for introducing me to real mathematics, and to Michael Crandall, for holding me to a higher standard than I held myself to.

Curriculum Vitæ

Jordan E. Fisher

Education

2008	Master of Science in Mathematics, University of California, Santa Barbara.
2006	Bachelor of Science, University of Kansas.

Selected Publications

"Efficient solutions to robust, semi-implicit discretizations of the immersed boundary method." *Journal of Computational Physics, 228(19):7137 - 7158, 2009.* (October 2009).

"A fast, robust, and non-stiff Immersed Boundary Method." *Journal of Computational Physics, 230(12):5133 - 5153, 2011.* (January 2011).

Abstract

Efficiently removing stiffness in the Immersed Boundary Method

Jordan E. Fisher

The Immersed Boundary Method introduced by Peskin is a popular technique for modeling interactions between a fluid and a dynamic immersed boundary. The method is well suited to a large range of applications. In practice, however, numerical simulations are often hampered by time stepping constraints (stiffness) induced by strong tangential, interfacial forces. This stiffness leads to prohibitive computational costs in many applications. Since the introduction of the method much effort has been devoted to remove this limitation. This effort has led to a greater understanding of the origin of the stiffness constraint, as well as to the discovery of stable implicit and semi-implicit schemes that robustly remove stiffness. In practice, however, these implicit schemes pose a formidable non-linear system of equations to solve. The cost of solving the systems overwhelms the cost savings gained by removing the stiffness restraint.

In this context, we revisit a robust semi-implicit discretization introduced by Peskin in the late 70's which has received renewed attention recently. This discretization, in which the spreading and interpolation operators are lagged, leads to

a linear system of equations for the interface configuration at the future time, provided the interfacial force is linear. However, this linear system is large and dense and thus it is challenging to streamline its solution. Moreover, while the same linear system or one of similar structure could potentially be used in Newton-type iterations, nonlinear and highly stiff immersed structures pose additional challenges to iterative methods. In this work, we address these problems and propose cost-effective computational strategies for solving Peskin's lagged-operators type of discretization. We explore methods for accelerating the computation of fluid-structure interactions, as well as a range of iterative methods that utilize the fluid-structure computation to efficiently solve the implicit system.

These new methods work in a very wide range of instances, including a large class of interfacial forcing functions, Newtonian and complex fluids, and both 2D and 3D fluid domains. We show that for such applications they are nearly computationally optimal and highly stable. We demonstrate the efficacy and superiority of the methods over existing approaches with various applications.

Contents

Chapter 1

Introduction

The Immersed Boundary (IB) Method, introduced by Peskin [25], offers a flexible approach for the simulation of flow-structure interaction. It combines a Lagrangian representation of the immersed structures with an Eulerian flow description. The Lagrangian representation of the immersed boundaries endows the method with a versatile structure-building capability while the Eulerian flow description permits the use of fast flow solvers. The connection of the two representations is done seamlessly through *spreading* (of interfacial forces) and *interpolation* (of velocity at the immersed boundary) steps via mollified delta functions.

The flexibility of the Immersed Boundary Method has lead to its adoption for the computational simulation of a staggering number of diverse applications. A great deal of work has been done utilizing the method to simulate the cardiac fluid dynamics of blood flow through hearts [17, 15, 18, 16]. Indeed, one of the original applications by Peskin himself was to study flow past artificial heart valves [24].

The method has found use on smaller length scale applications as well, such as modeling the unsteady flows involved in insect flight [19, 20] and investigating air flow inside the cochlea [2]. The method has found use modeling biological cells, including aggregates of red blood cells [13]. Additional work is being carried out to extend the method to smaller regimes where Brownian motion plays an important role [1] in the fluid dynamics. The method is easily wed to models of complex fluids; for example, to model the peristalsis of polymeric fluids [28, 5], as well as to investigate the effects that viscoelastic fluid responses can have on the speed of microswimmers [29].

The immersed structures used in these and other applications often have very stiff components and as a consequence strong *tangential* forces are generated, which in turn induce severe time-step restrictions for explicit discretization [27, 26]. Fully implicit discretizations remove this hindering constraint but are seemingly impractical due to their elevated cost [33, 14].

Progress has been made eliminating stiffness for the case of simple periodic interfaces, as in the work of Hou and Shi [8, 9]. For the more general case involving non-periodic interfaces in 2D and 3D, some encouraging results have been obtained using Krylov subspace methods to solve suitable semi-implicit discretizations [14, 22, 11]. Krylov methods are often ineffectual when the relevant linear or linearized system is non-definite, which is a common occurrence for Immersed

Boundary applications. Krylov methods also often suffer from increased costs as the resolution of the simulation increases, requiring more iterations to achieve the same degree of convergence.

This work seeks to eliminate stiffness in the fully general case, where immersed structures need not be periodic or even continuous and can include cross links, tethers, and other important structure building components. In addition to differences in geometry, immersed structures also differ in the force they exert on the surrounding fluid. This force can be an arbitrary function of the structural configuration and time, and in general can make the task of removing stiffness arbitrarily difficult. Our methods developed here do not apply to all cases of force functions, but do apply to a wide range of cases, including important linear and non-linear cases, as well as cases where the Jacobian of the linearized force is both definite and non-definite. Additionally, we demonstrate that our iterative methods scale very well for high resolutions.

The methods developed here are iterative methods for solving robust, semi-implicit discretizations of the Immersed Boundary Method. Such discretizations are well known. Indeed, one such discretization was introduced by Peskin [25] in the late 70's, in which the spreading and interpolation operators are lagged, i.e. evaluated at the current interfacial configuration rather than at the future one. Newren, Fogelson, Guy, and Kirby proved that this scheme, in its first order

or second order Crank-Nicolson form, is unconditionally stable when inertia is neglected and the interfacial force is linear and self-adjoint [23]. Numerical experiments in [23], as well as our own extensive experiments, suggest the robustness of this discretization extends to the inertial case with nonlinear interfacial force.

This semi-implicit discretization leads to a system of equations, generally nonlinear, for the interface configuration at the future timestep. Solving this system is non-trivial and has been prohibitively expensive with previous methods. The heart of this work is developing fast methods for solving the implicit system, allowing for the efficient removal of stiffness from the IB Method.

There are two interrelated problems for removing robustly and efficiently the numerical stiffness of the IB Method:

1. The design of efficient solvers for the (nonlinear) system of the interface configuration.

2. The fast evaluation of flow-mediated interactions between immersed boundary points (flow-structure interactions).

In this work we will attack both problems. We will focus on Krylov solvers and Multigrid methods for our implicit solvers, as well as develop a novel implicit solver for a difficult case when the fiber force is non-linear with non-definite Jacobian.

The heart of this work, however, lies in the acceleration of key computations needed for these implicit solvers.

The predominant computational cost incurred by our solvers is the calculation of flow-mediated interactions between immersed boundary points. We will encapsulate this calculation in an operator which we refer to as the *flow-structure operator*. Accelerating the computation of this operator's output is a key component of our strategy.

In Chapter 4, we develop an approximate matrix representation of the flow-structure operator, as well as the entire implicit system. In many cases this representation allows for large acceleration of the implicit solvers. In addition, the matrix representation allows for the easy implementation of multigrid solvers. We provide both analytic and numerical evidence that the approximations employed do not degrade the overall accuracy of the IB Method.

In Chapter 5 we show that the flow-structure operator can be seen as a *multipole summation* with a suitable choice of potential. Using the Singular Value Decomposition and a new, efficient, iterative algorithm we compute L^2-optimal far field expansions of this potential to be used in an effective treecode strategy. The treecode method is in some sense an extension of the matrix method, and allows for even greater acceleration. Indeed, we will see that in some cases the

acceleration is so great that sophisticated solvers such as multigrid can be replaced by much simpler Krylov methods.

Chapter 2

The Immersed Boundary Method

In the following sections we will overview the continuous equations describing the IB Method, as well as one of the most popular explicit discretizations used in numerical simulations, known as the Forward Euler/Backward Euler Method (FE/BE). In Section 2.3 we will then discuss the problem of stiffness.

2.1 The continuous equations

We review the IB Method in its simplest form. We consider an incompressible, Newtonian fluid occupying a domain $\Omega \subset \mathbb{R}^2$ or $\Omega \subset \mathbb{R}^3$. Inside this domain we assume that there is an immersed, neutrally buoyant, elastic structure. This structure may be a 1D filament or a 2D surface, but may also be a more complex, dense 3D mesh, or some combination of all these elements. The structure need not be closed or even continuous. We refer to the set of points comprising the structure as Γ. We further assume there is some parametrization of Γ, given over

a parameter space B. The configuration of Γ at time t is then provided in the Lagrangian form $\mathbf{X}(s,t)$, where $s \in B$ is a Lagrangian parameter. The fluid and immersed structure form a coupled system evolving according to:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \mathbf{f}, \tag{2.1}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{2.2}$$

$$\frac{\partial \mathbf{X}}{\partial t} = \mathbf{u}(\mathbf{X},t), \tag{2.3}$$

where ρ and μ are the fluid density and fluid viscosity, respectively (both assumed to be constant). Here $\mathbf{u}(\mathbf{x},t)$ and $p(\mathbf{x},t)$ are the velocity field and the pressure, respectively, described in terms of the Eulerian, Cartesian coordinate $\mathbf{x}$. The term $\mathbf{f}$ represents the singularly supported tension force of the immersed structure acting onto the fluid. The system (2.1)-(2.3) is supplemented with initial and boundary conditions. Throughout this work, we consider only periodic boundary conditions. Typically we will take $\Omega = [0,1]^2$ or $\Omega = [0,1]^3$.

The crux of the IB Method and much of its versatility is the seamless connection of the Lagrangian representation of the immersed structure with the Eulerian representation of the flow. This is achieved via the identities:

$$\frac{\partial \mathbf{X}}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\mathbf{X}(s,t))d\mathbf{x}, \tag{2.4}$$

$$\mathbf{f}(\mathbf{x},t) = \int_B \mathbf{F}(s,t)\delta(\mathbf{x}-\mathbf{X}(s,t))ds, \tag{2.5}$$

where δ denotes the (two or three-dimensional) Dirac delta distribution. In (2.5), $\mathbf{F}$ represents the elastic force density of Γ and is described in Lagrangian coordinates. Typically, the fiber force at time t, $\mathbf{F}(\cdot, t)$, is given as a function of $\mathbf{X}(\cdot, t)$, the configuration of our immersed structure at time t. We denote this as

$$\mathbf{F} = \mathcal{A}(\mathbf{X}), \tag{2.6}$$

for some potentially nonlinear operator $\mathcal{A}$.

The IB Method is thus a coupled system. The underlying fluid and its associated velocity field $\mathbf{u}$ dictate the motion of the immersed structure $\mathbf{X}$. The configuration of $\mathbf{X}$, in turn, produces a force distribution $\mathcal{A}(\mathbf{X})$ which acts on the fluid.

2.2 The Forward Euler/Backward Euler explicit discretization

One of the most commonly used schemes with an explicit treatment of the immersed boundary is the so-called Forward Euler/Backward Euler (FE/BE) [26] in which the tension force is explicit (Forward Euler) and the viscous term is implicit (Backward Euler). We present the FE/BE method in this section.

We start by discretizing Ω as a uniform $N \times N$ or $N \times N \times N$ Cartesian grid $\mathcal{G}_\Omega$ with grid size $h = 1/N$. We represent our structure Γ as a collection of N_B

points and define $h_B = 1/N_B$. Note that different conventions may be used to discretize the Lagrangian parameter space B when Γ is not one-dimensional. We call our discretization $\mathcal{G}_B$ and take it to be a one dimensional index space, so that the discretized structure $\mathbf{X}$ may be considered as an array of N_B vectors, $\{\mathbf{X}_i\}_{i=1}^{N_B}$.

A common discretization used for the continuous equations (2.1)-(2.3) is the Forward Euler/Backward Euler discretization, written as

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}+\mathbf{u}^n\cdot\mathbf{D}_h\mathbf{u}^n\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^n), \tag{2.7}$$

$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0, \tag{2.8}$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^n}{\Delta t}=\mathcal{S}_n^*\mathbf{u}^{n+1}, \tag{2.9}$$

where a superscript n denotes a numerical approximation taken at the time $n\Delta t$ and Δt is the timestep. We use standard, second order finite differences for the spatial derivatives. The spatial operators $\mathbf{D}_h$ and L_h are the standard, second order approximations to the gradient and the Laplacian, respectively, and $\mathcal{A}_{h_B}$ is a suitable discrete version of $\mathcal{A}$.

$\mathcal{S}_n$ and $\mathcal{S}_n^*$ are the *lagged* spreading and interpolation operators, respectively. In 2D they are given by

$$(\mathcal{S}_n G)(\mathbf{x})=\sum_{s\in\mathcal{G}_B}G(s)\delta_h(\mathbf{x}-\mathbf{X}^n(s))h_B, \tag{2.10}$$

$$(\mathcal{S}_n^* w)(s)=\sum_{\mathbf{x}\in\mathcal{G}_\Omega}w(\mathbf{x})\delta_h(\mathbf{x}-\mathbf{X}^n(s))h^2, \tag{2.11}$$

where $\delta_h(\mathbf{x}) \equiv d_h(\mathbf{x}_0)d_h(\mathbf{x}_1)$ and d_h is an approximation of the one-dimensional Dirac delta distribution. In the 3D case we replace h^2 in (2.11) with h^3 and take $\delta_h(\mathbf{x}) \equiv d_h(\mathbf{x}_0)d_h(\mathbf{x}_1)d_h(\mathbf{x}_2)$.

These operators are called lagged because the interface configuration $\mathbf{X}^n$ is used instead of the future configuration $\mathbf{X}^{n+1}$.

There is flexibility in the choice of d_h but for concreteness in the presentation we choose Peskin's delta [25]:

$$d_h(r) = \begin{cases} \frac{1}{4h}\left(1 + \cos(\frac{\pi r}{2h})\right) & \text{if } |r| \leq 2h, \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

We note that while the continuous equations (2.1)-(2.3) present an instantaneous coupling between $\mathbf{u}$ and $\mathbf{X}$, the FE/BE method decouples this system into a tick-tock evolution. Holding $\mathbf{X}^n$ fixed we evolve the fluid to its next configuration $\mathbf{u}^{n+1}$ by solving (2.7)-(2.8). We then hold $\mathbf{u}^{n+1}$ fixed and calculate the updated fiber configuration $\mathbf{X}^{n+1}$ via (2.9). This allows us to perform FE/BE timesteps without solving any system of equations relating the interdependencies of $\mathbf{u}$ and $\mathbf{X}$.

2.3 Stiffness

The simplicity of the FE/BE method presented in the previous section has spurred its use in many IB applications. Unfortunately, while flexible enough to handle most IB applications, the method suffers from a severe timestep restriction when the immersed structure has high stiffness. Typically the timestep restraint is of the form $\Delta t = Ch\sigma^{-1/2}$, for some constant C dependent on the particular problem and where σ is a stiffness constant associated with the immersed structure which can be very large, $O(10^7)$ or more.

For a concrete example we consider the case of tether points. Tether points are fixed locations in the fluid domain to which fiber points are attracted via a Hookean spring. Each fiber point $\mathbf{X}_i$ is associated with a tether point $\mathbf{T}_i$ and the fiber force is then given by $\mathcal{A}_{h_B}\mathbf{X} = \sigma(\mathbf{T} - \mathbf{X})$. We consider the specific geometry of an square plate sitting in a 3D fluid with an induced flow, as described in Subsection 5.4.1. For explicit FE/BE simulations we determine what the timestep restrictions are (shown in Table 2.1), and, moreover, what are the total computational costs for the simulation in units of CPU time (shown in Table 2.2). The total simulation time is set at $T = 0.25$, which is roughly the time it takes for a fluid particle in the induced flow to traverse the fluid domain $\Omega = [0, 1]^3$ twice.

Table 2.1: Δt in the explicit simulations of the immersed plate for various values of N and σ. Δt is approximately the largest *stable* timestep, given by the formula $\Delta t = 10h\sigma^{-1/2}$.

	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N = 32$	$3.125{\cdot}10^{-5}$	$9.882{\cdot}10^{-6}$	$3.125{\cdot}10^{-6}$	$9.882{\cdot}10^{-7}$	$3.125{\cdot}10^{-7}$
$N = 64$	$1.563{\cdot}10^{-5}$	$4.941{\cdot}10^{-6}$	$1.563{\cdot}10^{-6}$	$4.941{\cdot}10^{-7}$	$1.563{\cdot}10^{-7}$
$N = 128$	$7.813{\cdot}10^{-6}$	$2.471{\cdot}10^{-6}$	$7.813{\cdot}10^{-7}$	$2.471{\cdot}10^{-7}$	$7.813{\cdot}10^{-8}$

Table 2.2: Total CPU time in hours for the explicit simulations of the immersed plate, with varying values of σ and N. * denotes an extrapolated value.

	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N = 32$	0.214	0.675	2.146	6.806	21.296
$N = 64$	4.072	12.906	40.813^*	129.063^*	408.133^*
$N = 128$	64.779^*	204.8481^*	647.787^*	2048.481^*	6477.867^*

Increasing either N or σ leads to steep timestep restrictions and large total CPU times, on the order of months. Such restrictions are typical for many IB applications. The exorbitant costs have relegated such simulations to large computers clusters, or have prevented the simulations from being pursued altogether. The intent of this work is to remove these large computational costs, opening the possibility for preforming simulations in smaller, more agile environments, as well as opening up the possibility of simulations previously too expensive to carry out at all.

Chapter 3

The Semi-Implicit Discretization

We present now the implicit discretization used in this work. In Section 3.2 we will then rearrange the resulting implicit system so that it is written in terms of the Lagrangian coordinate alone.

3.1 The semi-implicit discretization

We employ a implicit discretization very similar to the FE/BE, taking the following form

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t}+\mathbf{u}^{n}\cdot\mathbf{D}_{h}\mathbf{u}^{n}\right)=-\mathbf{D}_{h}p^{n+1}+\mu L_{h}\mathbf{u}^{n+1}+\mathcal{S}_{n}\mathcal{A}_{h_{B}}(\mathbf{X}^{n+1}), \tag{3.1}$$

$$\mathbf{D}_{h}\cdot\mathbf{u}^{n+1}=0, \tag{3.2}$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^{n}}{\Delta t}=\mathcal{S}_{n}^{*}\mathbf{u}^{n+1}. \tag{3.3}$$

Indeed, this discretization differs from the FE/BE discretization in only a single place, replacing the occurrence of $\mathbf{X}^n$ in (2.7) with the future configuration $\mathbf{X}^{n+1}$ in (3.1).

Importantly, $\mathcal{S}_n$ and $\mathcal{S}_n^*$ are the *lagged* spreading and interpolation operators, respectively. By lagged we mean that the location of the spreading and interpolation is determined by the old fiber configuration $\mathbf{X}^n$. This choice makes $\mathcal{S}_n$ and $\mathcal{S}_n^*$ linear operators. We stress that this linearity is critical to the numerical efficiency of our algorithm.

Newren, Fogelson, Guy, and Kirby proved that this scheme, among a family of other similar semi-implicit schemes, is unconditionally stable when inertia is neglected and the interfacial force is linear and self-adjoint [23]. Numerical experiments in [23], as well as our own extensive experiments, suggest the robustness of this discretization extends to the inertial case with nonlinear interfacial force.

The use of lagged spreading and interpolation operators in a semi-implicit scheme was originally proposed by Peskin [25] in the late 70's. However, it was not known that this scheme was stable until 2007 with the work of Newren, *et al.*

The thrust of the present work is to find efficient and robust methods for solving the specific system of equations (3.1)-(3.3). We note, however, that the methods developed here can be easily applied to many other semi-implicit schemes in the family of schemes proved stable by Newren et al. in [23]. In particular,

there is no difficulty in adapting our methods to handle the Crank-Nicolson discretization

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}+\mathbf{u}^n\cdot\mathbf{D}_h\mathbf{u}^n\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\frac{1}{2}(\mathbf{X}^n+\mathbf{X}^{n+1})), \tag{3.4}$$

$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0, \tag{3.5}$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^n}{\Delta t}=\mathcal{S}_n^*(\frac{1}{2}(\mathbf{u}^n+\mathbf{u}^{n+1})). \tag{3.6}$$

3.2 The Lagrangian system

The discretization (3.1)-(3.3) presents a formidable implicit system to solve. Before outlining a method of solution, we seek a simpler representation. In particular (3.1)-(3.3) is a system of both $\mathbf{X}^{n+1}$ and $\mathbf{u}^{n+1}$, defined on an Eulerian and Lagrangian grid respectively. It would be convenient if we could eliminate one of these variables and work on a single grid. In this section we provide a simplified form for (3.1)-(3.3) by eliminating the occurrence of $\mathbf{u}^{n+1}$.

Let us rewrite (3.1) as

$$\mathbf{u}^{n+1}=-\frac{\Delta t}{\rho}\mathbf{D}_h p^{n+1}+\nu\Delta t L_h\mathbf{u}^{n+1}+\mathbf{a}^{n+1}, \tag{3.7}$$

where $\nu=\mu/\rho$ and

$$\mathbf{a}^{n+1}=\frac{\Delta t}{\rho}\,\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n+1})+\mathbf{u}^n-\Delta t\ \mathbf{u}^n\cdot\nabla_h\mathbf{u}^n. \tag{3.8}$$

We can eliminate the pressure term in (3.7) using (3.2), by introducing a discrete projection P_h defined as

$$\mathbf{v} = P_h\mathbf{v} + \mathbf{D}_h\phi_v, \qquad \mathbf{D}_h \cdot P_h\mathbf{v} = 0, \qquad P_h\mathbf{D}_h\phi_v = 0, \tag{3.9}$$

for any smooth vector field $\mathbf{v}$ defined on the grid $\mathcal{G}_\Omega$. Applying P_h to (3.7), using (3.2), and the fact that for periodic boundary conditions L_h and P_h commute we get

$$\mathbf{u}^{n+1} = \nu\Delta t L_h\mathbf{u}^{n+1} + P_h\mathbf{a}^{n+1}, \tag{3.10}$$

that is

$$\mathbf{u}^{n+1} = (I - \nu\Delta t L_h)^{-1}P_h\mathbf{a}^{n+1}. \tag{3.11}$$

Let us denote

$$\mathcal{L}_h = (I - \nu\Delta t L_h)^{-1}P_h. \tag{3.12}$$

We refer to $\mathcal{L}_h$ as the *flow-structure operator*. Note that with periodic boundary conditions and a standard second order finite difference approximation for the spatial derivatives, $I - \nu\Delta t L_h$ is symmetric and positive definite and as a result so is its inverse. On the other hand, P_h is also symmetric and, being a projection, it is positive semi-definite. Moreover, P_h and $(I - \nu\Delta t L_h)^{-1}$ commute as can be shown using the discrete Fourier transform. Consequently, these two symmetric operators can be diagonalized by the same orthogonal matrix and thus the product, $\mathcal{L}_h$, is positive semi-definite.

For calculations of the form $\mathcal{L}(\mathbf{f})$ we rely on an FFT based method, although any suitable method could be employed. Using $\mathcal{L}_h$, the semi-implicit method (3.1)-(3.3) can be expressed as

$$\mathbf{u}^{n+1} = \mathcal{L}_h \mathbf{a}^{n+1}, \tag{3.13}$$

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathbf{u}^{n+1}, \tag{3.14}$$

where $\mathbf{a}^{n+1}$ is given by (3.8). Eliminating $\mathbf{u}^{n+1}$ in (3.14) we obtain a system of equations for the immersed boundary configuration $\mathbf{X}^{n+1}$:

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n, \tag{3.15}$$

where

$$\mathcal{M}_n = \alpha \, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n, \tag{3.16}$$

with

$$\alpha = \frac{(\Delta t)^2}{\rho} \tag{3.17}$$

and

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h [\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n]. \tag{3.18}$$

We have thus reduced (3.1)-(3.3) to a single system of equations involving only the unknown $\mathbf{X}^{n+1}$. After solving this system we can obtain $\mathbf{u}^{n+1}$ via (3.13). We call the linear operator $\mathcal{M}_n$ the *flow-structure operator*.

Assuming we can solve (3.15) for $\mathbf{X}^{n+1}$ we can then calculate $\mathbf{u}^{n+1}$ via (3.13).

Note that we could just as easily eliminate $\mathbf{X}^{n+1}$ and obtain a system involving only the unknown $\mathbf{u}^{n+1}$, but there are two compelling reasons to prefer a system in terms of $\mathbf{X}^{n+1}$. First, typically $N_B = |CG_B|$ is much smaller than $|\mathcal{G}_\Omega|$, the number of Eulerian grid points, which is $O(N^2)$ in 2D and $O(N^3)$ in 3D. By working in Lagrangian coordinates we are thus greatly reducing the number of free variables in our system.

Second, consider the fluid interaction between two fiber points $\mathbf{X}_j^{n+1}$ and $\mathbf{X}_k^{n+1}$. This interaction is independent of the location of any other fiber point, and depends only the locations of $\mathbf{X}_j^{n+1}$ and $\mathbf{X}_k^{n+1}$. We will make considerable use of this simplicity in the design of our solvers. If instead we had an equation in terms of $\mathbf{u}^{n+1}$ only, then the interaction between two fluid cells $\mathbf{u}_{j_1,k_1}^{n+1}$ and $\mathbf{u}_{j_2,k_2}^{n+1}$ would depend on the location of every fiber point comprising $\mathbf{X}^n$. This complicates the design of solvers considerably.

3.2.1 On the positive-definiteness of $\mathcal{M}_n$

We observe that the positive semi-definiteness of $\mathcal{L}_h$ extends to $\mathcal{M}_n$. Indeed, considering the 2D case, if we define the inner product on Ω as

$$(u, v)_\Omega = \sum_{\mathbf{x} \in \mathcal{G}_\Omega} u(\mathbf{x}) v(\mathbf{x}) h^2, \tag{3.19}$$

$$(\mathbf{u}, \mathbf{v})_\Omega = (u_1, v_1)_\Omega + (u_2, v_2)_\Omega, \tag{3.20}$$

and on B as

$$(F, G)_B = \sum_{\mathbf{x} \in \mathcal{G}_B} F(\mathbf{x}) G(\mathbf{x}) h_B, \tag{3.21}$$

$$(\mathbf{F}, \mathbf{G})_B = (F_1, G_1)_B + (F_2, G_2)_B, \tag{3.22}$$

then $\mathcal{S}_n$ and $\mathcal{S}_n^*$ as given in (2.10)-(2.11) are adjoints to each other and

$$(\mathbf{F}, \mathcal{M}_n \mathbf{F})_B = \alpha(\mathbf{F}, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n \mathbf{F})_B = \alpha(\mathcal{S}_n \mathbf{F}, \mathcal{L}_h \mathcal{S}_n \mathbf{F})_B \geq 0. \tag{3.23}$$

In fact, $(\mathbf{F}, \mathcal{M}_n \mathbf{F})_B > 0$ for $\mathbf{F} \neq 0$ unless the Eulerian force $\mathcal{S}_n \mathbf{F}$ is in the kernel of the projection, i.e. if it is a gradient field, or if there are too many Lagrangian points per Eulerian cell and injectivity of $\mathcal{S}_n$ is lost. There are physical situations, such as for example a circular drop under uniform surface tension at equilibrium, where the Eulerian force is a gradient field at the continuous level (to balance the gradient of the pressure). However, as it is well known, the IB method spreading of the force fails in general to produce a discrete gradient which results in the generation of non-zero velocities referred to as spurious currents [32]. Ironically, this defect of the IB approach has the benefit of rendering $\mathcal{M}_n$ positive definite provided $\mathcal{S}_n$ remains injective. This is something that we will exploit via the splitting method, as detailed in Chapter 6.

3.3 Solving the implicit system

Within our framework, there are two main difficulties to produce a cost-efficient and robust, non-stiff IB Method. The first is the heavy cost of applying and inverting the flow-structure operator $\mathcal{M}_n$. The second is to produce an effective, in general nonlinear, iterative solver for the implicit system (3.15). Because $\mathcal{A}_{h_B}$ can be a very general nonlinear operator, the second problem can prove difficult.

In the present work we address both of these problems. Methods for reducing the computational cost of evaluating expressions of the form $\mathcal{M}_n\mathbf{F}$ will be developed in the following parts. In Chapter 4 we will construct a matrix form for $\mathcal{M}_n$, and show that, especially in 2D, this form can greatly accelerate iterative methods used to solve (3.15). In Chapter 5 we will instead compute expressions of the form $\mathcal{M}_n\mathbf{F}$ by first casting $\mathcal{M}_n\mathbf{F}$ as a multipole summation and then accelerating the summation via a fast treecode. In both 2D and 3D this method reduces the cost of evaluating $\mathcal{M}_n\mathbf{F}$ to an even greater extent than the matrix method. In fact, with the treecode in hand we can construct implicit solvers that are asymptotically faster than a single timestep of FE/BE, while maintaining a much larger Δt than a FE/BE simulation can afford.

The second problem, designing effective iterative solvers for the implicit system (3.15), is more application dependent. In the case that $\mathcal{A}_{h_B}$ is linear and

negative-semidefinite a plethora of iterative solvers are available. Krylov methods in particular have been used to good success in prior work [22]. With the development of the matrix method and treecode we open up the possibility of using Gauss-Seidel and Multigrid solvers, which we will show are much more efficient than the Krylov methods (Section 4.2).

For non-linear choices of $\mathcal{A}_{h_B}$ we can solve the implicit system via Newton's iterations, provided the Jacobian J of $\mathcal{A}_{h_B}$ is negative-semidefinite. In Section 4.2.1) we will show that such an approach converges well. At each stage of Newton's method we are left with a linear system identical in nature to the case when $\mathcal{A}_{h_B}$ is linear and negative-semidefinite. Newton iterations can then be solved via the multigrid we develop, or via Krylov methods, as was demonstrated by Mori and Peskin in [22].

Handling a non-definite J is more challenging. In Chapter 6 a splitting algorithm is proposed which performs well for the wide class of functions $\mathcal{A}_{h_B}$ such that the eigenvalues of J are all large in magnitude. Additional work remains to be done for the case where the eigenvalues of J are mixed in magnitude.

3.3.1 Measuring computational cost

With the operator $\mathcal{L}_h$ in hand, as defined in (3.12), we can rewrite the FE/BE method as

$$\mathbf{u}^{n+1} = \mathcal{L}_h \left[\frac{\Delta t}{\rho} \mathcal{S}_n \mathcal{A}_{h_B}(\mathbf{X}^n) + \mathbf{u}^n - \Delta t \ \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n \right], \tag{3.24}$$

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathbf{u}^{n+1}. \tag{3.25}$$

The main cost of this scheme per time step is the fluid solver, i.e. the operation involving $\mathcal{L}_h$. On the other hand any solution method for our implicit system (3.15) requires the evaluation of $\mathbf{b}^n$, given by (3.18), and thus at least one fluid solver operation. Hence, it seems appropriate to measure the cost of iterative methods for (3.15) relative to that of one FE/BE time-step and we will refer to this unit as one FE/BE. Our goal is to present robust solution methods to (3.15) that have cost of just a few FE/BE's. Naturally, because the semi-implicit scheme allows for time steps several orders of magnitude larger than those permitted by the FE/BE scheme in a stiff problem, the extra computational work per time step will be more than compensated and a speed-up of several orders of magnitude could be achieved.

Chapter 4

The Matrix Method

Consider the simplified case where $\mathcal{A}_{h_B}$ is linear and negative-semidefinite. Then our implicit system (3.15) can be rewritten as

$$(I - \mathcal{M}_n \mathcal{A}_{h_B})(\mathbf{X}^{n+1}) = \mathbf{b}^n. \tag{4.1}$$

The most straightforward way to solve (4.1) is via a Krylov method, in particular Conjugate Gradient (CG). Each iteration of CG requires the evaluation of the operator $(I - \mathcal{M}_n \mathcal{A}_{h_B})$, which, in particular, requires the evaluation of $\mathcal{M}_n$ for some input. The operator $\mathcal{M}_n$, defined via (3.16), requires a fluid solve to evaluate. This fluid solve is on the same order of cost as a single timestep of the explicit FE/BE method. CG will often take a few dozen iterations to converge properly, thus rendering our implicit timestep 20 to 30 times more expensive than its explicit counterpart.

The implicit method has the advantage of taking much larger Δt, potentially making it cheaper than an explicit method overall. However, for non-linear $\mathcal{A}_{h_B}$,

CG will be used multiple times per timestep inside of a Newton's method, pushing up the cost of a single implicit timestep to 50 to 100 times that of an explicit method. This cost can destroy the computational savings the implicit method is intended to capture.

The following sections explore the idea of explicitly constructing an approximate matrix form for the linear operator $\mathcal{M}_n$ and its use in iterative methods. There are many benefits to doing so.

One of the advantages of having a matrix representation of the operator $\mathcal{M}_n$ is that we can gain access to a wider range of iterative methods. Multigrid smoothers like Gauss-Seidel or S.O.R., which are inaccessible with the spreading-fluid solver-interpolation approach, could be easily implemented if the matrix is available. Also, multigrid-based methods are greatly simplified when we have a matrix representation of $\mathcal{M}_n$, which allows us to employ a straightforward algebraic multigrid as opposed to a geometric one.

A second advantage of having a matrix representation of $\mathcal{M}_n$ is that we can obtain direct solutions of *coarse* (small) linear systems of the form $\mathcal{M}_n\mathbf{F} = \mathbf{Z}$ which could be useful in a number of situations; an instance of this will be discussed later in the context of the 2D heart valve model (see Section 6.1).

Perhaps the main advantage of a matrix form for $\mathcal{M}_n$, at least in 2D, is in reducing the cost when computing quantities of the form $\mathcal{M}_n\mathbf{F}$. If this linear

operator applied to $\mathbf{F}$ is computed via the operator sequence $\mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n$ (spreading, fluid solver, interpolation) then the cost is about one FE/BE or $O(N^2 \log N) + O(N_B)$, when N^2 is the number of Eulerian nodes and the fluid solver is based on the Fast Fourier Transform (FFT).

On the other hand, a matrix-vector multiplication involving $\mathcal{M}_n$ requires $O(N_B^2)$ operations. If $N_B \sim N$ as it is typical in 2D applications, then the second approach has lower computational complexity. While the elimination of the factor $\log N$ might seem like a modest gain, we find that in practice (for $N_B = 2N$) using the matrix representation to compute matrix-vector products is significantly faster than the spreading-fluid solver-interpolation approach, even at modest resolutions.

Table 4.1 gives the CPU time in units of FE/BE (average CPU time of one FE/BE time-step for a given $N \times N$ Eulerian grid and $N_B = 2N$) for matrix-vector multiplications computed using a matrix representation of $\mathcal{M}_n$. For example, at $N_B = 1024$, the computation of a matrix-vector product with this approach is 1/71 FE/BE, i.e. 71 times faster than it would be using the spreading-fluid solver-interpolation option. If in addition, we take into the account that many matrix-vector products are needed in the course of an iterative method then the computational savings are significant.

$N_B = 2N$	256	512	1024
Matrix-Vector product	1/26.8	1/58	1/71

Table 4.1: Average CPU time in FE/BE units for a matrix-vector multiplication involving $\mathcal{M}_n$ for given $N_B = 2N$.

We note however that as the ratio N_B/N increases the savings get reduced and the computational advantage of using the matrix to obtain the product could be lost eventually. For example, in our application of the 2D heart valve model where $N_B \sim 4N$, the speedup in using the matrix form drops to roughly 10 at moderate to fine resolutions. While not as dramatic as that in the $N_B = 2N$ case, it still leads to substantial savings in the overall algorithm. And, of course, the matrix form allows for the use of multigrid, which itself is invaluable.

4.1 An expedited computation of a matrix representation of $\mathcal{M}_n$

A direct computation of the matrix form of $\mathcal{M}_n$ is prohibitively expensive, requiring $O(Nb \times FE/BE)$ operations. Because of this high cost all related work to date has avoided construction of the matrix. In this work we overcome the high cost by finding a suitable *approximation* to the matrix form of $\mathcal{M}_n$, which can be found in $O(N_B^2)$ time.

4.1.1 Approximation from translation invariance

Recall $\mathcal{M}_n = \alpha\, \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n$ is a linear function from $\mathbb{R}^{3N_B}$ to $\mathbb{R}^{3N_B}$. For a componentwise calculation of $\mathcal{M}_n$, we focus on two fiber points located at $\mathbf{x} \in \Omega$ and $\mathbf{y} \in \Omega$. We will consider both the 2D and 3D case. We take $\mathbf{e}_1$, $\mathbf{e}_2$ to be the canonical basis vectors of $\mathbb{R}^2$, ($\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ to be the canonical basis vectors of $\mathbb{R}^3$). We then define the scalar field

$$\delta_{\mathbf{x}}(\mathbf{z}) \equiv \delta_h(\mathbf{z} - \mathbf{x}). \tag{4.2}$$

Hence, $\mathbf{e}_i \delta_{\mathbf{x}}$ is the field obtained by spreading the unit force $\mathbf{e}_i$ at $\mathbf{x}$ to the fluid domain. We denote the influence of this force field on the fluid velocity as $\mathbf{u}_{\mathbf{x}}^i \equiv \mathcal{L}_h \mathbf{e}_i \delta_{\mathbf{x}}$. The induced velocity on $\mathbf{y}$ is now just an interpolation of $\mathbf{u}_{\mathbf{x}}^i$ at $\mathbf{y}$. We perform this procedure for $i = 1, 2$ ($i = 1, 2, 3$) and store the two (three) resulting vectors as a 2×2 (3×3) matrix defined via

$$(G_h(\mathbf{x}, \mathbf{y}))_{ij} \equiv \alpha h_B \sum_{\mathbf{z} \in \mathcal{G}_\Omega} u_{\mathbf{x}j}^i(\mathbf{z}) \delta_h\left(\mathbf{z} - \mathbf{y}\right) h^3, \tag{4.3}$$

where $u_{\mathbf{x}j}^i$ is the j-th component of $\mathbf{u}_{\mathbf{x}}^i$ and the summation against δ_h provides the necessary interpolation. G_h is a tensor valued function, acting on $\Omega \times \Omega$ and returning a 2×2 (3×3) matrix that specifies how forces at $\mathbf{x}$ affect the fiber displacement at $\mathbf{y}$. If there is a force $\mathbf{f}$ on $\mathbf{x}$ then the induced displacement at $\mathbf{y}$

is simply $G_h(\mathbf{x}, \mathbf{y})\mathbf{f}$. Evaluating G_h for all ordered pairs $(\mathbf{X}_i, \mathbf{X}_j)$, we have that

$$(\mathcal{M}_n\mathbf{F})_i = \sum_{0 \le j \le N_B} G_h(\mathbf{X}_j, \mathbf{X}_i)\mathbf{F}_j, \qquad \text{for } 0 \le i \le N_B. \tag{4.4}$$

In practice, calculating $G_h(\mathbf{x}, \mathbf{y})$ for any pair of points $\mathbf{x}, \mathbf{y} \in \Omega$ is prohibitively expensive. It would require a fluid solve, i.e. $O(N^2 \log N)$ operations per pair of points in 2D, and $O(N^3 \log N)$ in 3D. To overcome this limitation we approximate $G_h(\mathbf{x}, \mathbf{y})$ by $G_h(\mathbf{0}, \mathbf{x} - \mathbf{y})$, which we denote simply as $G_h(\mathbf{x} - \mathbf{y})$. G_h can now be seen as a matrix field over Ω. We precompute and store the values of $G_h(\mathbf{z})$ for every $\mathbf{z} \in \mathcal{G}_\Omega$. This allows us to reduce future costs of evaluating $G_h(\mathbf{z})$ to $O(1)$. When $\mathbf{z} \notin \mathcal{G}_\Omega$ we use bilinear (trilinear) interpolation to approximate $G_h(\mathbf{z})$.

For a given set of parameters $(\Delta t, \mu, \rho, h, \Omega)$ defining the discretized fluid domain and the fluid material properties, G_h need only be calculated once and can be reused throughout a simulation and in fact, in any other simulations with identical fluid parameters. The price we pay for this computational speedup is the error incurred by assuming translational invariance, as well as by using interpolation between grid points. We will show in Section 4.1.4 and Section 4.1.5 that these errors do not degrade the overall accuracy of the IB Method.

We note that translation invariance here is dependent on the type of boundary conditions on our domain Ω. In particular, Ω must be either periodic or infinite ($\Omega = \mathbb{R}^2$ or $\Omega = \mathbb{R}^3$). This requirement is not as restrictive as it might seem as Dirichlet type of boundary conditions can be easily implemented in the IB

framework as immersed boundaries within a periodic domain. Indeed, despite using only periodic boundary conditions in this work, we easily accommodate Dirichlet boundary conditions in one of our test cases, as seen in Section 6.1.

4.1.2 Precomputing the G_h lookup table

We refer to the matrix field of G_h over $\mathcal{G}_h$ as a lookup table. The steps for creating the lookup table are as follows.

1. First, we spread a unit force $\mathbf{e}_i$ at the origin, obtaining a force field defined as $\mathbf{e}_i\delta(\mathbf{x})$ at any $\mathbf{x} \in \mathcal{G}_h$.

2. We calculate the influence of this force field on the fluid velocity by applying the operator $\mathcal{L}_h$, obtaining the vector field $\mathbf{u}_\mathbf{0}^i \equiv \mathcal{L}_h(\mathbf{e}_i\delta_h)$.

3. We spread the velocity $\mathbf{u}_\mathbf{0}^i$ at every point $\mathbf{x} \in \mathcal{G}_h$, defining

$$(G_h(\mathbf{x}))_i \equiv \alpha h_B \sum_{\mathbf{z}\in\mathcal{G}_\Omega} \mathbf{u}_\mathbf{0}^i(\mathbf{z})\delta_h\left(\mathbf{z}-\mathbf{x}\right)h^3. \tag{4.5}$$

4. Repeating the above for $i = 0, 1$ ($i = 0, 1, 2$), we arrive at our 2×2 (3×3) tensor lookup table G_h over $\mathcal{G}_h$.

We can now rapidly evaluate terms of the form $G_h(\mathbf{X}_j - \mathbf{X}_i)$ by looking up the value of $\mathbf{X}_j - \mathbf{X}_i$ on our lookup table. For vectors that do not lie exactly on an Eulerian intersection we make use of trilinear interpolation.

4.1.3 Constructing an approximation to the matrix form of $\mathcal{M}_n$

Now that we can cheaply approximate terms of the form $G_h(\mathbf{X}_j - \mathbf{X}_i)$ we can outline the procedure for constructing our approximation to the matrix form of $\mathcal{M}_n$.

We consider the 2D case, where $\mathcal{M}_n$ can be considered a $2N_B \times 2N_B$ matrix. We will likewise consider $\mathbf{X}$ to be a vector of length $2N_B$, structured as

$$\mathbf{X} = \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} X_1 \\ \vdots \\ X_{N_B} \\ Y_1 \\ \vdots \\ Y_{N_B} \end{bmatrix}. \tag{4.6}$$

Similarly we write $\mathbf{F} = \mathcal{A}_{h_B}(\mathbf{X}) = (F, G)^T$. $\mathcal{M}_n$ can likewise be broken into components. In particular, there exist four $N_B \times N_B$ matrices A, B, C, and D such that

$$\mathcal{M}_n\mathbf{F} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} F \\ G \end{bmatrix}. \tag{4.7}$$

$N_B = 2N$	256	512	1024
Matrix construction	1/3.3	1/1.8	1/2.5

Table 4.2: Average CPU time in FE/BE units to construct the matrix approximation to $\mathcal{M}_n$ for given $N_B = 2N$

The components of these matrices are precisely

$$\begin{bmatrix} A_{ij} & B_{ij} \\ C_{ij} & D_{ij} \end{bmatrix} = G_h(\mathbf{X}_i - \mathbf{X}_j). \tag{4.8}$$

Each evaluation of $G_h(\mathbf{z})$ costs only $O(1)$ and, consequently, we can construct in this manner the entire matrix representation of $\mathcal{M}_n$ at an optimal $O(N_B^2)$ cost. Table 4.2 gives the CPU time to obtain the matrix in units of FE/BE, when $N_B = 2N$.

4.1.4 Error estimate for approximation of the matrix representation of $\mathcal{M}_n$ in 2D

We proceed to obtain an error estimate for the above approximation of $\mathcal{M}_n$ in the case of a 2D fluid. In what follows we assume the periodic domain is $\Omega = [0,1] \times [0,1]$ and consider a uniform grid $\mathcal{G}_\Omega$ with grid size $h = 1/N$. We first require a bound on G_h, the Green's function for $\mathcal{L}_h$. Given the vector fields

$$\mathbf{g}_1(\mathbf{x}) = \begin{cases} (1/h^2, 0) & \text{if } \mathbf{x} = \mathbf{0} \\ (0,0) & \text{otherwise} \end{cases} \tag{4.9}$$

and

$$\mathbf{g}_2(\mathbf{x}) = \begin{cases} (0, 1/h^2) & \text{if } \mathbf{x} = \mathbf{0} \\ (0,0) & \text{otherwise} \end{cases} \tag{4.10}$$

defined for $\mathbf{x} \in \mathcal{G}_\Omega$, G_h is a 2×2 matrix valued function given by

$$G_h(\mathbf{x}) = \begin{pmatrix} u_1(\mathbf{x}) & u_2(\mathbf{x}) \\ v_1(\mathbf{x}) & v_2(\mathbf{x}) \end{pmatrix}, \tag{4.11}$$

where $(u_1, v_1) = \mathcal{L}_h(\mathbf{g}_1)$ and $(u_2, v_2) = \mathcal{L}_h(\mathbf{g}_2)$.

We make the following claims about the four components of G_h. In the estimates that follow, C stands for a generic constant, not necessarily the same.

Lemma 4.1.1. *If Δt is proportional to h then the* sup *norm of each component of G_h over $\mathcal{G}_\Omega$ is bounded asymptotically by $C\Delta t^{-1}\log(h^{-1})$, where C is a constant.*

Proof. By definition of the fluid operator $\mathcal{L}_h$, we have

$$(u_1, v_1) = (I - \nu\Delta t L_h)^{-1} P_h \mathbf{g}_1. \tag{4.12}$$

The right hand side of (4.12) can be readily obtained in Fourier space. Let $\tilde{\mathcal{G}}_\Omega = \{\mathbf{x} = (x, y)^T : \mathbf{x} \in \mathcal{G}_\Omega, x \neq 1, y \neq 1\}$ and $\mathcal{G}_F = \{\mathbf{k} = (k_1, k_2)^T : |k_1|, |k_2| < N/2\}$. Then, we can define the discrete Fourier transform (DFT) of a doubly periodic function F on $\tilde{\mathcal{G}}_\Omega$ by

$$\widehat{F}(\mathbf{k}) = \sum_{\mathbf{x} \in \tilde{\mathcal{G}}_\Omega} F(\mathbf{x}) e^{-2\pi i \mathbf{x}\cdot\mathbf{k}} \tag{4.13}$$

for $\mathbf{k} \in \mathcal{G}_F$ with discrete Fourier inverse

$$F(\mathbf{x}) = \frac{1}{N^2} \sum_{\mathbf{k} \in \mathcal{G}_F} \widehat{F}(\mathbf{k}) e^{2\pi i \mathbf{x} \cdot \mathbf{k}} \tag{4.14}$$

for $\mathbf{x} \in \tilde{\mathcal{G}}_\Omega$. Then, for $\mathbf{k} \in \mathcal{G}_F$ and $|\mathbf{k}| \neq 0$, direct calculation gives us:

$$|\widehat{P_h \mathbf{g}_1}(\mathbf{k})| = |\hat{\mathbf{g}}_1(\mathbf{k}) - |\widehat{D}_h|^{-2} \widehat{D}_h \widehat{D}_h \cdot \hat{\mathbf{g}}_1(\mathbf{k})| \leq |\hat{\mathbf{g}}_1(\mathbf{k})| = 1/h^2 \tag{4.15}$$

and we set the zero mode ($\mathbf{k} = \mathbf{0}$) of the discrete projection to zero. Using (4.12) together with the inverse DFT (4.14) we have

$$\|u_1\|_\infty = \frac{1}{N^2} \max_{\mathbf{x} \in \mathcal{G}_\Omega} \left| \sum_{\mathbf{k} \in \mathcal{G}_F} e^{2\pi i \mathbf{x} \cdot \mathbf{k}} \hat{u}_1(\mathbf{k}) \right| \leq \frac{1}{N^2} \sum_{\mathbf{k} \in \mathcal{G}_F} \left| \frac{1/h^2}{1 - \nu \Delta t \widehat{L_h}(\mathbf{k})} \right|, \tag{4.16}$$

Here the Fourier symbol of the five-point Laplacian L_h can be written as

$$\begin{aligned} -\widehat{L}_h(\mathbf{k}) &= \frac{1}{h^2} \left[4 - 2\cos(2\pi k_1 h) - 2\cos(2\pi k_2 h) \right] \\ &= \frac{4}{h^2} \left[\sin^2(\pi k_1 h) + \sin^2(\pi k_2 h) \right]. \end{aligned} \tag{4.17}$$

Furthermore, using the inequality $\sin(x/2) \geq x/\pi$ for $0 \leq x \leq \pi$ and bounding the resulting sum by an integral, we get

$$\begin{aligned} \|u_1\|_\infty &\leq \sum_{\mathbf{k} \in \mathcal{G}_F} \frac{1}{1 + 16\nu \Delta t |\mathbf{k}|^2} \leq C \int_0^{\frac{\sqrt{2}}{2h}} \frac{1}{1 + 16\nu \Delta t r^2} r dr \\ &= \frac{C}{16\nu \Delta t} \log[1 + 8\nu \Delta t h^{-2}]. \end{aligned} \tag{4.18}$$

Thus, if $\Delta t \propto h$ then $\|u_1\|_\infty \leq C \Delta t^{-1} \log(h^{-1})$ for sufficiently small h. The remaining three components of G_h can be shown to have the same type of bound via similar calculations. □

Consider now two fiber points, say $\mathbf{X}_1$ and $\mathbf{X}_2$, with a point horizontal force on $\mathbf{X}_2$ of unit magnitude. If we spread this force we obtain a vector field $(f, 0)$ where

$$f(\mathbf{x}) = \delta_h(\mathbf{x} - \mathbf{X}_2)h_B. \tag{4.19}$$

Substituting this vector field into our fluid solver we obtain $(u, v) = \mathcal{L}_h(f, 0)$, where

$$u(\mathbf{x}) \equiv \sum_{\mathbf{x}_2 \in \mathcal{G}_\Omega} u_1(\mathbf{x} - \mathbf{x}_2)\delta_h(\mathbf{x}_2 - \mathbf{X}_2)h^2 h_B \tag{4.20}$$

and u_1 is one of the components of the discrete Green's function (4.11). The interpolated velocity of u at $\mathbf{X}_1$ is

$$u(\mathbf{X}_1) \equiv \sum_{\mathbf{x}_1 \in \mathcal{G}_\Omega} u(\mathbf{x}_1)\delta_h(\mathbf{x}_1 - \mathbf{X}_1)h^2. \tag{4.21}$$

The value $A_{1,2} \equiv \alpha u(\mathbf{X}_1)$ is exactly the horizontal displacement of $\mathbf{X}_1$ induced by a unit horizontal force on $\mathbf{X}_2$, where $\alpha = (\Delta t)^2/\rho$ as before. $A_{1,2}$ is one of four entries in the matrix representation of $\mathcal{M}_n$ relating forces at $\mathbf{X}_2$ to displacement at $\mathbf{X}_1$. We will focus only on $A_{1,2}$. The remaining three values may be analyzed in a similar manner.

We dub our approximation to $A_{1,2}$ by $\tilde{A}_{1,2}$. We obtain our approximation by shifting both $\mathbf{X}_1$ and $\mathbf{X}_2$ an equal amount such that $\mathbf{X}_2$ lies exactly on an Eulerian intersection. That is we shift by $\mathbf{r} \in [-h/2, h/2) \times [-h/2, h/2)$ such that $\mathbf{X}_2 + \mathbf{r} \in \mathcal{G}_\Omega$. We then proceed as before, starting with a unit horizontal force at

$\mathbf{X}_2 + \mathbf{r}$. Spreading this force results in a vector field with x-component given by

$$\tilde{f}(\mathbf{x}) = \delta_h(\mathbf{x} - \mathbf{X}_2 - \mathbf{r})h_B. \tag{4.22}$$

Setting $(\tilde{u}, \tilde{v}) = \mathcal{L}_h(\tilde{f}, 0)$ we have

$$\tilde{u}(\mathbf{x}) \equiv \sum_{\mathbf{x}_2 \in \mathcal{G}_\Omega} u_1(\mathbf{x} - \mathbf{x}_2)\delta_h(\mathbf{x}_2 - \mathbf{X}_2 - \mathbf{r})h^2 h_B. \tag{4.23}$$

Finally we obtain our proposed approximation via $\tilde{A}_{1,2} \equiv \alpha\tilde{u}(\mathbf{X}_1 + \mathbf{r})$ where

$$\tilde{u}(\mathbf{X}_1 + \mathbf{r}) \equiv \sum_{\mathbf{x}_1 \in \mathcal{G}_\Omega} \tilde{u}(\mathbf{x}_1)\delta_h(\mathbf{x}_1 - \mathbf{X}_1 - \mathbf{r})h^2. \tag{4.24}$$

Let M_n be the exact matrix representation of $\mathcal{M}_n$ and $\tilde{M}_n$ be the approximate matrix representation of $\mathcal{M}_n$ arrived at via the methods given above. We are then in a position to state an estimate for the error $\|M_n - \tilde{M}_n\|_\infty$.

Theorem 4.1.1. *The following bound on translational error holds,*

$$|(G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y}))_{ij}| < \frac{2(\Delta t)^2}{\rho} \|(G_h)_{ij}\|_\infty h_B, \tag{4.25}$$

for all $\mathbf{x}, \mathbf{y} \in \Omega$. *Moreover,* $\|M_n - \tilde{M}_n\|_\infty \leq Ch^2 \log h^{-1}$ *when both* Δt *and* h_B *are proportional to* h *and where* C *is a constant.*

Proof. Using (4.20)-(4.21), the fact that $A_{1,2} \equiv \alpha u(\mathbf{X}_1)$, and (4.23)-(4.24) we have

$$\begin{aligned} A_{1,2} - \tilde{A}_{1,2} = \alpha \sum_{\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{G}_\Omega} u_1(\mathbf{x}_1 - \mathbf{x}_2) \left[\delta_h(\mathbf{x}_1 - \mathbf{X}_1)\delta_h(\mathbf{x}_2 - \mathbf{X}_2)\right. \\ \left. -\delta_h(\mathbf{x}_1 - \mathbf{X}_1 - \mathbf{r})\delta_h(\mathbf{x}_2 - \mathbf{X}_2 - \mathbf{r})\right] h^4 h_B \end{aligned} \tag{4.26}$$

hence,

$$\begin{aligned} |A_{1,2} - \tilde{A}_{1,2}| \leq \alpha \left\| u_1 \right\|_\infty \sum_{\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{G}_\Omega} & [\delta_h(\mathbf{x}_1 - \mathbf{X}_1)\delta_h(\mathbf{x}_2 - \mathbf{X}_2) \\ & + \delta_h(\mathbf{x}_1 - \mathbf{X}_1 - \mathbf{r})\delta_h(\mathbf{x}_2 - \mathbf{X}_2 - \mathbf{r})]\, h^4 h_B \\ = \frac{2(\Delta t)^2}{\rho} \left\| u_1 \right\|_\infty & h_B, \end{aligned} \tag{4.27}$$

where we have made use of the identity $\Sigma_{\mathbf{x} \in \mathcal{G}_\Omega} \delta_h(\mathbf{x} - \mathbf{y}) h^2 = 1$ for any shift $\mathbf{y} \in \Omega$ in the last step of the estimate. Suppose that $\Delta t \propto h$ and $h_B \propto h$, then from Lemma 4.1.1 we get that $|A_{1,2} - \tilde{A}_{1,2}| \leq Ch^2 \log h^{-1}$, for sufficiently small h.

It is straightforward to extend the above estimate to all entries in the matrix $M_n - \tilde{M}_n$. This gives the desired $\|M_n - \tilde{M}_n\|_\infty \leq Ch^2 \log h^{-1}$. □

This error for the proposed, approximated matrix $\tilde{M}_n$ is asymptotically smaller than the $O(h)$ error of the IB Method for points within a distance $O(h)$ of the immersed boundary [21]. Thus, the approximated matrix $\tilde{M}_n$ can be use without any deterioration of the overall accuracy of the IB Method.

We calculate numerically both matrices and then compute the maximum norm between their difference for the standard example of a relaxing elliptical bubble as described in Section 4.2.1. A log-log graph of the norm with respect to h is given in Fig. 4.1. Here, we have fixed $\Delta t = h$ and $N_B = 2N$. We see that the error is approximately $O(h^2)$. More importantly we can numerically verify our

approximation by looking at residuals. If we solve

$$\mathbf{X}^{n+1} = \tilde{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n, \tag{4.28}$$

for $\mathbf{X}^{n+1}$ we calculate the residual

$$\mathbf{r} = \mathbf{b}^n - \mathbf{X}^{n+1} + \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}). \tag{4.29}$$

Generally $\|\mathbf{r}\| < .001$ in our numerical experiments, which is typically on the order of our stopping criteria for our iterators. We note that this residual is highly dependent on the choice of delta function δ_h. Delta functions which are less sensitive to translation provide better residuals. We note as well that if smaller residuals are desired it is possible to modify the methods presented in this paper by incorporating full fluid evaluations with $\mathcal{M}_n$ when calculating residuals used within the iterators we present. We have also had success in constructing more elaborate, higher dimensional lookup tables to use in the construction of $\tilde{M}_n$ which can be used to make the introduced error as small as desired with minimal additional CPU cost.

4.1.5 Error estimate for approximation of the matrix representation of $\mathcal{M}_n$ in 3D

In the previous section it was shown that in 2D the error $|(G_h(\mathbf{x},\mathbf{y}) - G_h(\mathbf{x}-\mathbf{y}))_{ij}|$ is smaller than $O(h)$. We sketch here the extension of this result to the 3D case.

First, we require a Green's function g_h associated with the operator $\mathcal{L}_h$. G_h, the Green's function relating the influence of one fiber point on another, can be

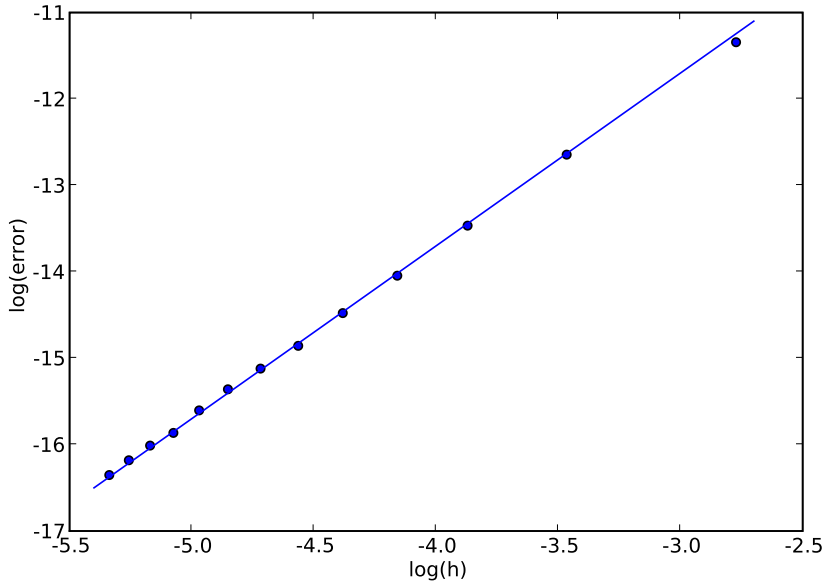


Figure 4.1: Log-log plot of $\left\|M_n - \tilde{M}_n\right\|_\infty$ with $\Delta t = h$. The continuous line is a fit with a line of slope 2.

written as a summation involving g_h. Bounds on g_h will lead to bounds on the translation error associated with G_h.

We define three force fields

$$\mathbf{f}_i(\mathbf{x})_j = \begin{cases} 1/h^3 & \text{if } \mathbf{x} = \mathbf{0}, i = j \\ 0 & \text{otherwise,} \end{cases} \tag{4.30}$$

for $i, j = 1, 2, 3$. These force fields correspond to a suitably scaled point force along a cardinal direction centered at the origin. With these we define the 3×3 tensor field

$$g_h(\mathbf{x})_{ij} = \mathcal{L}_h(\mathbf{f}_i)_j. \tag{4.31}$$

Following Lemma 4.1.1 we may bound the components of g_h via

$$\begin{aligned} \|(g_h)_{ij}\|_\infty &\leq \sum \frac{1}{1 + 16\nu\Delta t|\mathbf{k}|^2} \\ &\leq C \int_0^{\frac{\sqrt{2}}{2}N} \frac{1}{1 + 16\nu\Delta t r^2} r^2 \, dr < C\frac{N}{\Delta t}, \end{aligned} \tag{4.32}$$

where the sum is over the set $\{\mathbf{k} \in \mathbb{Z}^3 \mid |\mathbf{k}_i| < N/2\}$, and the integral is obtained from a spherical coordinate transformation.

Consider now two fiber points, located at $\mathbf{x}$ and $\mathbf{y}$. We wish to bound the tensor components of the error $G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y})$. By Theorem 4.1.1 we have

$$|(G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y}))_{ij}| < \frac{2(\Delta t)^2}{\rho} \|(g_h)_{ij}\|_\infty h_B. \tag{4.33}$$

Assuming $\Delta t \propto h$ and $h_B \propto h^2$, then (4.33) combined with (4.32) gives us the bound

$$|G_h(\mathbf{x}, \mathbf{y}) - G_h(\mathbf{x} - \mathbf{y})|_{ij} < Ch^2, \tag{4.34}$$

which is smaller than the $O(h)$ error of the IB Method.

4.1.6 Additional considerations and optimizations concerning $\tilde{M}_n$

There are a few additional optimizations possible when constructing $\tilde{M}_n$, the matrix representation of $\mathcal{M}_n$. Note first that on a square domain Ω we have via symmetry that

$$G_h(x, y)_{00} = G_h(y, x)_{11}, \tag{4.35}$$

$$G_h(x, y)_{01} = G_h(y, x)_{10}. \tag{4.36}$$

On a cubic domain we have similar symmetries, such as

$$G_h(x, y, z)_{00} = G_h(y, x, z)_{11} = G_h(z, y, x)_{22}, \tag{4.37}$$

$$G_h(x, y, z)_{01} = G_h(y, x, z)_{10} = G_h(x, z, y)_{02}. \tag{4.38}$$

Using these symmetries we can reduce the size of the precomputed lookup table for G_h. For rectangular or otherwise irregular grids these symmetries do not hold, we do however have other symmetries; most importantly $G_h(\mathbf{x}) = G_h(-\mathbf{x})$. This

symmetry implies that the matrix of $\mathcal{M}_n$ is symmetric. This observation reduces the cost of computing the matrix roughly by half.

The cost could be reduced further if interpolation were not used between grid points when looking up $G_h(\mathbf{z})$ for arbitrary $\mathbf{z}$. Unfortunately, the resulting error introduced in the simulation would be significant. A simple compromise is to use linear interpolation when calculating $G_h(\mathbf{z})$ if $\|\mathbf{z}\|_2$ is small and direct lookup otherwise. This is inexpensive, with only $O(N_B)$ interpolations needed, because most fiber points are distant from each other, as well as accurate, because G_h decays rapidly away from the origin.

Alternatively, we could employ an adaptive mesh on which to construct the lookup table for G_h, taking a dense grid around the origin where most of the structure lies. This would allow us to use direct lookup for all entries of $\mathcal{M}_n$ without much loss of accuracy. We do not pursue these strategies here and only utilize the symmetry of the matrix of $\mathcal{M}_n$ to expedite the construction. This shortcut was used in calculating the costs given in Table 4.2.

4.2 Multigrid

With an approximate matrix form of $\mathcal{M}_n$ in hand we can now move beyond Krylov methods and consider alternative iterative methods to solve our implicit

system (3.15). In particular we can now easily implement Gauss-Seidel. In the case that $\mathcal{A}_{h_B}$ is linear and negative-semidefinite we define $B = I - \mathcal{M}_n\mathcal{A}_{h_B}$, which itself is positive-definite. Our system (3.15) is then just the matrix problem

$$B\mathbf{X}^{n+1} = \mathbf{b}^n. \tag{4.39}$$

This problem is amenable to Gauss-Seidel and Successive over-relaxation (SOR). Unfortunately the number of iterations required to achieve adequate convergence is typically prohibitive. However, SOR functions well at reducing high frequencies errors, and is thus suitable as a smoother in a multigrid. In this section we overview briefly a simple multigrid implementation. We then show the results of numerical experiments detailing the efficiencies of the multigrid method.

For simplicity in presentation, we consider a simple elliptical interface immersed in a 2D fluid. For details of the multigrid method applied to a more complicated geometry see Section 6.1.

Suppose that $N_B = 2^m$ for some natural number m. We can then form a collection of Lagrangian grids $\mathcal{G}^l$, $l = 1, \dots m$, with number of nodes equal to $2^m, 2^{m-1}, \dots, 2, 1$. Our original grid corresponds to the first level, $l = 1$, with coarser grids coming afterward. For a given level l of the multigrid hierarchy, the prolongation operator $\mathcal{P}_l$ takes a fiber $\mathbf{X}_{l+1}$ at level $l+1$ and adds a node between each pair of consecutive nodes equidistant between each. As a matrix, $\mathcal{P}_l$ has

dimensions $2 \cdot 2^{m-l+1} \times 2 \cdot 2^{m-l}$ and has the form

$$\mathcal{P}_l = \begin{pmatrix} 1 & 0 & 0 & \\ .5 & .5 & 0 & \dots \\ 0 & 1 & 0 & \\ 0 & .5 & .5 & \\ & \vdots & & \ddots \end{pmatrix}. \tag{4.40}$$

Restriction operators are taken to be the transposes of prolongation operators. Calling $\mathcal{K}_1 = (I - \mathcal{M}_n \mathcal{A}_{h_B})$ we define recursively $\mathcal{K}_l = \mathcal{P}_{l-1}^T \mathcal{K}_{l-1} \mathcal{P}_{l-1}$. All together, a single restriction, prolongation, and smoothing step looks like

- Given a guess $\mathbf{X}_l$ to the linear problem

$$\mathcal{K}_l \mathbf{X}_l = \mathbf{b}_l \tag{4.41}$$

  calculate the residual and restrict it to the next lowest level

$$\tilde{\mathbf{r}}_{l+1} = \mathcal{P}_l^T (\mathbf{b}_l - \mathcal{K}_l \mathbf{X}_l) \tag{4.42}$$

- Find the correction on the coarse grid by solving or approximating

$$\mathcal{K}_{l+1} \mathbf{X}_{l+1} = \tilde{\mathbf{r}}_{l+1} \tag{4.43}$$

- Correct our initial guess

$$\mathbf{X}_l \leftarrow \mathbf{X}_l + \mathcal{P}_l \mathbf{X}_{l+1} \tag{4.44}$$

- Smooth the high frequency errors in $\mathbf{X}_l$ via SOR.

The number of iterations needed depends more on the desired degree of accuracy than a requirement for stability. Typically a single iteration of a Full Multigrid Step with one V-cycle per level is sufficient to maintain stability.

4.2.1 Case I: An initially elliptical drop with linear $\mathcal{A}_{h_B}$

We consider as our first example what has been the canonical test for methods intended to remove the severe stiffness of the IB method. This is the case of a closed, continuous membrane $\mathbf{X}(s,t)$ with a force distribution given by $\mathcal{A}(\mathbf{X}) = \sigma\mathbf{X}_{ss}$, where σ is a (large) constant. We take our domain as $\Omega = [0,1]\times[0,1]$ with periodic boundary conditions and we fix $\mu = \rho = 1$. Initially, we have an elliptical drop and zero velocity and the drop relaxes toward the equilibrium configuration. Figure 4.2 shows the initial and final configurations of the interface.

We discretize $\mathcal{A}(\mathbf{X})$ as

$$(\mathcal{A}_{h_B}\mathbf{X})_i = \frac{1}{h_B^2}\left(\mathbf{X}_{i+1} - 2\mathbf{X}_i + \mathbf{X}_{i-1}\right), \tag{4.45}$$

where we have omitted the parentheses in the discrete force operator $\mathcal{A}_{h_B}$ to emphasize that it is linear.

We set the elasticity constant $\sigma = 10^5$ and take $N_B = 2N$. The initial configuration of the fiber is an ellipse given by

$$\mathbf{X}(s,0) = (0.5 + 0.3\cos(2\pi s h_B), 0.5 + 0.2\sin(2\pi s h_B)), \tag{4.46}$$

for $s \in [0,1]$. As time elapses, the interfacial tension drives the ellipse toward a circular (cylindrical) configuration. The velocity of the fiber can reach roughly 500 units before it slows down toward equilibrium. Taking a standard length to be the geometric mean of our radii, 0.245, we calculate the Reynolds number of our fluid to be approximately 100.

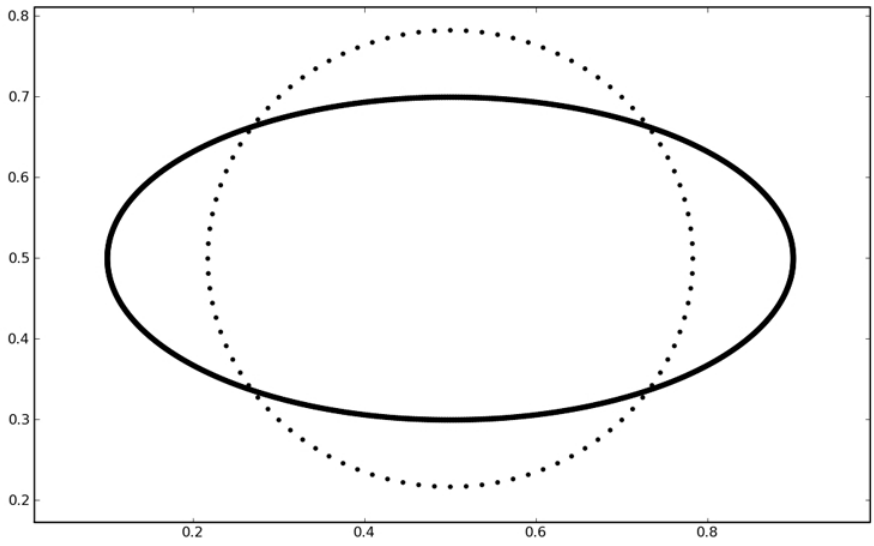


Figure 4.2: Initial configuration of fiber in bold and final rest configuration in dotted line.

4.2.2 Case I: Numerical results

Solving our implicit system (3.1)-(3.3) via the multigrid described in Section 4.2 has five primary costs.

1. First, when constructing the implicit system (3.15) we must calculate $\mathbf{b}^n$ via a fluid solve, with a cost of roughly one FE/BE.

2. Second, the cost of constructing the approximate matrix form of M_n is roughly one half FE/BE.

3. Third is the cost of initializing the multigrid solver, i.e. calculating the coarse representations of M_n on the various levels of our grid. Use of sparse data structures is critical here to maintain $O(N_B^2)$ cost.

4. Fourth, is the cost of our iterations to solve the linear system. Because matrix-vector multiplication is significantly cheaper than FE/BE (see Table 4.1) a full multigrid step can be much more economical than one FE/BE and this advantage grows as N increases.

5. The final cost is in computing $\mathbf{u}^{n+1}$ once $\mathbf{X}^{n+1}$ is known. This step involves another fluid solve with a cost of roughly one FE/BE.

We will see that the sum of these costs for the semi-implicit method results in a single time-step that costs approximately 4 FE/BE. However, the stability re-

Table 4.3: Elliptical drop relaxation for Navier Stokes. The average CPU time per time-step and the total CPU time up to a simulation time of $T = 0.005$ is given. Δt is the time-step taken and is the maximum allowed while maintaining stability.

	Explicit			Implicit		
N	Δt	Average	Total	Δt	Average	Total
128	$1.95 \cdot 10^{-6}$	0.03	73.35	$1.17 \cdot 10^{-4}$	0.10	4.34
256	$9.76 \cdot 10^{-7}$	0.14	730.21	$7.81 \cdot 10^{-5}$	0.55	35.06
384	$6.51 \cdot 10^{-7}$	0.34	2613.19	$5.21 \cdot 10^{-5}$	1.24	118.98
512	$4.87 \cdot 10^{-7}$	0.63	6491.95	$3.91 \cdot 10^{-5}$	2.34	299.10

straint on Δt for the FE/BE scheme for a stiff problem like this one is orders of magnitude smaller than that required by the semi-implicit discretization. Thus, our proposed approach yields a much superior computational strategy.

As a reference for comparison, for fiber-explicit simulations we use the FE/BE scheme with Δt as large as stability permits, which we find to be approximately $\Delta t = 0.00025h$. For our implicit scheme we make use of a linear multigrid with stopping criteria of $\left\|\mathbf{r}^k\right\|_\infty < 30\Delta t$, where $\mathbf{r}^k$ is the residual $\mathbf{b}^n - (I - M_n A_{h_B})\mathbf{X}^{n+1,k}$ at the kth iterate and $\|\cdot\|_\infty$ is the sup norm. Here Δt is chosen to comply with the CFL condition as the convection term is treated explicitly. With a conservative estimate for the maximum velocity, the time-step is given approximately by $\Delta t = 0.02h$.

The results of multiple simulations with varying N for the explicit and implicit methods are given in Table 4.3. The rows in the table correspond to identical

simulation runs with different N. The columns under the title Explicit relate data from the FE/BE simulations, whereas the Implicit columns relate data from the implicit simulations. The two columns marked Average give the average CPU time of a single timestep. The columns marked Total give total CPU time for the entire simulation, up to a simulation time of $T = 0.005$.

We see in Table 4.3 that the implicit scheme costs approximately 4 times as much per time-step than FE/BE. However, because Δt can be taken up to the CFL restraint, the scheme is roughly 20 times faster than the explicit scheme in terms of total CPU time. Here, the performance of the semi-implicit approach is limited by the presence of large convection (relative to viscous dissipation). Of course, one could employ a suitable implicit discretization of the convection terms to remove the CFL constraint. Effective options in this context have been proposed by Mori and Peskin [22] and Hou and Shi [8]. We do not pursue this here as our intent is to focus on the treatment of the tension forces. Alternatively, if we consider Stokes flow the semi-implicit method is no longer limited by a CFL restriction and we can take arbitrarily sized time-steps while maintaining stability. We compare the performance of the two methods for Stokes flow in Table 4.4.

In these simulations we have increased the total simulation time to $T = .05$. This simulation time is enough that the explicit methods may require unreasonable amounts of CPU time. We approximate the total CPU time for these longer

Table 4.4: Ellipse relaxation for Stokes flows. The average CPU time per timestep and the total CPU time up to a simulation time of $T = 0.05$ is given. Δt is the timestep taken. For the explicit scheme this is the maximum allowed while maintaining stability. The implicit scheme is unconditionally stable; the timestep taken is held constant as N varies. * denotes an extrapolated value.

	Explicit			Implicit		
N	Δt	Average	Total	Δt	Average	Total
128	$1.95 \cdot 10^{-6}$	0.03	683.96	0.001	0.09	9.34
256	$9.76 \cdot 10^{-7}$	0.15	7410.50	0.001	0.53	53.08
384	$6.51 \cdot 10^{-7}$	0.35	26555.30*	0.001	1.24	123.86
512	$4.87 \cdot 10^{-7}$	0.71	72669.41*	0.001	2.32	232.05

simulations by calculating the average CPU time of a single timestep and multiplying by the number of timesteps the explicit simulation would require to complete. We use an asterisk here and in following tables to denote values based on these extrapolated values.

While Δt for our implicit simulation can be chosen with disregard to stability, care must still be taken to ensure accuracy. Here we only require the loose condition that $\Delta t < h$. This is sufficient to preserve the qualitative behavior of our immersed structure. We note that there are implicit discretizations of the IB method with higher order accuracy in time, see for example [23]. Many of these methods require very little additional work to implement and do not require much additional CPU time per timestep. These methods may allow IB simulations to maintain high accuracy while exploiting the large timesteps allowed by an implicit discretization.

4.2.3 Case II: An initially elliptical drop with nonlinear $\mathcal{A}_{h_B}$

We consider now the case of the simple, initially elliptical membrane with a nonlinear force and detail how to adapt the method from the previous linear case.

We follow Mori and Peskin in [22] for the setup of this nonlinear $\mathcal{A}_{h_B}$ example. At the continuous level, the fiber force distribution is given by

$$\mathbf{F} = \frac{\partial}{\partial s}(T\mathbf{t}) \tag{4.47}$$

where $\mathbf{t}(s,t)$ is the tangential unit vector to $\mathbf{X}(s,t)$ and $T(s,t)$ is the tension given by

$$T(s,t) = \left|\frac{\partial \mathbf{X}}{\partial s}(s,t)\right| + \left|\frac{\partial \mathbf{X}}{\partial s}(s,t)\right|^2. \tag{4.48}$$

To discretize this interfacial force we introduce the following difference operators for a function f defined over $\mathcal{G}_B$:

$$D_s^+ f(s) = \frac{f(s+1) - f(s)}{h_B}, \tag{4.49}$$

$$D_s^- f(s) = \frac{f(s) - f(s-1)}{h_B}. \tag{4.50}$$

We then define the discrete force distribution as

$$\mathbf{F} = D_s^+ \left(\left(\left|D_s^- \mathbf{X}\right| + \left|D_s^- \mathbf{X}\right|^2 \right) \frac{D_s^- \mathbf{X}}{|D_s^- \mathbf{X}|} \right). \tag{4.51}$$

Severe stiffness in this model is manifested when the interfacial elastic force is much larger than the viscous forces. Following Mori and Peskin [22] we set $\rho = 1$

and either $\mu = .05$ or $\mu = .005$. As before the domain is $\Omega = [0,1] \times [0,1]$ which we discretize as Ω_h, an $N \times N$ uniform Eulerian grid. As in the linear case, the initial velocity of the fluid is zero everywhere. The fiber's initial configuration is given by

$$\mathbf{X}(s,0) = (\frac{1}{2} + \frac{1}{3}\cos(2\pi s h_B), \frac{1}{2} + \frac{1}{4}\sin(2\pi s h_B)). \tag{4.52}$$

With our implicit strategy we attempt to solve at each time step the *nonlinear* system (3.15)

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n,$$

where $\mathcal{M}_n$ and $\mathbf{b}^n$ are as in the linear case. We could simply approximate $\mathcal{A}_{h_B}$ linearly and solve the resulting linear system. This is easily accomplished because the Jacobian J of $\mathcal{A}_{h_B}$ is negative semidefinite, hence $I - \mathcal{M}_n J$ is positive definite. Such linear approximation is simple and reasonably robust but not as stable as the implicit scheme with full nonlinear term. This linear approximation is the method employed by Mori and Peskin [22] for their first order scheme. However, with our economical computational approach we can afford the extra stability (while maintaining accuracy) by solving the nonlinear system (3.15) via Newton iterations. Specifically, at each time-step we perform Newton iterations until the sup norm of the residual is less than 10^{-4}. This typically requires only 2 or 3 iterations. Note that a single iteration per timestep would be identical to the linear approximation used by Mori and Peskin. We expedite further the computation by

approximating the solution to the linear system at each one of Newton's iteration with 3 Multigrid cycles.

4.2.4 Case II: Numerical results

We choose this nonlinear test problem following Mori and Peskin [22] to have their results as a reference. However, there is a slight difference in the model in addition to the different discretizations employed; Mori and Peskin use an immersed fiber with finite mass while our own fiber is neutrally buoyant. We present the results from our numerical experiments in a format similar to that presented in [22] to facilitate comparisons.

Let N_T be the number of timesteps taken, with total simulation time fixed at 1. We fix $N_B = 2N$ as before. Table 4.5, for $\mu = 0.05$, and Table 4.6, for $\mu = 0.005$, summarize the total computational cost in units of FE/BE's. The total CPU time for the same cases is displayed in Table 4.7 and Table 4.8, respectively. In all four tables the columns under FE/BE hold the explicit result while the columns under $N_T = 8$ and $N_T = 16$ hold the implicit results, with N_T as specified.

As noted in [22], the semi-implicit method becomes more efficient compared to the explicit (FE/BE) scheme as N increases and μ decreases (more stiffness). For $N_T = 8$ and $N = 512$ the proposed semi-implicit strategy is about 45 times faster than the explicit approach in the case of $\mu = 0.05$ and about 90 times faster

Table 4.5: Total CPU cost for the nonlinear ellipse model with $\mu = 0.05$. Values given are total CPU time divided by average CPU time of a single FE/BE timestep for the given N.

N	FE/BE	$N_T = 8$	$N_T = 16$
64	166	38.2	75.5
128	500	49.7	89.7
256	1333	53.0	97.2
512	2666	59.3	92.6

Table 4.6: Total CPU cost for the nonlinear ellipse model with $\mu = 0.005$. Values given are total CPU time divided by average CPU time of a single FE/BE timestep for the given N.

N	FE/BE	$N_T = 8$	$N_T = 16$
64	333	83.5	112.4
128	1000	74.8	113.1
256	2666	71.7	112.6
512	5333	59.2	101.9

for $\mu = 0.005$. In contrast, the fully implicit approach in [22] gives cost ratios in the range $12 - 16$ for the same parameters.

4.3 Conclusion

In this chapter we developed a novel methodology for efficiently solving implicit discretizations of the IB Method, namely the matrix method and the resulting multigrid. We showed that for certain simple applications we can efficiently remove the stiffness in the IB Method and drastically decrease the compu-

Table 4.7: Total CPU time for the nonlinear ellipse model with $\mu = 0.05$.

N	FE/BE	$N_T = 8$	$N_T = 16$
64	2.94	0.67	1.33
128	20.89	2.08	3.75
256	227.85	9.05	16.59
512	1930.62	42.94	67.02

Table 4.8: Total CPU time for the nonlinear ellipse model with $\mu = 0.005$.

N	FE/BE	$N_T = 8$	$N_T = 16$
64	5.31	1.33	1.79
128	42.06	3.14	4.75
256	455.11	12.24	19.22
512	3823.40	42.45	73.10

tational cost of simulations. In Chapter 6, we will show that the matrix method is amenable to much more complex problems.

4.3.1 Strengths of the matrix method

When should the matrix method be used? Certainly in the case of a 2D fluid there are many potential applications. If $N_B \ll N^2$ then a large savings can be had by forming an approximate matrix representation of $\mathcal{M}_n$ at each timestep. This savings is present regardless of whether the implicit system is amenable to multigrid.

When applicable, however, multigrid is a powerful solver. Provided that $\mathcal{A}_{h_B}$ or its Jacobian matrix is negative-semidefinite then we can efficiently solve (3.15)

via multigrid, with the multigrid used inside a Newton iteration in the case that $\mathcal{A}_{h_B}$ is nonlinear. In Chapter 6 we will extend the methodology for a class of cases where the Jacobian is non-definite.

4.3.2 Limitations of the matrix method

While there are many advantages to the matrix method in 2D, the advantages disappear in a 3D application. If our immersed structure is a 2D membrane we would have $N_B \sim N^2$ leading to a cost of $O(N^4)$ for a matrix-vector multiplication using the matrix while obtaining the same product via spreading-fluid solver-interpolation would be $O(N^3 \log N)$. Indeed, some 3D applications may use substantially more fiber points. The heart model proposed in [15] employs nearly $N_B \sim N^3$ fiber points. For such applications we will develop a different methodology, presented in Chapter 5.

We stress that it is very difficult to construct an efficient multigrid without the matrix method. Gauss-Seidel is prohibitively expensive, and alternative smoothers such as Jacobi iterations do not perform well. If a multigrid is to be employed then we will need to find an efficient means of calculating the Gauss-Seidel iteration. The treecode developed in Chapter 5 allows for exactly this.

Chapter 5

The Treecode Method

5.1 Multipole summations and treecodes

Treecodes are efficient ways to evaluate certain sums. Suppose we have an expression of the form

$$\sum_{j=1}^{N_B} \phi(\mathbf{X}_j, \mathbf{X}_i)\mathbf{F}_j, \tag{5.1}$$

where $\{\mathbf{F}_j\}_{j=1}^{N_B}$ is a collection of forces, and ϕ is some tensor valued potential. Expression (5.1) is referred to as a multipole summation. Evaluating (5.1) for all $1 \leq i \leq N_B$ directly requires $O(N_B^2)$ operations. However, treecodes can reduce the overall computational cost to $O(N_B \log N_B)$, provided ϕ is sufficiently regular.

Given the function G_h, as constructed in Chapter 4, we can cast our fluid evaluations as multipole summations of the form

$$(\mathcal{M}_n\mathbf{F})_i = \sum_{0 \leq j \leq N_B} G_h(\mathbf{X}_j - \mathbf{X}_i)\mathbf{F}_j. \tag{5.2}$$

Calculating this sum directly is equivalent to the matrix method proposed in Chapter 4, and requires $O(N^4)$ operations when $N_B \sim N^2$. The goal now is to apply a treecode algorithm to accelerate the evaluation of this sum, ideally reducing the cost from $O(N_B^2)$ to $O(N_B \log N_B)$. In the following two subsections we overview the basics of treecodes. We will elaborate in Section 5.1.3 on the far field expansions of G_h that enable a treecode strategy to perform well for our particular problem. Following that we will detail two applications of the treecode method in Section 5.4.

5.1.1 Overview

For those unfamiliar with treecodes we briefly review the main ideas here. We recommend an alternative exposition by Li, Johnston, and Krasny in [12]. We also present a more detailed and technical overview of the treecode algorithm in Section 5.2.

The general strategy is to make use of far field expansions of G_h to compress the outgoing effect of clusters of fiber points. Assume we have two subsets of our domain Ω, Ω_{in} and Ω_{out}, such that G_h has a valid expansion in $\Omega_{in} \times \Omega_{out}$ given by

$$G_h(\mathbf{x}-\mathbf{y}) \approx \sum_{k=1}^{p} A_k(\mathbf{x}) B_k(\mathbf{y}), \qquad \text{for } \mathbf{x} \in \Omega_{in} \text{ and } \mathbf{y} \in \Omega_{out}. \tag{5.3}$$

Ω_{out} serves to restrict the location of our pole, the *outgoing* influence, while Ω_{in} serves to restrict the point of evaluation, the *incoming* position. Our expansion terms $\{A_k\}_{k=1}^p$ and $\{B_k\}_{k=1}^p$ are collections of 3×3 matrix (tensor) fields defined on Ω_{in} and Ω_{out} respectively, with multiplication between two tensors defined componentwise, and multiplication between a tensor and a vector defined via the usual matrix-vector multiplication. If we further define

$$B_{in} = \{1 \leq i \leq N_B | \mathbf{X}_i \in \Omega_{in}\}, \tag{5.4}$$

$$B_{out} = \{1 \leq i \leq N_B | \mathbf{X}_i \in \Omega_{out}\}, \tag{5.5}$$

then we can consider the subproblem of calculating the influence of all the fiber points in Ω_{out} on the fiber points in Ω_{in}:

$$\sum_{j \in B_{out}} G_h(\mathbf{X}_i - \mathbf{X}_j)\mathbf{F}_j, \qquad \text{for } i \in B_{in}. \tag{5.6}$$

Calculating this summation for all i in B_{in} requires $O(|B_{in}| \cdot |B_{out}|)$ operations, where $|\cdot|$ denotes the number of elements in a set. We seek to reduce this cost. Substituting (5.3) in (5.6) yields

$$\begin{aligned} \sum_{j \in B_{out}} G_h(\mathbf{X}_j - \mathbf{X}_i)\mathbf{F}_j &\approx \sum_{j \in B_{out}} \left(\sum_{k=1}^{p} A_k(\mathbf{X}_i) B_k(\mathbf{X}_j) \right) \mathbf{F}_j \\ &= \mathbf{E}^T \sum_{k=1}^{p} A_k(\mathbf{X}_i) \sum_{j \in B_{out}} B_k(\mathbf{X}_j) \tilde{F}_j. \end{aligned} \tag{5.7}$$

Here $\tilde{F}_j$ is a 3×3 matrix (tensor) where $(\tilde{F}_j)_{ab} = (\mathbf{F}_j)_b$ and $\mathbf{E}$ is a 3-vector whose components are all one. The additional complication of defining $\tilde{F}_j$ arises because

the product of A_k and B_k in (5.3) is understood component-wise. The vector $\mathbf{E}$ serves to collapse the final sum into a vector.

The summation over B_{out} in (5.7) may be calculated independently of i. Thus, given

$$H_k \equiv \sum_{j \in B_{out}} B_k(\mathbf{X}_j)\tilde{F}_j, \tag{5.8}$$

we can efficiently calculate (5.6) via

$$\mathbf{E}^T \sum_{k=1}^{p} A_k(\mathbf{X}_i)H_k, \qquad \text{for } i \in B_{in}. \tag{5.9}$$

Using (5.9) to compute (5.6) requires $O(p|B_{in}| + |B_{out}|)$ operations, which may be substantially fewer than $O(|B_{in}| \cdot |B_{out}|)$ if $p \ll |B_{out}|$. This compression is one of the main ingredients of a treecode. The remaining difficulty is that Ω_{in} and Ω_{out} generally do not cover our entire domain. There may be fiber points $\mathbf{X}_i$ and $\mathbf{X}_j$ where i, j do not belong to B_{out}, B_{in}, hence the interaction between $\mathbf{X}_i$ and $\mathbf{X}_j$ would not be accounted for in (5.6).

To remedy this we must choose multiple pairs of $(\Omega_{in}, \Omega_{out})$ so that every ordered pair of fiber points $(\mathbf{X}_i, \mathbf{X}_j)$ is represented exactly once. This is essentially an organizational problem, and the standard procedure is to use binary space partitioning. In 3D this is known as an octree whereas in 2D it is called a quadtree.

5.1.2 The octree

An octree is constructed by successively subdividing our domain $\Omega = [0,1]^3$ into smaller cubic domains called *panels*. Starting with the domain itself, called the *Root Panel*, we divide it into eight equal octants called child panels (of the root) then recursively we define the eight children of each of those child panels and so on. This subdivision process is stopped for a panel which has fewer that an (arbitrarily set) minimum number of fiber points (10 here).

The totality of the Root Panel and all of its branches is collectively known as the octree. Visualizing an octree in 3D is difficult. For demonstrative purposes we present in Figure 5.1 a drawing of a quadtree in 2D.

Given a panel P we refer to its domain as Ω_{out}^P. A point $\mathbf{x}$ is *well-separated* from a panel P if its distance to the center of the panel is at least 3/2 the size of the panel [see (5.23)]. We refer to the domain of all points well-separated from P as Ω_{in}^P. We assume that points in Ω_{in}^P are sufficiently far from P that a multipole expansion could be used to calculate the effect of the *outgoing* influence of all the fiber points in P. This assumption is equivalent to assuming that we can expand G_h over $\Omega_{in}^P \times \Omega_{out}^P$, arriving at two collections of coefficient functions $\{A_k^P\}_{k=1}^p$ and $\{B_k^P\}_{k=1}^p$.

Suppose now that we are given a fiber force $\mathbf{F}$ and wish to evaluate its influence at points $\mathbf{x}$ in the fluid domain. For each $\mathbf{x}$ this influence is given by

$$\sum_{1 \le j \le N_B} G_h(\mathbf{x} - \mathbf{X}_j)\mathbf{F}_j. \tag{5.10}$$

To evaluate this efficiently, we first loop over each panel P and calculate the far field expansion of all the poles located in P. The k-th term of this expansion is given by

$$H_k^P \equiv \sum_{j \in B_{out}^P} B_k^P(\mathbf{X}_j)\tilde{F}_j. \tag{5.11}$$

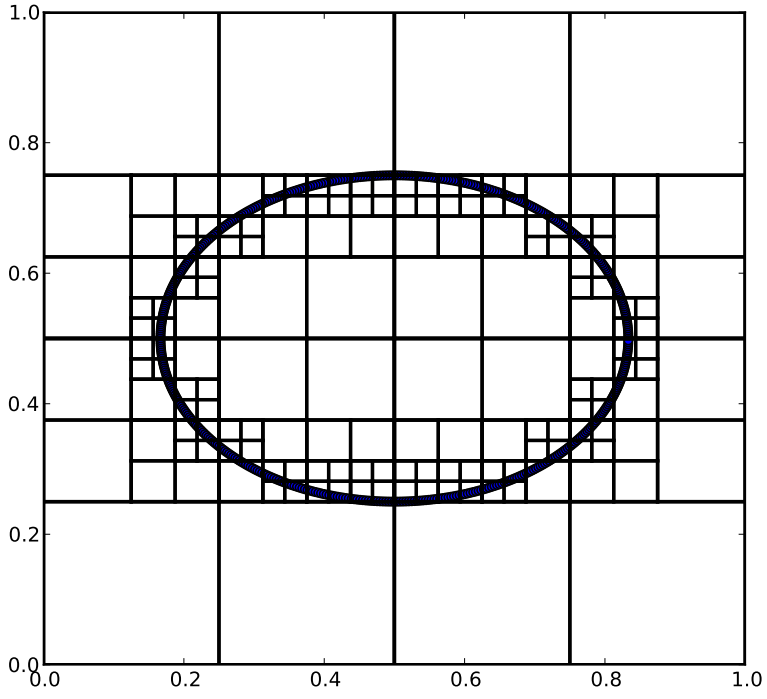


Figure 5.1: An ellipse shaped fiber $\mathbf{X}$, with a quadtree decomposition of the space $[0,1] \times [0,1]$ containing it.

If $\mathbf{x}$ is well separated from P, then the incoming effect on $\mathbf{x}$ from P is given by

$$\mathbf{E}^T \sum_{k=1}^{p} A_k^P(\mathbf{X}_i) H_k^P. \tag{5.12}$$

We can now evaluate the entire influence of $\mathbf{F}$ on a point $\mathbf{x}$, recursively. We start with the panel $P = \mathit{Root}$. If $\mathbf{x}$ is well separated from P then we evaluate the influence of P on $\mathbf{x}$ via (5.12). Otherwise, if P has children we recursively apply this process to those child panels. If instead P is childless, we directly evaluate the influence of every fiber point in P on $\mathbf{x}$, using (5.6). Summing all these influences from every branch of the recursion provides the desired total influence on $\mathbf{x}$.

Note that (5.11) only needs to be calculated once per panel, and may be reused for calculating the influence at multiple points $\mathbf{x}$. Because each fiber point $\mathbf{X}_i$ is contained in at most $\log N$ panels, the total cost of computing H_k^P for all panels P is at most $O(N_B \log N_B)$. Calculating the influence at a given $\mathbf{x}$ involves at most $O(\log N)$ panels, and hence the cost is at most $O(p \log N)$. The total cost of evaluating $\mathcal{M}_n\mathbf{F}$ is thus $O(pN_B \log N_B)$.

A detailed description of the octree creation and point evaluation is presented in 5.2.

5.1.3 Expansions

In order to make use of a treecode we must be able to find expansions (5.3) of G_h associated with particular pairs $(\Omega_{in}, \Omega_{out})$. Here, G_h is a summation of

discrete stokeslets. In free space, the continuous stokeslet has convenient analytic expansions useful for fast summation, see for instance [31]. The same is not true for the discrete stokeslet corresponding to a periodic domain, and much less so for the particular summation of stokeslets that yields G_h. Fortunately, an analytic expansion is not indispensable in practice. What we seek are collections of tensor valued function $\{A_k\}_{k=1}^{\infty}$ and $\{B_k\}_{k=1}^{\infty}$ defined on Ω such that

$$G_h(\mathbf{x}-\mathbf{y}) = \sum_{k=1}^{\infty} A_k(\mathbf{x})B_k(\mathbf{y}), \tag{5.13}$$

and, moreover, such that truncating the above expansion to p terms yields an adequate approximation, provided that $\mathbf{x}$ and $\mathbf{y}$ are well separated in some sense. That is, given two disjoint subsets Ω_{in} and Ω_{out} of our domain, we hope that for some small value of p and for a specified tolerance ϵ that

$$\left\| \left(G_h(\mathbf{x}-\mathbf{y}) - \sum_{k=1}^{p} A_k(\mathbf{x})B_k(\mathbf{y}) \right) \mathbf{f} \right\|_2 < \epsilon \left\|\mathbf{f}\right\|_2, \tag{5.14}$$

for all $\mathbf{f} \in \mathbb{R}^3$, $\mathbf{x} \in \Omega_{in}$, and $\mathbf{y} \in \Omega_{out}$.

For a given p, finding the optimal expansion that allows for the smallest ϵ that satisfies (5.14) for all $\mathbf{f}, \mathbf{x}, \mathbf{y}$ is an open question, and is likely computationally intractable.

We solve instead a least squares problem. We will search for the individual components of our tensors separately. Looking at each ab components, for $a, b = x, y, z$, we seek the expansion coefficients $\{(A_k)_{ab}\}_{k=1}^{p}$ and $\{(B_k)_{ab}\}_{k=1}^{p}$ such

that we minimize the L^2-norm of the difference between G_h and our approximate expansion. That is, we seek to minimize

$$\left\| \left(G_h - \sum_{k=1}^{p} A_k B_k \right)_{ab} \right\|_2^2 \equiv \iint \left(G_h(\mathbf{x}-\mathbf{y}) - \sum_{k=1}^{p} A_k(\mathbf{x}) B_k(\mathbf{y}) \right)_{ab}^2 d\mathbf{x} d\mathbf{y}, \quad (5.15)$$

where $A_k B_k$ is understood to be a function over $\Omega_{in} \times \Omega_{out}$, and the integrals are taken over $\mathbf{x} \in \Omega_{in}$ and $\mathbf{y} \in \Omega_{out}$. We approximate the integral in (5.15) as

$$\sum \left(G_h(\mathbf{x}-\mathbf{y}) - \sum_{k=1}^{p} A_k(\mathbf{x}) B_k(\mathbf{y}) \right)_{ab}^2 h^6, \quad (5.16)$$

where the outer sum is taken over the Eulerian grid points in our subsets, $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h$ and $\mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. Note that this minimization problem is decoupled with respect to the components of our tensors. We may thus consider the task of minimizing (5.16) as nine separate minimization problems.

It is useful to view our problem as a statement about matrices. To do this we first write $(G_h)_{ab}$ as an $N^3 \times N^3$ matrix, which we refer to as $\tilde{G}$ and which is given by the relation

$$\tilde{G}_{ij} = G_h((j_0, j_1, j_2)h - (i_0, i_1, i_2)h), \quad (5.17)$$

where $i = i_0 + i_1 N + i_2 N^2$ and $j = j_0 + j_1 N + j_2 N^2$ for any $0 \le i_l < N$, $0 \le j_l < N$, with $l = 0, 1, 2$. We may likewise write $(A_k)_{ab}$ and $(B_k)_{ab}$ as N^3-vectors labeled $\tilde{A}_k$ and $\tilde{B}_k$. We now stitch our collections of vectors $\{\tilde{A}_k\}$ and $\{\tilde{B}_k\}$ into matrices. Given p, we define an $N^3 \times p$ matrix U, a $p \times N^3$ matrix V, and a $p \times p$ diagonal

matrix Σ via

$$U_{jk} = \frac{(\tilde{A}_k)_j}{\left\|\tilde{A}_k\right\|_2}, \qquad V_{kj} = \frac{(\tilde{B}_k)_j}{\left\|\tilde{B}_k\right\|_2}, \qquad \Sigma_{kk} = \left\|\tilde{A}_k\right\|_2 \left\|\tilde{B}_k\right\|_2, \tag{5.18}$$

for $1 \leq j \leq N^3$ and $1 \leq k \leq p$.

The p columns of U are simply the normalized vector encodings of the p coefficient functions $\{A_k\}_{k=1}^p$. The rows of V are likewise formed from $\{B_k\}_{k=1}^p$. Note that we may reorder our indices such that $\{\Sigma_{kk}\}_{k=1}^p$ is a decreasing sequence.

We can now express our sum (5.16) as $\left\|\tilde{G} - U\Sigma V\right\|_F^2 h^6$, where $\|\cdot\|_F$ is the Frobenius matrix norm. Minimizing the Frobenius norm here is a well-studied least squares problem. It is known that if we take $p = N^3$ and find the optimal expansion minimizing (5.16) then we recover $\tilde{G}$ exactly. That is, $\tilde{G} = U\Sigma V$. This is precisely the Singular Value Decomposition (SVD) of $\tilde{G}$.

The key property of this decomposition for our needs is that, for a given $p < N^3$, truncating the SVD to p terms provides the minimizing expansion for (5.16). That is, given the SVD of $\tilde{G}$, we can obtain the optimal p-term expansion coefficient functions $\{A_k\}_{k=1}^p, \{B_k\}_{k=1}^p$ from the first p columns of $U\sqrt{\Sigma}$ and the first p rows of $\sqrt{\Sigma}V$, respectively. This result is known as the Eckart-Young theorem [6] and the resulting expansion is the so-called rank-1 decomposition.

Using this optimal p-term expansion, the L^2-error (5.16) is given by the square sum of the neglected singular values,

$$\left(\sum_{k=p+1}^{N^3} \Sigma_{kk}^2 \right)^{1/2}. \tag{5.19}$$

The accuracy of our p-term expansion is thus directly related to the rate of decay of the singular values of $\tilde{G}$. The faster the singular values decay the fewer terms we require in our expansion to accurately capture the behavior of $\tilde{G}$. Figure 5.2 shows the sharp decay of the singular values of $\tilde{G}$, where we have encoded in $\tilde{G}$ the xx and xy components of G_h respectively. It is this marked decay that allows for an efficient treecode approach.

Calculating the SVD of $\tilde{G}$ can be expensive. Fortunately, as with the construction of G_h, this is a one time cost that can be spread over multiple simulations. However, some care must still be taken as a direct approach to calculating the SVD of $\tilde{G}$ would require $O(N^9)$ operations. In Section 5.2.2 we propose an efficient strategy to reduce the cost to $O(pN^3 \log N)$ operations.

Finally, turning back to our original optimization problem of minimizing (5.15), we approximate $A_k(\mathbf{x})$ and $B_k(\mathbf{x})$ at arbitrary positions $\mathbf{x} \in \Omega_{in}$ and $\mathbf{y} \in \Omega_{out}$ by using trilinear interpolation between the surrounding Eulerian grid points. G_h is smooth away from the origin, and its singular vectors are likewise smooth. Trilinear interpolation thus introduces an error of at most $O(h^2)$. For a given interaction between two fiber points we commit this error p times, for a total error on the

order of $O(ph^2)$. Assuming h is sufficiently small, this error is smaller than the $O(h)$ error of our expansion and the $O(h)$ error of the IB Method. A numerical verification of the error rate of our expansion is presented below in Section 5.3.

5.1.4 Decomposition group

We have seen how to arrive at a decomposition of G_h over a subdomain $\Omega_{in} \times \Omega_{out}$. However, our treecode makes use of $O(N \log N)$ different subdomains. Each subdomain will require a suitable decomposition. Fortunately, we may recycle many of these decompositions for use over multiple subdomains.

Specifically, given two panels P^1 and P^2 at the same depth in our octree, we may take their expansion coefficient functions to be identical up to translation. That is, given $\mathbf{z} \in \Omega$,

$$A_k^{P_1}(\mathbf{c}^{P_1} + \mathbf{z}) = A_k^{P_2}(\mathbf{c}^{P_2} + \mathbf{z}), \tag{5.20}$$

$$B_k^{P_1}(\mathbf{c}^{P_1} + \mathbf{z}) = B_k^{P_2}(\mathbf{c}^{P_2} + \mathbf{z}), \tag{5.21}$$

provided the functions are defined at the points of evaluation. Here, $\mathbf{c}^{P_i}$ denotes the center of panel P_i, $i = 1, 2$.

Due to this equivalence we only need $\log N$ decompositions: one for each level of our octree. The total precomputational cost of calculating our lookup tables is thus $O(pN^3 \log^2 N)$, and the total memory requirement is $O(pN^3 \log N)$.

5.2 The treecode algorithm

The exposition on the treecode in the previous section was intentionally high level. The intent was to provide an understanding for what a treecode is, why it works, and why it is a good fit for the IB Method. Unfortunately, the details of a treecode implementation are somewhat technical. This section is devoted to providing a more detailed exposition, but is not critical to understanding the thrust of the methods developed in this chapter.

We develop in more detail two components of the treecode algorithm. First, in Section 5.2.1, we expand on the algorithms for creating the octree and for using it to evaluate multipole summations. Then, in Section 5.2.2, we expand on the algorithm used to arrive at the far field expansions of G_h.

5.2.1 The octree

Note that for this section we adopt a more programmatic syntax, replacing Ω_{in}^P with $P.\Omega_{in}$, $\mathbf{c}^P$ with $P.\mathbf{c}$, and so on. This is to facilitate the exposition of pseudocode.

We start by defining the notion of a *panel*. A panel is simply a cubic subset of Ω, together with an optional collection of 8 child panels that divide the parent into equal octants. Let P be a panel. To P we associate three vectors, $P.\mathbf{c}, P.\mathbf{a}, P.\mathbf{b} \in$

Ω. $P.\mathbf{a}$ and $P.\mathbf{b}$ serve to define the domain of P, given by

$$P.\Omega_{out} = [P.\mathbf{a}_0, P.\mathbf{b}_0] \times [P.\mathbf{a}_1, P.\mathbf{b}_1] \times [P.\mathbf{a}_2, P.\mathbf{b}_2] \subseteq \Omega. \tag{5.22}$$

The center of P is denoted $P.\mathbf{c} = (P.\mathbf{a} + P.\mathbf{b})/2$. This will be the center of expansion for poles inside P. We also associate to P a collection of fiber points $P.\mathbf{Y}$ consisting of those points of the fiber configuration $\mathbf{X}$ lying within the domain $P.\Omega_{out}$. The children of P, if it has any, are denoted by $P.Child[i]$, for $0 \leq i < 8$. The children divide $P.\Omega_{out}$ equally, by splitting it into 8 pieces via the 3 planes $x = P.\mathbf{c}_0$, $y = P.\mathbf{c}_1$, and $z = P.\mathbf{c}_2$.

Given a domain $\Omega = [0,1]^3$, we define an initial panel called *Root*, specified by the bounds $Root.\mathbf{a} = (0,0,0)$ and $Root.\mathbf{b} = (1,1,1)$. We now recursively define child panels, creating the 8 children of *Root*, then the 8 children of each of those children, and so on. For a panel P with few fiber points in its domain, say $|P.\mathbf{Y}| < MinPoints$, we truncate the recursive process, leaving P childless. For a pseudocode version of this process, see Function 5.2.1 CreatePanel. For our simulations we take $MinPoints = 10$. A rigorous accounting of the treecode costs can lead to an optimal value of $MinPoints$, but we did not pursue such an analysis here.

The totality of *Root* and all of its branches is collectively known as the octree. We now introduce the concept of *well separatedness*. Given a point $\mathbf{x}$ and a panel

P we say that $\mathbf{x}$ is well separated from P provided that

$$|\mathbf{x}_i - P.\mathbf{c}_i| \geq 3(P.\mathbf{c}_i - P.\mathbf{a}_i), \qquad \text{for } i = 0, 1, 2, \tag{5.23}$$

that is, provided that $\mathbf{x}$ is sufficiently far from the center of P. We now consider the pair of subsets $(P.\Omega_{in}, P.\Omega_{out})$, where $P.\Omega_{in}$ is the collection of points in Ω that are well separated from P. We assume that we can expand G_h over $P.\Omega_{in} \times P.\Omega_{out}$, arriving at two collections of coefficient functions $\{P.A_k\}_{k=1}^{p}$ and $\{P.B_k\}_{k=1}^{p}$. In addition, we define

$$P.B_{out} = \{i \in B | \mathbf{X}_i \in P.\mathbf{Y}\}, \tag{5.24}$$

as the collection of indices associated with fiber points in $P.\mathbf{Y}$.

Suppose now that we are given a fiber force $\mathbf{F}$ and wish to evaluate its influence at points $\mathbf{x}$ in the fluid domain. For each $\mathbf{x}$ this influence is given by

$$\sum_{1 \leq j \leq N_B} G_h(\mathbf{x} - \mathbf{X}_j)\mathbf{F}_j. \tag{5.25}$$

To evaluate this efficiently, we first loop over each panel P and calculate the far field expansion of all the poles located in $P.\Omega_{out}$. The k-th term of this expansion is given by

$$P.H_k \equiv \sum_{j \in P.B_{out}} P.B_k(\mathbf{X}_j)\tilde{F}_j. \tag{5.26}$$

If $\mathbf{x}$ is well separated from P, then the incoming effect on $\mathbf{x}$ from P is given by

$$\mathbf{E}^T \sum_{k=1}^{p} P.A_k(\mathbf{X}_i) P.H_k. \tag{5.27}$$

We can now evaluate the entire influence of $\mathbf{F}$ on a point $\mathbf{x}$. The process is best described in recursive form. A pseudocode is presented in Function 5.2.2 Evaluate.

Because our treecode must be invoked multiple times per timestep, it is critical to streamline its evaluation. There are two key points to keep in mind. First, calculations of the expansion coefficients can be recycled. In Section 5.1.3 we will detail the form of the coefficient functions A_k and B_k. Each evaluation requires an expensive trilinear interpolation. Avoiding duplications of these calculations provides substantial speedup. Second, using the treecode to evaluate the influence at a particular point $\mathbf{x}$ requires traversing the octree. This traversal can be stored as a template so that additional evaluations at $\mathbf{x}$ do not require a recursive call.

There are thus two different function calls: the actual evaluation function, which takes in a fiber force $\mathbf{F}$ and returns the influence over the entire structure $\mathbf{X}$; and a pre-evaluation function, which does a variety of computations used to streamline the evaluation function. The cost of the pre-evaluation function is much higher than that of the evaluation function itself. However, the pre-evaluation function is called only once per timestep.

We mention one further implementation detail regarding the treatment of the domain $P.\Omega_{in}$ for a given panel P. Rather than treating this is as a single domain, we break it into eight equally sized pieces, corresponding to the eight octants around the panel's center $P.\mathbf{c}$. For each piece we must then have separate collec-

tions of coefficient functions $\{P.A_k\}_{k=1}^p$, $\{P.B_k\}_{k=1}^p$, as well as a separate expansion $\{P.H_k\}_{k=1}^p$.

This additional detail may seem costly at first, but in fact it increases the speed of the treecode substantially. By restricting our attention to a smaller domain it is possible to find expansions which converge much more rapidly.

5.2.2 Far field expansions: calculating a truncated SVD of restrictions of G_h

The far field expansions of G_h that we employ are known as rank-1 decompositions, and can be obtained via the Singular Value Decomposition. Note that when we decompose G_h we consider it properly as a function defined on $\Omega \times \Omega$, not as a field over Ω. For actual evaluations of G_h, however, we still assume translation invariance, treating G_h as a field and making use of our lookup table on $\mathcal{G}_h$.

Treating G_h as a function defined only on $\mathcal{G}_h \times \mathcal{G}_h$, we may write G_h as an $N^3 \times N^3$ matrix and seek its SVD. Typical SVD algorithms would require $O(N^9)$ operations. This is too great a cost, even for a precomputation. The two key ingredients for accelerating the computation of the SVD are, first, that we only require the first p singular values of G_h, not all N^3 of them, and, second, that our approximation to G_h is convolutional. We present an algorithm that takes advantage of these ingredients. While matrices are convenient for relating to the

function 5.2.1 CreatePanel($\mathbf{a}, \mathbf{b}$)

$P \leftarrow$ **new** $Panel$

$P.\mathbf{a} = \mathbf{a}$

$P.\mathbf{b} = \mathbf{b}$

$P.\mathbf{c} = (\mathbf{a} + \mathbf{b})/2$

if $P.Parent$ exists **then**

$P.\mathbf{Y} = P.Parent.\mathbf{Y} \cap P.\Omega_{out}$

else

$P.\mathbf{Y} = \mathbf{X}^n \cap P.\Omega_{out}$

if $|P.\mathbf{Y}| > MinPoints$ **then**

$P.Child[0] = \text{CreatePanel}(P.\mathbf{a}, P.\mathbf{c})$

$P.Child[1] = \text{CreatePanel}([P.\mathbf{c}_0, P.\mathbf{a}_1, P.\mathbf{a}_2], [P.\mathbf{b}_0, P.\mathbf{c}_1, P.\mathbf{c}_2])$

$P.Child[2] = \text{CreatePanel}([P.\mathbf{a}_0, P.\mathbf{c}_1, P.\mathbf{a}_2], [P.\mathbf{c}_0, P.\mathbf{b}_1, P.\mathbf{c}_2])$

$P.Child[3] = \text{CreatePanel}([P.\mathbf{c}_0, P.\mathbf{c}_1, P.\mathbf{a}_2], [P.\mathbf{b}_0, P.\mathbf{b}_1, P.\mathbf{c}_2])$

$P.Child[4] = \text{CreatePanel}([P.\mathbf{a}_0, P.\mathbf{a}_1, P.\mathbf{c}_2], [P.\mathbf{c}_0, P.\mathbf{c}_1, P.\mathbf{b}_2])$

$P.Child[5] = \text{CreatePanel}([P.\mathbf{c}_0, P.\mathbf{a}_1, P.\mathbf{c}_2], [P.\mathbf{b}_0, P.\mathbf{c}_1, P.\mathbf{b}_2])$

$P.Child[6] = \text{CreatePanel}([P.\mathbf{a}_0, P.\mathbf{c}_1, P.\mathbf{c}_2], [P.\mathbf{c}_0, P.\mathbf{b}_1, P.\mathbf{b}_2])$

$P.Child[7] = \text{CreatePanel}(P.\mathbf{c}, P.\mathbf{b})$

return P

function 5.2.2 Evaluate($\mathbf{x}, P$)

if $\mathbf{x} \in P.\Omega_{in}$ **then** {$\mathbf{x}$ is well separated from P}

return $\mathbf{E}^T \sum_{k=1}^{p} P.A_k(\mathbf{X}_i) P.H_k$

else

if P has children **then**

$S = 0$ {the 3×3 zero tensor}

for $i = 0$ to 7 **do**

$S = S +$ Evaluate($\mathbf{x}, P.Child[i]$)

return S

else {do a direct summation}

return $\sum_{j \in P.B_{out}} G_h(\mathbf{x}, \mathbf{X}_j) \mathbf{F}_j$

SVD, in what follows we treat G_h directly as a function, and abandon any use of matrices.

We will focus on the discretized domain $\mathcal{G}_h$ and a particular pair of subsets, $(\Omega_{in}, \Omega_{out})$, over which we wish to decompose G_h. We will require a few definitions. Given two functions f and g defined over $\Omega_{in} \cap \mathcal{G}_h$ and $\Omega_{out} \cap \mathcal{G}_h$ respectively, we define the outer product $f \otimes g$ via the relation

$$(f \otimes g)(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})g(\mathbf{y}), \tag{5.28}$$

for $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h$, $\mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. If f' is another function defined over $\Omega_{in} \cap \mathcal{G}_h$, and both f and f' are 3×3 tensor fields, then we define the componentwise inner product $\langle f, f' \rangle$ to be a 3×3 tensor given by

$$\langle f, f' \rangle_{ij} = \sum f_{ij}(\mathbf{x}) f'_{ij}(\mathbf{x}), \tag{5.29}$$

where the sum is taken over all $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h$.

For a scalar function f defined on $\mathcal{G}_h$, the L^2 norm is given by

$$\|f\|_2^2 = \sum_{\mathbf{x} \in \mathcal{G}_h} f(\mathbf{x})^2. \tag{5.30}$$

For functions with other domains the norm is implicitly taken over the entire domain of the function. For a 3×3 tensor valued function g on $\mathcal{G}_h$, we define $\|g\|_2$ to be a 3×3 tensor given by $(\|g\|_2)_{ij} = \|g_{ij}\|_2$.

Now, we are seeking a collection of $2p$ functions, $\{A_k\}_{k=1}^p$ and $\{B_k\}_{k=1}^p$, each a 3×3 tensor field over $\mathcal{G}_h$, such that the rank-1 decomposition

$$\sum_{k=1}^{p} A_k \otimes B_k \tag{5.31}$$

is as close as possible to G_h in the L^2 sense. From SVD theory we know that given this truncated decomposition we will have simultaneously minimized

$$\left\| G_h - \sum_{k=1}^{q} A_k \otimes B_k \right\|_2 \tag{5.32}$$

for any q such that $1 \leq q \leq p$, where the norm is taken over the domain $\Omega_{in} \cap \mathcal{G}_h \times \Omega_{out} \cap \mathcal{G}_h$. We may take advantage of this result by first finding A_1 and B_1 such that (5.32) is minimized for $q = 1$. Next, we can hold A_1 and B_1 fixed as we seek the A_2 and B_2 that minimize (5.32) for $q = 2$. Assuming we have iterated this procedure up to $q = l - 1$, we detail the procedure for finding A_l and B_l.

First, A_l and B_l are exactly the tensor fields that minimize

$$L \equiv \left\| G_h - \sum_{k=1}^{l} A_k \otimes B_k \right\|_2^2 = \| G'_h - A_l \otimes B_l \|_2^2 , \tag{5.33}$$

where

$$G'_h \equiv G_h - \sum_{k=1}^{l-1} A_k \otimes B_k. \tag{5.34}$$

If we fix B_l then we can minimize L in (5.33) by the method of least squares. We fix a $\mathbf{z}$ in Ω_{in} and derive L with respect to $A_l(\mathbf{z})$, obtaining

$$-\frac{1}{2}\frac{\partial L}{\partial A_l(\mathbf{z})} = -\frac{1}{2}\frac{\partial}{\partial A_l(\mathbf{z})} \sum \left(G'_h(\mathbf{x}, \mathbf{y}) - A_l(\mathbf{x}) B_l(\mathbf{y}) \right)^2 , \tag{5.35}$$

where the sum is taken over $\mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h, \mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. Dropping the terms independent of $A_l(\mathbf{z})$ leaves

$$\begin{aligned} &\sum \left(G'_h(\mathbf{z},\mathbf{y}) - A_l(\mathbf{z})B_l(\mathbf{y})\right) B_l(\mathbf{y}) \\ &= \left[\sum G'_h(\mathbf{z},\mathbf{y})B_l(\mathbf{y})\right] - A_l(\mathbf{z}) \left\|B_l\right\|_2^2, \end{aligned} \tag{5.36}$$

where the sum is now over $\mathbf{y} \in \Omega_{out} \cap \mathcal{G}_h$. The sum in (5.36) is nearly a convolution. We seek to recast it as such.

First, we define the indicator functions

$$\mathbf{I}_{in}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_{in} \cap \mathcal{G}_h \\ 0 & \text{otherwise,} \end{cases} \tag{5.37}$$

and

$$\mathbf{I}_{out}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \Omega_{out} \cap \mathcal{G}_h \\ 0 & \text{otherwise.} \end{cases} \tag{5.38}$$

A_l is a field over $\Omega_{in} \cap \mathcal{G}_h$, but we may identify $A_l \mathbf{I}_{in}$ with a field defined over all of $\mathcal{G}_h$ in the natural way. We may likewise identify $B_l \mathbf{I}_{out}$ with a field defined over all of $\mathcal{G}_h$. These identifications allow us to express the sum in (5.36) as a convolution

against G_h over all of $\mathcal{G}_h$, minus some correction.

$$\begin{aligned}
&\left[\sum G'_h(\mathbf{z},\mathbf{y})B_l(\mathbf{y})\right] \\
&= \left[\sum G_h(\mathbf{z}-\mathbf{y})B_l(\mathbf{y})\right] - \sum_{k=1}^{l-1}\sum A_k(\mathbf{z})B_k(\mathbf{y})B_l(\mathbf{y}) \\
&= \left[(G_h * (B_l\mathbf{I}_{out}))\mathbf{I}_{in}\right](\mathbf{z}) - \sum_{k=1}^{l-1} A_k(\mathbf{z})\left\langle B_k, B_l\right\rangle .
\end{aligned} \tag{5.39}$$

Requiring that the derivatives of L be zero in (5.36), we solve for A_l as

$$A_l = \frac{1}{\|B_l\|_2^2}\left[(G_h * (B_l\mathbf{I}_{out}))\mathbf{I}_{in} - \sum_{k=1}^{l-1} A_k\left\langle B_k, B_l\right\rangle\right]. \tag{5.40}$$

If we fix A_l we can solve the equivalent least squares problem for B_l, obtaining

$$B_l = \frac{1}{\|A_l\|_2^2}\left[(G_h * (A_l\mathbf{I}_{in}))\mathbf{I}_{out} - \sum_{k=1}^{l-1} B_k\left\langle A_k, A_l\right\rangle\right]. \tag{5.41}$$

Equations (5.40) and (5.41) must be solved simultaneously. We approximate the solution to this system via the iterative method

$$\begin{cases}
A_l^{n+1} = \dfrac{1}{\|B_l^n\|_2^2}\left[(G_h * (B_l^n\mathbf{I}_{out}))\mathbf{I}_{in} - \displaystyle\sum_{k=1}^{l-1} A_k\left\langle B_k, B_l^n\right\rangle\right], \\
B_l^{n+1} = \dfrac{1}{\|A_l^{n+1}\|_2^2}\left[(G_h * (A_l^{n+1}\mathbf{I}_{in}))\mathbf{I}_{out} - \displaystyle\sum_{k=1}^{l-1} B_k\left\langle A_k, A_l^n\right\rangle\right].
\end{cases} \tag{5.42}$$

We require an initial guess for this iteration, call it B_l^0, with the restriction that B_l^0 not be a linear combination of $\{B_k\}_1^{l-1}$.

(5.42) is a type of power method. Power methods are known to converge quickly provided the singular values of G_h are sufficiently separated. For our

particular problem (5.42) converges in 20 to 30 iterations. A stopping criteria can be improvised based on $\left\|B_l^{n+1} - B_l^n\right\|$, but for simplicity we simply fix the number of iterations at 30.

We have written (5.42) in a form that allows for a minimum of computational effort. The convolutions against G_h may be efficiently computed in discrete Fourier space at an $O(N^3 \log N)$ cost, while the inner products require only $O(N^3)$ operations each. Thus, the total cost per iteration of (5.42) is $O(N^3 \log N)$. We require $O(\log N)$ decompositions, hence the total precomputational cost is $O(30pN^3 \log^2 N)$.

5.3 Treecode benchmarks

In this section we present a numerical validation and performance evaluation of the treecode. To this end, we use a test problem which consists of an immersed, flat plate with fiber points $\mathbf{X}_i$ tethered to fixed points $\mathbf{T}$ and with a fiber force $\mathcal{A}_{h_B}(\mathbf{X}) = \sigma(\mathbf{T} - \mathbf{X})$ (see Section 5.4.1 for details on this problem setup). We fix $N = 128$, $\sigma = 10^7$, $\Delta t = 0.002$, and vary N_B and p.

The performance of the treecode is analyzed through the performance of two different function calls: the evaluation function, which takes in a fiber force $\mathbf{F}$ and returns the influence at every fiber point $\mathbf{X}$; and a pre-evaluation function, which

does a variety of computations used to streamline the evaluation function. The pre-evaluation function is called only once per timestep

We examine accuracy first. There are two confounding factors that degrade the accuracy of our treecode. The first is that we approximate G_h as a truncated rank-1 decomposition. The second is that we introduce error into the simulation by assuming translation invariance of G_h. In order to analyze the first component separately we consider an immersed plate where the fiber points lie exactly on Eulerian intersection. This avoids the introduction of any error from translation.

We calculate the error according to the following procedure. We first make an $O(h)$ perturbation of the fiber $\mathbf{X}$. This perturbation generates a force $\mathbf{F}$ on the fiber. We then evaluate $\mathcal{M}_n\mathbf{F}$ using both the treecode and a direct fluid solve and take the sup norm of the difference.

The blue line in Figure 5.3 shows the resulting decrease in error for increasing values of p. If we do not restrict the fiber points to Eulerian intersections then the above procedure returns a different decay line, seen as the green line in Figure 5.3. Note that past $p = 10$ no additional reduction in error is achieved. This is the point at which the $O(h)$ error from our approximation to G_h overwhelms the error introduced by using a truncated expansion. In actual simulations fiber points can not be restricted to Eulerian intersections, hence any value of $p > 10$ would be computationally wasteful.

If we fix $p = 10$ and compute the error for various values of h we see that it does indeed decrease at least as fast as $O(h)$, as seen in Figure 5.4, consistent with the error bounds presented in Section 4.1.4 and Section 4.1.5.

We now analyze the performance of the treecode. All units of time are given as multiples of the average time to perform a fluid solve with $N = 128$. We first fix $p = 10$ and vary N_B. The resulting CPU time for the evaluation of $\mathcal{M}_n\mathbf{F}$ for an arbitrary $\mathbf{F}$ is shown in Figure 5.5. The corresponding CPU time for the pre-evaluation call is given in Figure 5.6. Both scale nearly linearly in N_B, as expected.

Most analytic expansions used for treecodes and FMM codes, including Taylor series expansions, are not optimal, in the sense that a different expansion would yield higher accuracy for the same number of terms. We approximate directly the L_2-optimal expansion. We note that the extension of treecodes and FMM codes to generic kernels, and the associated use of the SVD decomposition in particular, has been studied since the early 90s, see e.g. [7] for a more detailed exposition.

5.4 Krylov Methods

The treecode method can be easily adapted to compute Gauss-Seidel iterations for the implicit system (3.15). In this sense the treecode is a generalization of the

matrix method: it allows for all the iterative methods that the matrix method allows for, while providing additional speedups, especially in the 3D case. In particular we can use the treecode to implement a multigrid solver.

Importantly, however, for many applications evaluation of terms of the form $\mathcal{M}_n\mathbf{F}$ via the treecode is so efficient that sophisticated solvers such as multigrid are not necessary. In this section we present two applications of the treecode in 3D. The iterative method employed is a simple conjugate gradient. We will see, however, that solving (3.15) is nonetheless very cheap, amounting to a fraction of the total cost of an implicit timestep.

5.4.1 Case III: An immersed plate.

For the first test of our proposed methodology we simulate flow past an immersed plate. The plate is a flat square of dimensions 1/2 by 1/2 and is discretized as an $(M+1)\times(M+1)$ grid of fiber points, where $M = \lfloor N\sqrt{2}/2 \rfloor$. This yields $N_B = 529$, 2116, and 8281 for $N = 32$, 64, and 128 respectively.

For each j, k such that $0 \le j \le M, 0 \le k \le M$ we have a fiber point $\mathbf{X}_{j,k} \in \Omega$. Each fiber point $\mathbf{X}_{j,k}$ is tethered to a corresponding fixed point $\mathbf{T}_{j,k} \in \Omega$ given by

$$\mathbf{T}_{j,k} = (0.25, 0.25, 0.5) + \frac{1}{2M}(j, k, 0). \tag{5.43}$$

The tethers induce a force $\mathcal{A}_{h_B}(\mathbf{X}) = \sigma(\mathbf{T} - \mathbf{X})$ which acts to restore the plate to equilibrium, as well as to bind the plate against the fluid flow pushing against

it. The initial configuration of the plate is taken to be the equilibrium state $\mathbf{X}^0 = \mathbf{T}$. Note that, among non-empty Eulerian grid cells, the average number of fiber points per cell is roughly 2. This is the minimum density needed to prevent unreasonable spurious currents across the plate.

A fluid flow is induced by adding a time dependent forcing vector $\mathbf{f}(t)$ to the right hand side of (3.1). We take $\mathbf{f}(t) = 100(0, \sin\theta(t), \cos\theta(t))$, where $\theta(t) = 4\cos(6\pi t/T)/\pi$ and $T = 0.25$ is the total simulation time. The addition of $\mathbf{f}(t)$ alters the explicit term $\mathbf{b}^n$ in our implicit system to

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h[\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n + \Delta t \mathbf{f}(t^n)]. \tag{5.44}$$

The induced flow has a Reynolds number of about 10.

$\mathcal{A}_{h_B}$ is affine, not linear, thus we cannot directly apply CG to solve the implicit system. Suppose we denote our initial guess for $\mathbf{X}^{n+1}$ as $\mathbf{X}^{n+1,0}$. We define $\bar{\mathbf{X}}^{n+1} = \mathbf{X}^{n+1} - \mathbf{X}^{n+1,0}$ and recast our implicit system in terms of $\bar{\mathbf{X}}^{n+1}$. We define $\mathcal{A}'_{h_B}(\mathbf{X}) = -\sigma\mathbf{X}$ and a new explicit term

$$\bar{\mathbf{b}}^n = \mathbf{X}^n - \mathbf{X}^{n+1,0} + \Delta t \mathcal{S}_n^* \mathcal{L}_h[\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n + \mathcal{A}_{h_B}(\mathbf{X}^{n+1,0}) + \Delta t \mathbf{f}(t^n)]. \tag{5.45}$$

Our implicit system then becomes

$$\bar{\mathbf{X}}^{n+1} = \mathcal{M}_n \mathcal{A}'_{h_B}(\bar{\mathbf{X}}^{n+1}) + \bar{\mathbf{b}}^n. \tag{5.46}$$

Here $\mathcal{A}'_{h_B}$ is linear negative definite, thus $I - \mathcal{M}_n \mathcal{A}'_{h_B}$ is positive definite and we can solve (5.46) via CG. For the CG we use a convergence tolerance of 0.0001,

Table 5.1: Δt in the explicit simulations of the immersed plate for various values of N and σ. Δt is approximately the largest *stable* timestep, given by the formula $\Delta t = 10h\sigma^{-1/2}$.

	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N = 32$	$3.125{\cdot}10^{-5}$	$9.882{\cdot}10^{-6}$	$3.125{\cdot}10^{-6}$	$9.882{\cdot}10^{-7}$	$3.125{\cdot}10^{-7}$
$N = 64$	$1.563{\cdot}10^{-5}$	$4.941{\cdot}10^{-6}$	$1.563{\cdot}10^{-6}$	$4.941{\cdot}10^{-7}$	$1.563{\cdot}10^{-7}$
$N = 128$	$7.813{\cdot}10^{-6}$	$2.471{\cdot}10^{-6}$	$7.813{\cdot}10^{-7}$	$2.471{\cdot}10^{-7}$	$7.813{\cdot}10^{-8}$

which is at least 10 times smaller than the error of the method. For an initial guess we take $\mathbf{X}^{n+1,0} = \mathbf{X}^n$.

We perform both explicit and implicit simulations for $N = 32, 64, 128$ and various values of σ. The explicit simulation uses a standard Forward Euler/Backward Euler (FE/BE) discretization, i.e. implicit in the viscous term but explicit in all the other terms, including the tension force. The largest stable timestep is given by the empirical formula $\Delta t = 10h\sigma^{-1/2}$. Table 5.1 provides a list of the values of Δt required for a stable simulation using the FE/BE discretization. Note that even for modest resolutions the timestep is prohibitively small.

In marked contrast, for the semi-implicit, lagged operators discretization a constant timesteps of $\Delta t = 0.002$ is sufficient to maintain both stability and accuracy for all resolutions and values of σ. Our proposed fast solution strategy yields total computation times that are several orders of magnitude smaller than those for the popular FE/BE method. The total CPU time is shown in Table 5.2.

Table 5.2: Total CPU time in hours for the explicit and implicit simulations of the immersed plate, with varying values of σ. * denotes an extrapolated value.

		$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N = 32$	Implicit	0.012	0.011	0.011	0.013	0.012
	Explicit	0.214	0.675	2.146	6.806	21.296
		$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N = 64$	Implicit	0.109	0.109	0.106	0.111	0.108
	Explicit	4.072	12.906	40.813*	129.063*	408.133*
		$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$	$\sigma = 10^{10}$	$\sigma = 10^{11}$
$N = 128$	Implicit	0.896	0.889	0.897	0.892	0.896
	Explicit	64.779*	204.8481*	647.787*	2048.481*	6477.867*

The savings are striking; computations that would take over a month to perform with the FE/BE (even with a modest $N = 128$) can be done in minutes using the proposed new approach. Note also that as σ increases the total CPU time using the new methodology is almost invariant, whereas for the explicit FE/BE simulations the total CPU time grows markedly.

In Table 5.3 we provide a break down of the computational costs associated with a single implicit timestep for the case of the immersed plate. All units of time are given as multiples of the average time to perform a fluid solve for the given value of N. Of note is that the cost of performing Conjugate Gradient, typically requiring 5 to 10 iterations, is only a small contribution to the overall

Table 5.3: A break down of average CPU time for different components of the implicit timestep. Time is given as multiples of the average time to perform a fluid solve. Included in the Tree Initialization is the pre-evaluation cost.

	$N = 32$	$N = 64$	$N = 128$
Fluid Solves	2.000	2.000	2.000
Tree Initialization	1.166	1.125	1.107
Conjugate Gradient	0.394	0.327	0.295
Total	3.560	3.452	3.403

computational cost. The predominant costs come from the fluid solves and the treecode initialization and pre-evaluation.

Figure 5.7 presents a depiction of the flow using streamlines in a sequence of snapshots. A cross-section of the z-component of the velocity is also plotted below the plate. The flow has the expected periodic behavior while the structure of the plate is maintained throughout the simulation.

5.4.2 Case IV: An oscillating spheroid

In the immersed plate test described above, the geometry of the structure is trivial and the deformations are negligible. Of course, the true power of the IB methodology lies in its seamless handling of both rigid *and* dynamic, flexible interfaces and in its structure-building capability. As an example of a simulation with a dynamic, flexible interface we consider now an immersed, oscillating spheroid

given by

$$\frac{(x-0.5)^2}{(0.2+0.05\sin\theta(t))^2}+\frac{(y-0.5)^2}{(0.2+0.05\sin\theta(t))^2}+\frac{(z-0.5-0.05\cos\theta(t))^2}{(0.2-0.1\sin\theta(t))^2}=1, \tag{5.47}$$

where $\theta(t) = 2\pi t/T$, t is the current simulation time and $T = 0.25$ is the total simulation time. Equation (5.47) yields a spheroid centered at $(0.5, 0.5, 0.5 + 0.05\cos\theta)$ with an equatorial radius of $0.2 + 0.05\sin\theta$ and a polar radius of $0.2 - 0.1\sin\theta$. The prescribed motion induces a flow with a Reynolds number of about 10.

The shape of the spheroid is maintained by tethers. At time t^n each fiber point $\mathbf{X}_j^n$ is tethered to its respective location on the spheroid $\mathbf{T}_j^n$. The sphere itself is discretized by triangulating a regular octahedron, yielding $N_B = 578$, 2502, and 10406 fiber points when $N = 32$, 64, and 128, respectively. When solving for $\mathbf{X}^{n+1}$ in the semi-implicit, lagged operators discretization, we take our initial guess to be $\mathbf{X}^{n+1,0} = \mathbf{T}^{n+1}$. We use the same 0.0001 convergence tolerance for CG as we do in the immersed plate simulation.

As in the previous example, we compare (explicit) FE/BE simulations with the proposed, fast, semi-implicit approach. For the FE/BE method the stable timestep is determined by the empirical formula $\Delta t = 0.5 \cdot 10^{-1} h\sigma^{-1/2}$. Table 5.4 provides a list of stable, explicit timesteps for this problem. The required Δt for a stable FE/BE simulation is even smaller than that in the plate example. Again,

Table 5.4: Δt in the FE/BE explicit simulations of the immersed spheroid for various values of N and σ. Δt is approximately the largest stable timestep, given by the formula $\Delta t = 0.5 \cdot 10^{-1} h \sigma^{-1/2}$.

	$\sigma = 10^5$	$\sigma = 10^6$	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$
$N = 32$	$4.941{\cdot}10^{-6}$	$1.563{\cdot}10^{-6}$	$4.941{\cdot}10^{-7}$	$1.563{\cdot}10^{-7}$	$4.941{\cdot}10^{-8}$
$N = 64$	$2.471{\cdot}10^{-6}$	$7.813{\cdot}10^{-7}$	$2.471{\cdot}10^{-7}$	$7.813{\cdot}10^{-8}$	$2.471{\cdot}10^{-8}$
$N = 128$	$1.235{\cdot}10^{-6}$	$3.906{\cdot}10^{-7}$	$1.235{\cdot}10^{-7}$	$3.906{\cdot}10^{-8}$	$1.235{\cdot}10^{-8}$

such direct, FE/BE simulations are impractical and would require a massive computational effort even for modest resolutions. For the implicit, lagged operators discretization it is again sufficient to fix $\Delta t = 0.002$ to maintain both accuracy and stability for all resolutions, $N = 32$, 64, and 128. A comparison of the total CPU time for all simulations is presented in Table 5.5. As in the plate example, the CPU time for the fast, semi-implicit simulations is almost invariant as σ increases. The numbers are even more striking than in the preceding example; for $N = 128$ and $\sigma = 10^9$, the proposed approach is six orders of magnitude faster than the commonly used FE/BE approach. A depiction of the flow obtained using the new fast, semi-implicit approach is presented in Figure 5.8.

5.4.3 Accuracy

There are a number of differences between our implicit methodology and the standard FE/BE explicit methodology. Each difference is a potential source of

Table 5.5: Total CPU time in hours for the explicit and semi-implicit simulations of the immersed spheroid, with varying values of σ. * denotes an extrapolated value.

		$\sigma = 10^5$	$\sigma = 10^6$	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$
$N = 32$	Implicit	0.013	0.015	0.015	0.016	0.016
	Explicit	4.405	13.675	43.467	135.742*	429.701*
		$\sigma = 10^5$	$\sigma = 10^6$	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$
$N = 64$	Implicit	0.127	0.134	0.140	0.138	0.137
	Explicit	83.155*	253.626*	822.769*	2595.119*	8213.271*
		$\sigma = 10^5$	$\sigma = 10^6$	$\sigma = 10^7$	$\sigma = 10^8$	$\sigma = 10^9$
$N = 128$	Implicit	1.057	1.125	1.141	1.151	1.167
	Explicit	1326.0*	4170.7*	13904.8*	43827.9*	134516.2*

additional numerical error: the larger timestep taken, the implicit discretization itself, the assumption of translation invariance of G_h, the trilinear interpolation used to calculate values of the form $G_h(\mathbf{z})$, the far field expansion of G_h, and approximate nature of the Krylov subspace solvers. We have been extra careful to ensure that each of these errors is no more than $O(h)$, the underlying order of the IB Method, and to verify numerically that these errors do not accumulate.

To analyze error accumulation we again turn to our simulations of a plate and a sphere. For the following simulations we fix the immersed structure via tether points with spring constant $\sigma = 10^6$ and induce a simple flow by adding a constant force $\mathbf{f} = (0, 0, 1)$ to every point of the fluid domain. We perform the simulation

first for $N = 128$ using the explicit method and store the resulting fluid velocity at time $T = 0.1$ as u_{128}. This velocity field will serve as a our standard by which we gauge the accuracy of other simulations.

We now perform the same simulation twice for $N = 32$, 64, and 96, once using the explicit method and once using our implicit methodology. We calculate the relative error between a velocity field $\tilde{u}$ and our standard u_{128} by taking the l^2-norm, $\|u_{128} - \tilde{u}\|_{l^2} \,/\, \|u_{128}\|_{l^2}$. For simplicity we downsample all velocity fields to a $32 \times 32 \times 32$ grid. The resulting errors can be seen in Table 5.6.

The relative errors from our implicit simulations are roughly the same as those from the explicit simulations, confirming that our fast implicit methodology is not generating any unacceptable inaccuracies.

Finally, we perform one more test to verify that approximating G_h as translation invariant does not lead to a degradation of the overall accuracy. We do a simulation using an identical setup as that used for the calculation of u_{128}, except we shift the immersed boundary upward by a distance of $h/2$. We store the resulting fluid velocity at time $T = 0.1$ as $\tilde{u}_{128}$.

If G_h were exactly translation invariant then the resulting simulation would be identical to the original simulation up to a shift of the velocity field. That is, we would have $u_{128}(\mathbf{z}) = \tilde{u}_{128}(\mathbf{z} + (0, 0, h/2))$, for any $\mathbf{z} \in \Omega_h$. This is the case with the continuous equations.

Table 5.6: The relative error between various low resolution simulations and a high resolution explicit simulation.

		$N = 32$	$N = 64$	$N = 96$
Plate	Implicit	0.160	0.0327	0.015
	Explicit	0.222	0.073	0.024

		$N = 32$	$N = 64$	$N = 96$
Sphere	Implicit	0.058	0.015	0.005
	Explicit	0.089	0.029	0.009

For our discrete simulations G_h is not exactly translation invariant and there is a difference between $u_{128}(\mathbf{z})$ and $\tilde{u}_{128}(\mathbf{z} + (0, 0, h/2))$. Note that we use simple linear interpolation to calculate values of $\tilde{u}_{128}$ between grid points. Calculating the relative difference between u_{128} and a shifted $\tilde{u}_{128}$ using the l^2-norm as before, we see that this difference is less than 1%, or smaller than the error of the method. Shifts other than $(0, 0, h/2)$ yield similar results.

5.5 Conclusion

In Chapter 4, novel expedited methods for the semi-implicit system (3.1)-(3.3) were proposed. We showed that these methods were highly efficient for many 2D applications. The cost of the implicit solvers presented were on the order of the cost of a fluid solve, allowing for efficient implicit timestepping with computational

cost on the same order as that of an explicit timestep. The direct extension of that methodology to 3D was unfeasible due to the large number of fiber points common to 3D IB applications.

In this chapter we presented an entirely new, alternative methodology suitable for the 3D case and for when there is a large number of immersed boundary points. We showed that the efficiency of the proposed fast semi-implicit solver, relative to the cost of a fluid solve, is asymptotically superior to the solvers in the 2D case. Indeed, in our test problems, we demonstrated that solving the implicit system was not the predominant cost for the semi-implicit timestep, but was rather overshadowed by the cost of the necessary fluid solves. Thus, we have shown that the stiffness inherent in many IB applications can be eliminated via a robust, semi-implicit discretization for a minimal cost. More importantly, the proposed approach scales very well as N_B increases, allowing the new methodology to be applied to a wide range of complex structures. The computational savings obtained with the new methodology are enormous. IB Method computations that would typically require weeks or months using the standard, FE/BE approach can now be performed in just minutes.

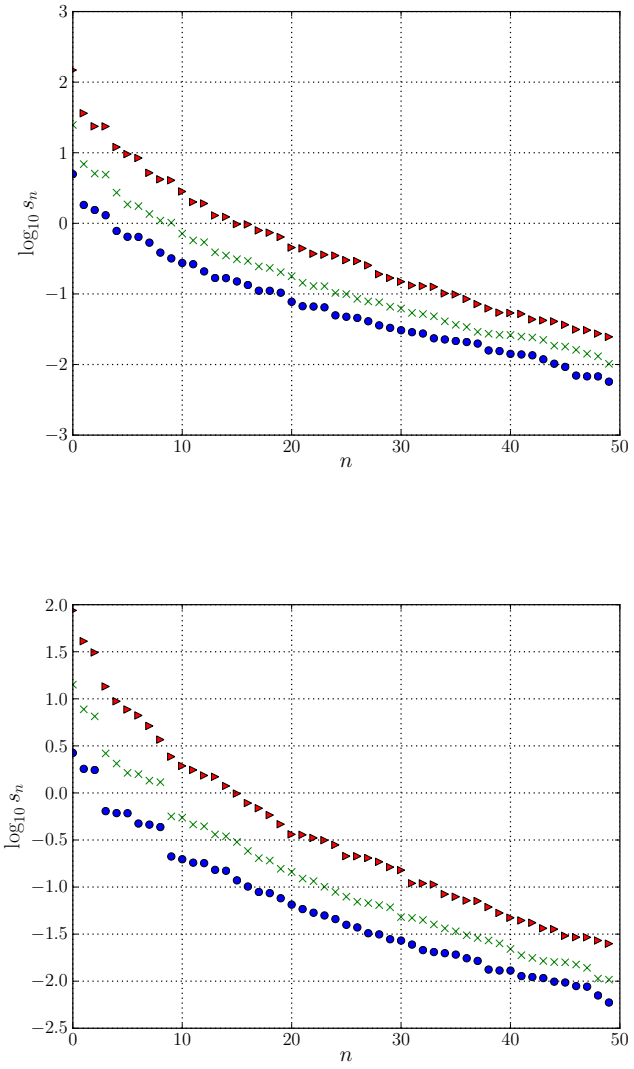


Figure 5.2: Singular values of the discrete Green's function: $(G_h)_{xx}$ component (top) and $(G_h)_{xy}$ component (bottom). 'o' markers are for $N = 32$, 'x' markers are for $N = 64$, and '$\triangle$' makers are for $N = 128$.

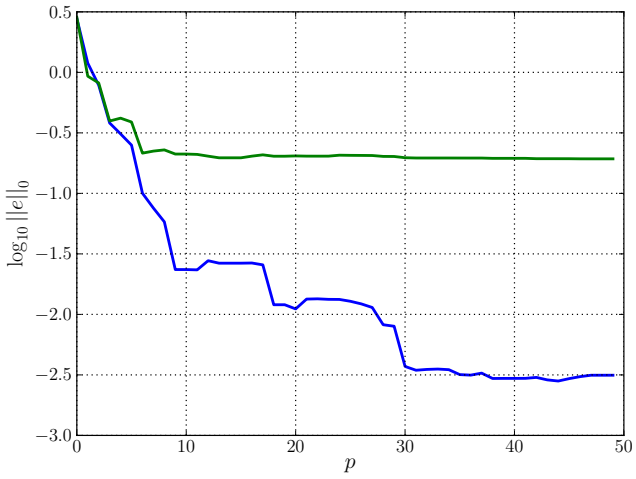


Figure 5.3: Log of the sup norm of the difference between $\mathcal{M}_n\mathbf{F}$ calculated via a fluid solve and via the treecode for various values of p, where p is the number of terms in the treecode's far field expansion. $N = 128$ and $N_B = N^2/4$. The blue line is for a plate with fiber points aligned to the Eulerian grid. The green line is for a plate with no restrictions on the fiber point locations.

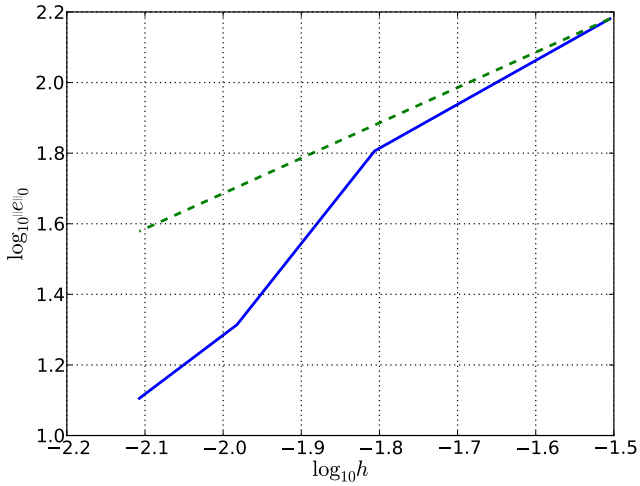


Figure 5.4: Log of the sup norm of the difference between $\mathcal{M}_n\mathbf{F}$ calculated via a fluid solve and via the treecode for various values of h, with $p = 10$. The solid blue line is the error. The dashed green line is a reference line with slope 1.

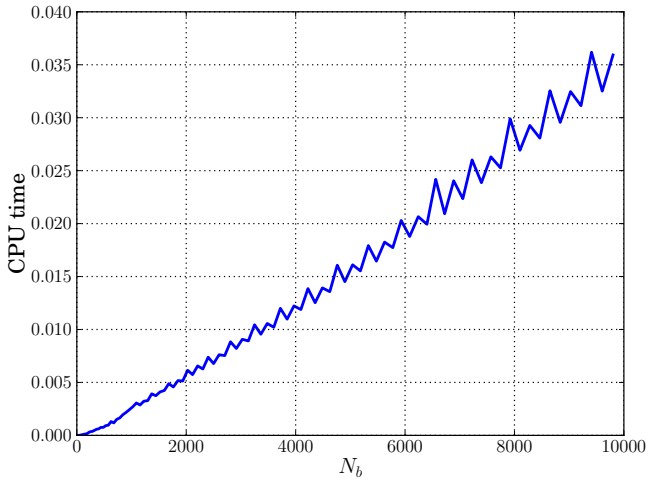


Figure 5.5: CPU time for the evaluation of $\mathcal{M}_n\mathbf{F}$ via the treecode for increasing values of N_B, where $p = 10$. Time is given as multiples of the average time to perform a fluid solve, for $N = 128$.

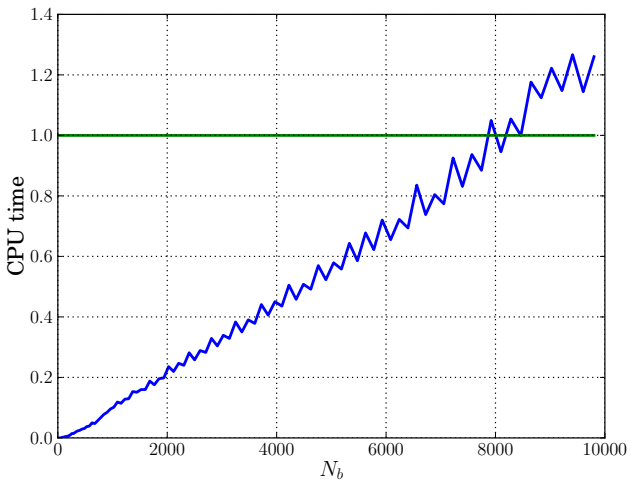


Figure 5.6: CPU time for the treecode pre-evaluation for increasing values of N_B, where $p = 10$. Time is given as multiples of the average time to perform a fluid solve, for $N = 128$. The green line represents the cost of one fluid solve.

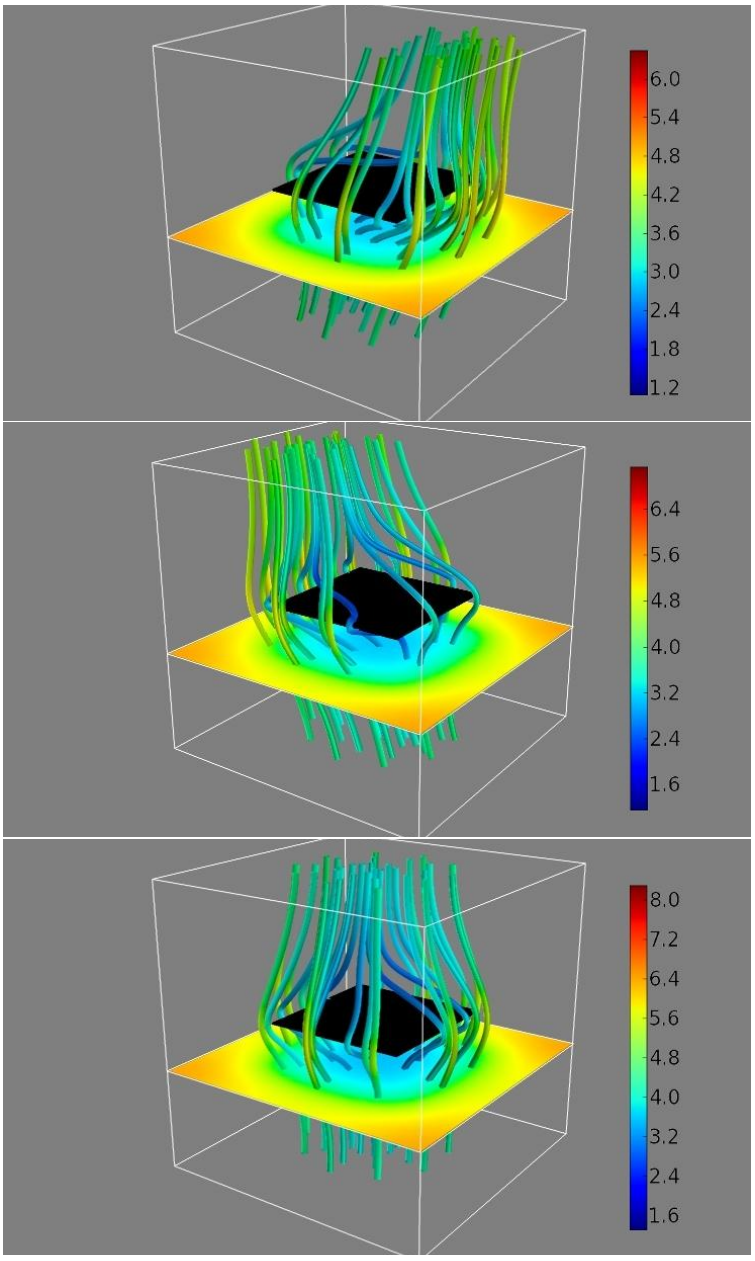


Figure 5.7: Plot of the immersed plate with flow lines. The plate is drawn in black. Below the plate is a cross-section of the z-component of the velocity field with an associated color bar. $N = 128$, and $N_B = 8192$. Frames shown at total simulation time $T = 0.020$, 0.142, and 0.208, from top to bottom.

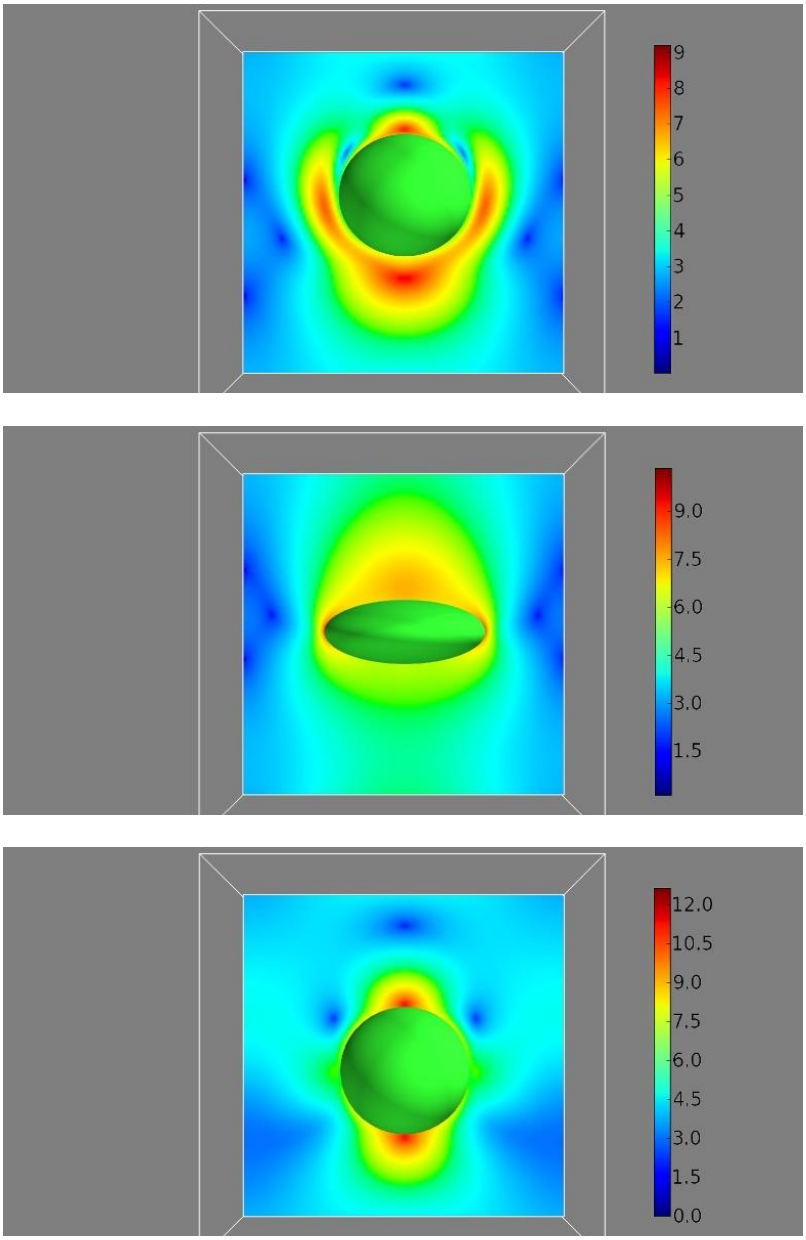


Figure 5.8: Plot of the immersed spheroid, drawn in green. Cutting the spheroid is a cross-section of the velocity magnitude scalar field with an associated color bar. $N = 128$, and $N_B = 10406$. Frames shown at total simulation time $T = 0.006$, 0.062, and 0.126, from top to bottom.

Chapter 6

The Splitting Method

In Chapter 4 and 5 we focused on methods to accelerate computations of the form $\mathcal{M}_n\mathbf{F}$, as well as iterative methods to solve the linear system (3.15) in the case that $\mathcal{A}_{h_B}$ or its Jacobian, J, is negative-semidefinite. In the more difficult case that J is nondefinite we must find alternative iterative methods to solve (3.15).

In this chapter we explore a special case where J is nondefinite but has eigenvalues that are all large in magnitude. This case includes applications modeling rigid bodies. We will develop the methodology, known as the Splitting Method, in conjunction with a specific application, namely flow past a rigid valve.

6.1 Case V: A model of a heart valve

6.1.1 The model

We turn now to a more challenging application of the IB Method in which there are rigid immersed structures, tethered points, and crossed links. We consider a 2D model of a rigid valve immersed in blood flowing through an artery. The valve is indirectly restricted in motion by two hinges but is allowed to rotate. The valve, artery walls, and hinges will all be modeled as immersed springs. The flow-structure interaction will be captured via the IB Method.

We select the computational domain to be $\Omega = [0,2] \times [0,1]$ with periodic boundary conditions and discretize it with a $2N \times N$ uniform grid. The geometry of the problem is represented in Fig. 6.1 and a detailed depiction of the valve's linked structure is shown in Fig. 6.2. The top and bottom of the artery walls include cushions in the shape of two hills. We simulate a horizontal flow through the artery by adding a forcing vector $\mathbf{f}_{j,k} = (v_{flow}, 0)$ to the right hand side of (3.1), where in general v_{flow} may be time dependent. This changes the explicit term $\mathbf{b}^n$ in our implicit system to

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h \left[\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n + \frac{\Delta t}{\rho} \mathbf{f} \right]. \tag{6.1}$$

All nodes in the wall and the two hinges are modeled as tethers, with one end of the tether fixed for all time. If $\mathbf{X}_i$ is a tethered point with base given by $\bar{\mathbf{X}}_i$

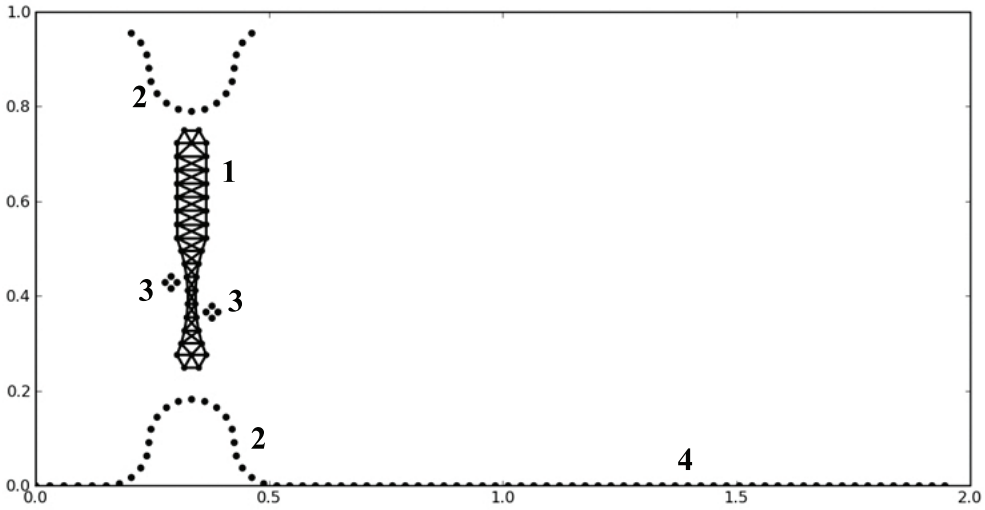


Figure 6.1: Configuration of heart valve model. Larger nodes represent tether points. 1: Valve; 2: Cushions; 3: Hinges; 4: Artery wall.

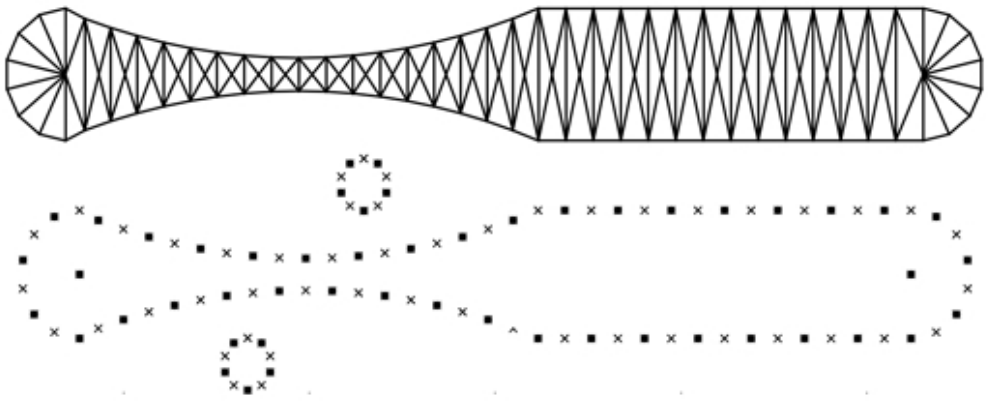


Figure 6.2: Top: A fine grid approximation to the valve, showing only linkage. Bottom: 'x's mark the prolongation of a coarse valve marked in 'o'.

then the force generated by the tether is given by

$$\mathbf{f}_i = -k_i(\mathbf{X}_i - \bar{\mathbf{X}}_i) \tag{6.2}$$

where k_i is the spring constant. For the body of the heart valve we will use springs between nodes. Suppose $\mathbf{X}_i$ is connected to multiple other nodes. If $\mathbf{X}_i$ is connected to $\mathbf{X}_j$ then let $k_{i,j}$ and $L_{i,j}$ be respectively the spring constant and resting length of the spring connecting them. If no connection exists between $\mathbf{X}_i$ and $\mathbf{X}_j$ then take $k_{i,j}$ to be zero. The force at $\mathbf{X}_i$ then is given by

$$\mathbf{f}_i = \sum_{j=1}^{N_B} -k_{i,j}\frac{\mathbf{X}_i - \mathbf{X}_j}{|\mathbf{X}_i - \mathbf{X}_j|}(|\mathbf{X}_i - \mathbf{X}_j| - L_{i,j}). \tag{6.3}$$

Note that the force operator (4.45) employed for the elliptical membrane problem is exactly the force given by (6.3) for a loop of fiber points connected sequentially by springs with suitable spring constants and zero resting length. However, for our valve model we require non-zero resting lengths to preserve the structure and as a consequence we end up with a nonlinear force density. Additionally, the problem is severely stiff. To maintain the rigidity of a solid body, we find that the spring constants must be $O(10^9 h^{-2})$. Similarly, for the tether points representing the artery walls, the spring constants must also be very large to portray the tautness of the biological fibers. Furthermore, because the forcing flow acts to bend the valve as it is penned between its hinges, larger values of v_{flow} require larger spring constants to preserve the structure. To illustrate the stiffness, a time integration

of the equations of motion with an explicit treatment of the interfacial force and with a modest spatial resolution ($N = 256$) requires the time-step to be $O(10^{-7})$.

Removing this stiffness for this more prototypical IB Method problem is considerably more challenging than in the case of the simple elliptical fiber. The implicit system we would like to solve is still given by (3.15):

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n.$$

As noted above, due to the nonzero resting lengths of the springs comprising the valve, $\mathcal{A}_{h_B}$ is nonlinear. Moreover, unlike the case of the elliptical interface with nonlinear force density, the Jacobian J of $\mathcal{A}_{h_B}$ is not semidefinite and the resulting matrix $I - M_n J$ can be shown to lack definiteness as well.

The breakdown of the definiteness of J can be seen in a very simple case. Consider four immersed boundary points at $(1, 0), (0, 1), (-1, 0), (0, -1)$, forming a square with side lengths $\sqrt{2}$ and with links connecting the perimeter of the square. We vary the resting length l of the connecting springs and compute the eight eigenvalues of the resulting force density's Jacobian. The results are plotted in Fig. 6.3. We see that as the resting length approaches and surpasses the side length of the square the Jacobian loses its negative semi-definiteness. This seems to be generic for most immersed structures. For our 2D valve model, we take the resting length to be the starting length of the links comprising the valve. Thus, we lose immediately negative semi-definiteness once we perturb the valve structure.

Due to the lack of positive definiteness of $I - M_n J$ the Conjugate Gradient Method does not converge. The Biconjugate Gradient Method converges but may take in excess of 100 iterations to attain a satisfactory residual. A major obstacle in accelerating convergence is that the natural preconditioner $I - M_n' J$ (where M_n' is the diagonal part of M_n) does not capture well the dynamics of this system. Indeed, Jacobi iterations have very poor convergence, requiring strong underrelaxation and thousands of iterations. Without adequate smoothers, an efficient multigrid for the linear system in Newton's iteration is not a viable option. Smoothers for the nonlinear system (3.15) were likewise difficult to uncover, seemingly ruling out a nonlinear multigrid. We propose next a very different approach, which consists of splitting the problem to take simultaneous advantage of the fast convergence of Newton's method and the efficiency of multigrid.

6.1.2 Solving the implicit system via fixed point iterations

Perhaps the simplest iterator for (3.15) is the fixed point iteration given by

$$\mathbf{X}^{n+1,k+1} = M_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k}) + \mathbf{b}^n. \tag{6.4}$$

However, this method fails spectacularly. This is not entirely surprising. If our initial guess is $\mathbf{X}^{n+1,0} = \mathbf{X}^n$ then $\mathbf{X}^{n+1,1}$ is the explicit update provided by the FE/BE scheme which is unstable for practical Δt. Additional iterations of (6.4)

exacerbate the instability: small errors in $\mathbf{X}^{n+1,k}$ are amplified enormously by $\mathcal{A}_{h_B}$ resulting in large errors in $\mathbf{X}^{n+1,k+1}$.

The eigenvalues of J are at least on the order of the spring constants comprising our valve. It should seem natural to take advantage of this by reversing the fixed point iteration (6.4). Consider the alternative fixed point iteration

$$\begin{cases} M_n \mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n, \\ \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1}) = \mathbf{F}^{k+1}. \end{cases} \tag{6.5}$$

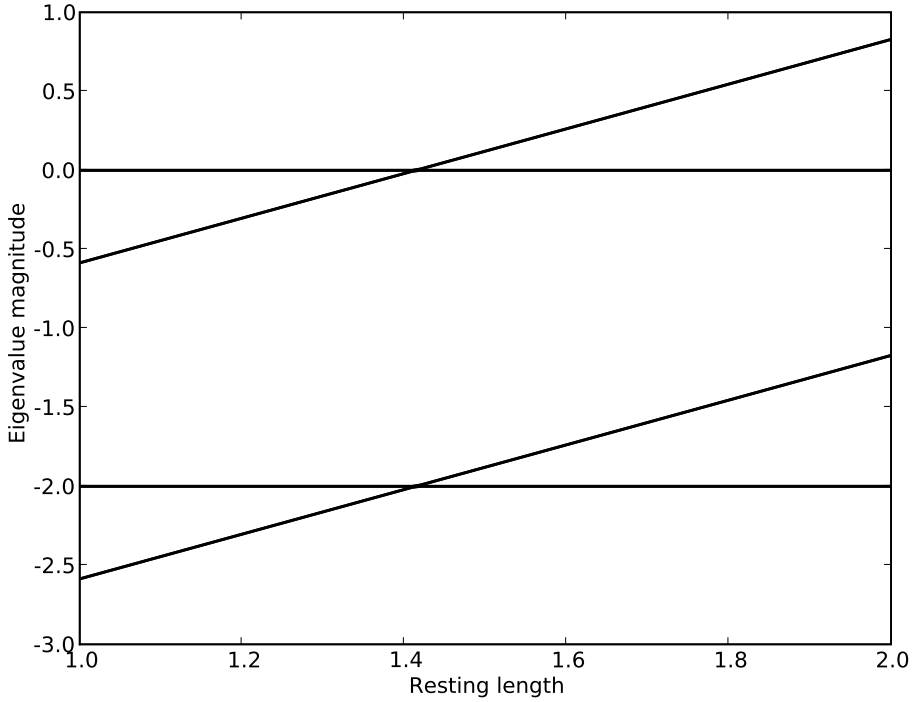


Figure 6.3: Eigenvalues for the Jacobian of a fiber forcing function for four points linked as a square with varying resting length.

First, we solve for $\mathbf{F}^{k+1}$, the force distribution that would move the fiber from $\mathbf{X}^n$ to $\mathbf{X}^{n+1,k}$ after a single timestep. The linear system we must solve only involves the linear positive definite operator $\mathcal{M}_n$, thus we can efficiently solve it using (linear)multigrid. Second, we determine what configuration $\mathbf{X}^{n+1,k+1}$ gives rise to this force through $\mathcal{A}_{h_B}$. In many applications there is a unique $\mathbf{X}^{n+1,k+1}$ such that $\mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1}) = \mathbf{F}^{k+1}$. For the current problem this is not the case.

To see this note that valve is modeled as a neutrally buoyant object, detached from any fixed points, and the internal forces generated by perturbations in the valve's structure are unaffected by translation. More precisely let $\mathcal{G}_V$ be all indices l such that $\mathbf{X}_l$ is a fiber point belonging to the valve. We consider then two translation vectors $\mathbf{V}^1$ and $\mathbf{V}^2$ given by

$$\mathbf{V}_l^1 = \begin{cases} (1,0) & \text{if } l \in \mathcal{G}_V, \\ (0,0) & \text{otherwise,} \end{cases} \tag{6.6}$$

$$\mathbf{V}_l^2 = \begin{cases} (0,1) & \text{if } l \in \mathcal{G}_V, \\ (0,0) & \text{otherwise.} \end{cases} \tag{6.7}$$

$\mathbf{V}^1$ and $\mathbf{V}^2$ are horizontal and vertical translation vectors, $\mathbf{X} + \mathbf{V}^1$ representing the same configuration as $\mathbf{X}$ with the valve shifted right by one unit. Shifting has no influence on the force density,

$$\mathcal{A}_{h_B}(\mathbf{X} + a_1\mathbf{V}^1) = \mathcal{A}_{h_B}(\mathbf{X} + a_2\mathbf{V}^2) = \mathcal{A}_{h_B}\mathbf{X}, \tag{6.8}$$

where a_1 and a_2 are arbitrary scalars. Thus, $\mathcal{A}_{h_B}$ is not injective and we may not have a unique solution to (6.5). In fact, $\mathcal{A}_{h_B}$ is not surjective either so (6.5) may have no solution at all. This follows physically from conservation of linear momentum and angular momentum: $\mathcal{A}_{h_B}$ cannot directly generate forces that move or rotate the valve. Translation and rotation can only be introduced via the fluid interaction.

To analyze rotation we introduce the operator R_θ which takes in a configuration $\mathbf{X}$ and returns the configuration obtained by rotating the valve by θ about some fixed point. This point of rotation is arbitrary and we take it to be the center of the valve at the previous time-step:

$$\mathbf{x}_c = \left(\frac{(\mathbf{X}^n, \mathbf{V}^1)_B}{(\mathbf{V}^1, \mathbf{V}^1)_B}, \frac{(\mathbf{X}^n, \mathbf{V}^2)_B}{(\mathbf{V}^2, \mathbf{V}^2)_B} \right). \tag{6.9}$$

We define also

$$\mathbf{V}^3(\mathbf{X}) = \frac{\partial}{\partial \theta} R_\theta(\mathbf{X}). \tag{6.10}$$

We may equivalently define $\mathbf{V}^3$ via

$$\mathbf{V}^3_l = \begin{cases} (-Y_l + y_c, X_l - x_c) & \text{if } l \in \mathcal{G}_V, \\ (0,0) & \text{otherwise,} \end{cases} \tag{6.11}$$

where $\mathbf{X}_k = (X_k, Y_k)$ and $\mathbf{x}_c = (x_c, y_c)$. $\mathbf{V}^3$ is the Jacobian of R_θ. Note that $\mathbf{V}^3$ depends on a configuration $\mathbf{X}$. We use the notation $\mathbf{V}^3(\mathbf{X})$ to emphasize this dependence.

The vectors $\mathbf{V}^1, \mathbf{V}^2, \mathbf{V}^3$ represent points outside the image of $\mathcal{A}_{h_B}$ associated with translation and rotation. Then, we have that there exist solutions to (6.5) only when $\mathbf{F}^{k+1}$ does not act to translate or rotate the valve. That is, a vector $\mathbf{F}$ lies in the image of $\mathcal{A}_{h_B}$ only when

$$(\mathbf{F}, \mathbf{V}^j)_B = 0 \quad \text{for } j = 1, 2, 3. \tag{6.12}$$

Indeed, consider first the simplest case of only two immersed fiber points

$$\mathbf{X} = \begin{pmatrix} (x_1, y_1) \\ \\ (x_2, y_2) \end{pmatrix} \tag{6.13}$$

connected with a single link. Then, the force they generate is

$$\mathbf{F} = T \begin{bmatrix} (x_2 - x_1, y_2 - y_1) \\ \\ (x_1 - x_2, y_1 - y_2) \end{bmatrix} \tag{6.14}$$

for some tension scalar T. The rotation vector about the origin is simply

$$\mathbf{V}^3 = \begin{bmatrix} (-y_1, x_1) \\ \\ (-y_2, x_2) \end{bmatrix}. \tag{6.15}$$

We have then that

$$(\mathbf{F}, \mathbf{V}^3)_B = -y_1(x_2 - x_1) + x_1(y_2 - y_1) - y_2(x_2 - x_1) + x_2(y_1 - y_2) = 0. \tag{6.16}$$

A similar calculation shows that $(\mathbf{V}^1, \mathbf{F})_B = (\mathbf{V}^2, \mathbf{F})_B = 0$. For our more complicated valve structure we simply have a summation of such forces, thus the

corresponding force density must satisfy (6.12) where $\mathbf{F} = \mathcal{A}_{h_B}(\mathbf{X})$ for any configuration $\mathbf{X}$. In general, an arbitrary vector $\mathbf{F}$ may not satisfy (6.12) and we would be unable to find a solution to (6.5). To remedy this we must factor out those components outside the image of $\mathcal{A}_{h_B}$. This is easier to do with the linearized $\mathcal{A}_{h_B}$, its Jacobian, where we may simply project onto the vector space free of the problematic vectors $\mathbf{V}^1, \mathbf{V}^2, \mathbf{V}^3$. These observations suggest the modified Newton iteration to approximately solve for $\mathbf{X}^{n+1,k+1}$ in (6.5):

$$\begin{cases} \mathbf{X}^{n+1,k+1,0} = \mathbf{X}^{n+1,k}, \\ J(\mathbf{X}^{n+1,k+1,l})(\mathbf{X}^{n+1,k+1,l+1} - \mathbf{X}^{n+1,k+1,l}) \\ \qquad = P(\mathbf{F}^{k+1} - \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1,l})), \end{cases} \tag{6.17}$$

where

$$P(\mathbf{F}) = \mathbf{F} - \sum_{j=1}^{3} \mathbf{V}^j(\mathbf{X}^{n+1,k})(\mathbf{F}, \mathbf{V}^j(\mathbf{X}^{n+1,k}))_B \tag{6.18}$$

is the projection operator onto rotation and translation free force distributions. Note that because each node in the valve is linked to only a few other nodes, J is $O(N_B)$ sparse. We can solve systems involving J efficiently with various sparse solvers. Additionally, the structure of J never changes throughout a simulation, though its values do, so we may make additional optimizations in the sparse solver if desired.

Linear systems of the form $J\mathbf{X} = P\mathbf{F}$ are, strictly speaking, over determined by three degrees of freedom. To rectify this we simply isolate three degrees of

freedom in $\mathbf{X}$ and fix them. These variables must be associated with the valve but are otherwise arbitrary. We could, for instance, fix the x and y component of a single node in the valve as well as the x component of an additional node. Once fixed, we may proceed to solve for $\mathbf{X}$ such that $J\mathbf{X} = P\mathbf{F}$ is satisfied at all other free points. What is important, however, is that iterations of (6.17) in such a manner will converge to an $\mathbf{X}$ that satisfies $\mathcal{A}_{h_B}\mathbf{X} = P\mathbf{F}$ at all points, even at those we fixed. In this sense we arrive at an updated configuration $\mathbf{X}^{n+1,k+1}$ which satisfies (6.5) up to components outside the image of $\mathcal{A}_{h_B}$.

The sequence $\mathbf{X}^{n+1,0}, \mathbf{X}^{n+1,1}, \mathbf{X}^{n+1,2}, \ldots$ formed by iterations of (6.5) typically converges quadratically in the l^2 norm. As an aside, it is important to note that the iterative method (6.5) converges precisely because the standard fixed point iteration (6.4) diverges. As we modify the parameters of our simulation, for instance, if we were to drastically decrease the spring constants, then (6.4) may converge whereas (6.5) would diverge. There are situations with mixed large and small spring constants where neither iteration converges. The crosslinks of our valve model is one such example. In the case of large spring constants everywhere (6.5) converges rapidly and increasing the spring constants aids in the convergence.

The limit of our iterations is a stable update of the immersed boundary configuration but is not in general a solution to (3.15). We need to reintroduce translation and rotation to correct for this discrepancy. We can achieve this in

many different ways. Perhaps the simplest method is to use the location and angle of the valve configuration obtained from an explicit update. To accomplish this we may simply take our initial guess $\mathbf{X}^{n+1,0}$ for iterations of (6.17) to be

$$\mathbf{X}^{n+1,0} = R_\theta \mathbf{X}^n + \mathbf{V}^1(\tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{V}^1)_B + \mathbf{V}^2(\tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{V}^2)_B \tag{6.19}$$

where

$$\theta = (\tilde{\mathbf{X}} - \mathbf{X}^n - \mathbf{V}^1(\tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{V}^1)_B - \mathbf{V}^2(\tilde{\mathbf{X}} - \mathbf{X}^n, \mathbf{V}^2)_B, \mathbf{V}^3(\mathbf{X}^n))_B \tag{6.20}$$

and

$$\tilde{\mathbf{X}} = \mathbf{X}^n + \mathcal{M}_n \mathcal{A}_{h_B} \mathbf{X}^n \tag{6.21}$$

Here, we are taking the FE/BE predictor $\tilde{\mathbf{X}}$ and extracting its location and angle. We then take the current configuration $\mathbf{X}^n$ and modify the location and angle of the valve to match that in $\tilde{\mathbf{X}}$. After this, we may proceed to apply iterations of (6.17) which will preserve the location and angle we extracted from the FE/BE predictor.

Other predictors may be employed as well. For instance we may rediscretize our problem on a coarser grid, solve for the updated time-step, prolong the solution to the original fine grid and extract its gross properties. Other predictors may involve higher order explicit updates $\tilde{\mathbf{X}}$ or treating the valve as a true rigid body and predicting its motion through rigid body motion.

6.1.3 Reintroducing translation and rotation iteratively

Many of the explicit predictors, such as those discussed above, behave quite satisfactorily and in conjunction with (6.5) provide accurate and stable solutions to (3.15). In a number of applications this is sufficient. If we employ (6.17) after using the initial guess (6.19) we can obtain efficiently an approximate solution to (3.15) within truncation error with just a few iterations.

In the current application, however, extra care must be taken. The close proximity of the valve to its hinges can quickly lead to unphysical collisions if small errors in translation propagate throughout the simulation. This proximity can also seriously affect the behavior of an explicit predictor, leading to highly inaccurate translation and rotation. Of course, even exact solutions to (3.15) with Δt large may still produce collisions due to inaccuracies and hence this problem is not intrinsic to the proposed methodology.

We seek a means to reintroduce more accurately the three degrees of freedom we have removed in applying (6.5). Suppose that we have a solution $\mathbf{X}^{n+1}$ to the original nonlinear system (3.15). We must have then

$$(\mathbf{F}^{n+1}, \mathbf{V}^j(\mathbf{X}^{n+1}))_B = 0 \text{ for } j = 1, 2, 3. \tag{6.22}$$

where $\mathbf{F}^{n+1}$ is such that

$$\mathcal{M}_n \mathbf{F}^{n+1} = \mathbf{X}^{n+1} - \mathbf{b}^n. \tag{6.23}$$

This is because if $\mathbf{X}^{n+1}$ solves (3.15) then $\mathbf{F}^{n+1} = \mathcal{A}_{h_B}\mathbf{X}^{n+1}$ and $\mathcal{A}_{h_B}$ cannot generate forces with nonzero components along $\mathbf{V}_j$, $j = 1, 2, 3$. Condition (6.22) suggests that before each iteration of (6.5) we simply shift and rotate $\mathbf{X}^{n+1,k}$ such that $(\mathbf{F}^{k+1}, \mathbf{V}_j(\mathbf{X}^{n+1,k}))_B = 0$ for $j = 1, 2, 3$, where $\mathcal{M}_n\mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n$ as before. This can be approximately accomplished by finding (a_1, a_2) and θ, a translation vector and an angle, such that for $j = 1, 2, 3$

$$(\mathcal{M}_n^{-1}(\mathbf{X}^{n+1,k} + a_1\mathbf{V}^1 + a_2\mathbf{V}^2 + \theta\mathbf{V}^3(\mathbf{X}^{n+1,k}) - \mathbf{b}^n), \mathbf{V}^j(\mathbf{X}^{n+1,k}))_B = 0. \quad (6.24)$$

Or, equivalently,

$$\begin{aligned} \begin{bmatrix} (\mathbf{W}^1, \mathbf{V}^1)_B & (\mathbf{W}^2, \mathbf{V}^1)_B & (\mathbf{W}^3, \mathbf{V}^1)_B \\ (\mathbf{W}^1, \mathbf{V}^2)_B & (\mathbf{W}^2, \mathbf{V}^2)_B & (\mathbf{W}^3, \mathbf{V}^2)_B \\ (\mathbf{W}^1, \mathbf{V}^3)_B & (\mathbf{W}^2, \mathbf{V}^3)_B & (\mathbf{W}^3, \mathbf{V}^3)_B \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \theta \end{bmatrix} = \\ - \begin{bmatrix} (\mathbf{F}^{k+1}, \mathbf{V}^1)_B \\ (\mathbf{F}^{k+1}, \mathbf{V}^2)_B \\ (\mathbf{F}^{k+1}, \mathbf{V}^3)_B \end{bmatrix}, \end{aligned} \quad (6.25)$$

where $\mathcal{M}_n\mathbf{W}^j = \mathbf{V}^j$ for $j = 1, 2, 3$. Then, we construct an updated configuration

$$\mathbf{X}^{n+1,k} \leftarrow R_\theta\mathbf{X}^{n+1,k} + a_1\mathbf{V}^1 + a_2\mathbf{V}^2 \quad (6.26)$$

after which we proceed to calculate $\mathbf{X}^{n+1,k+1}$ via (6.17) using an unmodified $\mathbf{F}^{k+1}$. In this manner we obtain an iteration which converges rapidly to a solution of (3.15). This iteration converges nearly as quickly as (6.5) does for a fixed valve

with no rotation or translation. There is an additional cost per iteration though. The predominant cost comes from solving the linear systems involving $\mathcal{M}_n$, which we must now do four times to calculate $\mathbf{W}^j$ for $j = 1, 2, 3$ as well as $\mathbf{F}^{k+1}$.

Fortunately, these added costs are not significant in practice. The vectors $\mathbf{W}^1$ and $\mathbf{W}^2$ only need to be calculated once per timestep. For our simulations, we calculate $\mathbf{W}^3$ only once per timestep as well, even though $\mathbf{V}^3$ depends on the current guess $\mathbf{X}^{n+1,k}$. We find that this does not degrade the final residual beyond the limits of accuracy provided by our multigrid solver. Moreover, $\mathbf{W}^j$ for $j = 1, 2, 3$ will change only incrementally from one time-step to the next and hence these same vectors can be reused as good initial guesses in the following several time-steps. At the end, the predominant cost of the proposed iterators is that of initializing the linear system and multigrid itself.

6.1.4 Near boundary-boundary interactions

There is an additional subtlety that we wish to point out. The prolongation and restriction operators we use in our multigrid for solving $\mathcal{M}_n\mathbf{W}^j = \mathbf{V}^j$ are geometrically inspired, relying on the underlying structure of the immersed boundary configuration. However, this approach fails to give adequate weight to the fluid interactions *between* immersed boundaries, in particular the interaction

between the ends of the valve and the cushions as well as the interaction between the middle of the valve and the hinges.

The linear multigrid we employ still converges in the current situation but not sufficiently fast at key points. To capture accurately all the required dynamics between the valve and hinges to prevent collision and protrusion it is necessary to increase significantly the number of multigrid cycles. While a robust algebraic multigrid algorithm with more appropriate prolongation and restriction operators might remedy this problem we opt here for a simple yet effective local alteration to our smoother.

We maintain the standard Gauss-Seidel relaxation but in addition we also perform a direct solve, via Gaussian elimination, of a small subset of the problem. Because we are only concerned with the high accuracy needed to prevent collision between the valve and hinge we isolate those points close to the hinge (see Fig. 6.4). Holding all other points outside this region fixed we then proceed to solve the resulting reduced system. Afterward we perform the standard Gauss-Seidel sweep to smooth the interface between the points inside and outside the solved region. Because of the small size of the selected local region, this procedure is quite inexpensive and does not contribute in a significant manner to the cost of the multigrid iteration.

6.1.5 Numerical results

The initial condition is a rest configuration of all links and tethers. The valve rests between the two hinges and is parallel to the y-axis. The imposed flow is $v_{flow} = 100$ and all spring constants are taken to be $k = 10^9 l^{-2}$, where $l = h/2$ is the length between nodes in our valve. The magnitude of the velocities here reaches approximately 10 units, much smaller than that in the elliptical membrane case. Taking a characteristic length to be the length of the heart valve we obtain a Reynolds number of about 5. The smaller speeds lead to a mild CFL constraint and this allows our semi-implicit approach to show its power, even though now

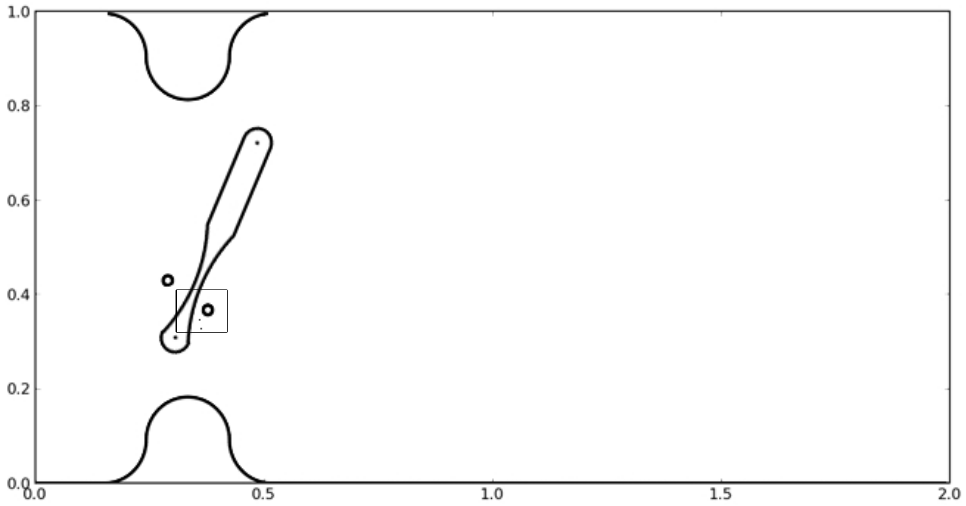


Figure 6.4: Configuration of our valve system after some period of time. The small box contains the subsystem of fiber points we solve exactly.

$N_B \approx 4N$ and the cost of an implicit time-step is roughly 20 times that of an explicit one.

For the FE/BE simulation Δt is again taken to be the maximum allowable while retaining stability, with $\Delta t = 30hk^{-1/2}$. For the proposed semi-implicit approach we fix $\Delta t = 0.0025$, well below the CFL restraint. With the constant imposed flow, the valve will simply open. The CPU times for the explicit simulations are given in Table 6.2 with the results from the implicit simulations following in Table 6.3. With a less restrictive CFL constraint the semi-implicit approach becomes $O(1000)$ faster than FE/BE. Stability-wise, Δt may be taken even larger for the semi-implicit method, up to the limit of the CFL restraint, leading to even greater savings in CPU time. However, this will typically increase leakage through the immersed boundary and eventually allow the valve to collide and pass through the hinges.

To open and close the valve we consider a time dependent imposed flow $v_{flow}(t)$. We define

$$v_{flow}(t) = \begin{cases} 60000t(0.1-t) & t < 0.1 \\ -60000(t-0.1)(0.2-t) & t \geq 0.1 \end{cases}. \tag{6.27}$$

A sequence of snap-shots of the motion of the valve as it opens and closes is depicted in Fig. 6.5 where also flooded contours of the vorticity are shown. Initially, there is a noticeable generation of (positive) vorticity at the upper and lower tips

of the valve as the valve opens as well as in the upper cushion. As the valve reverses its motion the vorticity in those same areas evolves toward positive values and becomes large and localized as the valve closes.

Table 6.1: The number of Lagrangian nodes N_B and the maximum stable timestep Δt for a FE/BE method are given for increasing values of N.

N	N_B	Δt
128	520	$1.15 \cdot 10^{-7}$
256	1053	$2.89 \cdot 10^{-8}$
384	1592	$1.29 \cdot 10^{-8}$
512	2122	$7.24 \cdot 10^{-9}$

Table 6.2: Heart valve simulation with Forward Euler/Backward Euler scheme. The total CPU time taken to run the simulation up to time T is given, with the average CPU time per timestep given. * denotes an extrapolated value.

N	Average	$T = 0.01$	$T = 0.05$	$T = 0.1$	$T = 0.5$
128	0.024	2110.49	10508.95	21017.90*	105089.54*
256	0.109	37687.16*	188435.83*	376871.66*	1884358.32*
384	0.249	193168.57*	965842.85*	1931685.70*	9658428.53*
512	0.503	694291.69*	3471458.49*	6942916.98*	34714584.93*

Table 6.3: Heart valve simulation with implicit scheme. The total CPU time taken to run the simulation up to time T is given, with the average CPU time per timestep given. $\Delta t = 0.0025$.

N	Average	$T = 0.01$	$T = 0.05$	$T = 0.1$	$T = 0.5$
128	0.588	2.001	10.436	21.155	117.767
256	2.014	7.640	38.171	76.297	402.749
384	4.747	18.735	93.266	186.270	949.409
512	9.538	34.984	174.702	348.531	1907.635

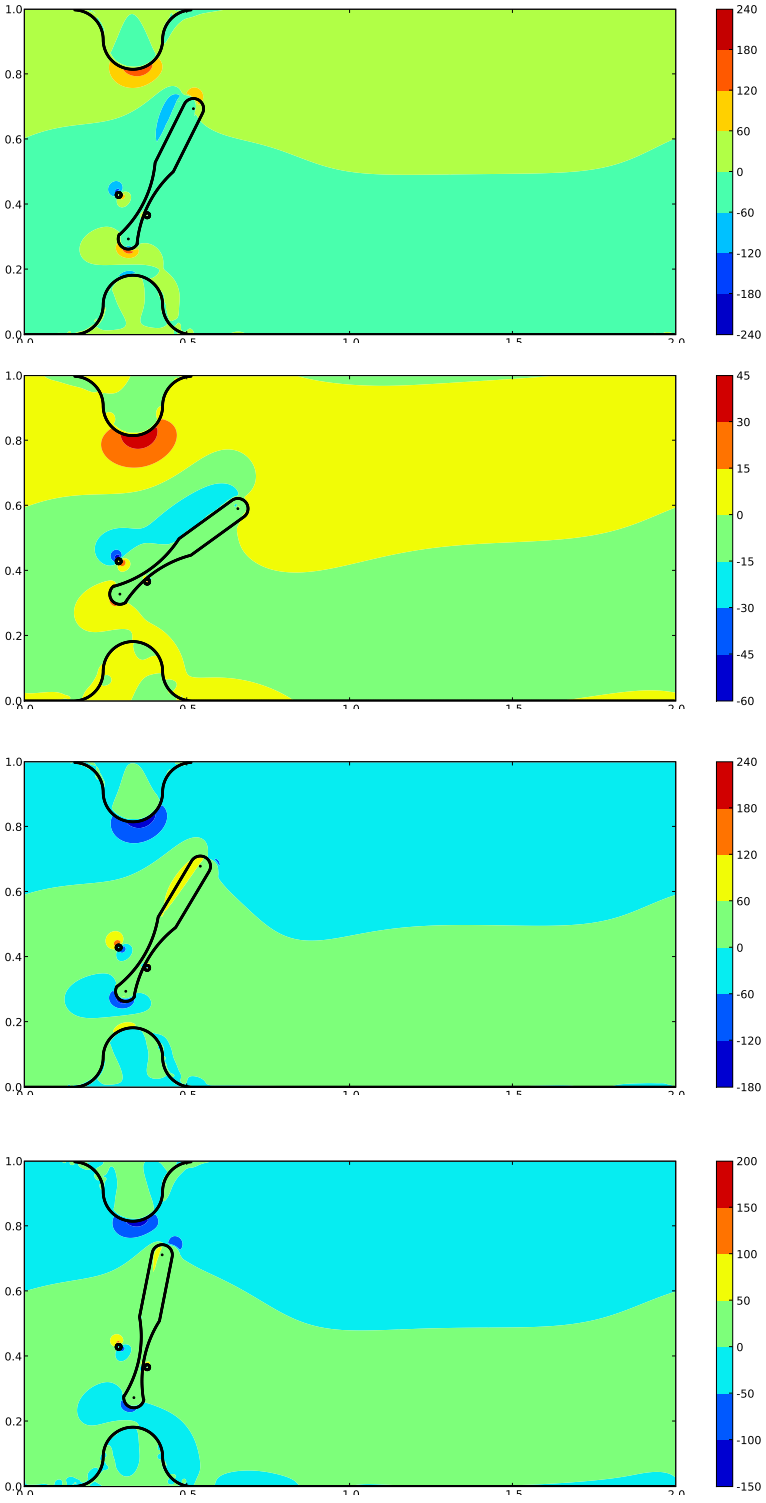


Figure 6.5: Vorticity field of our valve system through time. $N = 512, N_B = 2082, \Delta t = 0.0025$. Frames shown at total simulation time $T = 0.0625, 0.1125, 0.1750, 0.2000$

Chapter 7

Peristaltic pumping, an application

We turn now to a fully featured application of the developed implicit methodology: peristalsis of a viscoelastic fluid. Peristaltic pumping is a mechanism for transporting fluid prevalently utilized in both biological organisms and human designed systems. The mechanism induces flow in a channel by contracting the channel's walls in a set pattern. Peristaltic pumping of Newtonian fluids is well understood, and recent investigations have begun exploring the nature of peristalsis of non-Newtonian fluids. Here we investigate the particular case of peristalsis of an Oldroyd- B fluid via an Immersed Boundary method. Previous studies have investigated this same model, but have been hampered by stiffness constraints. Our intent here is to more fully explore the entire parameter regime pertaining to the model, including regimes where the polymeric stress is many orders of magnitude larger than previously accessible to numerical experiments. In particular, we

consider large Weissenberg numbers and high channel occlusions. A new dynamic behavior is observed with substantial effects on the pumping flow rate, including evidence for a finite-time blow up of the Oldroyd- B equations.

7.1 Introduction

Peristalsis is the predominant mechanism of action in a plethora of biological phenomena, from earthworm mobility to gastrointestinal and esophageal transport. Peristalsis is utilized in many mechanical fluid pumps, often because of their ability to effectively transport highly viscous fluids. In both biological and mechanical systems, the fluid internal to the pump may be non-Newtonian. Such is the case for peristalsis in the oviduct and uterus where the transported biological fluid is highly complex.

Recent work has begun the investigation of peristalsis of an Oldroyd B (OB) fluid [28, 5]. A marked difference has been reported with respect to the Newtonian flow counterpart, particularly the non-linear relationship between mean flow through the pump and the Weissenberg number of the fluid. It has been noted in these works that the polymeric stress of the transported fluid can lead to very strong normal stresses at the interface with the pump. These stresses present a difficult numerical problem. In order to correctly model the structure of the

pump wall very stiff forces must be employed, leading to very small time-step constraints. Such numerical constraints (stiffness) are particularly pernicious for explicit Immersed Boundary (IB) methods, as used in [28, 5]. There, the numerical stiffness limited the range of parameters ameanable to investigate. In Section 7.3, we present a new numerical method coupling a viscoelastic fluid solver to a novel, highly efficient semi-implicit implementation of the IB method, presented in more detail in [4, 3]. The implicit methodology allows us to explore extreme parameter regimes previously out of reach of numerical algorithms. In particular, when the waves of peristalsis have very large amplitude, nearly occluding the channel, the normal stresses on the wall become extremely large. In addition, for very large Weissenberg numbers the transported fluid can develop very strong stresses, even for moderate occlusions. The implicit methodology allows us to robustly probe both of these regimes. Previous work has hinted at interesting behavior as the pump occlusion is increased, including substantial effects on the pump's rate of flow. We probe these effects for substantially larger occlusions than previously possible. In Section 7.4 we detail new behavior observed, including evidence for a finite-time blow up of the OB equations.

7.2 The Peristaltic Pump and Viscoelastic Fluid Models

We consider a peristaltic pump immersed in a period 2D domain $\Omega = [0,1]^2$. We model the peristaltic pump as two disconnected sinusoidal curves,

$$\mathbf{X}(t) = \left\{ \left(\xi, \frac{1}{2} + d(\xi, t) \right) \middle| \xi \in [0,1] \right\} \cup \left\{ \left(\xi, \frac{1}{2} - d(\xi, t) \right) \middle| \xi \in [0,1] \right\} \tag{7.1}$$

where

$$d(x,t) = \frac{\alpha}{2\pi}[1 + \chi \sin 2\pi(\xi - t)]. \tag{7.2}$$

Both the spatial and temporal period of the pump is fixed at 1. As time progresses the waves of peristalsis move from left to right, forcing the fluid to flow to the right (in aggregate). The parameter χ represents the occlusion ratio of the pump. The value $\chi = 0$ corresponds to a straight channel , with no waves of peristalsis, while $\chi = 1$ correspond to a completely occluded channel, with the peaks of each sinusoidal curve meeting at some point along the horizontal line $y = 1/2$. The parameter α controls the aspect ratio of the channel. For this work we fix $\alpha = 1.5$.

We model the interior and exterior of the valve as one continuous Oldroyd B (OB) incompressible fluid. The interaction between the valve and the fluid is

captured via the Immersed Boundary Method. The continuous equations are then

$$Re\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \nabla^2\mathbf{u} + \beta\nabla\cdot\mathbf{S} + \mathbf{f}, \tag{7.3}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{7.4}$$

$$\frac{\partial \mathbf{X}}{\partial t} = \mathbf{u}(\mathbf{X}, t), \tag{7.5}$$

$$\mathbf{S}^{\nabla} = -We^{-1}(\mathbf{S} - \mathbf{I}). \tag{7.6}$$

Here $\mathbf{f}$ can be seen as a Lagrange multiplier used to enforce the prescribed motion of the peristaltic waves. Due to the no slip boundary condition (7.5) we can only modulate $\mathbf{X}$ by modifying the fluid velocity $\mathbf{u}$. In practice, we will only enforce the prescribed position of $\mathbf{X}$ approximately by taking $\mathbf{f}$ to be a very stiff force binding the current configuration $\mathbf{X}$ to the desired prescribed position given by (7.1).

In (7.3), Re is the Reynolds number which is a measure of the viscous dissipation relative to inertial forces. The dimensionless term β specifies the strength of the viscoelastic force $\nabla\cdot\mathbf{S}$. Here, $\mathbf{S}$ is the deviatoric part of of the viscoelastic stress tensor and evolves according to the OB constitutive equation, given in (7.6). $\mathbf{S}^{\nabla}$ denotes the upper convected derivative of $\mathbf{S}$, namely

$$\mathbf{S}^{\nabla} = \frac{d\mathbf{S}}{dt} + \mathbf{u}\cdot\nabla\mathbf{S} - \nabla\mathbf{u}\cdot\mathbf{S} - \mathbf{S}\cdot\nabla\mathbf{u}^T. \tag{7.7}$$

We is a dimensionless parameter giving the ratio of the relaxation time of the polymeric stress $\mathbf{S}$ against some characteristic time scale of the fluid. We is

referred to as the Weissenberg number of the fluid. In the limit as $We \to 0$ the polymeric stress is fixed as the identity tensor $\mathbf{I}$ and the fluid becomes Newtonian. In general the larger the value of We the larger the non-Newtonian effect on the fluid.

Finally, the product βWe is the ratio of the polymeric viscosity to the solvent viscosity. Following Teran, Fauci, Shelly [28] we fix $\beta We = \frac{1}{2}$. As mentioned above, we choose the characteristic length scale to be 1, the width of our fluid domain Ω and the characteristic timescale we also take it to be 1, the period of the peristaltic pump. We fix the Reynolds number of our fluid at $Re = 1$. Throughout this work the only fluid parameter we change is the Weissenberg number We.

7.3 Methodology

We discretize the peristaltic pump $\mathbf{X}$ as a collection of N_B immersed points $\{\mathbf{X}_j\}$. The position of these points are not directly prescribed, rather we construct an artificial force to approximately constrain the immersed points to their respective positions. For each point $\mathbf{X}_j$ we define $\mathbf{T}_j$ to be the desired target position. We then induce a force $\mathbf{F}$ defined on immersed points given by

$$\mathbf{F} = \sigma(\mathbf{T} - \mathbf{X}). \tag{7.8}$$

The scalar variable σ is a numerical parameter. In the limit as $\sigma \to \infty$ we exactly constrain $\mathbf{X}$ to the desired configuration. In practice we take σ to be some large value. This can lead to instability in the resulting numerical simulations. We counter this by employing a new semi-implicit methodology as detailed in [4, 3]. With the semi-implicit method we can use values of σ multiple orders of magnitude larger than previously possible. For the large values of We and χ explored in this work we are required to take $\sigma = O(10^6)$ to maintain the structure of the pump. This large stiffness coefficient would lead to prohibitively small time-steps for previous explicit methods. In our numerical experiments our choice of σ reduces deviations in $\mathbf{X}$ from the target position $\mathbf{T}$ to less than .0005 units, even when the normal polymeric stresses at the boundary rise to values of 1000 and more.

We briefly overview the semi-implicit method here. The method is based on a semi-implicit discretization of the Navier-Stokes equations given by

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \mathbf{u}^n \cdot \nabla \mathbf{u}^n = -\mathbf{D}_h p^{n+1} + L_h \mathbf{u}^{n+1} + \mathbf{f}, \tag{7.9}$$

$$\mathbf{D}_h \cdot \mathbf{u}^{n+1} = 0, \tag{7.10}$$

$$\frac{\mathbf{X}^{n+1} - \mathbf{X}^n}{\Delta t} = \mathcal{S}_n^* \mathbf{u}^{n+1}. \tag{7.11}$$

Here a superscript n denotes a numerical approximation taken at the time $n\Delta t$ and Δt is the timestep. The spatial operators $\mathbf{D}_h$ and L_h are the standard, second order approximations to the gradient and the Laplacian, respectively. The

convection term $\mathbf{u}^n \cdot \nabla \mathbf{u}^n$ is handled separately via a third-order essentially non-oscillatory (ENO) scheme. The force $\mathbf{F}$ in (7.8) is defined at the immersed points and can not be directly used in the fluid equations (7.3)-(7.6). First we must *spread* the force onto the surrounding Eulerian grid. Likewise in equation (7.11) the fluid velocity is not given at the immersed points, so we must *interpolate* the fluid velocity. To achieve the spreading and interpolation we define

$$(\mathcal{S}_n G)(\mathbf{x}) = \sum_{s \in \mathcal{G}_B} G(s) \delta_h(\mathbf{x} - \mathbf{X}^n(s)) h_B, \tag{7.12}$$

$$(\mathcal{S}_n^* w)(s) = \sum_{\mathbf{x} \in \mathcal{G}_\Omega} w(\mathbf{x}) \delta_h(\mathbf{x} - \mathbf{X}^n(s)) h^2, \tag{7.13}$$

known as the spreading and interpolation operators, respectively. Here $\delta_h(\mathbf{x}) \equiv d_h(\mathbf{x}_0) d_h(\mathbf{x}_1)$ is an approximation to the two-dimensional Dirac delta distribution and d_h is given by

$$d_h(r) = \begin{cases} \frac{1}{4h}\left(1 + \cos(\frac{\pi r}{2h})\right) & \text{if } |r| \leq 2h, \\ 0 & \text{otherwise.} \end{cases} \tag{7.14}$$

We refer to these operators as lagged because the interface configuration $\mathbf{X}^n$ is used instead of the future configuration $\mathbf{X}^{n+1}$.

Utilizing $\mathcal{S}_n$ and $\mathcal{S}_n^*$ we now specify the form of $\mathbf{f}$ in (7.9).

$$\mathbf{f} = \sigma \mathcal{S}_n(\mathbf{T}^{n+1} - \mathbf{X}^{n+1}) + \beta \mathbf{D}_h \cdot \mathbf{S}^n, \tag{7.15}$$

encapsulating both the artificial force on the immersed points, as well as the additional force coming from the polymeric stress. We thus consider the polymeric

stress fixed as we update the fluid. Once we have an updated fluid velocity $\mathbf{U}^{n+1}$ we will then calculate an updated value for the stress $\mathbf{S}^{n+1}$.

We may formally eliminate $\mathbf{u}^{n+1}$ from the equations (7.3)-(7.6), arriving at a *linear* system of the form

$$\mathbf{X}^{n+1} = \sigma \mathcal{M}_n(\mathbf{T}^{n+1} - \mathbf{X}^{n+1}) + \mathbf{b}^n, . \tag{7.16}$$

We refer to this as the Lagrangian system. $\mathcal{M}_n$ is an operator acting on a force distribution $\mathbf{F}$ and returning the resulting displacement of immersed points due to the induced fluid flow from the spread force $\mathcal{S}_n\mathbf{F}$. Critically, both $\mathcal{M}_n$ and $\mathbf{b}^n$ can be explicitly constructed in an efficient manner, yielding a $2N_B \times 2N_B$ matrix and $2N_B$ vector respectively. We detail this construction in our previous papers [4, 3]. The resulting system can be rewritten as a simple matrix inversion problem

$$(I + \sigma \mathcal{M}_n)\mathbf{X}^{n+1} = \sigma \mathbf{T}^{n+1} + \mathbf{b}^n, . \tag{7.17}$$

The matrix $I + \sigma\mathcal{M}_n$ is positive-definite and the system (7.17) can be inverted in a number of ways, the most efficient being multigrid. For the present work conjugate gradient suffices to provide a nearly optimal solver. Once we have solved (7.17) for the updated configuration $\mathbf{X}^{n+1}$ we may then calculate the updated fluid velocity via (7.3)-(7.10).

Finally we must calculate the updated polymeric stress $\mathbf{S}^{n+1}$. Recall the constitutive equation (7.6) given as $\mathbf{S}^{\nabla} = -We^{-1}(\mathbf{S} - \mathbf{I})$. We discretize this in space

as

$$\frac{d\mathbf{S}}{dt} + \mathbf{u}^{n+1} \cdot \nabla \mathbf{S}^n = \mathbf{D}_h \mathbf{u}^{n+1} \cdot \mathbf{S}^n + \mathbf{S}^n \cdot \mathbf{D}_h (\mathbf{u}^{n+1})^T + \frac{1}{We}(\mathbf{I} - \mathbf{S}^n). \tag{7.18}$$

Here the convective term $\mathbf{u}^{n+1} \cdot \nabla \mathbf{S}^n$ is calculated via the third-order ENO scheme, as with the convection in the momentum equation (7.9). We further discretize in time via a second-order total variation diminishing (TVD) Runge-Kutta method. If we define the Euler update operator

$$E(\mathbf{S}) = \mathbf{S} + \Delta t \left[\mathbf{D}_h \mathbf{u}^{n+1} \cdot \mathbf{S} + \mathbf{S} \cdot \mathbf{D}_h (\mathbf{u}^{n+1})^T + \frac{1}{We}(\mathbf{I} - \mathbf{S}) - \mathbf{u}^{n+1} \cdot \nabla \mathbf{S} \right], \tag{7.19}$$

then our Runge-Kutta update is given as $\mathbf{S}^{n+1} = (\mathbf{S}^n + E(E(\mathbf{S}^n)))/2$.

7.3.1 Summary of algorithm

Given $\mathbf{X}^n$, $\mathbf{u}^n$, $\mathbf{S}^n$ at time $t = n\Delta t$, a complete timestep may be summarized as follows

1. Calculate the fluid matrix $\mathcal{M}_n$ and the explicit term $\mathbf{b}^n$.

2. Solve for the updated pump configuration $\mathbf{X}^{n+1}$ via the matrix problem $(I + \sigma \mathcal{M}_n)\mathbf{X}^{n+1} = \sigma \mathbf{T}^{n+1} + \mathbf{b}^n$.

3. Calculate the updated fluid velocity $\mathbf{u}^{n+1}$ via the Stokes problem (7.9)-(7.10).

4. Calculate the updated polymeric stress $\mathbf{S}^{n+1}$ via the second-order TVD Runge-Kutta method, $\mathbf{S}^{n+1} = (\mathbf{S}^n + E(E(\mathbf{S}^n)))/2$.

7.4 Results

Here we summarize the results of our numerical simulations. We first present evidence to validate our methodology and then proceed to explore the extreme parameter regimes corresponding to very large wave amplitudes (occlusions) and very large Weissenberg numbers. We observe a new rich behavior which includes the potential formation of a finite time blow-up in the Oldroyd B model.

Two important values we will be observing are the flux and the normalized mean flow. We define the flux as

$$Q(x,t) = \int_{0.5-d(x,t)}^{0.5+d(x,t)} u(x,y)dy, \tag{7.20}$$

which is the total mass flux within the peristaltic pump across a specified vertical line. In this work we always take $x = 0.5$ and write $Q(t) = Q(x,t)$ for convenience. Closely related to the flux is the normalized mean flow, given by

$$\Theta(t) = \frac{\pi}{\alpha\chi}\int_{t}^{t+1} Q(t)dt. \tag{7.21}$$

Here we simply average and normalize the flux over one period of the peristaltic pump. The normalization leads to a dimensionless value, and is chosen such that $\Theta = 1$ when $\chi = 1$, regardless of the nature of the underlying fluid.

7.4.1 Resolution study and comparison to analytical results

We seek to validate our simulations through two approaches, by performing a spatial resolution study as well as by comparing simulations to known analytical results.

For the resolution study we consider the case when $\chi = 0.25$ and $We = 1$. We fix the time-step $\Delta t = 0.00025$ and run the simulation for a range of N from $N = 256$ up to $N = 2048$. We consider the $N = 2048$ run as our reference solution ("exact") and compute the errors of the other simulations in relation to this $N = 2048$ case. The results are given in Figure 7.1. In Figure 7.1(a) we examine the difference in flow Q at time $t = 1$. We observe less than a 1% relative error in Q for $N \geq 1024$. In Figure 7.1(b) we consider the sup norm of the difference of the xx-component of the stress. In both cases we observe slightly better than first order convergence, as expected from the convergence of the IB method.

There is a known analytic approximation of the mean flow Θ for the case of a Newtonian Stokes flow ($Re = 0$ and $We = 0$) due to Jaffrin and Shapiro [10]. The formula is given as

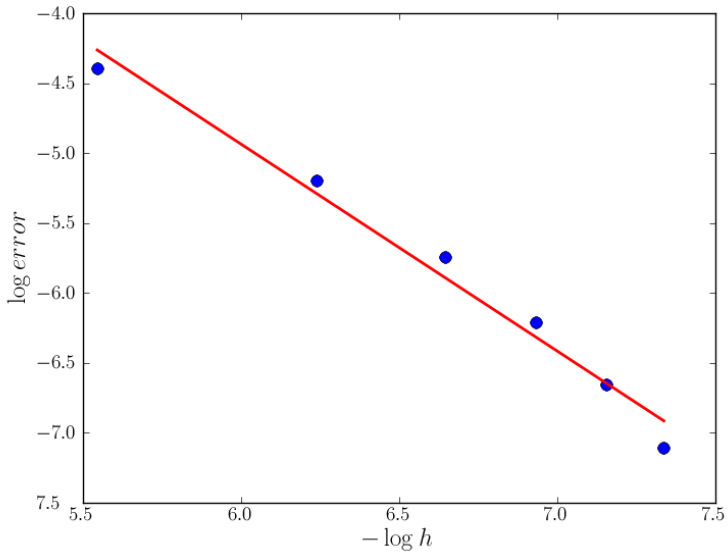
$$\Theta_J = \frac{15\chi^2 + 2\alpha^2[4(1-\chi^2)^{5/2} + (7\chi^2 - 4)(1-\chi^2)]}{\chi[5(2+\chi^2) + 6\alpha^2\chi^2(1-\chi^2)]} \tag{7.22}$$

In the Newtonian Stokes case Θ is a constant, thus the time of evaluation is not important so long as $T \geq 1$. In contrast our Navier-Stokes fluid has convection and takes a finite amount of time ($t < 1$) to reach a steady state. We take $t = 1$ and calculate Θ for a full range of χ, from 0 up to 0.95. We note that as $\chi \to 1$ the normal stresses on the walls of the pump become enormous. While the value $\sigma = 10^6$ is sufficient for $\chi = 0.95$, we note that for the case $\chi = 1$ even the extremely large value $\sigma = 10^8$ is insufficient to maintain the shape of the pump. In this extreme case the implicit methodology remains stable but the distortion of the geometry is great enough to warrant omitting the calculated mean flow.

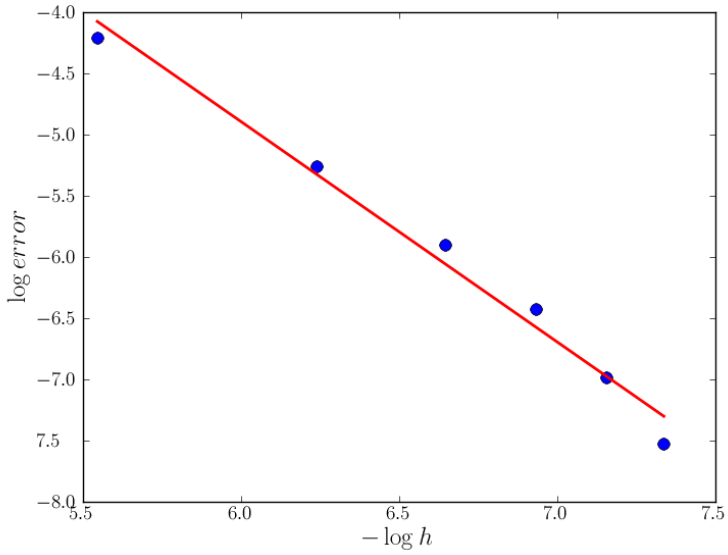
The comparison of our simulated results, which have been checked under time and space refinement, to Jaffrin and Shapiro's formula can be seen in Figure 7.2. We see reasonably good agreement. We note, however, that for small to moderate occlusion ratios, the Stokes-OB results in [28] and our own results from explicit simulations provide slightly better agreement with the analytic formula. From our experience, we tend to believe that this small difference might be attributed to the somewhat better volume conservation of the explicit method. Importantly, however, explicit simulations for the cases when $\chi > 0.5$ become impractical.

7.4.2 High occlusion, χ=.8

Previous investigations utilizing the IB method to study peristaltic pumping have been limited to occlusions $\chi \leq 0.5$. With our implicit methodology we are



(a) Error of normalized flow $|Q - \tilde{Q}|$, $m = -1.479$.



(b) Error of maximum stress $\left\|S_{xx} - \tilde{S}_{xx}\right\|_\infty$, $m = -1.800$.

Figure 7.1: Spatial resolution study for decreasing values of h. Value specified is plotted against h in a log-log plot. Variables with a tilde, $\tilde{\square}$, refer to values coming from an $N = 2048$ simulation taken to be an exact solution. Dashed lines are linear fits with specified slope m.

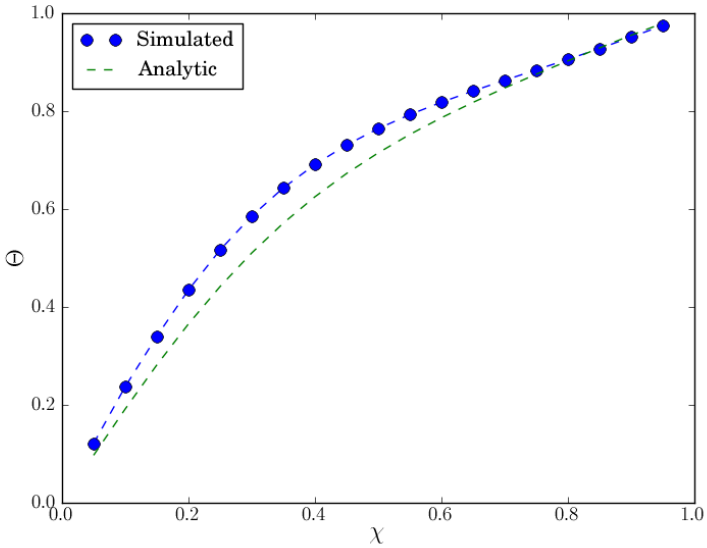


Figure 7.2: Normalized mean flow calculated both via Jaffrin and Shapiro's formula and numerical simulation for the full range of $\chi = 0$ to $\chi = 1$.

able to explore for the first time occlusions in the range $0.5 < \chi < 1$ which induce extremely large polymeric stress forces on the peristaltic pump. Here we focus specifically on the case $\chi = 0.8$ and $We = 5$.

In Figures 7.9 through 7.12 we plot the polymeric stress evolution over time, for $N = 1024$ and $\Delta t = 0.00025$. We see first that the xx component of stress develops a strong concentration at the channel's neck, reaching a value of 360 at time $t = 0.5$. Soon after, at time $t = 1.5$, the xx-component is further concentrated into a nearly horizontal line, reaching a peak value of 18000. The two additional components $\mathbf{S}_{xy}$ and $\mathbf{S}_{yy}$ develop strong interfaces in identical locations, reaching values of 2400 and 6400 respectively. These strong interfaces effectively block off

the inner interior of the peristaltic pump from any outside influences. This can be clearly seen in the vorticity plot, where the vorticity inside the interfaces is nearly zero. Outside the interfaces, near the boundary of peristaltic pump, we see strong, nearly uniform vorticity. As time progresses further, up to $t = 4.5$, we see the formation of more complex localized structures which again have strong effects on the vorticity. These fine structures eventually leads to instabilities and we are unable to reliably compute for longer than $t = 7$ for $N \geq 512$.

An important observation is that the strong thin line observed in the xx-component of the stress at time $t = 1.5$ dissipates over time and is significantly weaker at time $t = 4.5$. We suspect that this dissipation is due to numerical diffusion inherent to the ENO advection scheme. To investigate this further we perform a resolution study, focusing on the peak value of $\mathbf{S}_{xx}$ over the time interval $t = 0$ to $t = 1.5$.

First, in Figure 7.5(a) we plot $\mathbf{S}_{xx}$ over time for values of N ranging from $N = 256$ to $N = 2048$. We see that there is fast, nearly exponential growth, that levels off near $t = 1.5$. As N increases the maximum stress increases as well. We analyze the dependence on N in Figure 7.5(b). Here we plot $\|\mathbf{S}_{xx}\|_\infty$ at $t = 1.25$ versus $\log N$. We see that $\|\mathbf{S}_{xx}\|_\infty$ grows almost exactly proportional to $\log N$, yielding the approximate formula

$$\|\mathbf{S}_{xx}\|_\infty \approx 5286 \log N. \tag{7.23}$$

It is unclear if the relationship (7.23) persists for arbitrarily large N. If this were the case then it would be evidence for a finite time blow up in the Oldroyd-B model for peristaltic pumping. To provide additional evidence we attempt to fit an exponential growth model of the form $C_N e^{\beta_N t}$ to the stress $\|\mathbf{S}_{xx}\|_\infty$ over the time interval $(0.36, 0.70)$. For $N = 2048$ the best fit is given by $C = 10, \beta = 7.68$. The fit is qualitatively good, as seen in Figure 7.6(a). We proceed to compute the best fit for various values of N and investigate the sequence $\{\beta_N\}$. In Figure 7.6(b) we plot β_N with respect to $\log N$ and notice a distinct linear trend. A linear fit yields $\beta_N \approx 0.3533 \log N$. We thus again see sharper growth as $N \to \infty$. This apparently unbounded behavior for the exponent points in the direction of a potential finite-time singularity in the polymeric stress.

The simulation for $\chi = 0.8$ is very challenging, involving very large stress and equally large restorative forces from the immersed structure. Very sharp interfaces develop in both the vorticity and stress. We present evidence that we are properly resolving these sharp interfaces. In Figure 7.3 we plot a simulation at time $t = 1.5$ for the case $\chi = 0.8, We = 5$. We compare the vorticity fields for both $N = 512$ and $N = 1024$ and note that the structure is qualitatively identical. Closer examination of the $N = 512$ plot reveals slightly blurred interfaces, indicative of the greater numerical diffusion at lower resolutions. We observe similar results when we investigate the stress $\mathbf{S}$. We look at the Newtonian case in Fig-

ure 7.4 and note that the vorticity is well resolved. Indeed, we have observed that for the Newtonian case the vorticity is well resolved even for substantially lower resolutions, down to $N = 256$. The difference between the vorticity distributions of Newtonian and viscoelastic cases is striking.

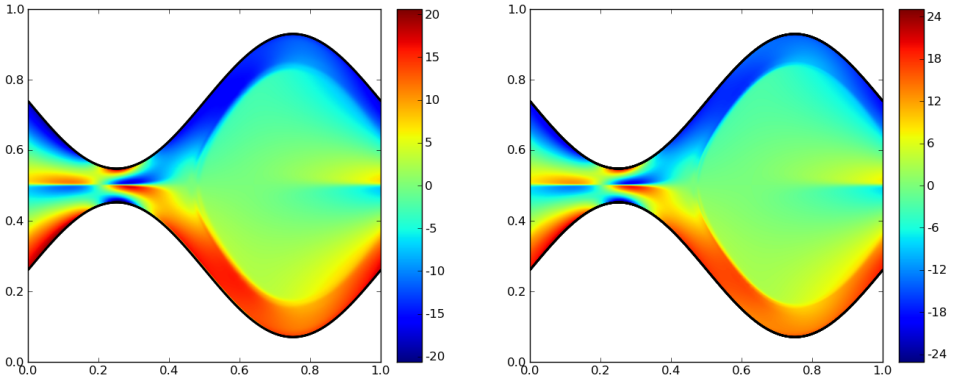


Figure 7.3: A comparison of the vorticity for a valve with $\chi = .8$ and $We = 5$ at time $t = 1.5$ for $N = 512$ and $N = 1024$ on the left and right respectively.

7.4.3 High Weissenberg number

Large values of We lead to large normal forces on the walls of the peristaltic pump. These forces in turn require large stiffness constants σ to properly maintain the prescribed shape of the walls. Previous investigations by Teran, Fauci and Shelley [28] and Chrispell and Fauci [5] were limited to Weissenberg numbers $We \leq 5$. With our superior methodology we are able to investigate for the first time higher Weissenberg numbers, up to and beyond $We = 100$.

We investigate first the case $We = 55$, $\chi = 0.5$. In Figures 7.13 through 7.16 we plot the polymeric stress evolution over time, for $N = 1024$ and $dt = .00025$. We note first that all components of the stress $\mathbf{S}_{xx}, \mathbf{S}_{xy}, \mathbf{S}_{yy}$ develop multilayered interfaces by time $t = 8.4$. As time progresses, however, these interfaces merge into smooth, simple regions. By time $t = 18.9$ no indication is left of the intricate,

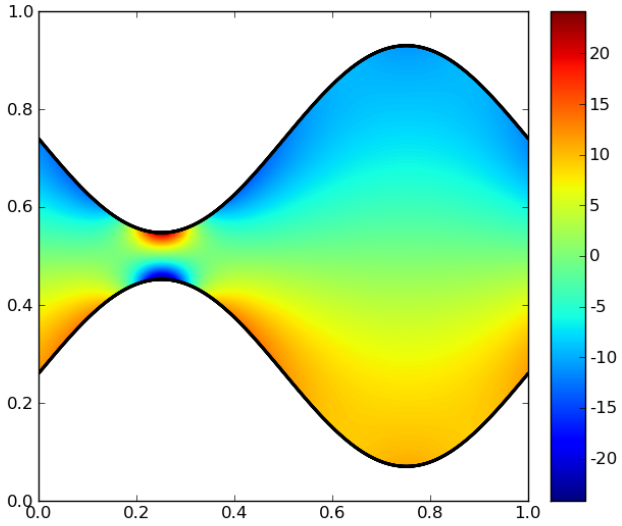
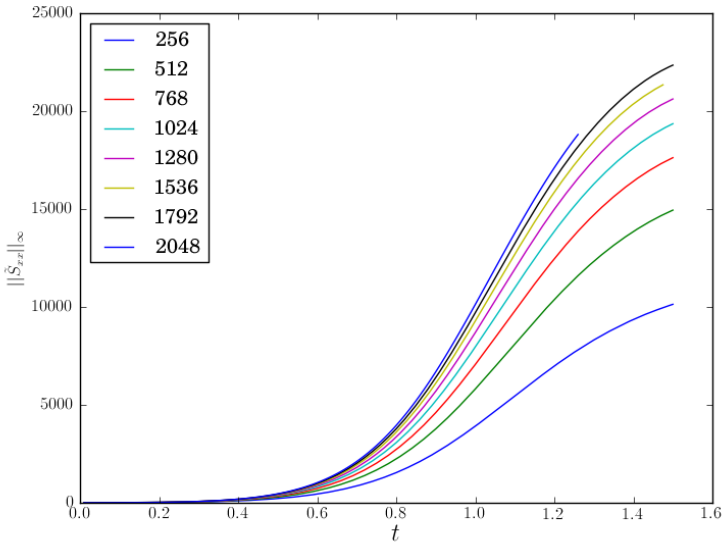


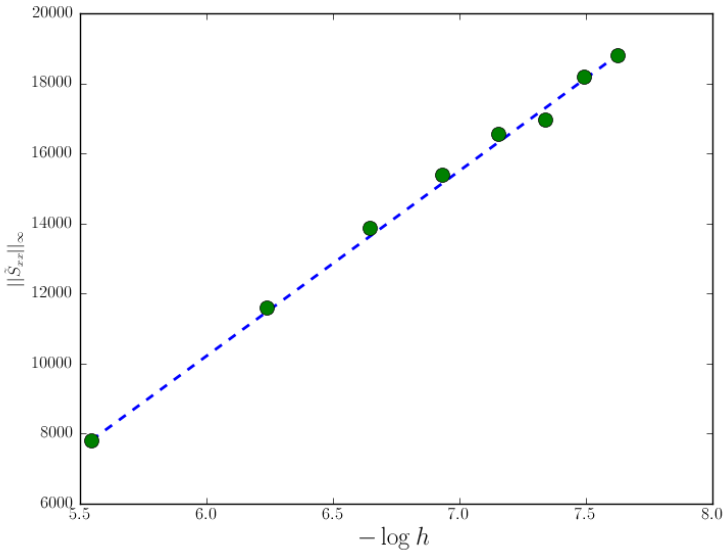
Figure 7.4: Vorticity for a valve with occlusion $\chi = .8$ in a Newtonian flow at time $t = 1.5$.

multilayered interfaces. We suspect that this smoothing is due to the numerical diffusion inherent to the ENO convection scheme.

An interesting question is how the normalized mean flow Θ responds to changes in We and χ. In Figure 7.7 we plot Θ at time $t = 15$ (Figure 7.7(a)) and $t = 50$ (Figure 7.7(b)) for $We = 0$ up to $We = 105$, and for $\chi = 0$ up to $\chi = 0.75$. We note a modest effect of the Weissenberg number on Θ, with higher values of We corresponding to higher mean flows. The dependence of Θ on t is more complicated. For $We \leq 5$ it appears that a semi-steady state is eventually reached. The time required to reach this semi-steady state was given to be $3We$ in [28, 5].

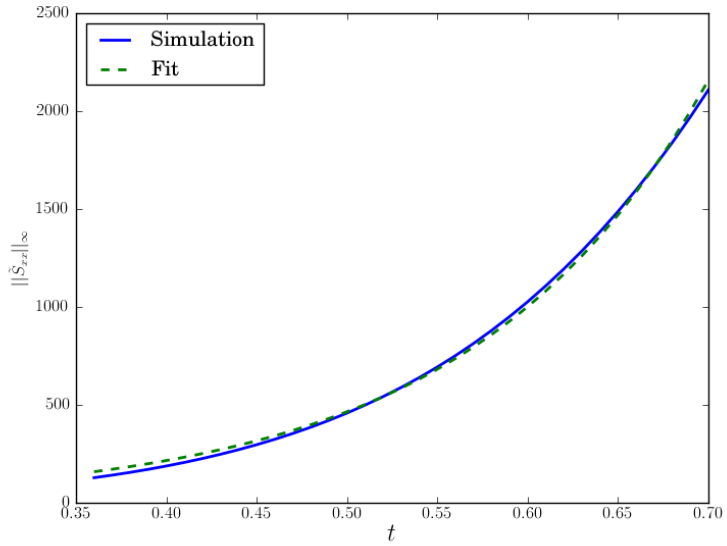


(a) $\left\|\bar{S}_{xx}(t)\right\|_\infty$ over the time interval $(0, 1.5)$ for various values of N.

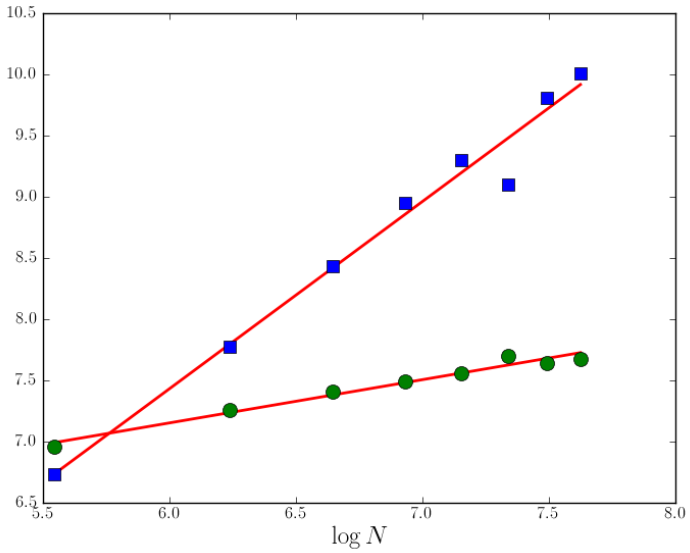


(b) $\left\|\bar{S}_{xx}(t)\right\|_\infty$ at time $t = 1.25$, plotted against $-\log h$, as N varies from 256 to 2048. Dashed line is a linear fit with slope $m = 5286$

Figure 7.5: Resolution study of the sup norm of the stress component S_{xx} along the horizontal line of symmetry, denoted as $\left\|\bar{S}_{xx}(t)\right\|_\infty$ at time t.



(a) $\left\|\bar{S}_{xx}(t)\right\|_\infty$ over the time interval $(0.36, 0.70)$ for $N = 2048$. Matched fit is the exponential curve $10e^{7.68t}$.



(b) β_N ($\circ$) and C_N ($\square$) with respect to $\log N$. The solid line is a linear fit with slope $m = 0.3533$

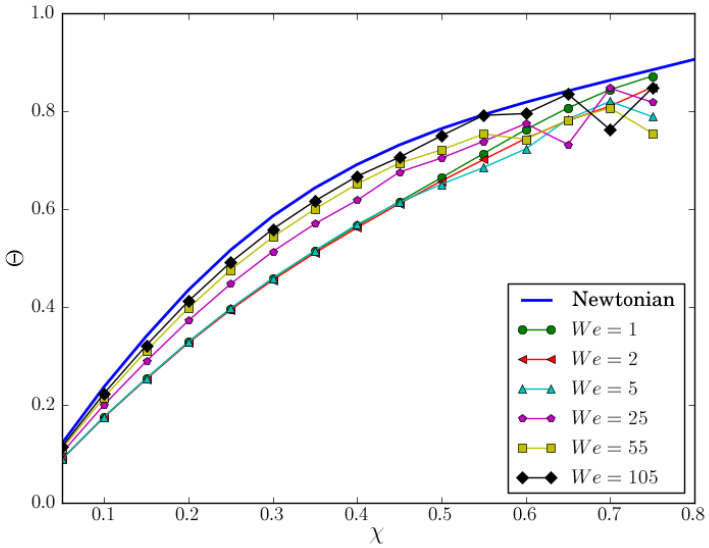
Figure 7.6: Consider $\left\|\bar{S}_{xx}(t)\right\|_\infty$ over the timer interval $(0.36, 0.70)$ for a specified resolution N. We compute a best fit of the form $C_N e^{\beta_N t}$, in the L^2 sense.

When we examine the same regimes as studied in [28, 5] ($\chi \leq 0.5$, $We \leq 5$), we see a similar requirement.

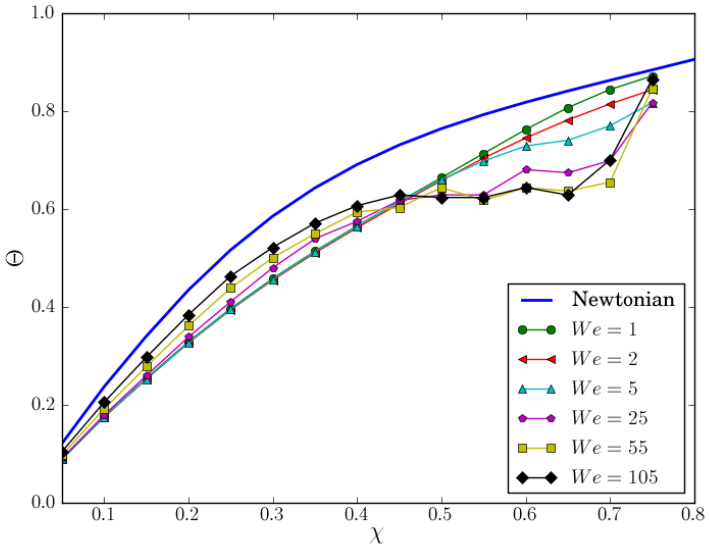
For higher We we report substantially different behavior. We examine the mean flow over time in Figure 7.8. For $\chi = 0.4$ and $We = 5$, displayed in Figure 7.8(a), we see relatively little variation in mean flow over time, consistent with previous results. For $We \geq 25$, however, we see substantially more complex behavior. For $We = 25$ the mean flow appears to become reasonably consistent after $t = 70$, despite large variations. For $We = 55$ and $We = 105$ no semi-steady state is reached for our simulation times ($T < 150$). For $\chi = 0.6$ we see long term variation in the mean flow even for the small value of $We = 5$. The flow appears to reach and maintain a semi-steady state for a long duration of time, from $t = 20$ to $t = 70$ (50 periods of the pump), but then has a sudden drop in mean flow at time $t = 70$.

Examining the structure of the stress over time we see that there is symmetry breaking in the stress field at approximately the same time as the drop in mean flow. We observe a similar break in symmetry for the other long time scale simulations. Such an instability is noted as well in [30]. There symmetry breaking is noted for higher Weissenberg numbers $We \geq 5$, as we observe here. We suspect that it is this instability that is preventing a semi-steady behavior from emerging

in the mean flow, and expect that the behavior will not become semi-steady even for greatly longer simulations.

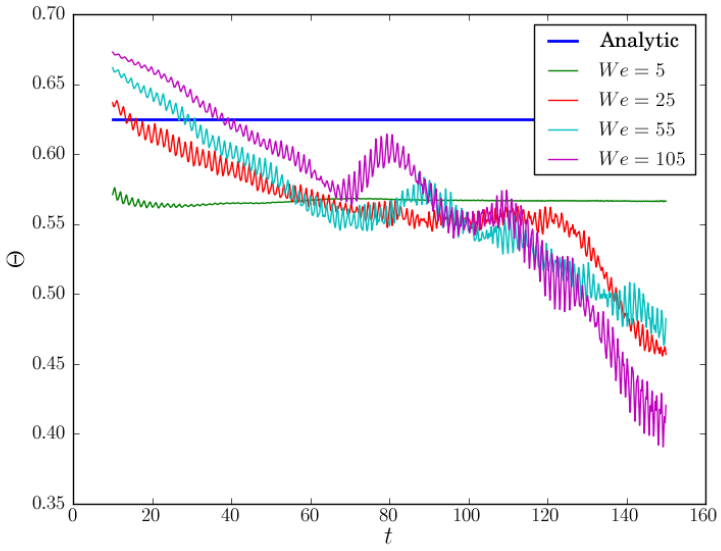


(a) Normalized mean flow at time $t = 15$ for various values of We and χ.

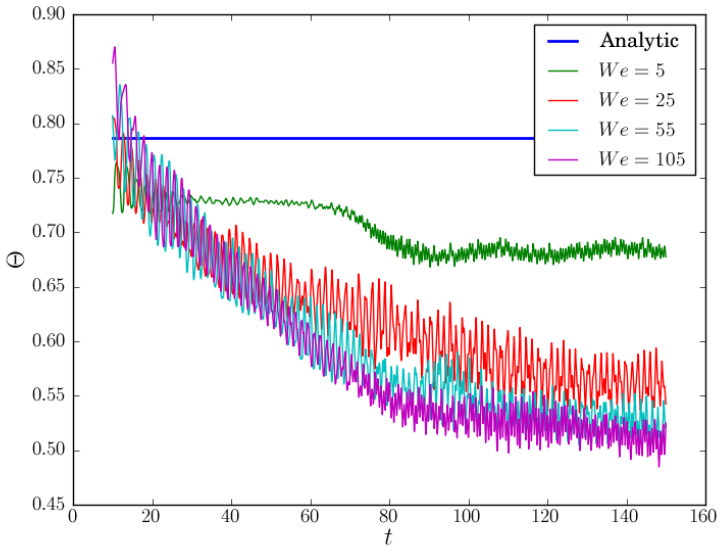


(b) Normalized mean flow at time $t = 50$ for various values of We and χ.

Figure 7.7: Normalized mean flow at times $t = 15$ and $t = 50$ for various values of We and χ.



(a) $\chi = 0.4$



(b) $\chi = 0.6$

Figure 7.8: Mean flow Θ over time for various values of We. Top and bottom plot are for $\chi = 0.4$ and $\chi = 0.6$ respectively.

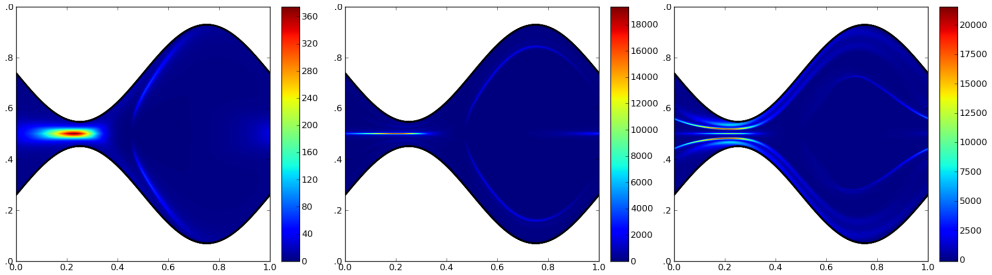


Figure 7.9: S_{xx} at times $t = .5$, $t = 1.5$, $t = 4.5$.

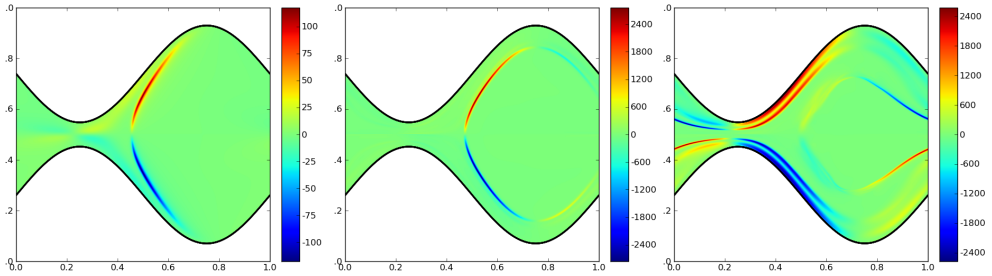


Figure 7.10: S_{xy} at times $t = .5$, $t = 1.5$, $t = 4.5$.

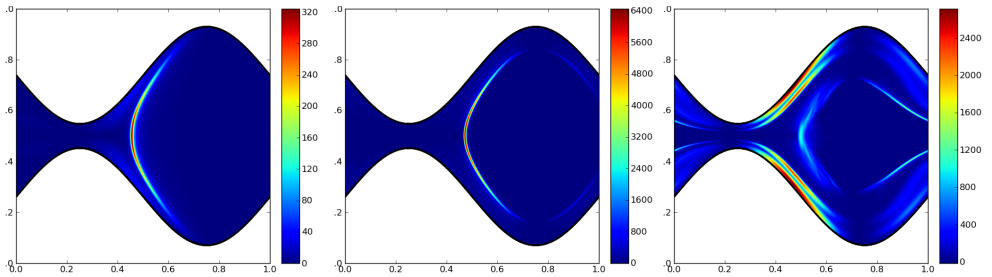


Figure 7.11: S_{yy} at times $t = .5$, $t = 1.5$, $t = 4.5$.

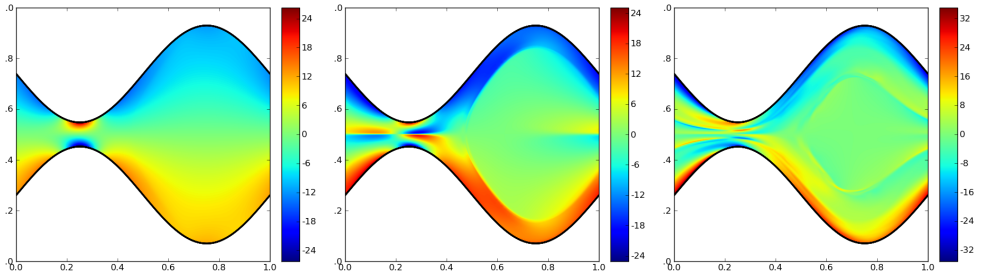


Figure 7.12: Vorticity of the fluid velocity at times $t = .5$, $t = 1.5$, $t = 4.5$.

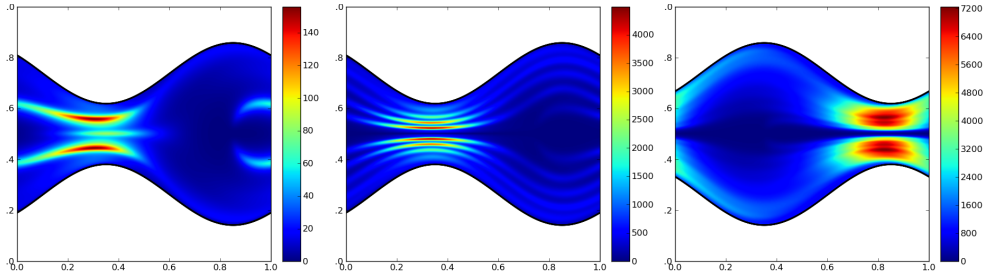


Figure 7.13: S_{xx} at times $t = 1.4$, $t = 8.4$, $t = 18.9$.

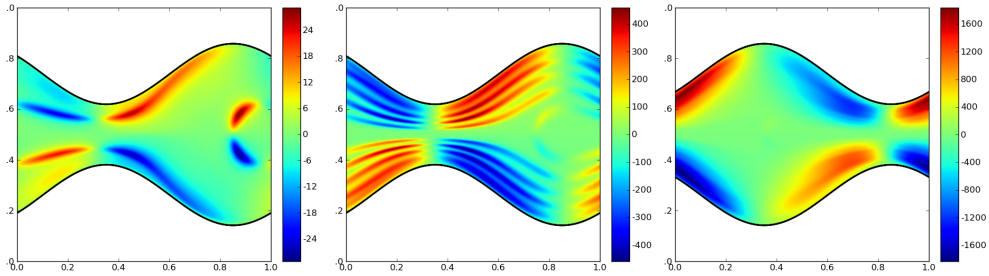


Figure 7.14: S_{xy} at times $t = 1.4$, $t = 8.4$, $t = 18.9$.

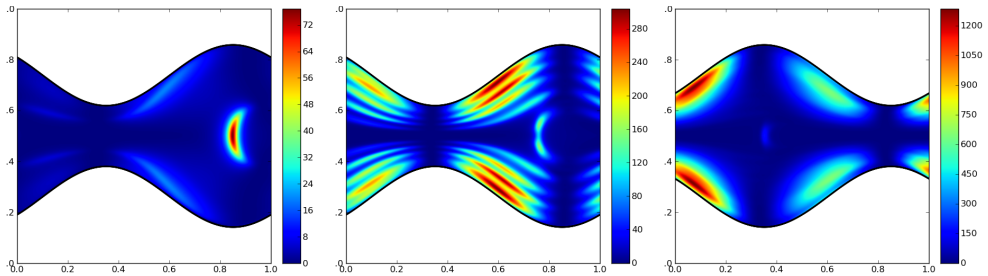


Figure 7.15: S_{yy} at times $t = 1.4$, $t = 8.4$, $t = 18.9$.

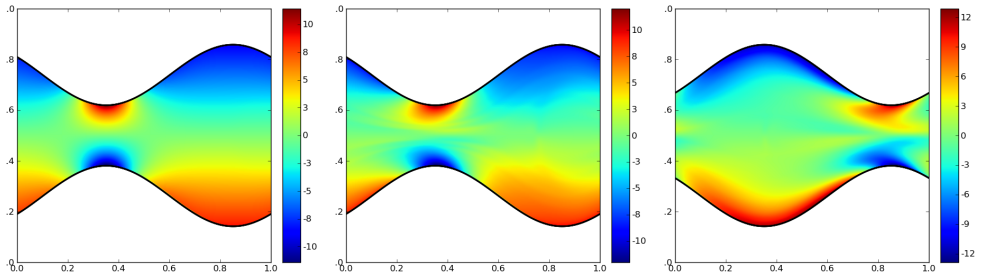


Figure 7.16: Vorticity of the fluid velocity at times $t = 1.4$, $t = 8.4$, $t = 18.9$.

7.5 Conclusion

Previous investigations into peristaltic pumping of OB fluids have revealed that the gross transport properties of the pump are highly sensitive to the underlying fluid parameters. Making use of a new implicit methodology we are able to explore a much broader parameter regime, observing new behavior both for high occlusion ratios and high Weissenberg numbers.

Future work will focus on extending this investigation to three dimensions, where preliminary results suggest new and different behavior. Importantly, in 3D we can investigate a plethora of novel asymmetrical pump geometries, including cork-screw like shapes. Preliminary results also suggest interesting differences when the OB fluid is replaced with a FENE-P fluid. Future work will explore these differences.

Chapter 8

Concluding Remarks

Implicit methods for alleviating the stiffness of the IB Method have been around for some time. Their implementation involves solving systems of equations typically thought to be prohibitively expensive. In this work we have explored new methods that allow for the efficient solution of the pertinent systems in a wide range of IB applications. In many cases we saw that the new methods allowed speedups of multiple orders of magnitude.

The approach used in this work was two fold. First, we developed new methods to accelerate computations of the form $\mathcal{M}_n\mathbf{F}$, either through a matrix method, shown in Chapter 4 to be well suited for many 2D applications, or through a treecode method, shown in Chapter 5 to be well suited for many 3D applications.

Second, we explored new iterative techniques for solving the linear system (3.15). In Chapter 4 we explored a multigrid approach, and in Chapter 6 we

explored the splitting method, suitable to cases where simple Krylov methods are inapplicable.

Ultimately, however, the specific form of $\mathcal{A}_{h_B}$ can lead to an arbitrarily difficult system of equations to solve. In particular, if the Jacobian of $\mathcal{A}_{h_B}$ is nondefinite and has eigenvalues of mixed magnitude, the iterative methods developed here will be ineffective. We are confident, however, that future work, by ourselves and others, will fill in such gaps in the methodology.

Bibliography

[1] P. Atzberger, P. Kramer, and C. Peskin. A stochastic immersed boundary method for fluid-structure dynamics at microscopic length scales. *Journal of Computational Physics*, 224(2):1255–1292, 2007.

[2] R. Beyer Jr. A computational model of the tcochlea using the immersed boundary method. *Journal of Computational Physics*, 98(1):145–162, 1992.

[3] H. Ceniceros and J. Fisher. A fast, robust, and non-stiff immersed boundary method. *Journal of Computational Physics*, 2011.

[4] H. D. Ceniceros, J. E. Fisher, and A. M. Roma. Efficient solutions to robust, semi-implicit discretizations of the immersed boundary method. *Journal of Computational Physics*, 228(19):7137 – 7158, 2009.

[5] J. Chrispell and L. Fauci. Peristaltic pumping of solid particles immersed in a viscoelastic fluid. *Bulletin of the American Physical Society*, 55, 2010.

[6] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[7] Z. Gimbutas and V. Rokhlin. A generalized fast multipole method for nonoscillatory kernals. *SIAM Journal on Scientific Computing*, 24(3):796, 2002.

[8] T. Y. Hou and Z. Shi. An efficient semi-implicit immersed boundary method for the Navier-Stokes equations. *J. Comput. Phys.*, 227:8968–8991, 2008.

[9] T. Y. Hou and Z. Shi. Removing the stiffness of elastic force from the immersed boundary method for 2D Stokes equations. *J. Comput. Phys.*, 227:9138–9169, 2008.

[10] M. Jaffrin and A. Shapiro. Peristaltic pumping. *Annual Review of Fluid Mechanics*, 3(1):13–37, 1971.

[11] D. Le, J. White, J. Peraire, K. Lim, and B. Khoo. An implicit immersed boundary method for three-dimensional fluid-membrane interactions. *Journal of Computational Physics*, 228(22):8427–8445, 2009.

[12] P. Li, H. Johnston, and R. Krasny. A cartesian treecode for screened coulomb interactions. *Journal of Computational Physics*, 228(10):3858 – 3868, 2009.

[13] Y. Liu and W. Liu. Rheology of red blood cell aggregation by computer simulation. *Journal of Computational Physics*, 220(1):139–154, 2006.

[14] A. A. Mayo and C. S. Peskin. An implicit numerical method for fluid dynamics problems with immersed elastic boundaries. In A. Y. Cheer and C. P. V. Dam, editors, *Fluid Dynamics in Biology: Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference on Biofluiddynamics*, pages 261–277. American Mathematical Society, 1993.

[15] C. McQueen, D.M. & Peskin. A three-dimensional computer model of the human heart for studying cardiac fluid dynamics. *ACM SIGGRAPH Computer Graphics*, 34(1):56–60, February 2000.

[16] D. McQueen and C. Peskin. Shared-memory parallel vector implementation of the immersed boundary method for the computation of blood flow in the beating mammalian heart. *The Journal of Supercomputing*, 11(3):213–236, 1997.

[17] D. McQueen and C. Peskin. Heart simulation by an immersed boundary method with formal second-order accuracy and reduced numerical viscosity. *Mechanics for a New Mellennium*, pages 429–444, 2002.

[18] McQueen, D.M. & Peskin, C.S. A Three-Dimensional Method for Blood Flow in the Heart: (II) Contractile Fibers. *J. Co?mput. Phys.*, 82(2):289–297, April 1989.

[19] L. Miller and C. Peskin. When vortices stick: an aerodynamic transition in tiny insect flight. *Journal of experimental biology*, 207(17):3073, 2004.

[20] L. Miller and C. Peskin. Flexible clap and fling in tiny insect flight. *Journal of Experimental Biology*, 212(19):3076, 2009.

[21] Y. Mori. Convergence proof of the velocity field for a Stokes flow immersed boundary method. *Communications on Pure and Applied Mathematics*, 61:1213–1263, 2008.

[22] Y. Mori and C. S. Peskin. Implicit second-order immersed boundary methods with boundary mass. *Comput. Methods Appl. Mech. Engrg.*, 197:2049–2067, 2008.

[23] E. P. Newren, A. L. Fogelson, R. D. Guy, and R. M. Kirby. Unconditionally stable discretizations of the immersed boundary equations. *J. Comput. Phys.*, 222:702–719, 2007.

[24] C. Peskin. Flow patterns around heart valves: a numerical method. *Journal of Computational Physics*, 10(2):252–271, 1972.

[25] C. S. Peskin. Numerical analysis of blood flow in the heart. *J. Comput. Phys*, 25:220–252, 1977.

[26] J. M. Stockie and B. R. Wetton. Analysis of stiffness in the immersed boundary method and implications for time-stepping schemes. *J. Comput. Phys.*, 154:41–64, 1999.

[27] J. M. Stockie and B. T. R. Wetton. Stability analysis for the immersed fiber problem. *SIAM J. Appl. Math*, 55(6):1577–1591, 1995.

[28] J. Teran, L. Fauci, and M. Shelley. Peristaltic pumping and irreversibility of a stokesian viscoelastic fluid. *Physics of Fluids*, 20:073101, 2008.

[29] J. Teran, L. Fauci, and M. Shelley. Viscoelastic fluid response can increase the speed and efficiency of a free swimmer. *Physical review letters*, 104(3):38101, 2010.

[30] B. Thomases and M. Shelley. Transition to mixing and oscillations in a stokesian viscoelastic flow. *Physical review letters*, 103(9):94501, 2009.

[31] A.-K. Tornberg and L. Greengard. A fast multipole method for the three-dimensional Stokes equations. *Journal of Computational Physics*, 227(3):1613 – 1619, 2008.

[32] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan. A front-tracking method for computations of multiphase flow. *J. Comput. Phys.*, 169:708–759, 2001.

[33] C. Tu and C. S. Peskin. Stability and instability in the computations of flows with moving immersed boundaries: a comparison of three methods. *SIAM J. Sci. Stat. Comput.*, 13(6):1361–1376, 1992.